Contraceptive Method Choice

CS 4300 Semester Project

Lexi Anderson

February 2022

Contents

List of Figures

List of Tables

1 Abstract

Artificial neural networks are powerful computing tools that mimic the inner workings of the human brain in order to solve a specified problem. When trained correctly, a neural network can produce highly accurate predictions. The accuracy of the model depends in part on its internal configuration–the hyperparameters, including batch size, training time, activation method, model capacity, and optimization. The model's accuracy also depends on the quality of the training dataset–the number of data points, the presence of various cases, the importance of each feature, the balance of the training set, etc.

In this project, the objective was to build a model that would solve a binary classification problem with high accuracy. Each model was hand-tuned in order to find the one that performed with the highest possible accuracy. For the most part, this project focused on optimizing the model's configuration rather than improving the dataset.

In the end, the most optimal model produced in this project reached an accuracy of about 80 percent. This value may have been higher if the training data were optimized so that unimportant input features were eliminated.

The neural network was built in Google Colab using TensorFlow Keras, a deep learning API written in Python. All project code was written in Python.

2 Introduction

The purpose of this project was to build a high-accuracy artificial neural network for solving a classification problem.

The first step was to choose a dataset to use to build the model. The dataset I chose to work on is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The sample population consisted of married women who were not confirmed to be pregnant at the time of the interview. The original purpose of this dataset was to predict a woman's current contraceptive usage based on certain demographic and socioeconomic factors, but in this project I used the women's age input feature for the target output and utilized the rest of the data collected to predict whether each woman in the dataset was over the mean age of 32.

After I chose my dataset, I began building a neural network model by starting from a basic one-layer, one-neuron model and incrementally adjusting its hyperparameters until I reached an optimal accuracy score. I tested various combinations of layers, neurons, activation methods, training periods, and optimizers to determine the maximum possible accuracy I could attain. I also evaluated the tested models according to their learning curves and ROC curves.

I tested an overfitted model that performed with near perfect accuracy on the training set. I used this model to compare with my candidate models to ensure that I did not overfit any of them.

Finally, I tested out two regularization techniques, early stopping and model checkpointing, to see if they could further refine my models.

3 The Dataset

I found this dataset to be a good candidate for binary classification due to the nature of its input features and its sample size. Including the classifying feature, there are ten features, all of which are either numerical or categorical, so it was simple to assign a range of numerical values to each attribute. In addition, the enumerable nature of each input feature in the dataset gave me ample options for which attribute to choose as the output. The dataset consists of over 1400 instances, so I am confident that the sample is large enough to draw reasonable conclusions from.

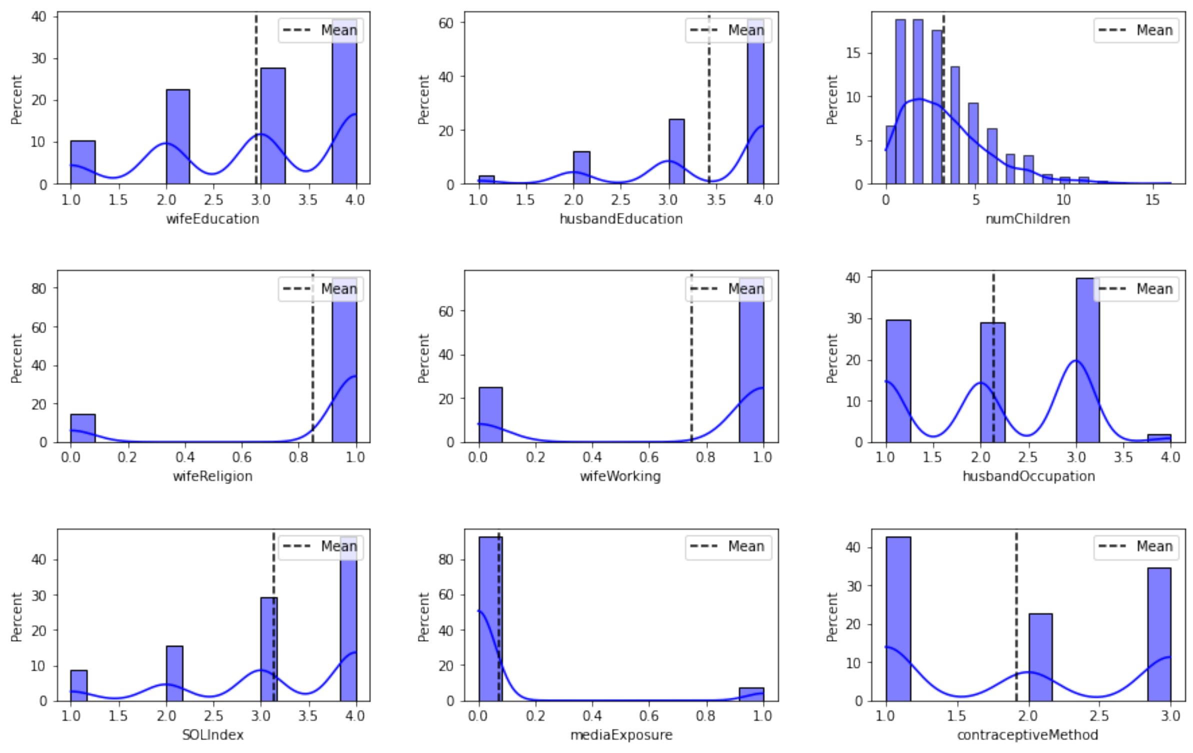

Figure 1: Visualization of each input feature

I chose the output feature to be the women's age. I polarized the data along the mean age, 32, such that every instance under 32 was assigned a value of 0, and every other instance was assigned a value of 1. When I did this, I found that the mean of the binary data was .513917, which was very close to 50 percent. Since the data was so evenly split along this feature, I decided that I would base my output feature on the women's age. The task would be to determine whether each woman was above the mean age of 32. In this case, the output would be 1, or else the output would be 0. I generated these binary values from the numerical data provided for the women's age feature such that each data point would produce its binary value based on whether the associated age was either <32 (represented by 0) or ≥ 32 (represented by 1). In this way, I produced a balanced distribution for my data.

