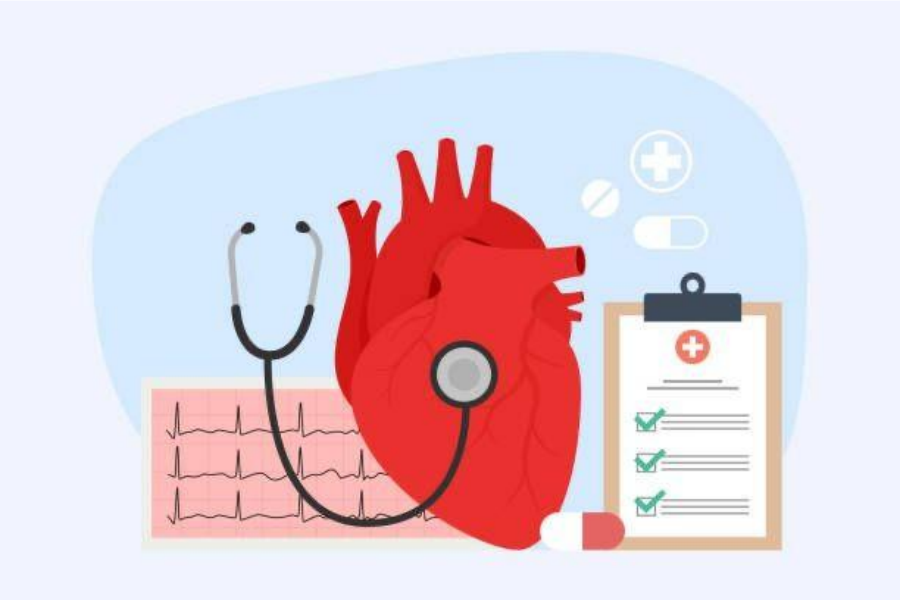
Heart Disease Classification Using Neural Networks (AI)

With Python and PyTorch

By: Khuram Chaudhary



Project Objective

- Develop a classification model which reads a dataset and predicts whether a patient presents heart disease or not.
- Split the dataset in order to assess the models performance.
- Explore, preprocess, and clean data to understand its features and prepare it for training and testing.
- Utilize different architectures to optimize model performance on PyTorch to build a classification model.
- Test and evaluate the models to report performance metrics (including Accuracy, Precision, Recall, AUC, and F1 Score).

Data Exploration

- Identified relevant characteristics and researched correlations to heart disease.

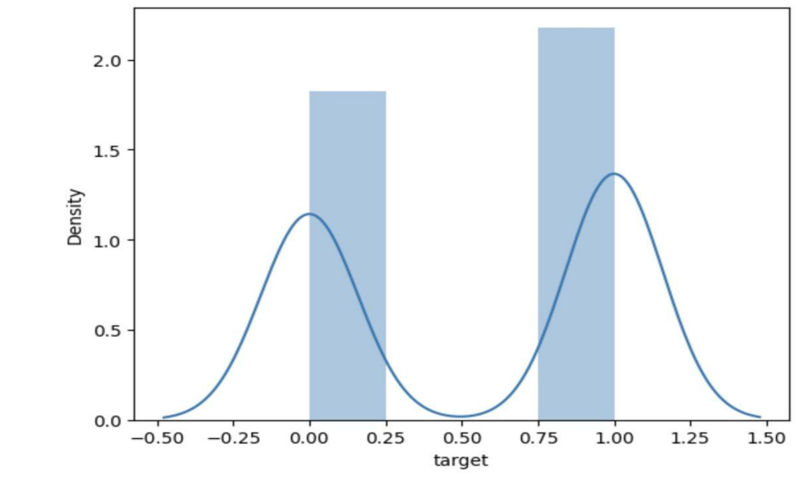
- Split data and analyzed target attribute statistics.

- Skewness of -0.1798 indicates a slight left skew, while kurtosis of -1.98 suggests a flatter distribution with thinner tails compared to normal.

- Correlation matrix suggests negative relationship between slope and oldpeak.

