Phase 3 Report

Megan Joseph

April 26, 2025

1 Summary

The goal of this phase is to train neural networks smaller than the one that overfits and pick the one that has the greatest accuracy on the validation set.

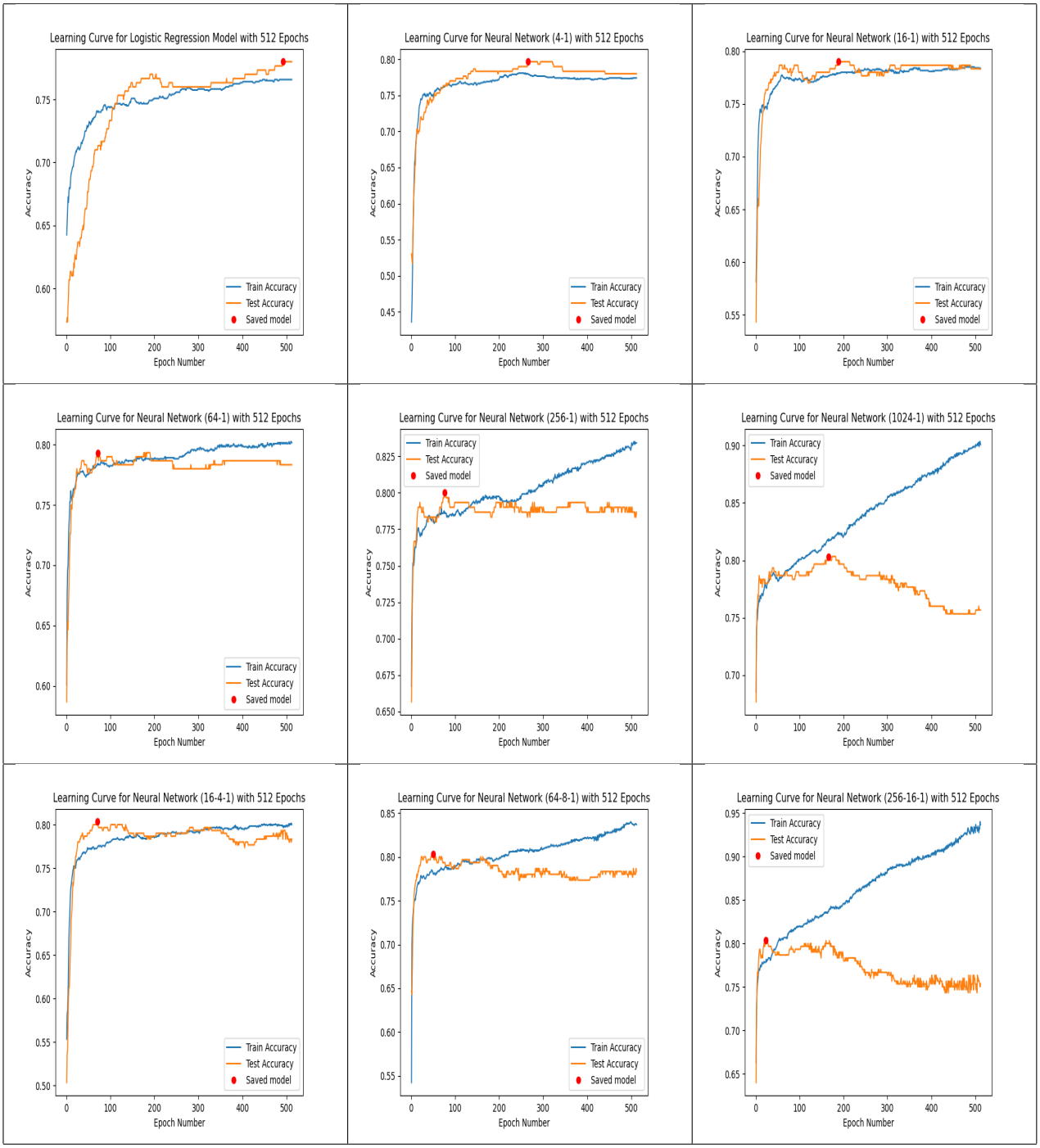
First, I did an 80/20 train test split. I shuffled the indices first before splitting the dataset. Then, I normalized the inputs. After, I created lists for the columns of the table to show the parameters and metrics of each model. The columns are the type of model, the number of epochs, train and test accuracy, train and test loss, precision, recall, F1 score, and the total number of parameters.

