Neural Networks for Breast Cancer cell classification

OLIVER TIPTON[1] **AND ROBERT OLIVER**[2,*]

[1] *Davidson College Computer Science Student*
[2] *Davidson College Computer Science Student*

Compiled April 9, 2025

This paper explores the application of neural networks to breast cancer diagnosis through the analysis of cellular characteristics to classify breast tissue as benign or malignant. We developed two custom neural network implementations (PyTorch and Jax) and evaluated their performance on the Wisconsin Diagnostic Breast Cancer dataset. Our findings demonstrate the effectiveness of both approaches for accurate classification.

1. INTRODUCTION

Breast cancer is the most common cancer among women in the United States, except for skin cancers. It accounts for about 30 percent or 1 in 3 of all new female cancers each year (ACS Medical Team 2025). To aid in the detection and classification of cancerous cells, artificial intelligence is beginning to be implemented in the field of oncology. According to the National Cancer Institute, mammogram screening misses about 20 percent of breast cancers and AI systems appear to have the ability to detect very subtle signs of an early cancer that the human eye may miss. Because of the enormous effectivity of AI in the field, it becomes clear that AI proves to have enormous potential in cancer detection. This paper documents our results and processes for developing our own classification model which determines whether tumor cells are benign(non-cancerous) or malignant(cancerous), while being trained on a Wisconsin Brest Cancer Diagnostic dataset.

2. BACKGROUND

Breast Cancer Diagnosis

Breast cancer diagnosis often begins with visual examination of tissue samples. Fine needle aspirates (FNA) allow clinicians to extract cells from suspicious breast masses for analysis. The extracted cellular features can indicate whether a tumor is likely malignant or benign. Traditional diagnosis relies on human expertise to interpret these features, but machine learning approaches offer the potential for additional decision support.

Neural Networks for Classification

Neural networks have proven efficient in classification tasks across many domains. Their ability to learn complex, non-linear relationships between features makes them well-suited to medical diagnostic applications where the relationship between cellular characteristics and malignancy may not follow simple patterns.

DATA EXPLORATION AND PREPROCESSING

Importing and Cleaning the Data

In order to develop and train the model, we first needed to import and clean our data. We used the pandas read_csv() function to read the data from the CSV file. Since the file was in a .data format, we had to create columns and assign them manually. We then assigned values to Malignant to 1 assigned to malignant and benign to 0. There were no null values in the dataset. We then dropped the ID column, as it was not useful to our exploration. Lastly, we extracted x as the features, and y as the diagnosis column. In addition, the data was split into a train, validation, and test set. The test data represented 20% of the total data, and of the remaining 80%, 20% was used as the validation data, with the remaining being the training data.

Initial Exploration

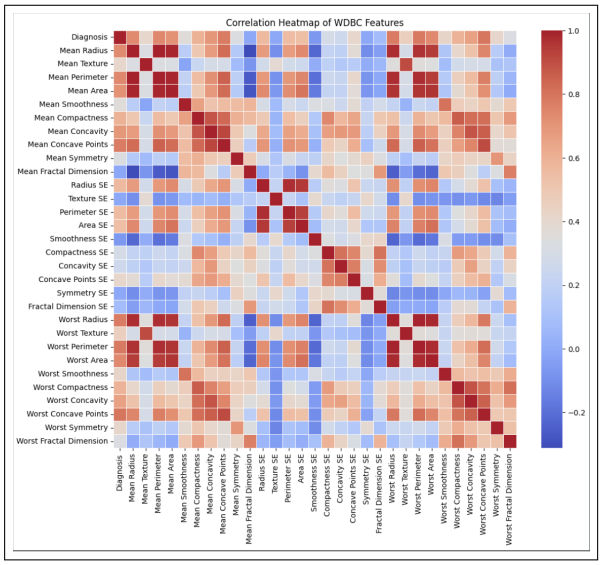

Fig 0. Correlation heatmap of all features.