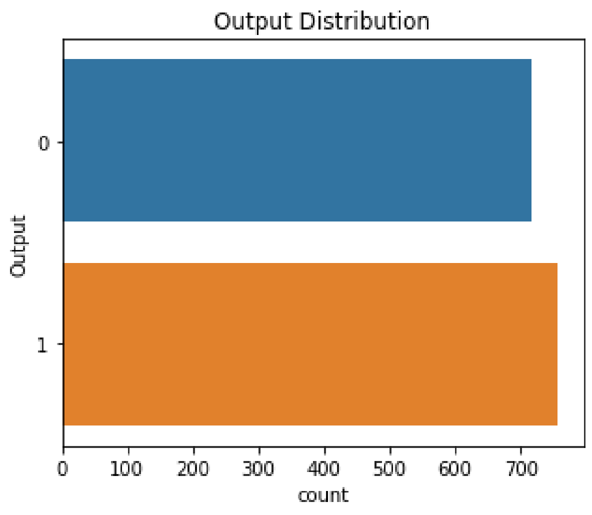

Figure 2: Distribution of the output feature, wife's age

I found the dataset in the UCI Machine Learning Repository.
Plain link: `https://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice`

To normalize my data, I used the mean normalization method to adjust the input values. This method adjusts each value with the formula

$$X = \frac{X - X_{mean}}{range(X)}$$

4 Building a Neural Network Model

I built several different neural network models to train on my dataset. I started with the most basic model, one containing a single neuron in a single layer. I branched out from there by adding more layers and more neurons, trying several different configurations to get an optimal accuracy. The last neural network model I built was created specifically to overfit the dataset, so I added several layers and increased the number of neurons in each layer.
Each model was built under similar circumstances: since the problem to be solved for the dataset was one of binary classification, the sigmoid function was used to activate the final layer of each model, and any previous layers were activated via the rectifier ("relu") function. Each model was trained over 512 epochs, except for the overfitted model, which was trained over 2000 epochs.
Each model was evaluated according to its accuracy score. The results can be seen in the following table, in which each model is described as a list $(x_1, ...)$, where x_i is the number of neurons in the ith layer:

Table 1: Neural Network Model Evaluation, Round 1

Model	Accuracy %	Precision %	Recall %	F1-Score
(1)	76.92	77.11	72.44	0.75
(20,10,1)	83.98	83.95	81.53	0.83
(20,15,10,5,1)	87.71	84.97	89.75	0.87
(8,4,2,1)	81.47	82.11	77.49	0.80
(200,80,60,40,1)	93.21	95.27	90.04	0.93

Predictably, the overfitted model has the highest accuracy at 93.21 percent. Of the rest, the most accurate model appeared to be the model with neuron layers (20, 15, 10, 5, 1), which had an accuracy score of 87.71 percent. These performances changed drastically with a simple change: setting the activation mode of the final layer of each model to be the rectifier function. When compiled and evaluated on accuracy, nearly each model showed a significant decrease:

Table 2: Model Accuracy by Final-Layer Activation

Model	sigmoid (%)	relu (%)
(1)	76.92	76.17
(20,10,1)	83.98	77.94
(20,15,10,5,1)	87.71	79.16
(8,4,2,1)	81.47	52.95
(200,80,60,40,1)	93.21	54.45

The base model was the only one that did not experience any decrease when activated with the ReLU function. It appeared that accuracy decreased as the number of layers and neurons increased. Even with the decrease, the (20, 15, 10, 5, 1) model remained the most accurate of the candidate models.

5 Further Model Selection & Evaluation

On reviewing the previously trained models, I decided to try a more incremental approach to finding the best fit. I fit and evaluated several new models, this time plotting their learning curves and ROC curves to better assess each model's performance. Each model produced the following performance metrics:

Table 3: Neural Network Model Evaluation, Round 2

	Training Set				Validation Set			
Model	Accuracy %	Precision %	Recall %	F1-Score	Accuracy %	Precision %	Recall %	F1-Score
(1)	75.48	80.05	76.07	77.00	71.63	78.57	0.74	0.78
(2,1)	75.78	76.64	76.56	74.35	71.63	72.45	0.74	0.73
(4,1)	77.23	75.51	75.10	69.82	78.87	79.08	0.77	0.74
(6,1)	78.39	78.23	77.96	75.51	76.86	75.51	0.77	0.76
(8,1)	79.65	79.14	79.83	76.26	77.26	77.04	0.79	0.77
(2,2,1)	77.91	74.83	75.14	68.89	80.89	79.08	0.78	0.74
(4,2,1)	78.10	78.00	78.65	76.47	74.85	72.96	0.77	0.75
(8,2,1)	78.97	78.23	78.00	75.77	78.47	75.00	0.78	0.75

5.1 Learning Curves

Each model's history was plotted to show how the accuracies of the training set and the validation set changed over time as well as how the losses of both sets changed over time. Ideally, a model's learning curve will show that over time, the training loss and the validation loss reach a minimum.

In an *overfitted* model, the training loss will continue decreasing while the validation loss reaches a minimum, then starts increasing again. This indicates that perhaps the model should be trained for a shorter time period.

In an *undertrained* model, both training loss and validation loss will continue decreasing rather than reaching a minimum point. In this case, the model may need to be trained for a longer period of time.

Another underfitted model with *insufficient capacity* will have a wide gap between the training loss and validation loss. This kind of model would need more neurons or layers.

