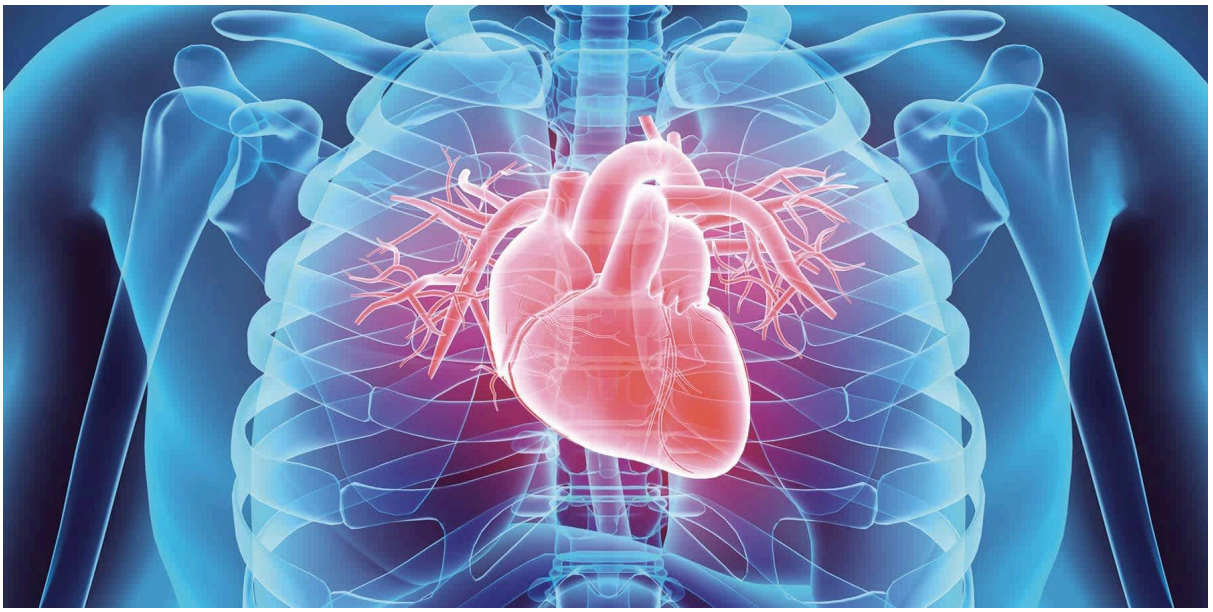
Heart Disease Prediction



James Stanton - 2541773

Ru’aan Maharaj - 2446659

James Stanton - 2541773 1
Ru'aan Maharaj - 2446659 1
The Dataset 3
Data Preprocessing 4
Model Choice and Implementation 5
Parameter Tuning 7
Batch Size Tuning 7
Epoch Tuning 9
Learning Rate 11
Hidden Layers and Optimizer 12
Final Model overview 13

The Dataset

Heart disease is a leading global cause of mortality, with heart failure often being the final result of various cardiovascular conditions. This dataset, sourced from Kaggle, includes clinical and demographic data that can be used to predict the likelihood of heart disease in patients.

The dataset contains 11 input features and one binary output label (HeartDisease) indicating whether the patient is diagnosed with heart disease (1) or not (0). The goal of this project is to develop and evaluate machine learning models, particularly a neural network that can accurately classify individuals based on these features.

Input Features:

- Age: age of the patient in years
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]
- Cholesterol: serum cholesterol [mm/dl]
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

Output Features:

- HeartDisease: Binary classification target (1: Heart disease present, 0: No disease)

Source:

https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data

Data Preprocessing

The data preprocessing begins by removing any rows with missing values to maintain data quality. Duplicate entries are also eliminated to prevent redundancy and bias. Categorical features - Sex, ChestPainType, RestingECG, ExerciseAgina, and ST_Slope - are encoded using one-hot encoding to convert them to a numerical format suitable for model training. Numerical features such as Age, RestingBP, Cholesterol, MaxHR, and Oldpeak are standardized using “Standard Scaler”, which centers the data around zero with unit variance. to ensure that each feature contributes equally during training.

