Improvement on Breast Cancer Cell Binary Classification

David Dai, Student Member, UCSD
Guanxin Li, Student Member, UCSD
University of California, San Diego
9500 Gilman Dr. , La Jolla, CA 92093

Abstract

Since the start of the twenty-first century, different implementations of neural networks have been implemented. One of the most breakthrough neural networks is the Convolutional Neural Network (CNN). Eager to improve the classification accuracy of whether the suspected breast cancer cells are invasive or non-invasiev, we decided to train the Breast Histology Images with different CNN implementations to test accuracy.

1.Introduction

In the current Medical field, deciding a cancer cell is benign or malignant takes more than a day to complete. It requires medical workers to take a sample from the suspicious cancer cell; then lab workers need to perform cell culture for more than 24 hours before deciding whether the cell is cancerous or not. If the cell is malignant, every minute and every second count; thus for this project, we are planning to train a Convolutional Neural Network to classify the cell is invasive or non-invasive within seconds.

Our program will focus on the diagnosis of breast cancer. The ultimate goal is to take an input of a 50*50*3 cell segmentation and classify it into invasive ductal carcinoma, which is a type of common breast cancer, or non-invasive cell. The training data is from *Kaggle*, with a dataset of 5547 images of size 50x50x3. And the data set contains labels of either positive (1) for invasive or negative (0) for non-invasive.

The following sections of this report are Methodology, Experiment, and Result. In the Methodology section, we will talk about how we select the best Convolutional Neural Network model to further improve the accuracy of the program. In the Experiment section, we are going to talk about how we choose the optimizer, learning rate and so forth to improve the accuracy of the CNN model we selected from the Methodology section. Then in the Result section, we are stating what we have learned from the program and whether our improved CNN model will perform better than machine learning algorithms.

2. Methodology

2.1 Testing and Training with Normalization

The dataset we found online regarding classifying breast cancer is invasive or non-invasive. And the dataset is set to be 5547 images of size 50x50x3. Because we want to select the batch size to be divisible for both the cardinality of the training data and the cardinality of the test data, we select the batch size to be 555. And because of our selection of the batch size, we decided to randomly select 3 images from the dataset and replicate

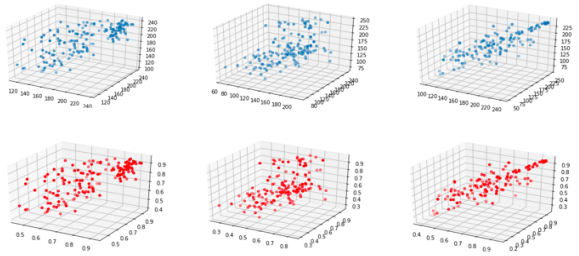
them to make a total of 5550 images of size 50x50x3. Then, for the testing and training data, we decided to randomly pick 1110 to be testing data and 4440 to be training data.

More importantly, since the overall properties of the images might be affected by a lot of outside factors, like the lighting when capturing the picture or shade of each person's cells due to different blood concentration. We decide to normalize the data according to the darkest and lightest pixel of each image to get the relative shading of each image to help us converge during the training process.

Using the training data as an example, the training dataset has a shape of 4440 x 50 x 50 x 3 and it has a range from the minimum value of 2.00 to a maximum value of 255. The dataset also has a mean of 185.18 and a standard deviation of 47.20.

If we did not normalize the input vectors, the range of distribution of our input values will be different for each feature. And since our ultimate goal is to train a Convolutional Neural Network, if we did not normalize the data, the back-propagation might adjust more weight values on one kind of weight dimension, and change less on a different kind of weight dimension. This will ultimately cause CNN to converge slowly or just unable to find the global/local maxima. Thus, we decided to normalize each input vector by dividing each input with its own largest value. This way, all the input vector will be in the range from 0.0078 to 1. As stated, after normalization, the training dataset has a min of 0.01, a mean value of 0.73 and a max of 1.00 with a standard deviation of 0.19. Figure1 below shows the result of the normalization.

Figure 1 *Visualization of random data points before (blue) and after (red) augmentation*



2.2 Performance Metrics:

The performance metric for our study is the testing dataset classification accuracy. We will also pay attention to the False Positive and False Negative ratio in the Experiment section.