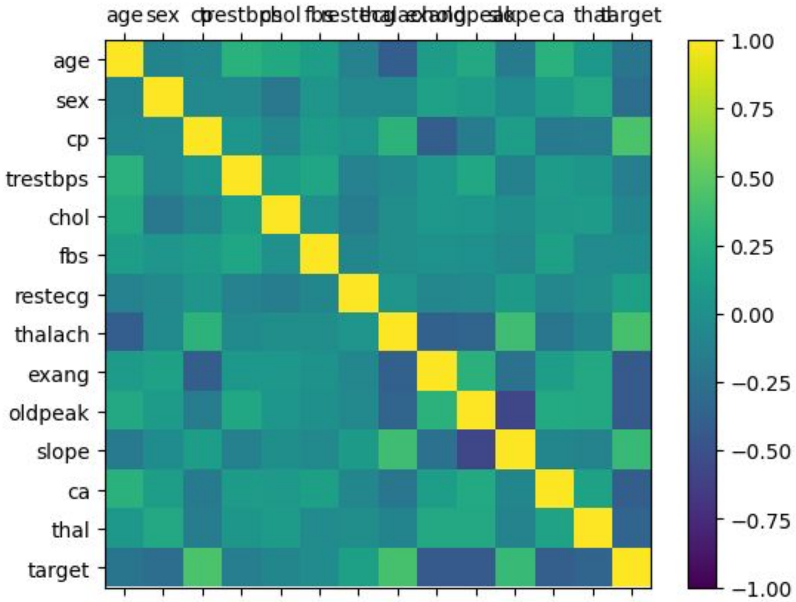
```
sns.distplot(df['target'])
```

```
<Axes: xlabel='target', ylabel='Density'>
```




```
[12] print("Skewness: %f" % df['target'].skew())
     print("Kurtosis: %f" % df['target'].kurt())
```

```
Skewness: -0.179821
Kurtosis: -1.980783
```



Data Preprocessing and Cleaning

- Searched for outliers within attributes relevant to heart health:
  - trestbps, chol, thalach, and oldpeak.
- Flagged outliers using the calculated thresholds: values below (Q1 - 1.5 * IQR) or above (Q3 + 1.5 * IQR).
- Filtered for and capped outliers to prevent unduly skew from our models.
- Applied one-hot encoding to categorical attributes (sex, cp, fbs, restecg, exang, slope, ca, thal) to maintain model integrity, as they are numerical but unordered.
- Used Min-Max scaling to normalize 'age' and 'ca' to preserve their original ranges.
- Used Z-score normalization for 'trestbps', 'chol', 'thalach', and 'oldpeak' to ensure consistency.
- Conducted feature selection by analyzing correlation between the features in our training data and the target variable.

Architecture of Neural Network

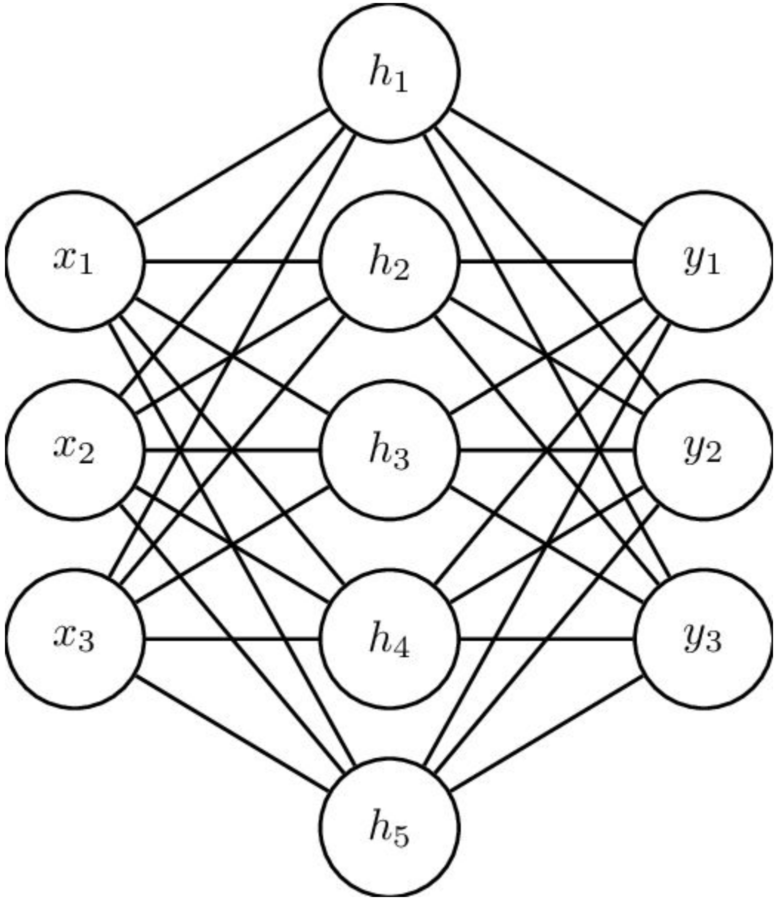
- Over 20 unique models were built, tested, and evaluated to ensure the highest performance metrics were achieved.