The models that I tested produced the following learning curves:

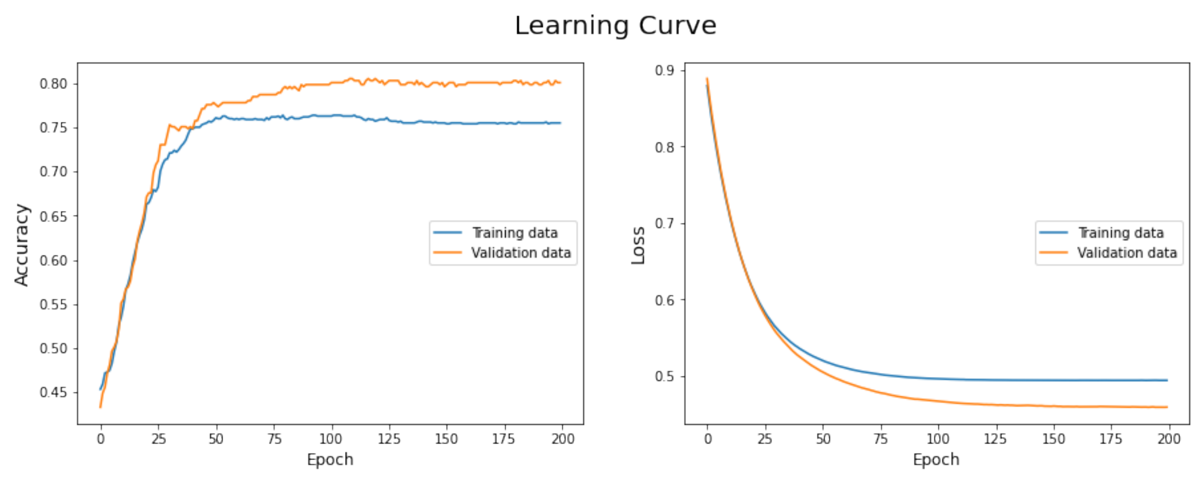

Figure 3: Base Model Learning Curves

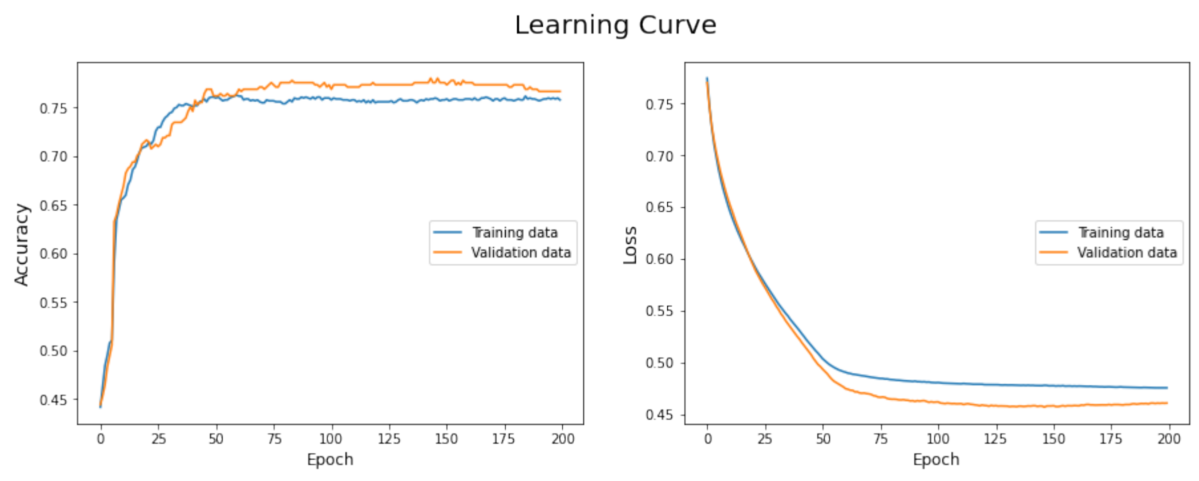

Figure 4: Model (2,1) Learning Curves

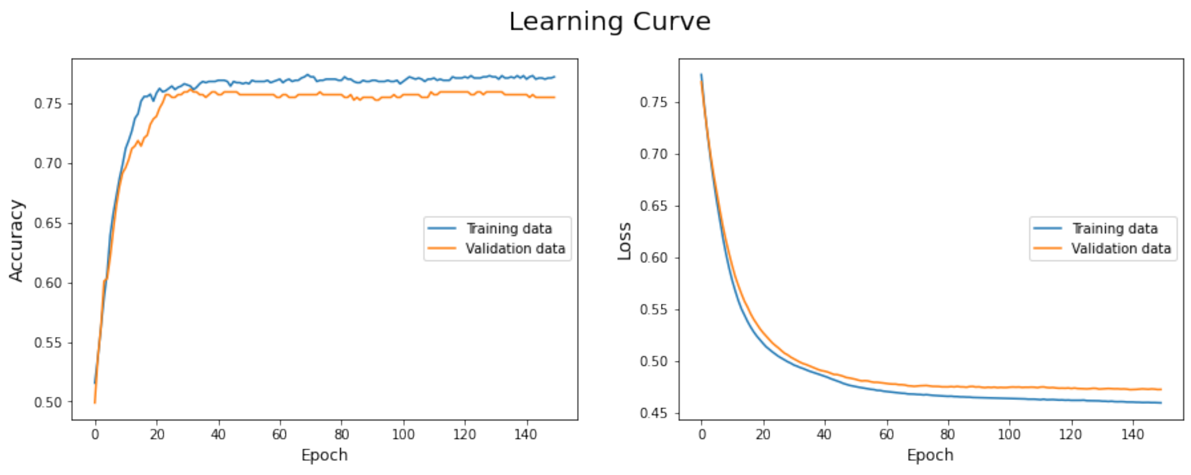

Figure 5: Model (4,1) Learning Curves

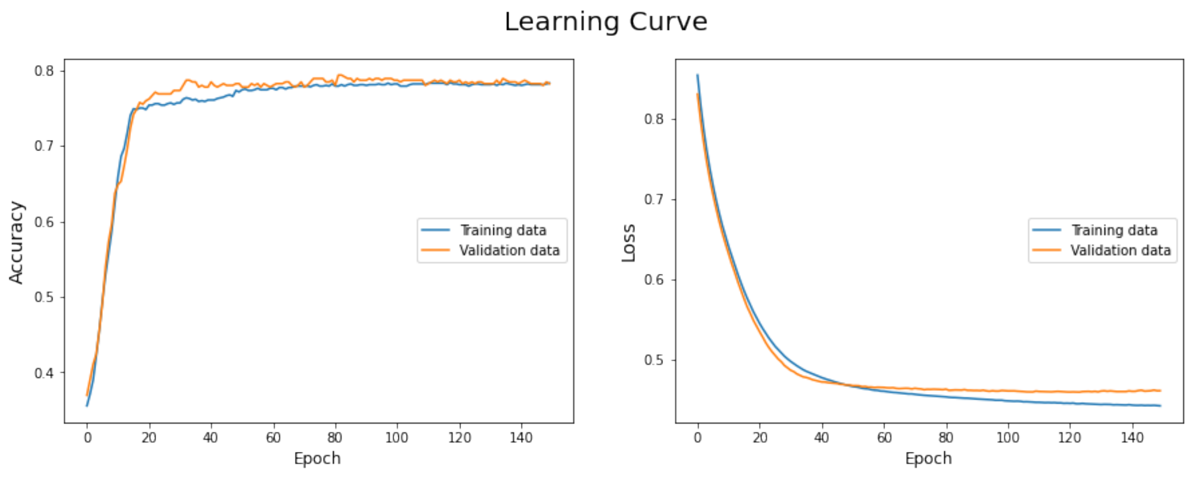

Figure 6: Model (6,1) Learning Curves

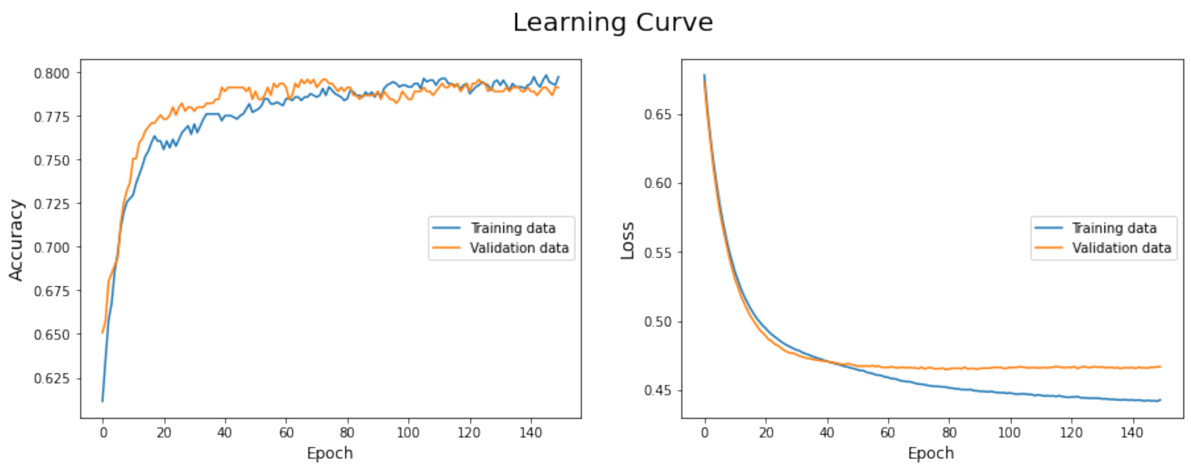

Figure 7: Model (8,1) Learning Curves

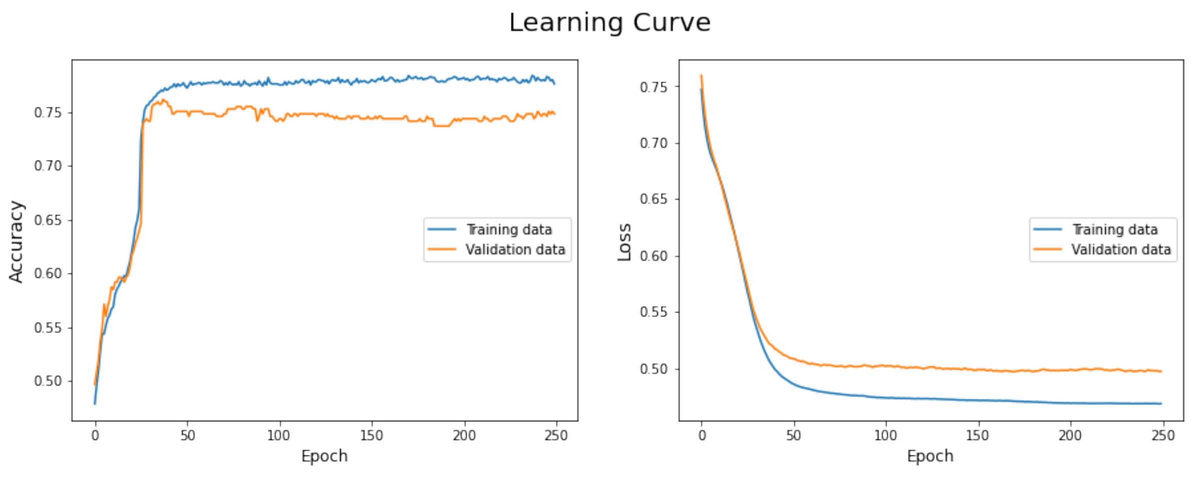

Figure 8: Model (2,2,1) Learning Curves

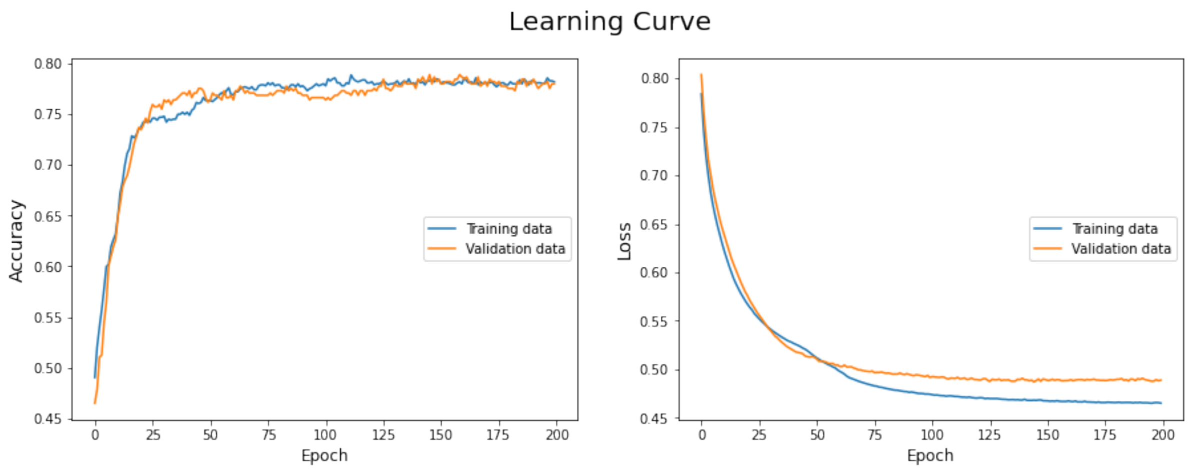