Once cleaned and encoded, The cleaned dataset is split into input features (x) and the binary target label (y, HeartDisease).. The data is then divided into three subsets: 60% for training, 20% for validation, and 20% for testing. The training data is fed to the model in batches using a PyTorch DataLoader, while the validation and test sets are evaluated in full to track performance. This systematic preprocessing ensures the model is trained on clean, balanced, and properly scaled data, enhancing its ability to generalize to new, unseen cases.

Model Choice and Implementation

Model Used:

We implemented a feedforward neural network using PyTorch, specifically designed for binary classification on the heart disease dataset. Feedforward neural networks (also called multilayer perceptrons or MLPs) are well-suited for structured/tabular data, as they can learn complex, non-linear relationships without requiring spatial or temporal context.

Why This Model?

Given the dataset consists of a mix of numerical and one-hot encoded categorical features, more specialized architectures such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) would not be appropriate. CNNs are optimized for spatial data like images, and RNNs for sequential data like text or time series. An MLP strikes the right balance of complexity and relevance for this kind of predictive task.

Architecture:

Our model, HeartNet, is a multilayer perceptron (MLP) with:

- **Input layer**: Number of neurons equals the number of input features (after one-hot encoding).
- **Hidden Layer 1**: 16 neurons with ReLU activation.
- **Dropout Layer**: 20% dropout to prevent overfitting during training.
- **Hidden Layer 2**: 8 neurons with ReLU activation.
- **Output Layer**: 1 neuron with Sigmoid activation, producing a probability between 0 and 1 for binary classification.

Loss Function & Optimizer:

We use Binary Cross-Entropy Loss (BCELoss), which is standard for binary classification problems, and Adam Optimizer with a learning rate of 0.001 for efficient gradient updates.

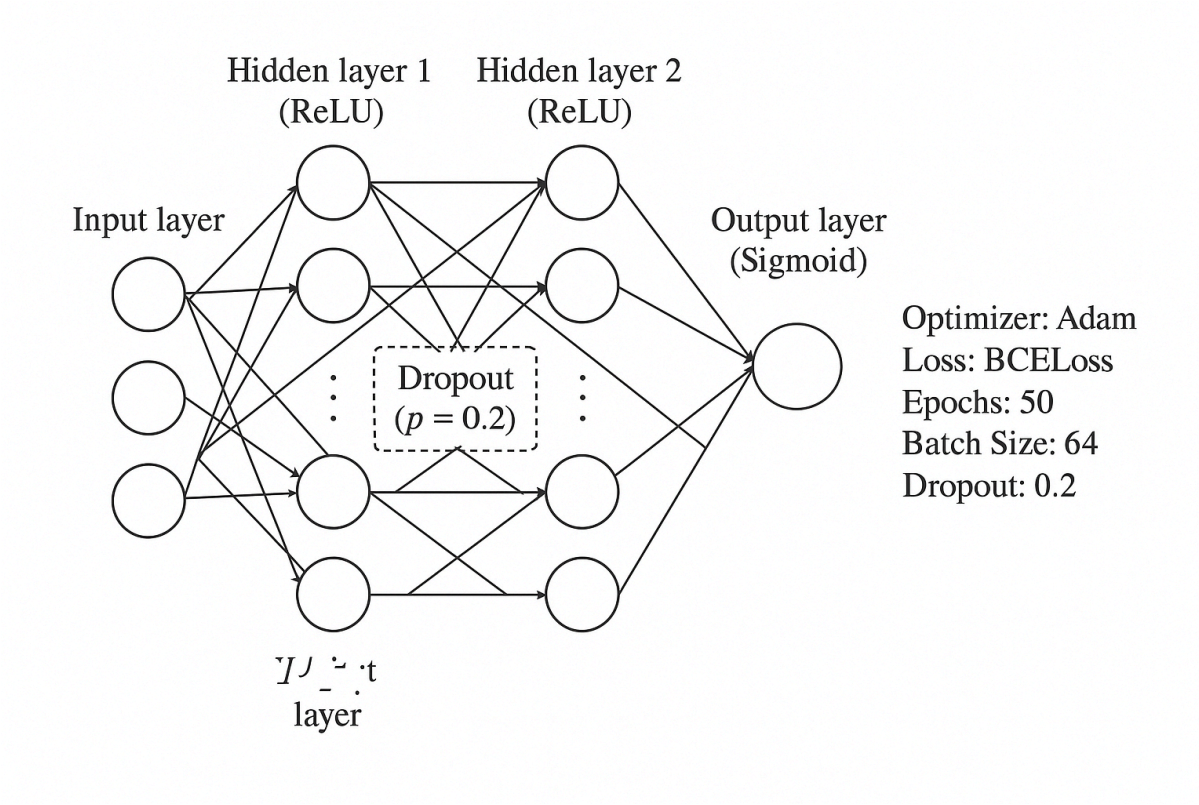
Training:

- We trained the model for 50 epochs, feeding mini-batches of 64 samples using a DataLoader.
- After each epoch, the model is evaluated on a separate validation set, and both training and validation losses are recorded for performance tracking.

- Once training concludes, the model is evaluated on a held-out test set (20% of the original data) to measure generalization performance using accuracy and a confusion matrix.

This architecture offers a strong baseline for this classification task. While we could experiment with deeper networks or alternative activation functions, this model already achieves an accuracy between 87–91%, indicating solid predictive power.

This is what the architecture looks like:



Parameter Tuning

Batch Size Tuning

Batch size determines how many samples are processed before the model updates its weights during training. Smaller batch sizes (like 1 or 8) update frequently and introduce gradient noise, which can help generalization but may lead to unstable training. Larger batch sizes (like 64 or 80) result in smoother updates but might overfit or underfit depending on the model and data.

Accuracy Results:

Batch Size	Test Accuracy
1	0.8424
8	0.8478
16	0.8315
32	0.8913
64	**0.8967**
80	0.8370

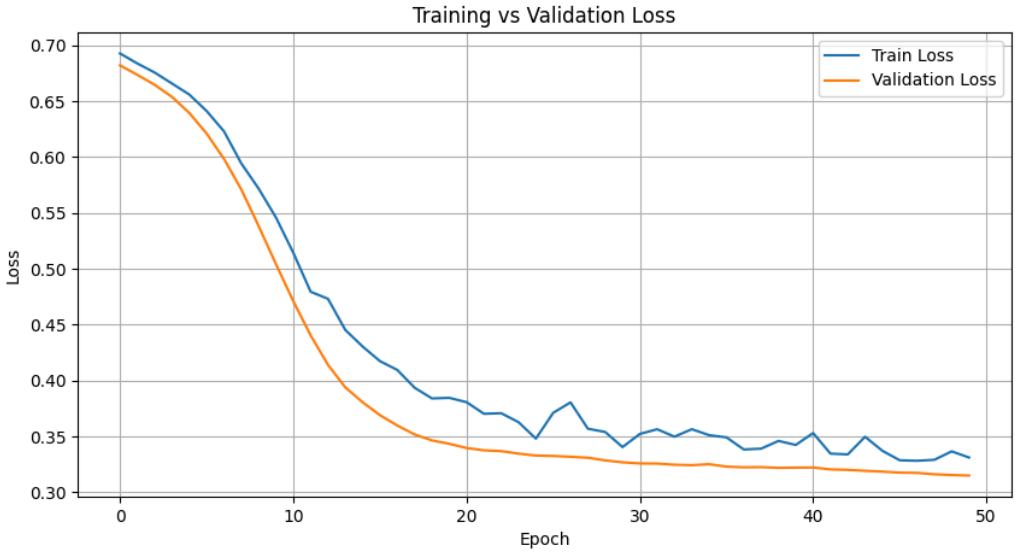
From this table, we can see that test accuracy initially increases as the batch size grows, peaking at batch size 64, after which it drops slightly. This suggests that batch size 64 provides the best generalization performance for this dataset and model configuration.

Graph comparison and Analysis:

To further explore this, we visualized the train and validation loss over 50 epochs for batch sizes 32 and 64 as the top two contenders.