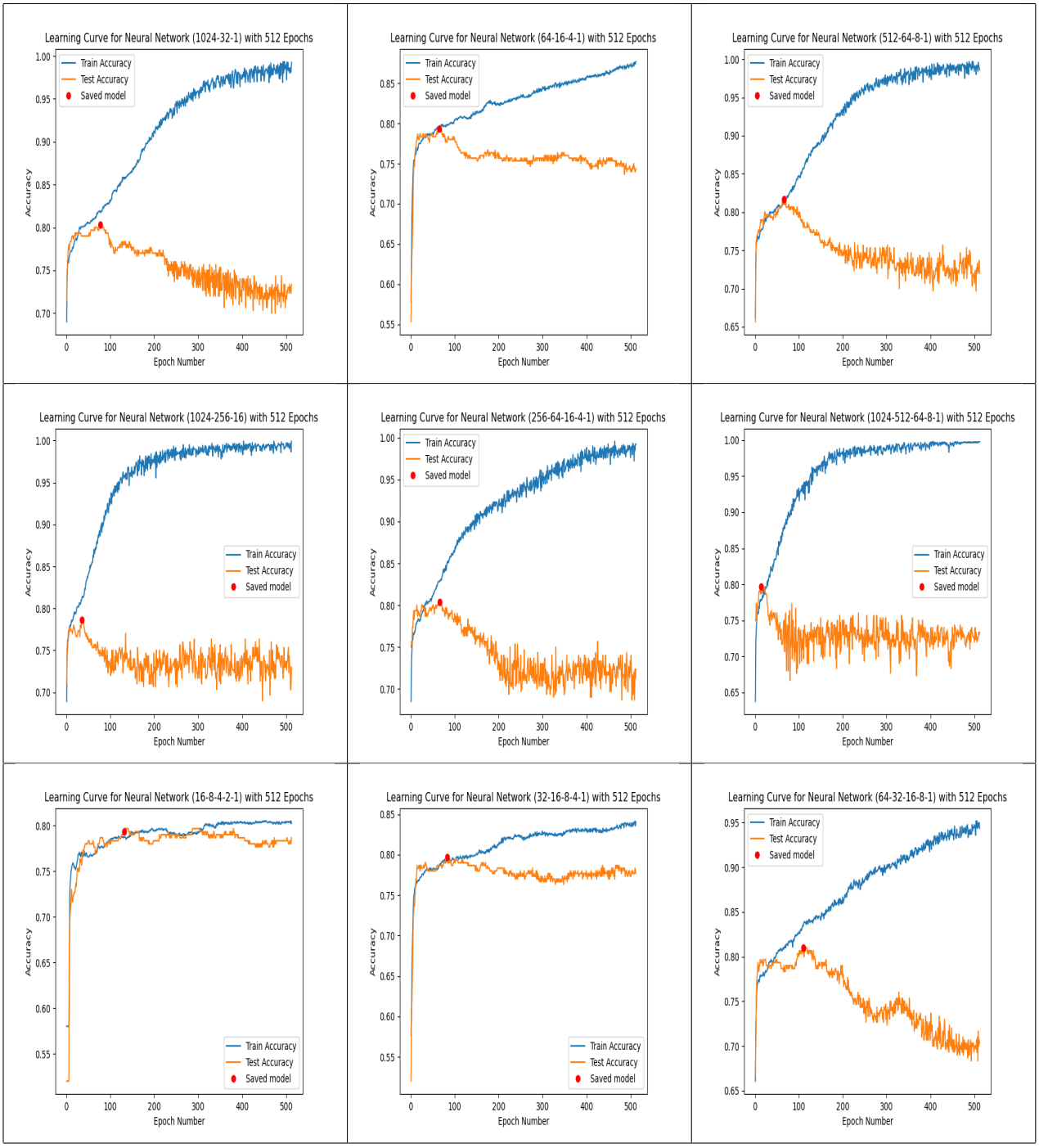
I started out with a random baseline classifier which classifies 50/50 1 or 0. Then, I made functions to make the neural networks part more clean. One function, create_model creates the neural network model. It incorporated model checkpointing by saving the model with the best test accuracy. The next function draws the learning curve with the x-axis as the number of epochs and the y-axis as the test accuracy since that is what we're optimizing. The last function adds the remaining metrics to the lists. The last two functions required me to check my files for the epoch of the best model and the file path.