Performance metrics (accuracy) has a range [0, 1], with higher the better. Since they are all in the same range, they are on the same comparable scales and we do not need to normalize them to compare them.

2.3 Baseline Algorithms:

Since our classification is binary (invasive or non-invasive), many of the machine learning algorithms can be applied to this case as well. One of the most common practices of the machine learning algorithm to do binary classifications is Logistic Regression. However, since our data input is an image. Thus performing logistic regression on the image data can be simplified if we flatten the data from a 3D array to a 1D array. With the corresponding 1D array and the training label, we performed the logistic regression and the result as in Table 1 below:

Baseline (LR)	**without normalization**	**with normalization**
ACC(%)	66.49	69.30

Table 1: ACC for LR with and without normalization

As shown above, we are going to set LR with normalization as the baseline result. In the next section, we will try to find methods that can perform better than this.

	Decision Tree	**Random Forest**	**Adaboost**	**Baseline (LR)**
ACC (%)	69.2	75.41	76.31	69.30

Table 3: ACC between different ML algorithm

2.4 Learning Algorithms:

2.4.1 NLP

The first idea that came to our mind is to improve on the logistic regression classifier with a little Natural Language Processing (NLP) technique. Using the previously flattened training set, we change the 1D float array into a string. Then, we introduced a text vectorizer to read each string in a trigram model, which basically is three numbers at a time. Then, with the expanded version of the training set, we decided to run logistic regression on it. Table 2 shows the result of the NLP approach.

	Vectorizer LR	**Baseline (LR)**
ACC(%)	66.04	69.30

Table 2: ACC comparison between NLP and LR

As the table has shown, the accuracy not only is lower than the logistic regression with normalization on the 1D array, it is also lower than the logistic regression without normalization. With the disappointing result, we decided to use of other machine learning algorithms to do binary classification.

2.4.2 ML Algorithms

Since in the previous part that we have already out our training data into a 1D array, we directly load them with training labels into different classifiers for reference.

Table 3 is our results machine learning algorithms after training on the normalized training set.

As shown in Table 3, random forest classifier and AdaBoost classifier from the sklearn ensemble learning classifier both generated results of over 75% accuracy on the test data. Thus, it puts pressure on us to train a Convolutional Neural Network that should perform even better than these ensemble learning classifiers.

2.4.3 CNN Networks

Then, the following are our testing stage of different kinds of Convolutional Neural Networks. For all the following CNN models, we decided to use a similar parameter. We decided to have the batch size as 555, the learning rate at 0.001, and the epoch as 500 iterations as default. We also picked Adam optimizer to update the parameters and weights of each layer. More detailed explanations will be in the "Experiment" section. The result of each CNN testing result will be listed in the table at the end of this section ("Learning Algorithms")

- *Alex Net*

Architecture: Alex Net consists of 5 Convolutional layers and 3 fully connected layers. All Convolutional layers are immediately followed by ReLu as the activation function. The first two and the last

layer are followed by Max Pooling layers. The output of the last layer would directly go to the fully connected layers, which also contains two Dropout and two ReLU layers in between.

Achievement: AlexNet dramatically scaled up the networks and demonstrated the ability to learn much more complex objects.

- *Mobile Net*

Architecture: Since we have a relatively small dataset, we think Mobile Net would be a good approach. Mobile Net is based on depth wise separable convolutions to build lightweight deep neural networks. All layers are followed by Batch Normalization and

ReLU with the exception of the finally connected layer which directly feeds into a softmax layer.

Achievement: Traditional nets are two big to run as they usually need to perform in the clouds. Mobile Net is very small, fast and easy to tune.

- *VGG*

Architecture: 3x3 convolutional layers stacked on top of each other in an increasing depth. Then the convolutional layers would be followed by ReLU and then Max-Pooling to reduce volume size. Here we trained VGG11 (with 11 weight layers) and VGG13 (with 13 weighted layers).

Achievement: Contradicts with the idea of that time to use large filters to capture features, VGG use multiple 3*3 convolution in sequence to emulate the effect of large receptive fields.

- *ResNet 18*

Architecture: Unlike other traditional sequential network architectures such as Alex Net and VGG, ResNet make use of the residuals functions and use a form of "network-in-network" architecture. Each ResNet block is 2 layer deep and each Convolutional layer is followed by a Batch Normalization.