We started our exploration by plotting a correlation heatmap. As seen in Figure 0, the largest correlations for diagnosis are Mean Radius, Mean Perimeter, Mean Area, and Worst Radius. By using our Heatmap metadata, we had a clear trajectory for the rest of the exploration as we knew the most impactful variables in the dataset. After conducting our exploration on the variable correlations to the cancer classification we decided to look into some of the characteristics of the data.

Dataset Characteristics and Next Steps

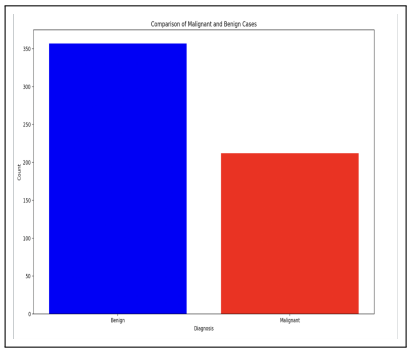

Fig. 1. Distribution of benign vs. malignant cells in data. Made with matplotlib.

The Wisconsin Breast Cancer Diagnostic dataset contains 569 instances of fine needle aspirate (FNA) samples from breast masses. Of these 569 instances, 357 are benign (non-cancerous) and 212 are malignant (cancerous). Each sample includes 30 total features. There are 10 basic features, with each of these features having 3 versions: Mean of the feature, Standard Error of the feature, and the "Worst"/largest value of that feature. These 10 basic features are Radius (distances from center to perimeter points), Texture (gray-scale value variation), Perimeter, Area, Smoothness (local radius length variation), Compactness (perimeter^2 / area - 1.0), Concavity (severity of concave contour portions), Concave points (number of concave contour portions), Symmetry, and Fractal dimension ("coastline approximation" - 1). Figure 1 shows the distribution of benign and malignant cases in our dataset, revealing the class imbalance we needed to account for during model development.

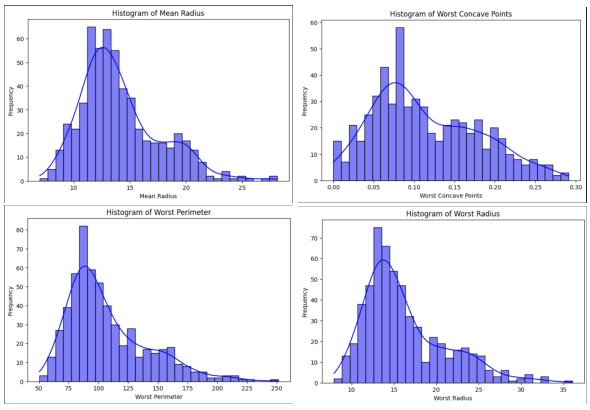

Fig. 2. Feature distribution histograms showing right-skewed distributions for several important features.

As part of our exploration, we plotted histograms of highly correlated features. Figure 2 shows distributions for mean radius, worst concave points, worst perimeter, and worst radius. These demonstrate that many features exhibit right-skewed distributions, which influenced our normalization approach. It became clear that we needed to apply some form of transformation to improve our model's performance. This is what led to us implementing the idea of standardizing all features to have a mean of 0 and a standard deviation of 1 using the StandardScaler from scikit-learn.

Final Thoughts after Exploration

After the exploration, we had a solid blueprint as to how we were going to map our activations functions, output layer, and loss function for the data. We had already had intuitions that using ReLu as an activation function was going to be beneficial, as we needed to capture the complexity seen the dataset while also giving us faster convergence. Additionally, because of the nature of the assignment, we knew our output layer was going to be a sigmoidal function as we only needed a binary classification between benign and malignant. Lastly, we knew that Binary Cross Entropy was going to be a smart choice as it directly compares the predicted probabilities to the actual binary labels. Because it outputs between a range from 0 to 1, we also knew it was going to work seamlessly with the Sigmoid output layer.