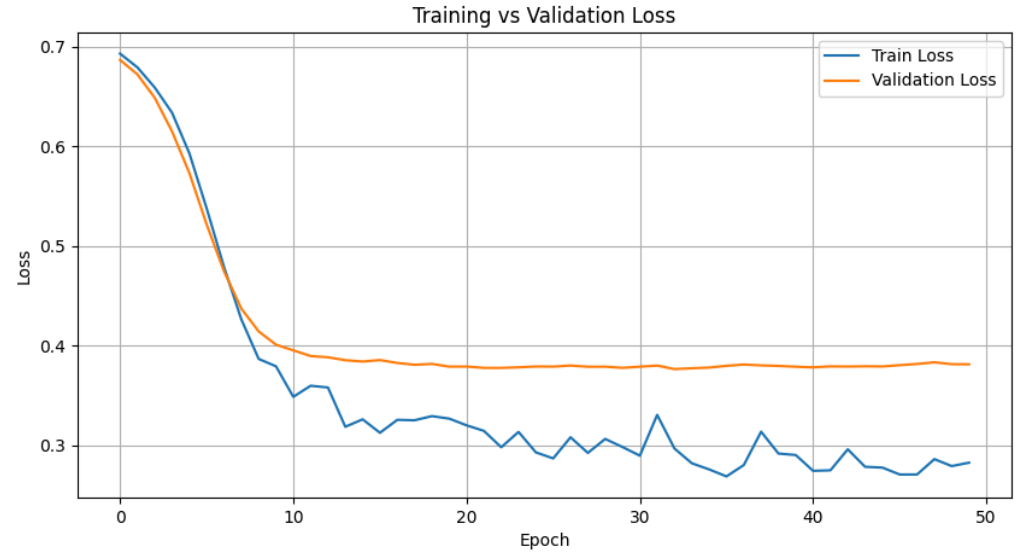
Batch Size 64:

- Final Train Loss: 0.3169
- Final Val Loss: 0.3150
- Accuracy: 0.8967
- Observation from the graph below: Training and validation losses show a steady downward trend, indicating consistent learning without overfitting.



Batch Size 32:

- Final train Loss: 0.1029
- Final Val Loss: 0.3814
- Accuracy: 0.8913
- Observation from graph below: While the accuracy is close to that of batch size 64, the validation loss is noticeably higher and more volatile, which may hint at minor overfitting or instability.



While both batch sizes performed well, batch size 64 was selected as the optimal value due to its highest accuracy, stable training behavior, and lower validation loss. It offered the best balance between training efficiency and generalization performance.

Epoch Tuning

To explore the impact of training duration on model performance, we experimented with a range of epoch values. The table below summarizes the test accuracy achieved at several different epoch counts:

Accuracy and Results:

Epochs	Test Accuracy
1	0.6902
5	0.8370
10	0.8533
20	0.8533
50	0.8641
60	**0.8696**
70	0.8098

At epoch 1, the model performs poorly, with a test accuracy of only 0.6902. This is expected, as the network has only just begun learning and weights are still near their random initializations, and the model hasn't yet captured any meaningful patterns from the data.

From this analysis, we observed that increasing the number of epochs generally improved accuracy up to a point, with 60 epochs yielding the highest test accuracy of 0.8696. However, after 60, performance began to decline, likely due to overfitting, as seen with the drop in accuracy at 70 epochs.

In-depth Comparison of Epoch 50 vs 60:

Although 60 epochs gave the best accuracy, we also analyzed 50 epochs in more detail since they had very competitive performance and slightly more stable validation loss. Below is the loss progression and confusion matrix for both:

Epoch 60:

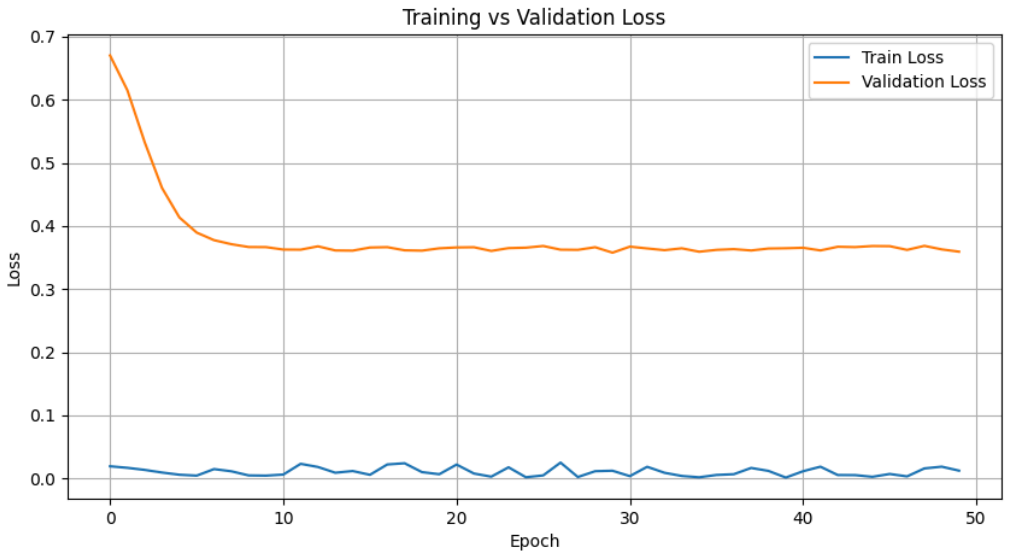
- Final Train Loss: 0.1839
- Final Validation Loss: 0.3705
- Test Accuracy: 0.869

Epoch 50:

- Final Train Loss: 0.4250
- Final Validation Loss: 0.3593
- Test Accuracy: 0.8641

While both models performed similarly, the epoch-60 model had slightly higher accuracy but also signs of increased training volatility (train loss fluctuating across epochs). In contrast, epoch-50 appeared more stable, potentially making it a safer choice if we were prioritizing generalization.

This can also be seen in this graph for 50 epochs:



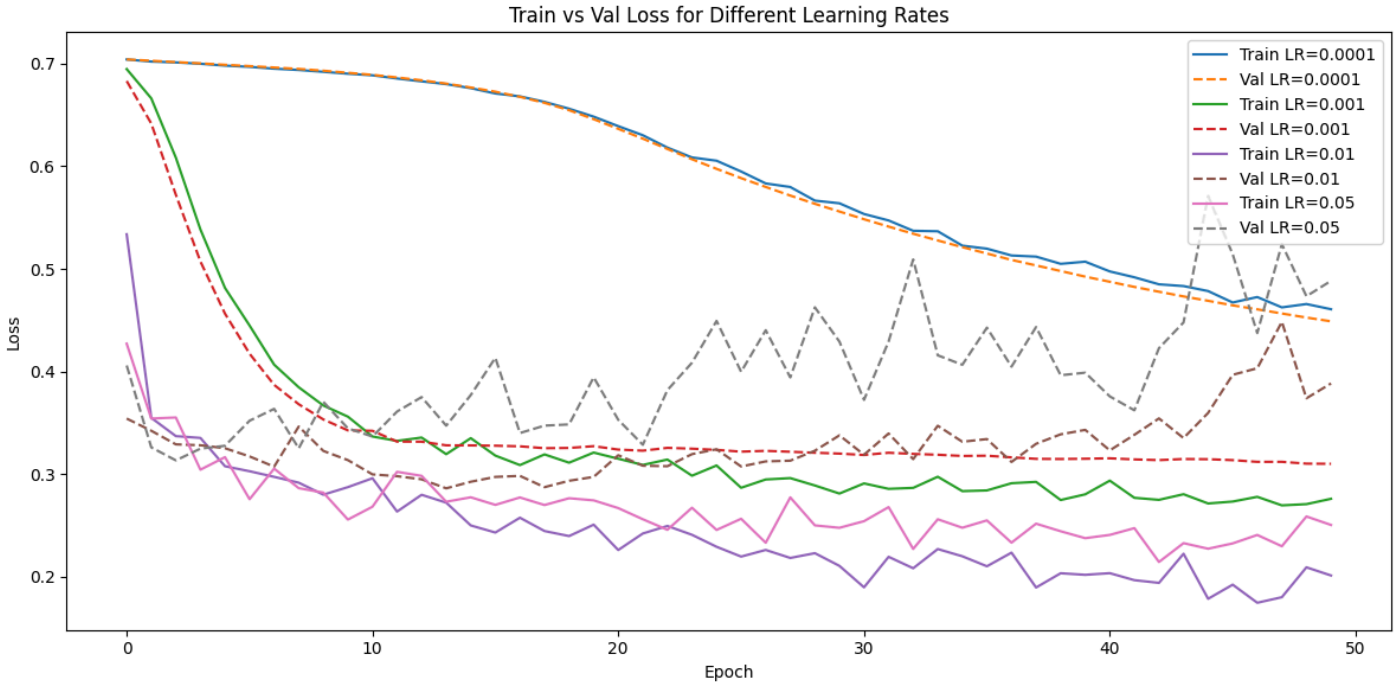
The graph reveals a classic *U-shape* in validation loss: it dips, levels off, and stops improving around epoch 50. Meanwhile, the fluctuations in train loss reflect batch variation, regularization effects (e.g., dropout), or optimizer noise.