Original Approach (8 models):

Attribute	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
# of Hidden Layers	1	1	1	1	3	3	3	3
# of Layers	5	5	10	10	5	5	10	10
Activation Function	ReLu	Sigmoid	ReLu	Sigmoid	ReLu	Sigmoid	ReLu	Sigmoid
Epochs	100	100	500	500	100	100	500	500

- nn.Linear
- Learning Rate = 0.1
- Loss function = BCEWithLogitsLoss
- Optimizer = SGD

Architecture of Neural Network



Revised Simplistic Approach (multiple models):

- Base model:
  - nn.Linear
  - Learning Rate = 0.01
  - Loss function = BCEWithLogitsLoss
  - Optimizer = Adam
  - No activation function originally used

Tested Following Hyperparameters on Model:

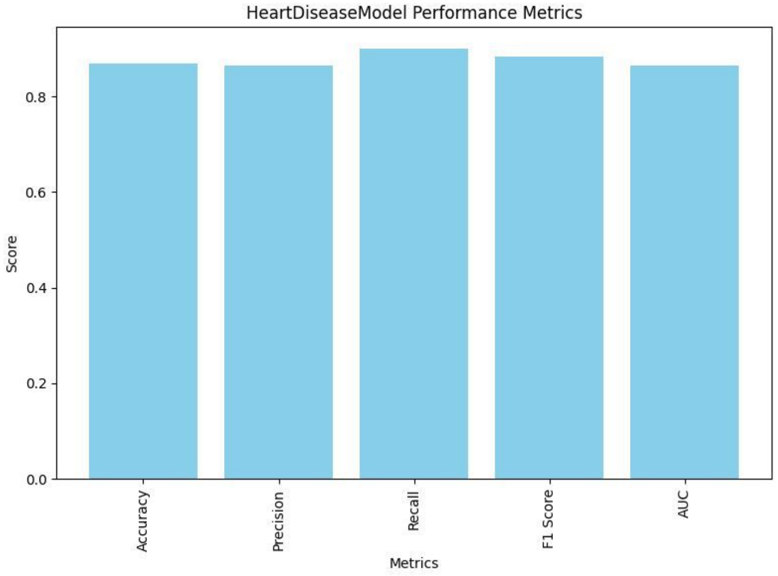
- 0-5000 Epochs
- 1-5 Layers
- 6-101 neurons
- 6 different activation functions (ReLU, Tanh, LeakyReLU, ELU, Mixed, and Sigmoid) tested

Evaluation and Parameter sensitivity analysis

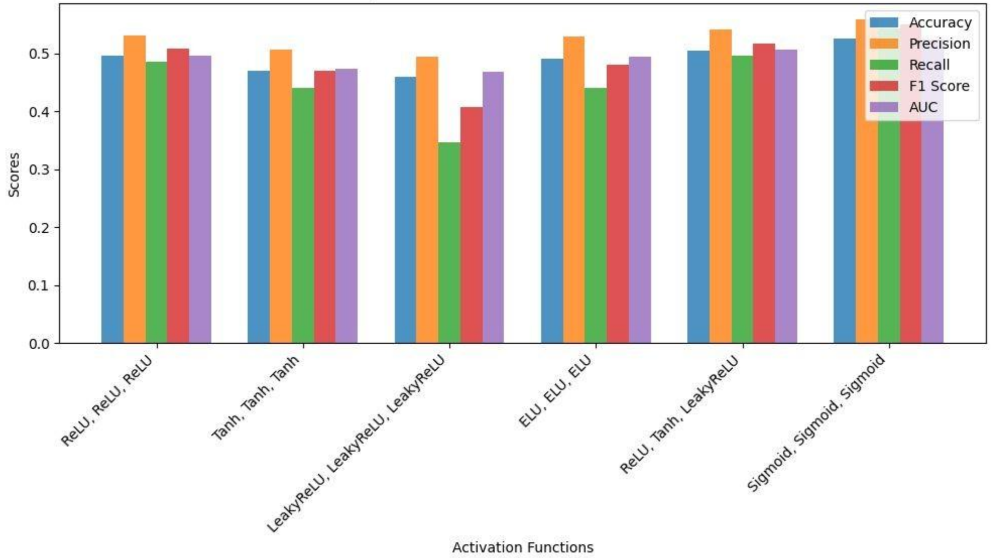
- Despite trying a variety of hyperparameters, the values implemented between models were too similar to one another and with the Original Model approach, it had very low metrics (55% or less accuracy) and it did not perform as well as anticipated.