Figure 9: Model (4,2,1) Learning Curves

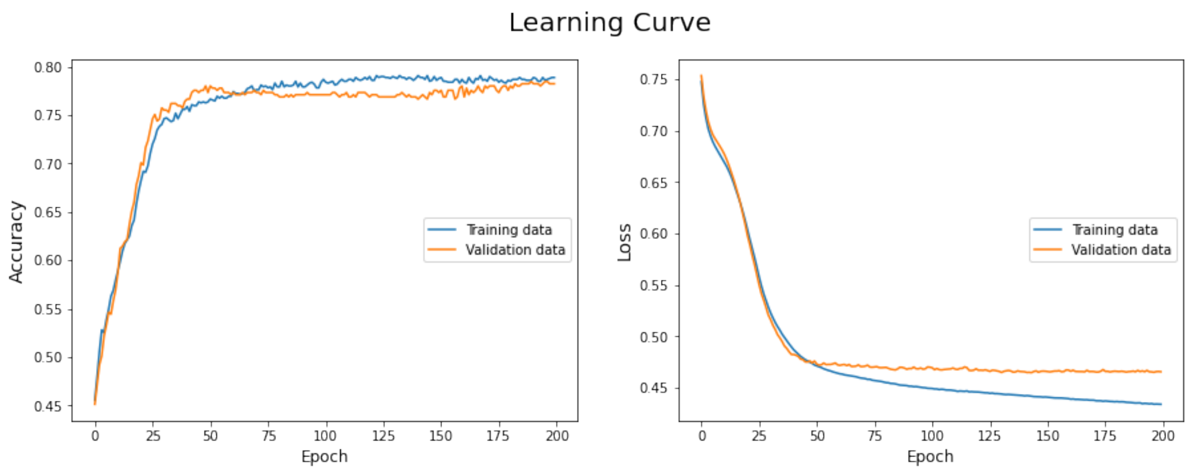

Figure 10: Model (8,2,1) Learning Curves

5.2 ROC Curves & Area Under the Curve (AUC)

The Receiver Operating Characteristics (ROC) curve and the AUC give information about how often a model is likely to make certain types of errors. In binary classification problems, there are two different types of errors that a model can make in predicting values. Type I is a false positive, in which the model makes a positive prediction for what turns out to be a negative value. Type II is a false negative, in which the model makes a negative prediction for what turns out to be a positive value. The ROC curve and the AUC focus on the first type of error. In an ROC curve, the false positive rate is plotted along the x-axis, and the true positive rate is plotted along the y-axis, forming a curve. The AUC is given by the area under the curve and above the line x = y. The ideal AUC value is 1; this suggests that the model distinguishes well between true positive and false positive values. In the worst case, the AUC value will be 0.

The models that I tested produced the following ROC curves and associated AUC values:

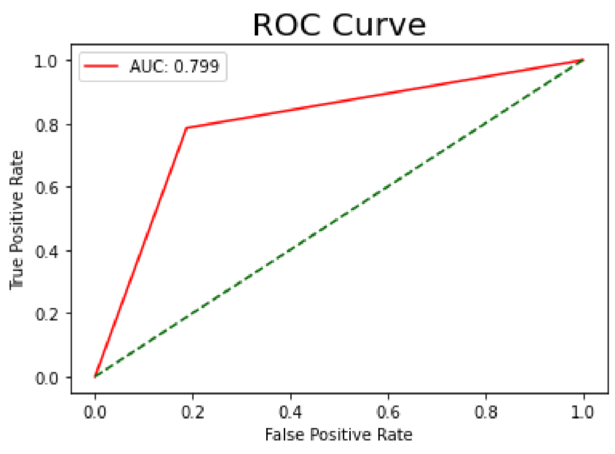

Figure 11: Base Model ROC Curve

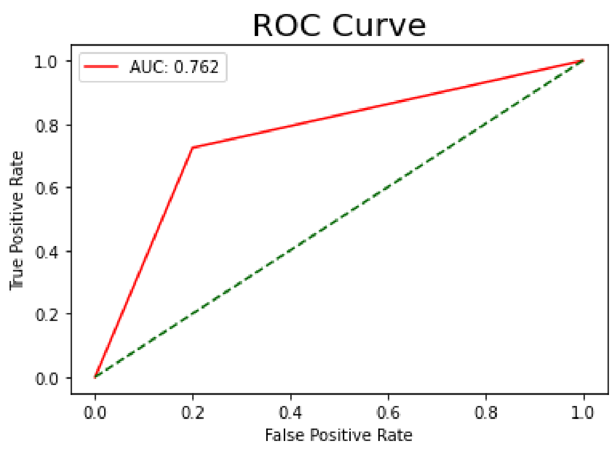

Figure 12: Model (2,1) ROC Curve

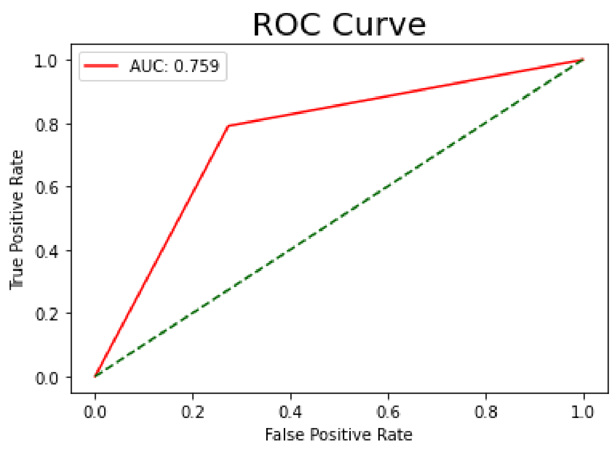

Figure 13: Model (4,1) ROC Curve

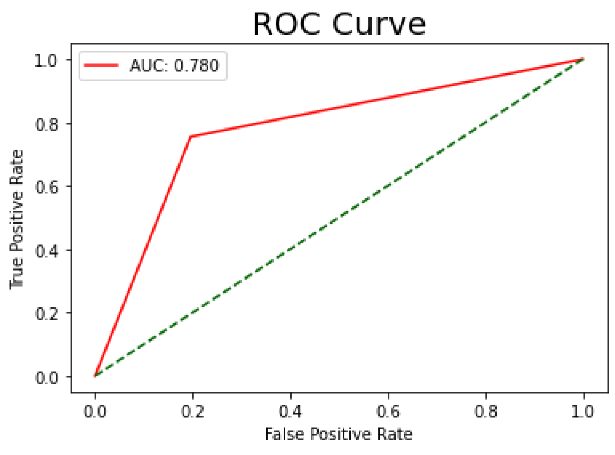

Figure 14: Model (6,1) ROC Curve

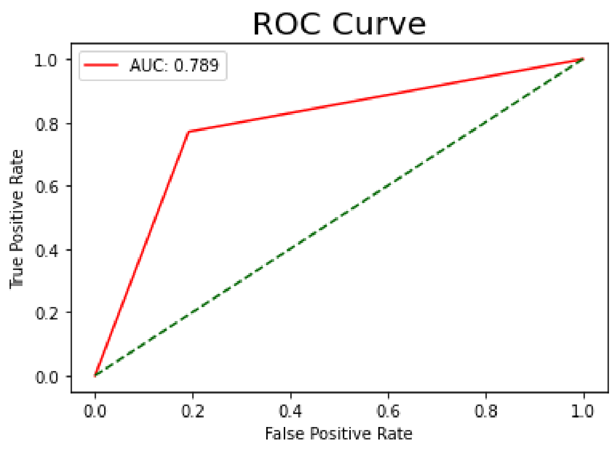

Figure 15: Model (8,1) ROC Curve

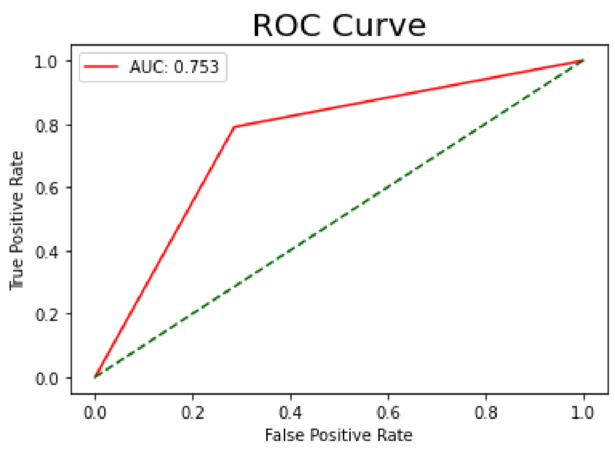

Figure 16: Model (2,2,1) ROC Curve

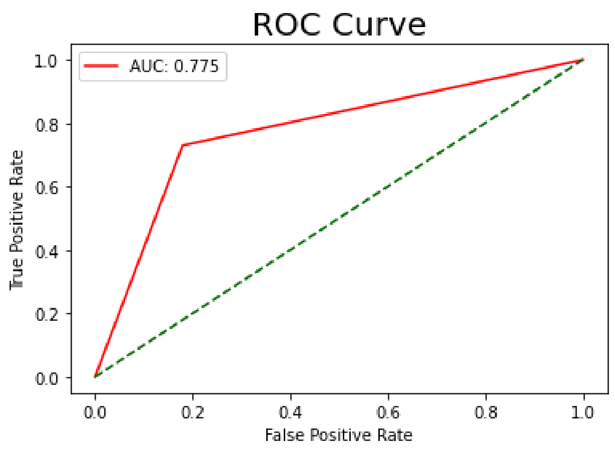

Figure 17: Model (4,2,1) ROC Curve

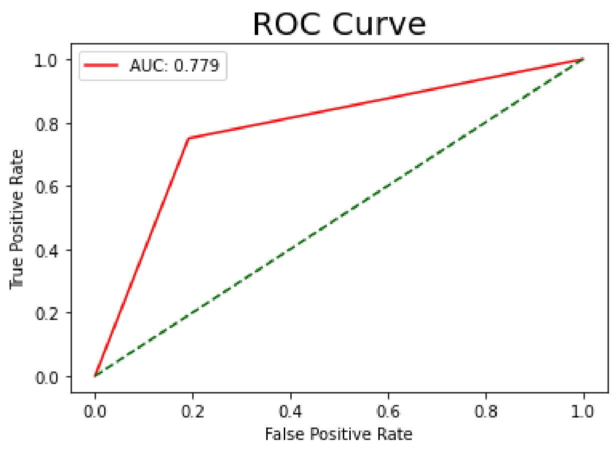

Figure 18: Model (8,2,1) ROC Curve

5.3 Choosing a Model

During the testing phase, I adjusted my hyperparameters to produce better models. I started with training each model for 500 epochs, but later lowered that amount to suit each individual model. In addition, I switched from ’rmsprop’ optimization to ’adam’ optimization after I found that ’adam’ produced better learning curves. I found that model performance began to suffer when the hidden layer count was increased to three, so I limited my candidates to models with two or fewer hidden layers. To choose a best fit from among the tested models, I first compiled a table of the models’ specifications and key performance metrics:

Table 4: Candidate Model Summary

Model	Epochs	Optimizer	Validation Accuracy %	AUC
base model	*200*	*adam*	*80.05*	*0.799*
(2,1)	200	adam	76.64	0.762
(4,1)	150	adam	75.51	0.759
(6,1)	150	adam	78.23	0.780
(8,1)	150	adam	79.14	0.789
(2,2,1)	250	rmsprop	74.83	0.753
(4,2,1)	200	adam	78.00	0.775
(8,2,1)	200	rmsprop	78.23	0.779

I aimed to choose a model whose accuracy on the validation set was close to 80 percent and whose AUC was close to 0.8. Based on these criteria, I eliminated (2,1), (4,1), and (2,2,1) from consideration. For the remaining candidates, I reviewed their learning curves to find ones that displayed a minimal gap between the training loss and validation loss. The model with the best learning curves appeared to be (6,1).