Achievement: A great progress in solving the degradation problem where when the network depth increase, accuracy would be saturated and degrades rapidly.

2.5 CNN Model Selection

After we trained our data in different nets with our default setting, we figured out that the best result is the ResNet 18 model, as shown in the Table 4 below.

Table 4: ACC between different Neural Nets

	Alex Net	Mobile Net	VGG 11	VGG 13	ResNet 18	Baseline(LR)
ACC (%)	75	73	76	77	78	69.30

In the next section, we are going to tune this CNN model hoping to accomplish three goals.

1. Generate a better accuracy than the Baseline
2. Generate a better accuracy than the Adaboost Ensemble Classifier (AEC)
3. Reach a high accuracy on the test dataset around 80%

3. Experiment

3.1 Data set preprocessing

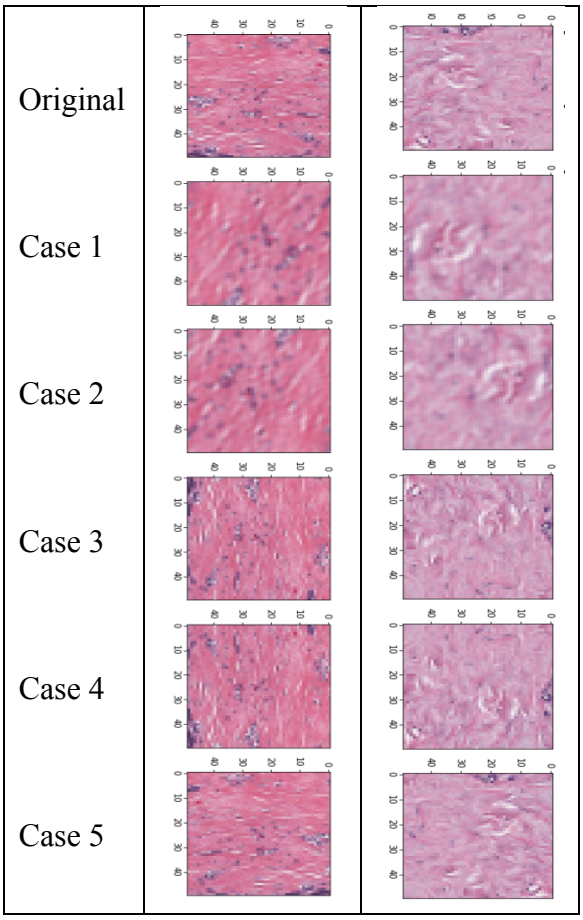
Since our dataset of 5550 images of size 50x50x3, we always encountered a problem that our dataset is way too small after the training and testing dataset split.

One of the most direct solutions is to do data augmentation. Using data augmentation can generate more relevant data in our training dataset. Since the property of a convolutional neural network, different orientations, translation, and the viewpoint will not affect the network to classify objects. These augmentation methods should be invariant to the result and the training label; however, at the same time, we create more different scenarios to feed into the Neural Network, ResNet 18 specifically.

As for the detail of the augmentation, we decided to do augmentation only on the training dataset. We replicate one image into 5 more different scenarios. The first case is to rotate the image clockwise by 60 degrees. The second case is to rotate the image counter-clockwise by 60 degrees. The third case is to rotate the image counter-clockwise by 90 degrees. The fourth case is to transpose the image, which is to flip the image diagonally. And the last case, the fifth case, is to rotate the transposed image by 90 degrees clockwise. The table with graphs below are two examples of the augmentation of image input.

	Positive	Negative	Total
Training Dataset before Aug	**2221**	**2219**	**4440**
Training Dataset after Aug	**13326**	**13314**	**26640**

Table 5: Training data before and after augmentation



And if we do the augmentation on the whole training dataset, we should expect the total number of data increase 6 times. As expected, Table 5 below is the result of the cardinality change of the training dataset after augmentation.

3.2 Optimizer

After augmenting the original dataset into a larger dataset, we can manipulate a different aspect of the ResNet 18. Optimizer, being the primary part to update the parameters and weights of the neural network, is extremely important. Thus, in this section, we will go through some of the optimizers we picked hoping to achieve good results.

