Reed Ballesteros
MSDS 458-DL: Artificial Intelligence & Deep Learning
Spring 2023
Prof. Syamala Srinivasan, Ph.D.

A.1: First Research/Programming Assignment

Abstract

This study details the development and comparison of several multilayer perceptron (MLP) models using the MNIST dataset. The most optimal model from the ones created will be chosen based on their accuracy using test data. While the first thought that simply adding more units to the single hidden layer would yield the best accuracy, integrating data dimensionality reduction processes such as principal component analysis (PCA) also yields significant accuracy gains. The combination of more units and additional data dimensionality reduction would show its limits as these models in rising complexity eventually start showing signs of accuracy performance degradation.

Introduction

The MNIST dataset is an image database containing 70,000 greyscale images of handwritten numeric characters at 28x28 pixel resolution. It is a dataset that is commonly used to train image processing systems. Led by LeNet developer Yann LeCun, the MNIST dataset was created further expanding on the convolutional neural network progress from LeNet as it was used by the United States Postal Service (USPS) to read handwritten zip codes on envelops. Today the MNIST dataset is used as an introduction to the deep learning process for people looking into the field of data science. Creating a CNN from a neural network trained by the MNIST dataset is considered to the 'Hello World!' equivalent in using deep learning programming tools such as TensorFlow and Kera.

This study will use the MNIST data to create single hidden layer CNNs of various configurations in terms of the number of units used in the hidden layer of the network and different applications of data reduction. From these models, we will choose the best one that yields the highest accuracy.

Literature Review

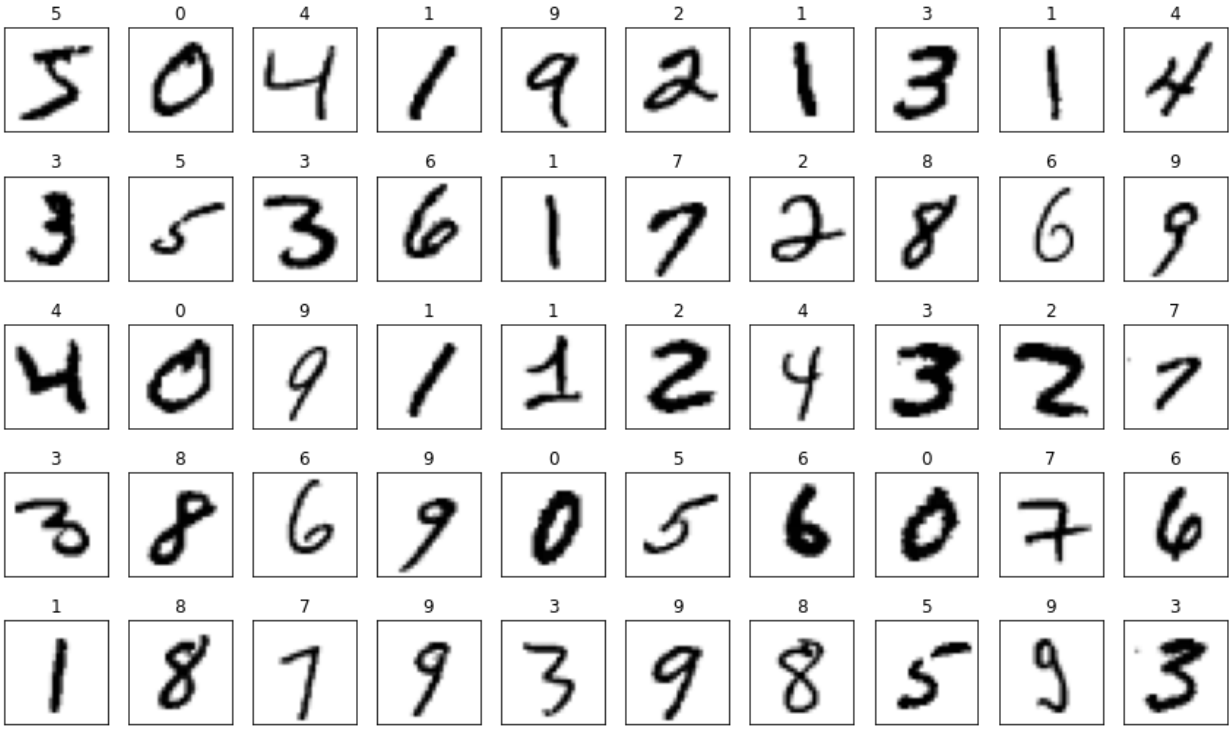
There have been many studies using the MNIST dataset to create a deep learning model that yield very high accuracy. A 2018 study by Akmaljon Palvanov and Young Im Cho used the MNIST data to train and compare 4 models: capsule network (CapsNet), deep residual learning network (ResNet), CNN, and multinominal logistic regression (Palvanov et al. 2018). Their results found that the capsule network-based model yielded high accuracy, trains quickly, and runs fast when applied to a real-time application. That being said, their CapNet model uses a more complex architecture that contains three hidden layers