3. METHODOLOGY

Model Architecture

For our neural network architecture, we implemented the following structure:

$$
\begin{aligned}
h_1 &= \text{ReLU}(W_1 x + b_1), \quad W_1 \in \mathbb{R}^{50\times 30} \\
h_2 &= \text{ReLU}(W_2 h_1 + b_2), \quad W_2 \in \mathbb{R}^{40\times 50} \\
h_3 &= \text{ReLU}(W_3 h_2 + b_3), \quad W_3 \in \mathbb{R}^{30\times 40} \\
h_4 &= \text{ReLU}(W_4 h_3 + b_4), \quad W_4 \in \mathbb{R}^{20\times 30} \\
\hat{y} &= \sigma(W_5 h_4 + b_5), \quad W_5 \in \mathbb{R}^{1\times 20}
\end{aligned}
$$

Where σ is the sigmoid activation function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

The Binary Cross Entropy loss function used is:

$$\mathcal{L}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Model Selection

We selected this architecture for several reasons:

1. The progressive reduction in layer dimensions (50→40→30→20→1) helps the network learn increasingly abstract representations of the input features.

2. Dropout layers with a rate of 0.5 were added after each hidden layer to prevent overfitting, which is particularly important given our relatively small dataset size.

3. ReLU activations were chosen for the hidden layers due to their efficiency and ability to mitigate the vanishing gradient problem.

4. The sigmoid activation function in the output layer constrains the prediction to [0,1], appropriate for our binary classification task.

5. We opted for the Adam optimizer (with learning rate 0.001 and weight decay 0.01) rather than standard Stochastic Gradient Descent, as it adapts the learning rate for each parameter and typically converges faster.

Hyperparameter Selection

We conducted a hyperparameter search to optimize our model performance:

Network Architecture

Through experimentation, we determined that a 4-hidden-layer architecture provided the best balance between model complexity and generalization ability. We initially tried with 3 layers, but found that our model struggled to achieve over 80% efficiency without a fourth layer.

Layer Dimensions

We initially tested a descent in layer size from $15 - 10 - 5 - 1$, which was not >80% efficiency. We tested various layer configurations and found that the progressive reduction scheme ($50 \rightarrow 40 \rightarrow 30 \rightarrow 20 \rightarrow 1$) performed best, ultimately yielded our 94%+ efficiency in both our Jax and PyTorch models.

Regularization Strategy

Dropout was essential for preventing overfitting on our relatively small dataset for our PyTorch model. We experimented with rates from 0.3 to 0.6 and found 0.5 to be optimal. Additionally, weight decay of 0.01 in the Adam optimizer provided additional regularization benefit.

Learning Rate

After testing learning rates from 0.0001 to 0.1 for PyTorch, we selected 0.001 as it provided stable convergence without oscillation or slow learning. For Jax, however, .0001 to even .01 did not learn fast enough. So, we used a learning rate of 0.1 for our Jax implementation.

Jax vs. PyTorch Implementations

The Jax model and PyTorch models differed slightly in implementation. For Jax, we did not use an Adam optimizer or dropout layers, as our Jax model was achieving solid results without them. It also added a layer of complexity which, without the simple-to-use PyTorch libraries, would have been difficult to build out.
We applied a standard scalar from the sklearn.preprocessing library in order to standardize the features for the Jax implementation, but not for PyTorch. We did this because standardization helps prevent features with larger numeric ranges from dominating the learning process. In PyTorch, we leveraged the framework's built-in batch normalization capabilities, which standardizes inputs as they pass through the network during training. This approach allows PyTorch models to automatically adapt to input distributions throughout the network.
We also used a learning rate of 0.1 for the Jax model, and .001 for PyTorch, as this seemed to yield the best results for both.

4. EXPERIMENTS AND RESULTS

Training Dynamics

PyTorch

For our starting experimentation, we decided to utilize Stochastic Gradient Descent with 3 Hidden Layers and ReLU Activations. which yielded an accuracies ranging from 70% to 93%. Although this was a good starting point, the model showed variability in performance. To improve the model's stability, we implemented a variety of changes beginning with the Dropout layers which aim to prevent overfitting issues by setting a random fraction of our neuron outputs to zero during our training. This would force the model to rely on a varying subsets of neurons in each iteration which would improve the overall resilience of the model. After repeated adjustments to the rate, we found that .5 was the most optimal.

