Breast Cancer Detection Classification Analysis

Milestone 2

Aquincum Labs

Isaac Klar - Mingi Kang - Karam Al-Askar - Shaamil Karim

Related Works

Janowczyk and Madabhushi's work on deep learning for digital pathology image analysis provides significant insights into our project. They employed deep learning techniques to analyze whole slide images, specifically focusing on Invasive Ductal Carcinoma (IDC) segmentation. Their approach is somewhat different from ours, as their goal was not only to classify IDC-positive and IDC-negative images but also to segment the exact regions containing the tumor and assign an aggression score. Their model used a stochastic gradient descent method, iterating over image patches and adjusting the network weights through backpropagation. This training approach helped the model learn to identify IDC regions more accurately.

While their work involves a more complex task of segmentation, their methodology offers valuable lessons for our project. The key takeaway is their use of a fixed batch size, learning rate annealing, and image patch-based model training, which could be applied to improve the accuracy and efficiency of our own classification model. Though our task is a binary classification (IDC-negative vs. IDC-positive), Janowczyk and Madabhushi's approach to image patch handling and model training is relevant and beneficial to our understanding of deep learning applications in pathology.

Data Understanding and Preparation Steps

The dataset for this project, obtained through KaggleHub, required careful restructuring to align image paths with the necessary classification labels. We utilized the glob library to systematically retrieve paths for IDC-negative and IDC-positive images, ensuring that we had organized data for both categories. The directory structure was as follows:

- /#### represents the patient ID folder,
- /0 or /1 indicates whether the image belongs to the IDC-negative or IDC-positive category,
- ####_idx#_x##_y####_class# represents the specific image file.

Using glob.glob(), we successfully collected all image paths for both categories, preparing them for further preprocessing.

Data Preprocessing

The raw image data needed to be converted into a format suitable for input into our convolutional neural network (CNN). We used the OpenCV library for image manipulation, performing several steps to ensure that all images were ready for the CNN model:

1. Reading images from their paths,
2. Color conversion to ensure proper RGB representation,
3. Resizing all images to 50x50 pixels for consistency,
4. Normalization by scaling pixel values between 0 and 1 by dividing by 255.

We created an image_to_numpy function that took the image paths as input and returned NumPy arrays of shape (50, 50, 3), suitable for feeding into the CNN. Additionally, we implemented an image visualization function using matplotlib to verify the integrity of the preprocessed images.

Choice of Data Format
Initially, we considered converting the images into tensors for input to the CNN. However, given the computational limitations of our environment, we chose to use NumPy arrays for their computational efficiency. This approach not only reduced memory consumption but also aligned with our familiarity from prior coursework, enabling smoother implementation.

Dataset Subsetting and Splitting
The dataset contained approximately 200,000 images, but due to resource limitations, we began by testing with a smaller subset. For initial experiments, we selected the first 2,000 images from each category (IDC-negative and IDC-positive). After preparing the features and labels as NumPy arrays, we used train_test_split from sklearn.model_selection to divide the data into an 80% training and 20% testing split. This subset allowed us to quickly iterate and evaluate the model before scaling to the full dataset.

Initial Model Creation & Evaluation
For the initial model, we adapted the deep learning architecture from our digit recognition task (HW4). The main modification was changing the number of output classes from 10 (for digits 0-9) to just 2 outputs (IDC-negative and IDC-positive). The activation function in the dense layer was switched to sigmoid, and the loss function was changed to binary cross-entropy to accommodate binary classification.

The initial model achieved around 80% accuracy after sufficient epochs of training on both the training and testing datasets. The model generated a probability score for each image, classifying it as IDC-negative if the probability was below 0.5 and IDC-positive if the probability was above 0.5. Although this model showed decent results, we identified areas for

improvement, such as experimenting with different activation functions, optimizers, and thresholds for positive classification.

More Data Analysis Steps, Implementing Models, and Evaluating Them

While the initial model achieved a respectable accuracy, we recognized that there was still room for improvement. This performance, although satisfactory, highlighted the complexity of IDC detection in medical images and prompted us to explore more advanced architectures and techniques. Through further research, we identified a deep learning model specifically designed for IDC detection, which had already demonstrated impressive results in similar tasks. This discovery offered a promising foundation for further optimization and adaptation to our project.

We adopted this model and made several key modifications to improve its performance. The first change was the introduction of multiple convolutional layers with progressively increasing depth. The first layer used 32 filters, the second 64 filters, and the third 128 filters. Each layer applied ReLU activation, followed by max pooling and dropout to combat overfitting. By increasing the depth of the network, the model could learn more complex features of the images, improving its ability to differentiate between IDC-positive and IDC-negative images.

Another key modification was the use of Global Average Pooling instead of the traditional flattening layer after the convolutional layers. This technique reduces the dimensionality of the feature maps while maintaining the essential information, making the network more efficient by decreasing the number of parameters.

We also incorporated L2 regularization in the fully connected layers to further mitigate overfitting and added a dropout layer (rate = 0.5) after the dense layer to ensure robust training. The output layer used a sigmoid activation function to perform the final binary classification.

To further improve the model's performance, we applied data augmentation techniques such as image rotation, flipping, and scaling. These techniques artificially increased the size and diversity of the training data, helping the model generalize better to new, unseen images.

After these modifications, the model's performance improved significantly, achieving an accuracy of around 94%. This increase in accuracy was accompanied by a detailed analysis of other metrics, including precision, recall, and the confusion matrix. These evaluations revealed specific areas where the model performed well and where it still had room for improvement, such as misclassifying some IDC-negative images as positive.