Course Code: COMP3220

ID: 620164974 & 620157584

1. Introduction

The goal of this project was to build machine learning models to predict whether a patient has heart disease using the provided dataset. The dataset included clinical features such as age, cholesterol level, chest pain type, resting blood pressure, and more. By analyzing these features, the models aimed to classify patients into two categories: heart disease present (1) or absent (0).

2. Methodology

2.1 Data Preprocessing and Normalization

The dataset heart_train.csv was loaded with pandas, and the first few rows were inspected with df.head(). A histogram of the HeartDisease column was generated to visualize the distribution of class values and ensure the data was binary (i.e., a patient either had heart disease or did not). One key note was that the dataset was imbalanced, with more positive (1) cases than negative (0). The dataset was checked for null values, and categorical columns were converted to numerical values using one-hot encoding for uniformity. StandardScaler was then implemented, which standardizes data or scales each feature to have a mean of zero (0) and a standard deviation of one (1). Additionally, data normalization was performed, in which the numerical features in a dataset are scaled to a consistent range. Finally, the HeartDisease column was selected as the target label, and the remaining processed columns formed the feature set for model training. The processed data was then split into 80% for model fitting and 20% reserved for unbiased evaluation.

2.2 Model Architectures

In creating the architectures, three distinct models were implemented to evaluate performance:

A. Logistic Regression (Baseline)

A standard Logistic Regression classifier was trained using scikit-learn. This serves as a linear baseline to determine if the complexity of a neural network is necessary for this problem. Because it is simple and easy to interpret, logistic regression is widely used in areas such as healthcare, finance, and marketing. It works well for problems where the goal is to distinguish between two categories, such as yes/no or 1/0 decisions. A logistic regression classifier was created using: LogisticRegression(random_state=42, max_iter=1000).

B. Small Neural Network

Input: 20 features

Hidden Layer 1: 16 units

Hidden Layer 2: 8 units

Output: 1 unit

C. Large Neural Network

Input: 20 features

Hidden Layer 1: 128 units

Hidden Layer 2: 128 units

Output: 1 unit

Both neural networks were trained using the Keras/Tensorflow framework, with consistent hyperparameter configuration to ensure a fair comparison:

Hyperparamter	**Value**
Optimizer	Adam
Activation Function	Tanh
Loss function	Binary Crossentropy
Batch Size	1
Epochs	100

3. Results and Data

Table showing test accuracy and test loss for three model architectures described in Section 2.2

Model Type	**Test Accuracy**	**Test Loss**
Logistic Regression	0.9000	N/A
Small Neural Network	0.8415	0.4013
Large Neural Network	0.8232	1.1532