- *SGD*

SGD stands for Stochastic Gradient Descent. The general equation is $\theta = \theta - \eta \nabla_\theta J\left(\theta, x^{(i:i+n)}, y^{(i:i+n)}\right)$ It takes the normal gradient descent but selects a random batch of samples per iteration to update the parameters. With the usage of momentum, the SGD method converges a lot faster than regular gradient descent. More importantly, it has a higher chance converge to the local min or local max. However, SGD does not have adaptive learning rate, and we need to manually pick a good learning rate for convergence. And more importantly, if there is a saddle point, it is still hard for SGD to get out even with the application of momentum.

- *AdaDelta*

Adadelta takes adaptive learning rate into consideration, which solves one of the problems that SGD is encountering. AdaDelta is fixing the accumulation of the squared gradient of the denominator. It is a good optimizer for natural language processing, but not a well known optimizer for image neural network. With the default parameters from AdaDelta of pytorch, the loss function will not decrease in our case. As we later manipulate different initial learning rates and other different values, the loss function did not seem to converge.

- *Adam*

Since Adam is the most common optimizer, it stands for Adaptive Moment Estimation. It is extremely useful to control the momentum not going too fast or too slow. And with the adjustment of the momentum, despite a stable the learning rate, it alleviated the problems we faced with SGD. However, the only drawback for Adam optimizer is that the learning rate have to be chosen on our own.

With all of the three optimizers in pytroch default setting, we started to train the ResNet. Since AdaDelta did not converge, we eliminate this optimizer from the selection pool. Lastly, since Adam converges faster and creates slightly better accuracy than SGD, we decided to use Adam optimizer. And the general loss function is attached on the side. Table 6 below shows our accuracy of the ResNet with different optimizers compared to Baseline.

	SDG	**Adam**	**Baseline**
Epoche to converge	~ 45	~ 85	---
ACC(%)	78	82	69.30

Table 6: ACC for ResNet under different optimizers

3.3 Hyper-parameter Tuning

After selecting the best optimizer for training our neural network, we want to maximize the training efficiency and continue to increase the accuracy.

3.3.1 Learning rate

As stated in the section above, Adam optimizer has to manually select the best learning rate. We have selected from the following list [0.01, 0.001, 0.0005 0.0001] for our learning rate.

And as we found out during the experiment, 0.01 is too big for the learning rate to actually generate a good quality result. 0.001 is also rather large. However, 0.0001 is too slow for convergence. Thus, we decided to pick the learning rate to be 0.0005. Table 7 summarize our results.

	0.01	**0.001**	**0.0005**	**0.0001**
ACC(%)	---	80	82	----

Table 7: ACC for ResNet under different learning rate

3.3.2 Epoch number

After select a reasonable learning rate, we decided to pick the right amount of iterations for the training. If the number of iterations is too small, we will end the training before the loss function even converges. On the other hand, if the number of iterations is too large, we will waste valuable time not making significant improvements on the parameter. Thus, we tried many different epoch number from 10 to 1000. And after different experiments, we found that for ResNet 18, 100 epochs is the most efficient. It decreases the loss from 0.813 to 0.000.

3.3.3 MaxPool vs AvgPool ResNet

In this part, we decided to change the pooling layer of the ResNet 18 to see the effect of the alteration. In the ResNet 18's forward section, it is assigned with an average pooling layer. And in this experiment, we decided to alter the average pooling to max pooling. Max pooling is useful preventing localization. Both max pooling and average pooling are effective deducting computational complexity and variance. Max pooling is better at selecting extreme features and average pooling is a more smooth approach. However, in this case, it turns out that Average Pooling, AvgPool, performs better than MaxPool, shown as in Table 8 below

	Max Pooling	**Average Pooling**
ACC(%)	84	84

Table 8: ACC for ResNet under different pooling

3.3.4 ReLU vs Sigmoid ResNet

In this part, we decided to experiment with different activation function. In the forward step, it has ReLU as the activation function, and we decided to substitute it with sigmoid to see the result. ReLU is the most used activation function. ReLU is half rectified ranging from 0 to infinity. The function and its derivative are both monotonic. And sigmoid is one of the oldest activation functions. It is differentiable and monotonic, but its derivative is not. It may cause the neural network to get stuck at a certain location. Then the result of both ReLU and Sigmoid activation with AvgPooling has shown that ReLU performs significantly better, shown as in table 9 below.