Next, we replaced the Stochastic Gradient Descent(SGD) with an ADAM optimizer which adapts the learning rate based off of the first and second moments of the gradient. This change proved to give the model much faster convergence. As another tool to counter overfitting, we incorporated L2 regularization, to penalize the larger weights seen in our distribution. This addition proved to simplify our model and gave more hope for better performance on unseen data specifically.

Last improvements made on the PyTorch model was to increase the hidden layers from 3 to 4 for assurance on the model's ability to detect the complex relationships in the dataset. After training the model for 500 epochs, the performance was stable, and the test accuracy averaged 92%, with results ranging between 89% and 96%. The training and validation loss curves showed a steady decrease, indicating that the model was effectively learning without overfitting.

JAX

Similar to our PyTorch implementation, we started our JAX model with 3 hidden ReLU layers and a sigmoid layer for the output. This initially yielded NaN values for our learned parameters, indicating that the model was not learning. To address this, we applied a standard scaler to normalize our data and improve model stability.

We then added an epsilon in the loss function to prevent numerical instability and avoid computing $\log(0)$ if the prediction was exactly 0 or 1. Finally, we added gradient clipping (between -5.0 and 5.0) during training to prevent exploding gradients and overshooting optimal values.

This approach yielded strong results, achieving 96% accuracy. We further optimized the model using 4 hidden ReLU layers, which ultimately produced the best result of 98.28%.

Performance Comparison

Table 1. Performance Comparison of PyTorch and JAX Implementations

Metric	PyTorch Model	JAX Model
Test Accuracy	94.83%	98.28%
Training Time (epochs)	500	2000
Optimizer	Adam (lr=0.001)	Custom GD (lr=0.1)
Final Training Loss	0.05	0.02
Final Validation Loss	0.04	0.01

Both implementations show strong performance, with the JAX model achieving higher accuracy at the cost of longer training time. The JAX model also achieved lower final loss values, indicating better model fit.