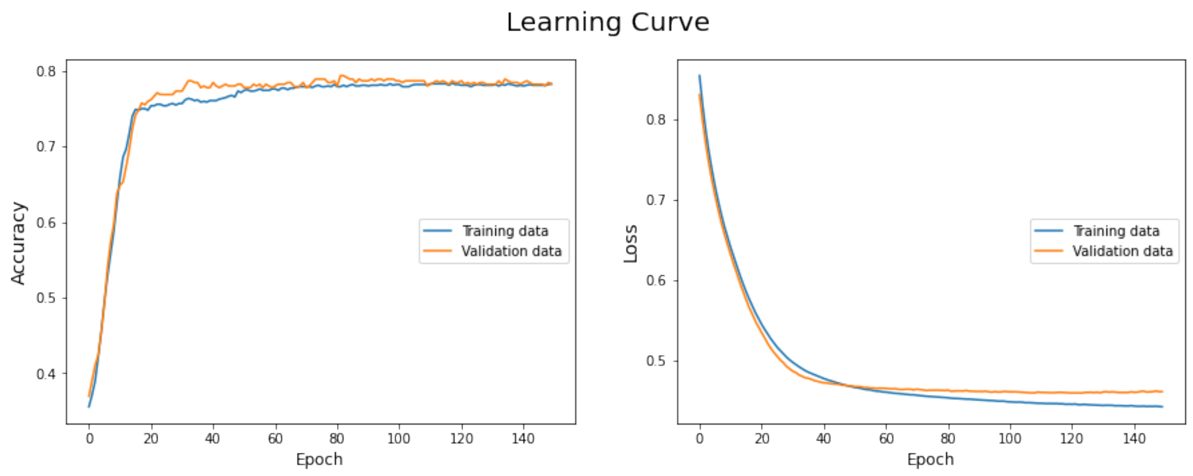

Figure 19: Learning curves of chosen model (6,1)

6 Model Overfitting

In my experiments, I found that it was very easy to overfit models by training for too long or by giving them too large of a capacity. I used these principles to purposely create an overfitted model. At first, I built a (50,50,1) model and trained it over 500 epochs, but this made debugging significantly slower, so instead I trained a (200,200,1) over 200 epochs. The larger capacity caused the model to overfit the training set much more quickly.

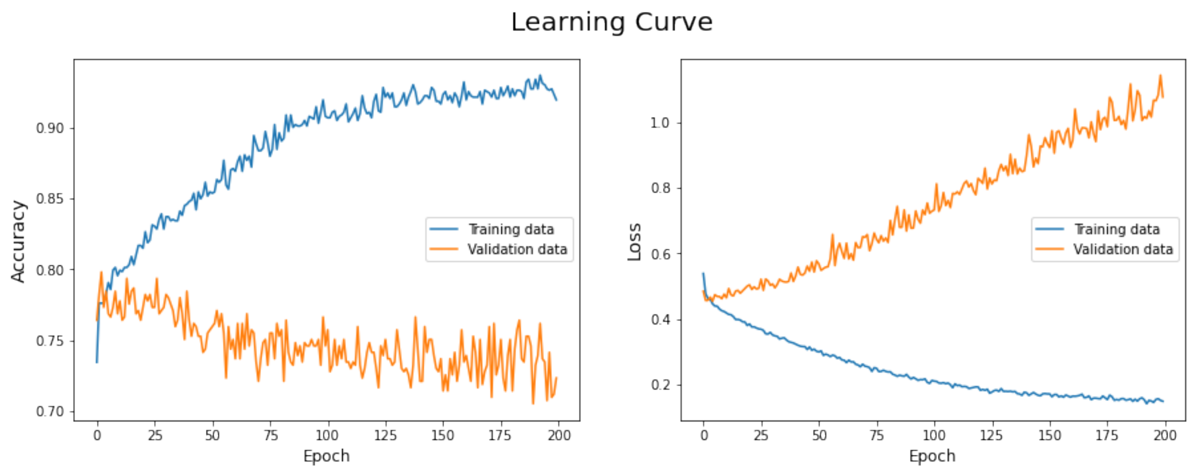

Figure 20: Overfitted Model Learning Curves

The overfitted model produced a ROC curve that looked similar to the curves of the other models, but its AUC value of 0.724 was moderately lower than the others'.

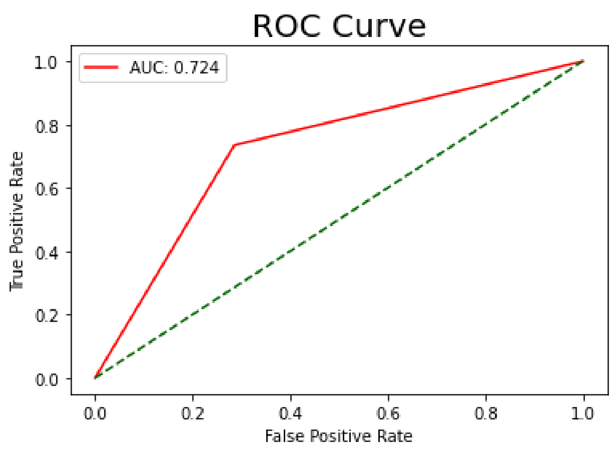

Figure 21: Overfitted Model ROC Curve

7 Checkpointing & Early Stopping

Two callback methods that help to prevent overfitting are model checkpointing and early stopping. In model checkpointing, the model's current best configuration is saved to file. In this case, the best configuration is defined as that which gives the highest accuracy on the validation set. As the model trains, if it reaches a new highest accuracy, then the configuration replaces the saved file. After the model is finished training, the best model checkpoint can be loaded from the file. I built the previous overfitted model as before, but this time I included model checkpointing. The resulting learning curves still showed overfitting, but the model's AUC value was moderately higher at 0.795.