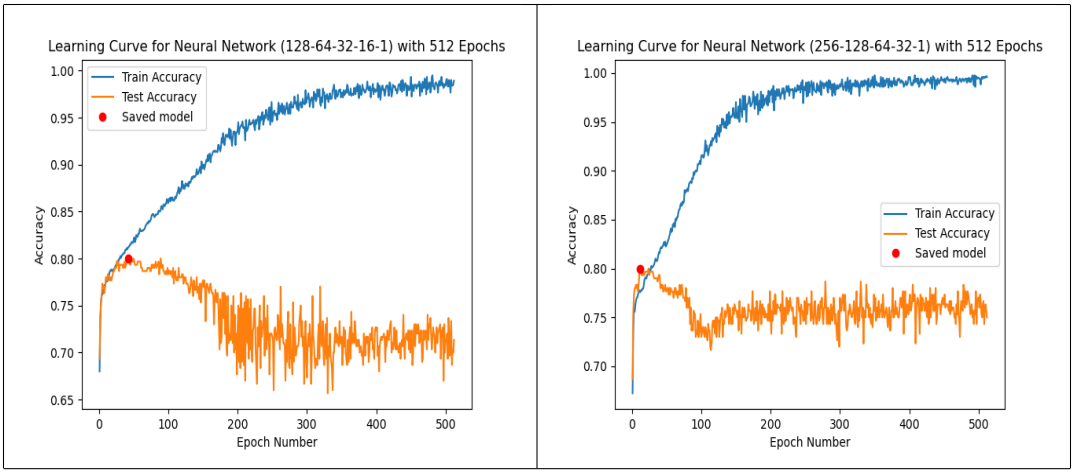
The neural network models I made had between 1 and 5 layers and neurons up to 1024. The following table shows the metrics for each model. The learning curves for each model are also shown below.

	Model	Number of Epochs	Train Accuracy	Train Loss	Test Accuracy	Test Loss	Precision	Recall	F1 Score	Total Parameters
0	random baseline classifier	--	0.500000	17.845270	0.500000	16.693915	0.50	0.44	0.47	--
1	Logistic Regression Model	512	0.765833	0.497768	0.780000	0.494166	0.81	0.71	0.76	8
2	Neural Network (4-1)	512	0.779167	0.477286	0.793333	0.481558	0.82	0.74	0.78	37
3	Neural Network (16-1)	512	0.779167	0.467139	0.790000	0.476660	0.80	0.76	0.78	145
4	Neural Network (64-1)	512	0.784167	0.464641	0.786667	0.477044	0.82	0.74	0.78	577
5	Neural Network (256-1)	512	0.786667	0.447799	0.800000	0.476504	0.81	0.76	0.78	2305
6	Neural Network (1024-1)	512	0.818333	0.394840	0.803333	0.494854	0.82	0.75	0.78	9217
7	Neural Network (16-4-1)	512	0.775000	0.483025	0.800000	0.484524	0.82	0.75	0.78	201
8	Neural Network (64-8-1)	512	0.780833	0.465341	0.800000	0.478991	0.81	0.76	0.78	1041
9	Neural Network (256-16-1)	512	0.781667	0.459659	0.803333	0.478630	0.81	0.76	0.78	6177
10	Neural Network (1024-32-1)	512	0.819167	0.388758	0.800000	0.501092	0.81	0.77	0.79	41025
11	Neural Network (64-16-4-1)	512	0.795833	0.440569	0.793333	0.482842	0.81	0.74	0.77	1625
12	Neural Network (512-64-8-1)	512	0.810833	0.394495	0.816667	0.501285	0.82	0.78	0.80	37457
13	Neural Network (1024-256-16)	512	0.814167	0.398530	0.783333	0.524125	0.81	0.73	0.77	274721
14	Neural Network (256-64-16-4-1)	512	0.829167	0.383657	0.800000	0.506182	0.80	0.79	0.79	19609
15	Neural Network (1024-512-64-8-1)	512	0.778333	0.466138	0.790000	0.491132	0.82	0.74	0.78	566353
16	Neural Network (16-8-4-2-1)	512	0.787500	0.459712	0.796667	0.475381	0.81	0.75	0.78	313
17	Neural Network (64-32-16-8-1)	512	0.837500	0.384278	0.803333	0.505955	0.82	0.78	0.80	3265
18	Neural Network (128-64-32-16-1)	512	0.812500	0.416376	0.796667	0.494844	0.81	0.76	0.78	11905
19	Neural Network (256-128-64-32-1)	512	0.777500	0.468310	0.796667	0.486753	0.81	0.76	0.78	45313
20	Neural Network (32-16-8-4-1)	512	0.793333	0.445318	0.790000	0.481553	0.79	0.78	0.78	961