Despite the last train loss spike, the validation loss is still among the lowest, and the test accuracy is strong (86.41%), with a solid confusion matrix. That's why Epoch 50 is chosen as it's the best compromise between performance and generalization.

Learning Rate

The learning rate controls how much the model's weights are updated during training. It determines the size of the steps taken towards minimizing the loss function. Choosing an appropriate learning rate is important to ensure that the neural network learns effectively from the data without diverging or learning too slowly.

To find the most suitable learning rate, we experimented with different values and observed the results:



Learning Rate	Test Accuracy
0.0001	0.8315
0.001	0.8641
0.01	0.8315
0.05	0.8261

Based on the results above, we decided 0.001 was the most appropriate choice for this model. The graph shows that this learning rate led to a steady and consistent decrease in both training and validation loss, without the instability seen at higher rates like 0.01 and 0.05. Those higher rates caused the validation loss to fluctuate or increase in later epochs, indicating overfitting or instability during training. On the other hand, the lowest learning rate, 0.0001, resulted in a very slow learning and a slow drop in loss, suggesting underfitting and inefficient learning.

The model trained with a learning rate of 0.001 achieved both the lowest final validation loss and the highest test accuracy. This suggests that it not only fit the training data well but also generalised better on unseen data. While 0.01 starts with fast improvements, it fails to sustain performance and ends up with higher validation loss. Similarly, 0.05 shows erratic loss behaviour, and 0.001 is behind in convergence speed and final accuracy.

Hidden Layers and Optimizer

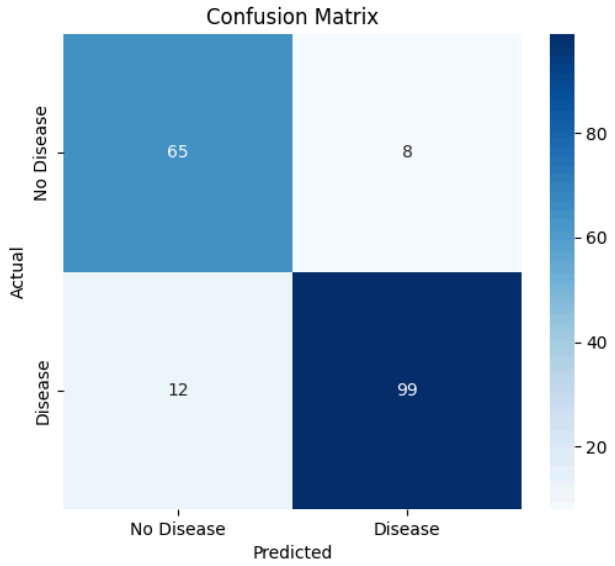
The number of hidden layers and the choice of optimizer did not lead to significant improvements in the model's performance. The dataset's clean, normalized inputs, low noise levels, and balanced class distribution helped the model learn effective decision boundaries without the need for extensive tuning. All tested optimizers were able to minimise the loss function efficiently, allowing the model to converge to suitable solutions regardless of the optimization method used. Likewise, changing the number of hidden layers had little effect, as the model consistently captured the underlying patterns in the data. Overall, careful tuning of other key parameters meant that variations in optimizer choice and network depth had minimal impact on the results.