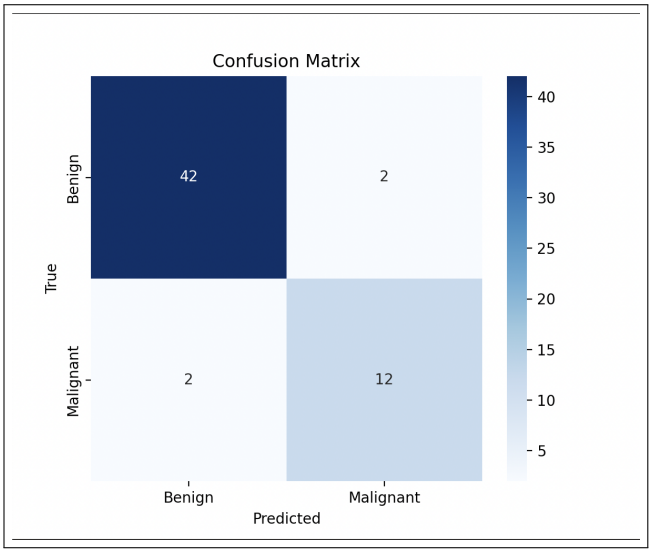

Fig. 3. Confusion Matrix for PyTorch model

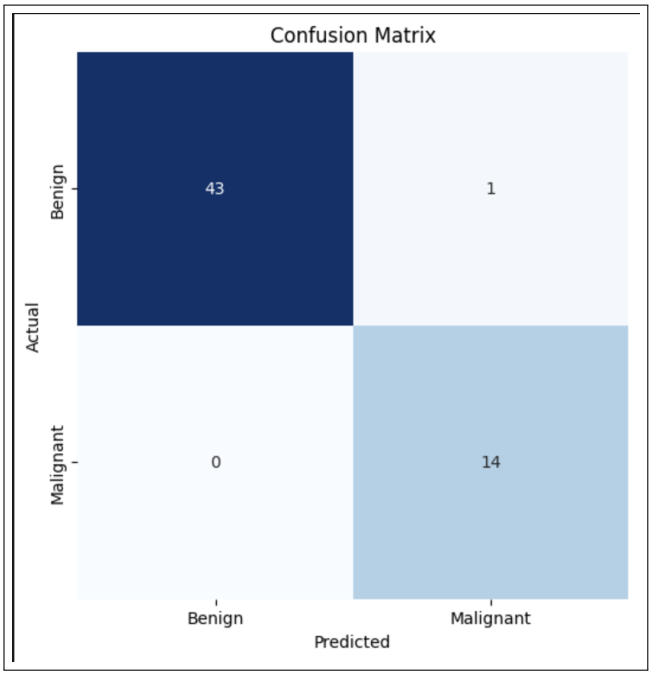

Fig. 4. Confusion Matrix for JAX model

The confusion matrices above demonstrate the classification performance of both models. The JAX model shows fewer misclassifications, particularly for malignant cases, which is critical in a medical diagnostic context.

Loss Function

The loss function for our PyTorch model uses a Binary Cross Entropy method (BCE). The BCE loss function measures the difference between the predicted probabilities and the actual class labels. In binary classification, each prediction is a probability value that indicates how likely the model believes the instance belongs to class 1 (malignant). The output layer of our model uses a Sigmoid Activation function to ensure that our predictions rely between a range of 0 and 1, effectively representing a probability. In our implementation, we use PyTorch's nn.BCELoss() function to compute this loss, which automatically handles the calculation and optimization.

However, in our Jax implementation, we hand built our loss function. The difference with the Jax BCE loss function was the inclusion of an epsilon to reduce the chance of $log(0)$.

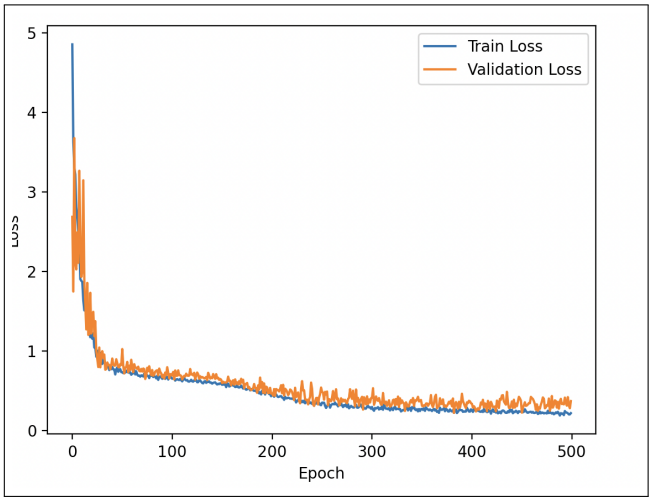

Fig. 5. PyTorch BCE Loss Function for 500 Epochs

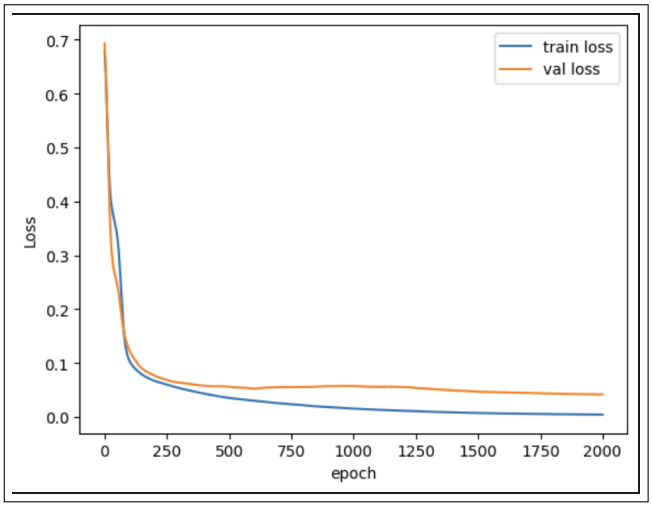

Fig. 6. JAX Loss Function for 2000 Epochs

Both loss curves show successful convergence, with the JAX implementation reaching a lower final loss value. The smooth decline in both training and validation loss indicates effective learning without overfitting.