Figure 1: Table of metrics for each model


Learning Curve for Logistic Regression Model with 512 Epochs
Learning Curve for Neural Network (4-1) with 512 Epochs
Learning Curve for Neural Network (16-1) with 512 Epochs
Learning Curve for Neural Network (64-1) with 512 Epochs
Learning Curve for Neural Network (256-1) with 512 Epochs
Learning Curve for Neural Network (1024-1) with 512 Epochs
Learning Curve for Neural Network (16-4-1) with 512 Epochs
Learning Curve for Neural Network (64-8-1) with 512 Epochs
Learning Curve for Neural Network (256-16-1) with 512 Epochs
Accuracy
Epoch Number
Train Accuracy
Test Accuracy
Saved model


Learning Curve for Neural Network (1024-32-1) with 512 Epochs
Learning Curve for Neural Network (64-16-4-1) with 512 Epochs
Learning Curve for Neural Network (512-64-8-1) with 512 Epochs
Learning Curve for Neural Network (1024-256-16) with 512 Epochs
Learning Curve for Neural Network (256-64-16-4-1) with 512 Epochs
Learning Curve for Neural Network (1024-512-64-8-1) with 512 Epochs
Learning Curve for Neural Network (16-8-4-2-1) with 512 Epochs
Learning Curve for Neural Network (32-16-8-4-1) with 512 Epochs
Learning Curve for Neural Network (64-32-16-8-1) with 512 Epochs
Train Accuracy
Test Accuracy
Saved model
Accuracy
Epoch Number



I noticed that not matter what I changed, the test accuracy would stay at around 0.8. I believe this may be due to the data being simulated and not from a real source. As a result, I ended up picking the Neural Network (512-64-8-1) as the best model since it scored the highest on the validation set. It had 37,457 total parameters which is much smaller than the overfit model. For all of the models, precision was higher than recall and the F1 scores are comparable.

The architecture required to overfit when the output is also a feature is a simple logistic regression model. It does not require too many epochs to overfit.

The functions to represent the model and serve as the prediction are below:

```
# represent model
def model(file_path):
    return keras.models.load_model(file_path)

# prediction model
def predictions(model, X_test):
    num_layers = 4
    curr_input = X_test
    for i in range(num_layers):
        weights, bias = model.layers[i].get_weights()
        if i == (num_layers - 1):
            output = 1 / (1 + np.exp(-((curr_input @ weights) + bias)))
            curr_input = output
        else:
            output = np.maximum(0, (curr_input @ weights) + bias)
            curr_input = output

    curr_input[curr_input >= 0.5] = 1
    curr_input[curr_input < 0.5] = 0

    return curr_input
```

Listing 1: Function to represent the model and function to serve as the prediction model.

The prediction function manually computes the forward pass by multiplying the input by the weights of the layer and adding the bias to it. It then updates the input to the updated version and repeats for the total number of layers in the model. The predictions from the function match the predictions when calling the predict function on the model.