Final Model overview

Selected configurations:

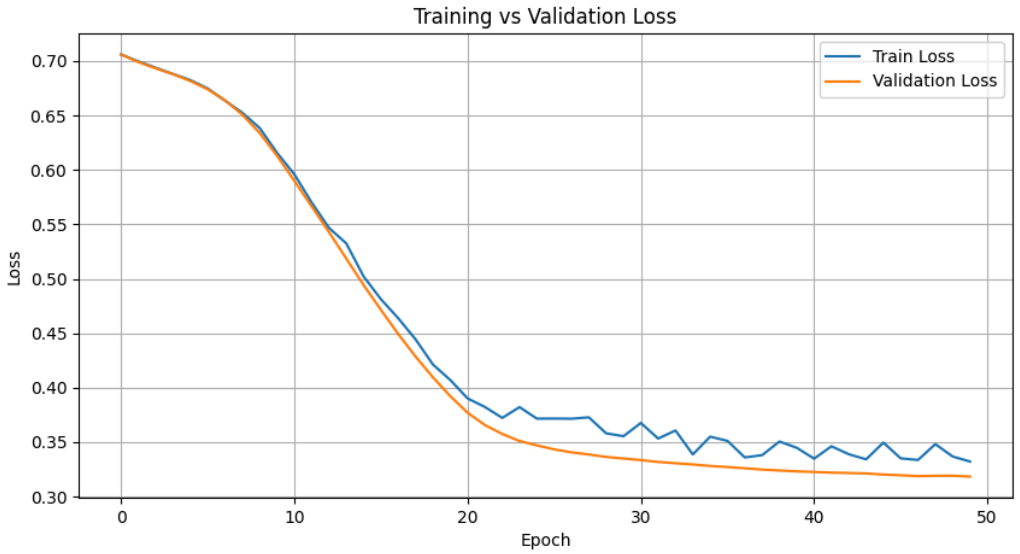
- Epochs: 50 - 60 (We use 50)
- Batch size: 64
- Learning rate: 0.001
- Final accuracy: 89.13%
- Performance range: typically between 87% to 91%

Confusion Matrix:



This confusion matrix shows 65 true negatives and 99 true positives. There were 8 false positives and 12 false negatives which is a strong indication that the model handles both classes well with minimal misclassification.

Graph Analysis:



The training and validation loss curves demonstrate a healthy and stable learning process:

- **Initial High Loss:** At epoch 1, both training loss (0.6858) and validation loss (0.6713) are high, which is expected when the model is untrained.
- **Rapid Improvement:** By epoch 10, both losses drop significantly (train: 0.4942, val: 0.4945), indicating the model is learning relevant patterns quickly and efficiently.
- **Steady Decline:** From epochs 10 to 50, both training and validation loss continue to decline gradually, showing that learning continues without stagnating.
- **Close Train/Val Loss:** The training and validation loss values stay relatively close throughout, especially by epoch 50 (train: 0.4898, val: 0.3238), which suggests that the model:
    - Isn't overfitting (i.e., it's not memorizing the training data)
    - Maintains generalization ability to unseen data
- **No Spikes or Instability:** There are no sudden jumps or oscillations in either curve, indicating that:
    - The optimizer is working well (Adam with appropriate LR)
    - The training process is smooth and controlled
    - The model architecture is well-suited to the data

Overall, these curves reflect a well-regularized model that's learning effectively over time and generalizing well to the validation set.

After extensive tuning of both epoch count and batch size, we found that a configuration of 50 epochs and a batch size of 64 produced the most balanced and generalizable results. The model achieved a test accuracy of 89.13% in this run, (but the maximum accuracy it achieved was 91.12%), with a confusion matrix showing a strong ability to correctly classify both positive and negative cases. The loss graph confirms stable training and suggests the model was well-tuned to avoid both underfitting and overfitting.
These results validate our preprocessing, model design, and hyperparameter tuning efforts, and suggest that a relatively simple feedforward neural network can perform remarkably well on this structured classification task.