	0.01	**0.001**	**0.0005**	**0.0001**
ACC(%)	---	80	82	----

Table 9: ACC for ResNet under different activation function

3.4 Selective Augmentation

From a medical perspective, it is really scary to miss diagnose a patient who actually have cancer. Thus, we actually want to have low false negative rate. One of the ideas that we think that can reduce the false negative rate is to slightly increase the number of positive cases. This way, with more information regarding positive cases, the neural network should capture more features about it and thus decrease the false negative rate.

Then, it came to the question of which data do we want to select to further augment the amount of data. As previous sections have indicated with the flattened data, adaboost

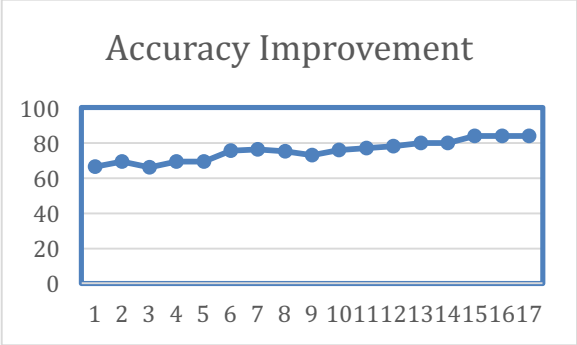
ensemble classifier has the highest accuracy rate on the testing dataset. Thus, we rearrange the flattened training dataset from the highest to the lowest.

Then, for each positive and negative case, we pick a different range of confidence levels to further augment the dataset. For the positive case, we decided to pick confidence range from 60% to 85%. If we pick the confidence below 50%, the label might not be really solid and the pictures might not have identifiable features. If we pick the confidence above 85%, then we are afraid that they might be misleading as well. As for the negative case, we decided to do the same thing to augment the dataset, because we want to keep the dataset labels balanced.

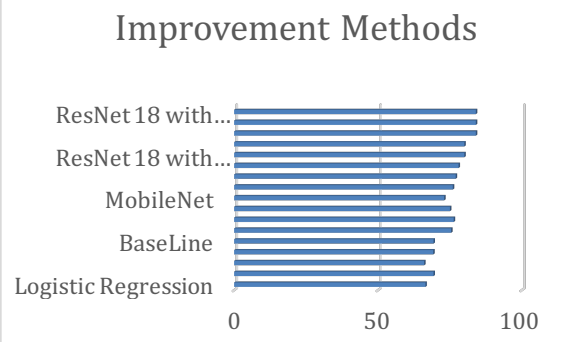
As a result, if the image with lab is within the 60 percent to 85% confidence range, we augment one image into 10 different images with different orientation, flip and zoom. If the image with label, however, is not in the confidence range, we still augment them by 6. The augmentation algorithm has been described above. The new training dataset is now having 34070 number of images.

4. Results

In conclusion, we achieved all three goals that we set which is to perform better than the Baseline logistic regression algorithm. More importantly, we actually further improved the accuracy from about 69 percent to 84 percent accuracy on the testing dataset. For preprocessing the data, we did data normalization and data augmentation. After comparing different neural nets with a raw default setting, we decided to use ResNet (with 78% acc) as our base neural network. After comparing between different optimizers and tuning hyper-parameters, we trained our ResNet under Adam with a 0.005 learning rate, which gives us an accuracy of 84% on the test data as the best performance. And the accuracy improvement graph is graph representation of the accuracy improvement.



Here is the table of each different iteration



5. Reference:

"Torchvision.models¶." *Torchvision.models - PyTorch Master Documentation*, pytorch.org/docs/stable/torchvision/models.html#torchvision.models.mobilenet_v2.

Douillard, Arthur. "3 Small But Powerful Convolutional Networks." *Towards Data Science*, Towards Data Science, 13 May 2018, towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d.

Sirena. “Optimizers for Training Neural Networks.” *Medium*, Data Driven Investor, 28 July 2018, medium.com/datadriveninvestor/optimizers-for-training-neural-networks-e0196662e21e.

Digital Histology. “Use Case 6: Invasive Ductal Carcinoma (IDC) Segmentation.” *Andrew Janowczyk*, 5 Jan. 2018, www.andrewjanowczyk.com/use-case-6-invasive-ductal-carcinoma-idc-segmentation/.