Research question: Recognizing handwritten numbers through a neural network model.

**Data preparation, exploration, visualization**

There is no missing data and all classes are balanced. I normalized all columns by dividing with 255. Later on in the model I use sparse categorical cross-entropy as the loss function, which allows me to avoid doing one-hot encoding on this data, since the values of each pixel is categorical.

The training set is then split 80-20 to a training set and a validation set, since the test sets are hidden.

**Model Development**

Neural networks have an input layer with the same number of nodes as the number of input features, an output layer with the same number of nodes as the number of output classes, and hidden layers in between, where the number of hidden layers and number of neurons at each hidden layer is decided by me.

This experiment uses a completely crossed design, with {1, 3} layers and {32, 128} nodes. In most practical use cases, 1 hidden layer should be sufficient. If we consider the possibility that this problem is a rare problem that requires more hidden layers, I include 3 hidden layers as an alternate choice. All layers use the ReLU activation function. The number of nodes chosen were based on factors of 256 (the number of possible values each input can take). Between each hidden layer, I include a DropOut layer that randomly removes 20% of the nodes, to minimize overfitting. The Softmax activation function is used for the output layer, that for every sample produces an array of probabilities for each class.

I use the Keras API to build each Sequential model with Dense layers. The model is then compiled with the Adam optimizer with a 0.005 learning rate. I use the sparse categorical cross-entropy loss function, to avoid one-hot encoding for the input features. The primary metric I use is accuracy, since that is what Kaggle uses as well. The model is then fitted on the training sets and scored on the validation sets for 20 epochs.

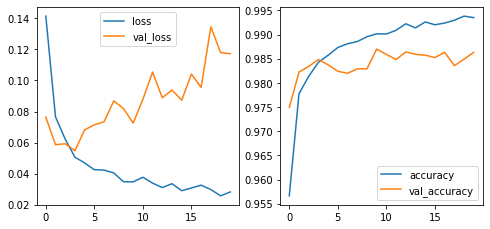
**Model Evaluation**

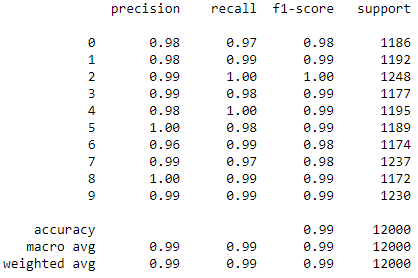
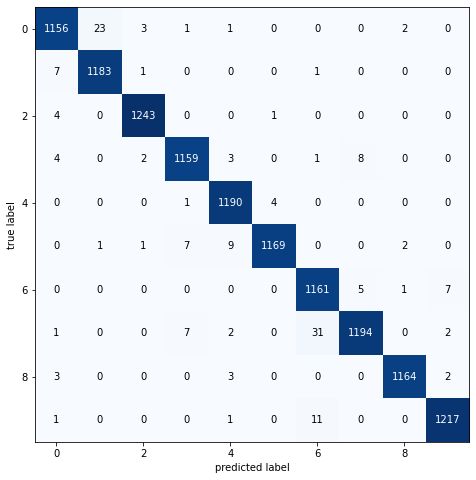
For each model, I produce a classification report and a multi-label confusion matrix (10X10 not 10 of 2X2 matrices). I also look at the graphs of loss and accuracy, plotting the training values with the validation values to see if overfitting occurs. The time for the model to fit is also considered. Models are then run on Kaggle against the hidden test sets.

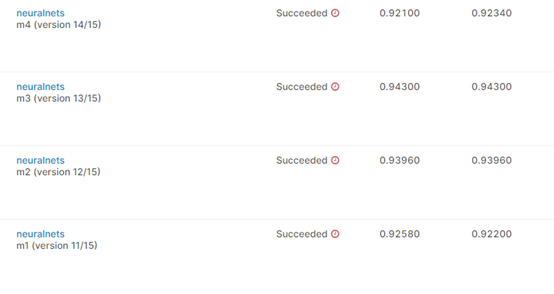
**Results**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hidden Layers | Nodes | Time(s) | Training Acc | Training Loss | Validation Acc | Validation Loss | Kaggle Private | Kaggle Public |
| 0 | 0 | 28.2 | 0.983021 | 0.055778 | 0.9645 | 0.137465 | 0.895 | 0.8974 |
| 1 | 32 | 31.13 | 0.995708 | 0.012361 | 0.9775 | 0.12441 | 0.9258 | 0.922 |
| 1 | 128 | 31.68 | 0.998708 | 0.003916 | 0.98633 | 0.117216 | 0.943 | 0.943 |
| 3 | 32 | 37.916 | 0.983583 | 0.058939 | 0.97333 | 0.108936 | 0.921 | 0.9234 |
| 3 | 128 | 35.42 | 0.993333 | 0.025099 | 0.98217 | 0.124997 | 0.9396 | 0.9396 |

Below are my output figures for the best result, 1 hidden layer with 128 nodes







**Insights**

The best result came from 1 hidden layer with 128 neurons. Almost all the runs were better than my Random Forest models from the previous week (0.9246 score). The loss-accuracy graphs do a good job of showing that after some time, more iterations lead to overfitting the training set without reducing the error from the validation set.

The fitting times show that increasing neurons (4X in this case) did not increase fitting time in all cases, but increase layers did. Although in this experiment, the fit time increase from 1 to 3 layers was only around 5 seconds, keep in mind that I only used 20 epochs, whereas in real usage scenarios hundreds to thousands of iterations are used. The small increase at each step will lead to much longer fitting time overall.

The model without any hidden layers is surprisingly decent, with training and validation scores within 0.01 of the other NNs and a Kaggle score that, without comparing to other contestants, isn’t too bad. This might suggest that plenty of features are linearly separable.