4. Discussion

4.1 Importance of Normalization

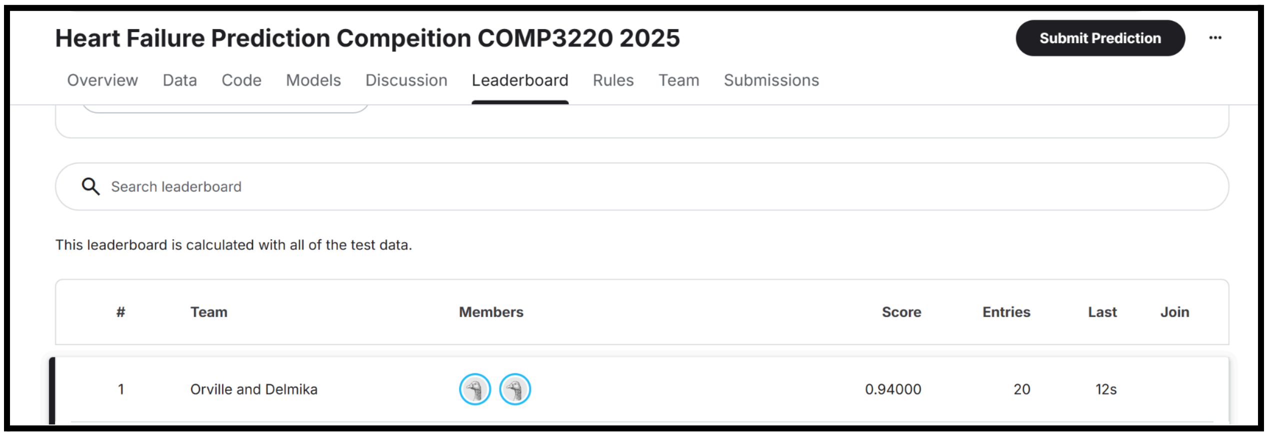
Data normalization was necessary because normalized data is easier to interpret and understand. When all features are on the same scale, the relationships between features become more explicit, enabling more meaningful comparisons. Similarly, normalizing data improves the performance and the accuracy of a model by preventing features with larger scales from dominating the model. For example, in the dataset, without normalization, the age range is fifty (50), the cholesterol range is six hundred (600), and the maxHR range is one hundred and forty (140). Without normalization, the algorithm might weigh features with larger scales more heavily. For instance, the feature with the larger range, like cholesterol, would dominate the distance calculations and gradient updates. Unnormalized data creates an elongated loss landscape. Gradient descent struggles to find the minimum, bouncing back and forth. Meanwhile, normalization makes the landscape spherical, allowing the optimizer to converge much faster. Moreover, large input values can lead to exploding gradients or saturated activation functions (such as Sigmoid saturating at 1.0), preventing the network from learning. As such, normalization is necessary to preserve proportionality and give a more accurate representation of the data without neglecting the smaller features that introduce nuance. It also ensures that the optimization process is more stable, allowing for the optimal solution to be found quickly. Without normalization, there is a higher risk of exploding gradients, where model updates get too big, and the model can not find the right solution because it keeps overshooting. There is also a risk of vanishing gradients, where the model stops learning because changes are too small to matter.

4.2 Overfitting Neural Networks

The experimental data were indicative of overfitting in the large neural network. While the small neural network achieved 84.15% accuracy with a loss of 0.40, increasing the hidden layer neuron count caused the performance to degrade significantly, with the accuracy dropping to 82.32% and the loss exploding to 1.15. The reason for this is likely because the dataset has approximately 800 entries, which is too small to support such a massive architecture. As a result, the overfitted model memorizes the training data rather than learning generalized patterns, and begins to learn overly specific patterns that do not apply to new inputs. Furthermore, the massive increase in Test Loss (1.15 vs 0.40) is particularly telling. It indicates that when the large model makes a mistake, it is "confidently wrong" (predicting a probability near 0 or 1 incorrectly), whereas the small model is more calibrated.

4.3 The Improved Model Architecture

To achieve the superior score of 94% on the Kaggle Leaderboard, a third optimized architecture was developed that addressed the shortcomings of both the small and large networks.



This leaderboard is calculated with all of the test data.

#	Team	Members	Score	Entries	Last	Join
1	Orville and Delmika		0.94000	20	12s	

Input: 20 features
Hidden Layer 1: 24 Neurons (ReLU)
Dropout: 0.5 (50% Drop rate)
Hidden Layer 2: 12 Neurons (ReLU)

Output: 1 Neuron (Sigmoid)

The primary differentiator was the inclusion of Dropout. The small network lacked the capacity to capture complex feature interactions, while the large network memorized the noise. By slightly widening the first layer to 24 neurons but applying an aggressive 50% Dropout, the network was forced to learn robust, redundant features. During training, the network could not rely on any single neuron, simulating an ensemble of many smaller networks. This allowed us to increase model capacity without suffering the overfitting penalties seen in Section 4.2.

5. Conclusion

The results showed that while Logistic Regression (90%) was the most robust baseline model on our internal test set, a carefully regularized Neural Network using Dropout was able to achieve a better performance of 94% on the external competition data. This highlighted that for tabular medical data, raw model size (number of neurons) is less important than regularization techniques that prevent overfitting.