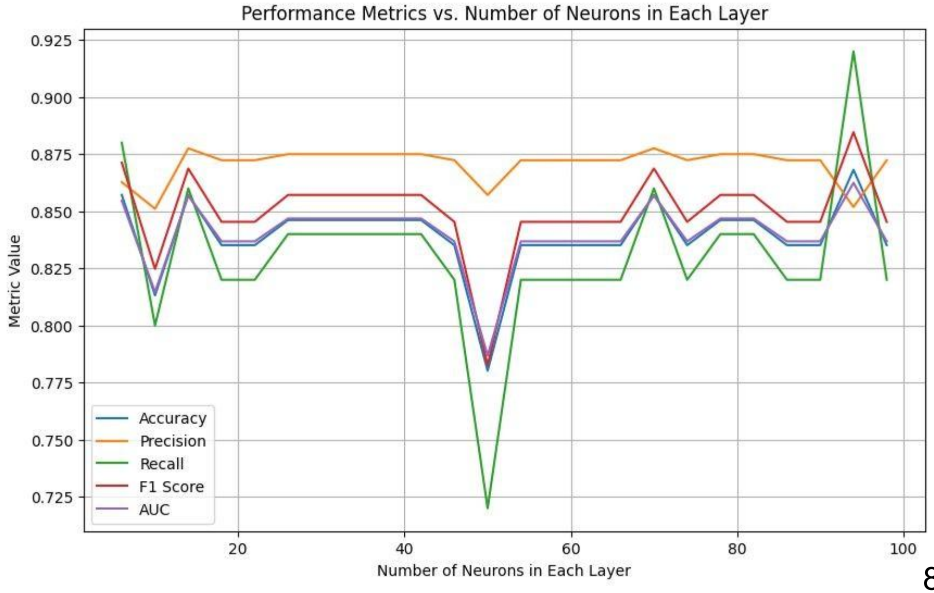
- The new simplistic approach used a wider variety of hyperparameters and was able to perform dramatically better than the original models.
  - This new model performed about 150% better than the original approach.

Evaluation and Parameter sensitivity analysis



Accuracy: 84%
Precision: 87%
Recall: 82%
F1 Score: 85%
AUC: 84%





Takeaways

- Our final result that the neural network achieved was positive and served and shows that the model is more linearly separable than expected.
  - This is a good final model because it has high performance metrics and every hyperparameter that was implemented is there for a specific purpose.
  - The model can be improved with more data to better train it and prevent any unintentional overfitting.
    - With a greater volume and variety of data, there is more information to gain from the model and more test cases that it can cover

Conclusion and Future Work

- Used Neural Networks and Deep Learning to build various models.
- Tested each model and identified optimal performance metrics:
  - The model with the highest performance metrics was the HeartDiseaseModel (Slide 8).
- Tested the models with a variety of different hyperparameters and activation functions like 'ReLU', 'Tanh', 'LeakyReLU', 'ELU', and 'Sigmoid':
  - With no activation function, the model acted like a linear regression model, which helped to capture the linear relationship between the inputs and target variable.
- In conclusion, this project was an excellent representation of the amount of testing and training required for creating an effective neural network.
  - This allowed for a wide variety of model architectures to be explored and increased the likelihood of the "best" model being selected to undergo the final implementation.

Thank You!