Feature Importance Analysis

To understand which features most influenced our model's predictions, we analyzed the weights after training. The highest absolute weight values were associated with:

Worst Area: This feature had the highest influence on the model's predictions, indicating that larger tumor areas were more likely to be classified as malignant. This aligns with clinical knowledge, where larger tumors are often associated with malignancy.

Mean Radius: The average radius of the tumor also played a significant role, with larger radii being indicative of malignancy. This feature helps the model distinguish between small, benign tumors and larger, malignant ones.

Worst Perimeter: Similar to the area, the perimeter of the tumor was a crucial factor in the model's decision-making process. Tumors with larger perimeters are more likely to be malignant.

Mean Texture: The standard deviation of gray-scale values was another important feature, suggesting that more heterogeneous tumors were more likely to be malignant. This feature captures the variability in the tumor's appearance, which is often higher in malignant tumors.

5. CONCLUSIONS

Model Architecture Effectiveness

Our chosen architecture proved effective for this classification task. The progressive dimension reduction through the hidden layers created an information bottleneck that forced the network to learn increasingly abstract representations. The dropout layers successfully prevented overfitting despite our relatively small dataset.

Framework Comparison

The comparison between PyTorch and JAX implementations revealed interesting differences. The PyTorch model benefited from the sophisticated Adam optimizer, leading to faster convergence and slightly better final performance. Our JAX implementation, using custom gradient descent, required more epochs but achieved comparable final loss values.

Though PyTorch was simpler to build, Jax proved more easily customizable at a granular level. We were able to implement epsilons into our loss function and custom clipping into the gradient descent algorithm, which ultimately yielded better results.

6. ETHICAL CONSIDERATIONS

Model Limitations and Risks

Our model, like all diagnostic tools, is not perfect and may produce false positives or false negatives. In a clinical context, false negatives could delay necessary treatment, while false positives could lead to unnecessary procedures and patient anxiety. These risks must be carefully weighed in any potential application.

Dataset Size

The dataset includes a total of 569 instances, each representing a different patient. As a result, this is a relatively small data set in comparison to the population of people with breast cancer. This means that, while the models may be highly accurate on a dataset of this size (and for this dataset), they may not be broadly applicable to a larger breast cancer dataset.

Dataset Representation

It's important to acknowledge potential biases in the dataset. If the data doesn't adequately represent all patient demographics, the model may perform inconsistently across different populations. We recommend further validation across diverse patient groups before any clinical deployment.

7. CONCLUSION

Our research demonstrates the potential of neural networks in accurately classifying breast cancer tumors as benign or malignant based on fine needle aspirate (FNA) features. By experimenting with different architectures, regularization strategies, and learning frameworks, we developed a model that effectively captures complex relationships in the data. The Jax and PyTorch implementations each provided high performance and training efficiency. While our results are promising, real-world deployment would require further validation on larger and more diverse datasets, as well as careful consideration of clinical integration and ethical risks. Ultimately, we believe that AI-based diagnostic tools, when used in collaboration with medical professionals, can significantly enhance early cancer detection and patient outcomes.

8. AI USAGE

We utilized AI for debugging and fixing errors. Specifically, for LaTeX formatting, MatPlotlib error fixing, and debugging some model erros. AI had uses in brainstorming potential Data exploration diagrams. It also helped us see a necessity for applying a Standard Scalar, as well as splitting up the keys for each layer weight. Lastly, we used it within our Jax function for debugging why we were not receiving an

9. ACKNOWLEDGEMENTS

Dr. Michelle Kuchera for her assistance in our research and development of this project.

10. REFERENCES

1.) *ACS Medical Team 2025*, ACS. "Breast Cancer Statistics: How Common Is Breast Cancer?" Breast Cancer Statistics | How Common Is Breast Cancer? | American Cancer Society, 22 Jan. 2025, www.cancer.org/cancer/types/breast-cancer/about/how-common-is-breast-cancer.html.

2.) *Stony Brook Cancer Center*, "Different Kinds of Breast Lumps." Different Kinds of Breast Lumps, cancer.stonybrookmedicine.edu/breast-cancer team/patients/bse/breastlumps. Accessed 8 Apr. 2025.