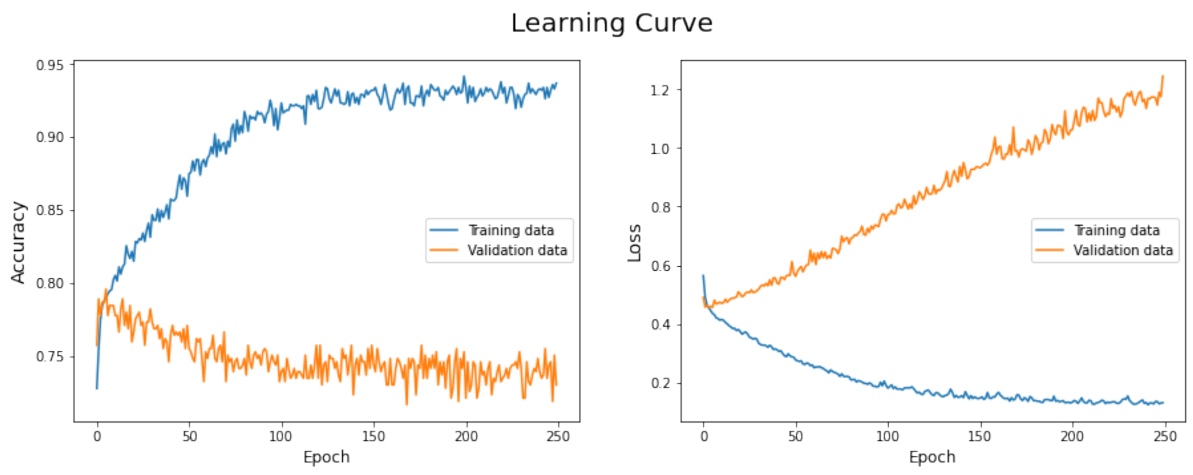

Figure 22: Learning Curves of Overfitted Model with Checkpointing

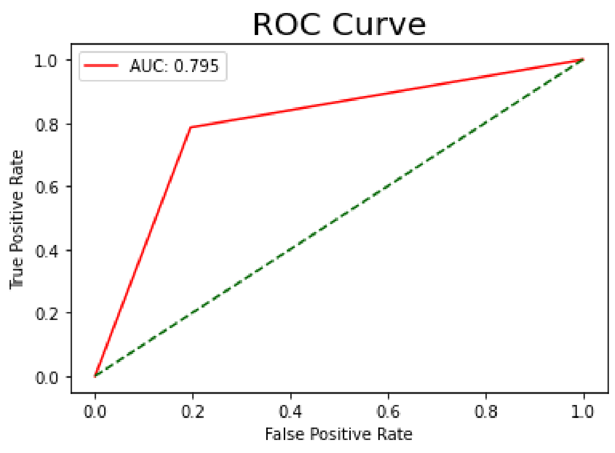

Figure 23: ROC Curve of Overfitted Model with Checkpointing

Early stopping is a callback that keeps track of a specific metric, in this case the minimal loss on the validation set, and stops training when the validation loss starts increasing. The aim is to catch the model when the loss is at its lowest. In the case of early stopping, the learning curves may be disregarded as they are no longer a useful illustration of loss over time. I tested early stopping on the overfit model, but since the validation loss did not strictly decrease with time, the model stopped training after only two epochs. The model achieved a validation accuracy of 80.05 percent and an AUC of 0.799.

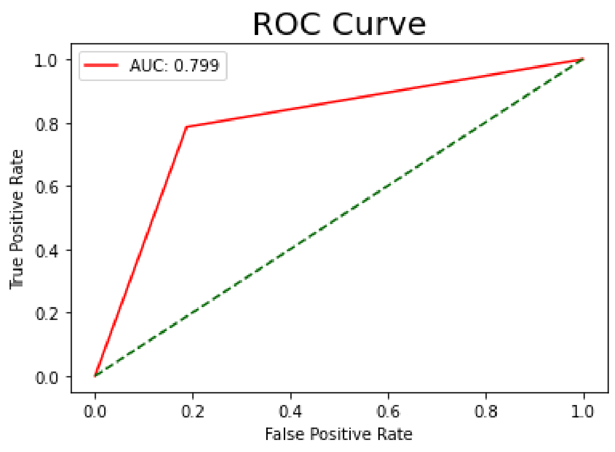

Figure 24: ROC Curve of Overfitted Model with Early Stopping

Finally, I used both checkpointing and early stopping on the overfitted model. Once again, the model stopped training after two epochs, but this time it achieved a validation accuracy of 79.59 percent and an AUC of 0.795.

Checkpointing and early stopping are useful tools that might help me to further hone my chosen model, but I do not think they were particularly useful for this case.

8 Conclusion

I set out to build a model that would perform significantly better than the base model. I started very basic, then branched out by adding more layers and more neurons, but in the end, the best model I built was a simple one with one hidden layers of six neurons. This model produced a validation accuracy of 78.23 percent, which wasn't particularly high when compared to the other models or even the base model, but it also produced a learning curve that stabilized with a minimal gap between the validation loss and training loss.

I believe that this model could have achieved a higher accuracy if the training set had been further optimized. I ensured that the training set contained a balanced amount of samples for each class, but I did not take into account the importance of each input feature and eliminate those that were less important. Doing this would have made the model more focused on the input features that correlated more with the output data. Given the time and resources, I could have also diversified the dataset by adding more samples of underrepresented cases, but this was not feasible since I did not have access to additional relevant data.

One circumstance that I suspect affected the accuracies of the models was the randomization of the training and validation sets in each test session. Whenever I ran a new session of the code, I fed in the original data file and randomly shuffled it, then split the randomized data into training and validation sets. So while I achieved accuracies close to 85 percent in some earlier sessions, it was impossible to replicate these results later on since the training set I was using was completely different. For consistency's sake, it might have helped to shuffle and split the data in the Excel spreadsheet before loading the file into my program.