of 512, 1024, and 784 units respectively. Our study will be limited to a single hidden layer of various unit sizes.

Modeling Methods

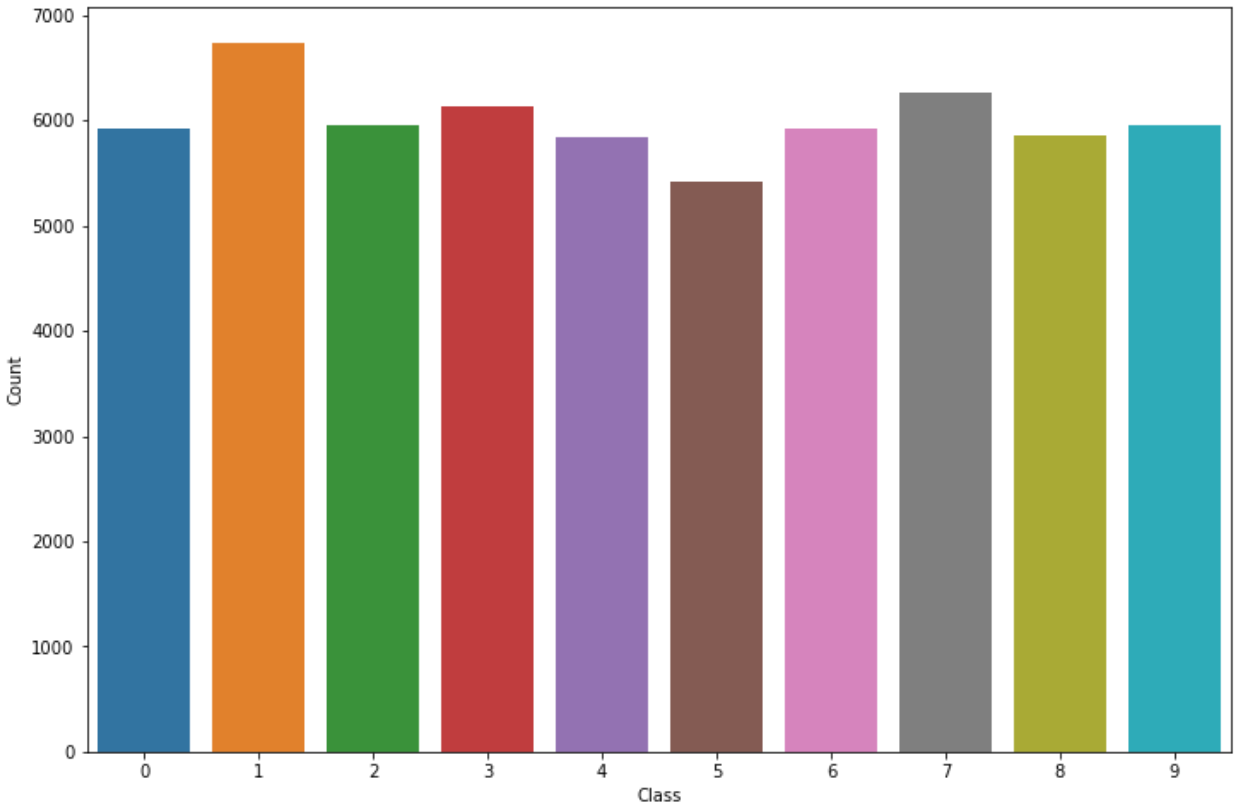
Exploration/Visualization

An explanatory data analysis (EDA) is conducted to observe the MNIST data. Below is a sample of the first 50 images in the dataset:



The sample above shows various handwritten numeric characters along with their corresponding label above. From just this relatively small sample from 70000 images one can already notice several variations of a number, such as how the number ‘2’ is written with or without a lower loop; the number ‘7’ is written with out without a middle cross; or the number ‘1’ is fully decorated with a ‘base’ and a ‘roof’ or is simply a vertical or slightly slanted line. The number ‘3’ from some samples above can have potential cause for misclassification due to questionable legibility of how they were written by hand.

The distribution of the MNIST training set labels yield the following chart:



Based on the chart above, the MNIST training dataset has a relatively even distribution of numeric class labels, with images classified at '1' having the most number samples and '5' having the least number of samples. That being said, their respective gaps do not seem to deviate very far from the average.

We can view a raw sample from the MNIST dataset. This is the data from the 30000 index of the training set, shown as a 28x28 matrix:

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  47 156 205 254 255 112   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  53 208 245 253 253 253 240 249  50   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   1  73 242 248 212 128  56  56 122 253  94   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 101 253 211  64   0   0   0   0  66 253 212   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 126 143  15   0   0   0   0   0  66 253 226   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 115 253 142   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  32 254 253 119   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   7 129 254 253 252 244  95   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 120 253 254 238 225 253 246  50   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  42 243 218  66  32   3 121 253 175   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 116  60   0   0   0   0 236 247  47   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 103 253 135   2   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  28 230 253  47   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 113 253 103   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  97 243 237  14   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   2  19  15   0   0   0   0   0   6 184 251 155   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  86 253 236  26   0   0   0  35 169 253 167   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  53 236 253  79   0  96 199 248 253 169  22   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89 252 249 216 240 248 221 103  17   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84 239 253 170  56   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

The integer values in the 28x28 matrix represent a greyscale value from 0 (white) to 255 (black) of each pixel in the image. From this particular data we can see the matrix form the number '3,' which is what this data is labeled.

Research design

Our modeling method will use a multi-layer perceptron (MLP), which is a feedforward neural network. The model will take each item in the MNIST dataset as an input, process it in a hidden layer containing a given number of nonlinear processing units, and exits to an outer layer containing 10 units, one for each numerical value, as the result. The values of all 10 units in the outer layer contain a probability of their respective numerical value, and the one containing the highest probability is the final result. The training dataset of 60,000 items enables the MLP to learn its classification task by minimizing errors through backpropagation and adjusting the weighted values between the units in each layer.

Our research design will create multiple MLPs using a various number of units in our hidden layers. Some models will use a transformed version of the MNIST dataset via dimensionality reduction techniques such as principal component analysis (PCA) or random forests. We will compare our models based on the accuracy of test data, as well as cross entropy loss.

Implementation

Our neural network models will be created using the Keras and TensorFlow frameworks that are available in Python. These are the most common tools used in developing such models based on their ease of use, the customization options available, and its scalability in how many units we can use for leach layer in the network. Each layer in our neural network are sequential dense layers from Karas. The single hidden layer uses a ReLU activation function as its simplicity in calculation can make the feedforward and backpropagation process fast, which can especially help when we develop models that

scale with a large number of units in the hidden layer. The outer layer function will use a softmax activation function, which is typically used for classification tasks, which is our case for classifying each image in the MNIST dataset as one of 10 numerical values.

The MNIST dataset will require some data transformation before it is input into our neural network models. As we observed in the raw dataset, an image is represented as a 28x28 matrix, or 2-dimensional data. The neural network requires the data to be reshaped into 1-dimensional data representing all features lined up in an array. The greyscale range of the pixel data (0 to 255) will also need to be scaled to a float value between 0 to 1 before the data can be input into the neural network. Depending on the model, will further transform the data using PCA or random forests for feature reduction but will still need to be input into the neural network as a 1-dimensional array, though with less features. The training dataset will be split where 5000 samples will be set aside for validation testing during the model fitting process.

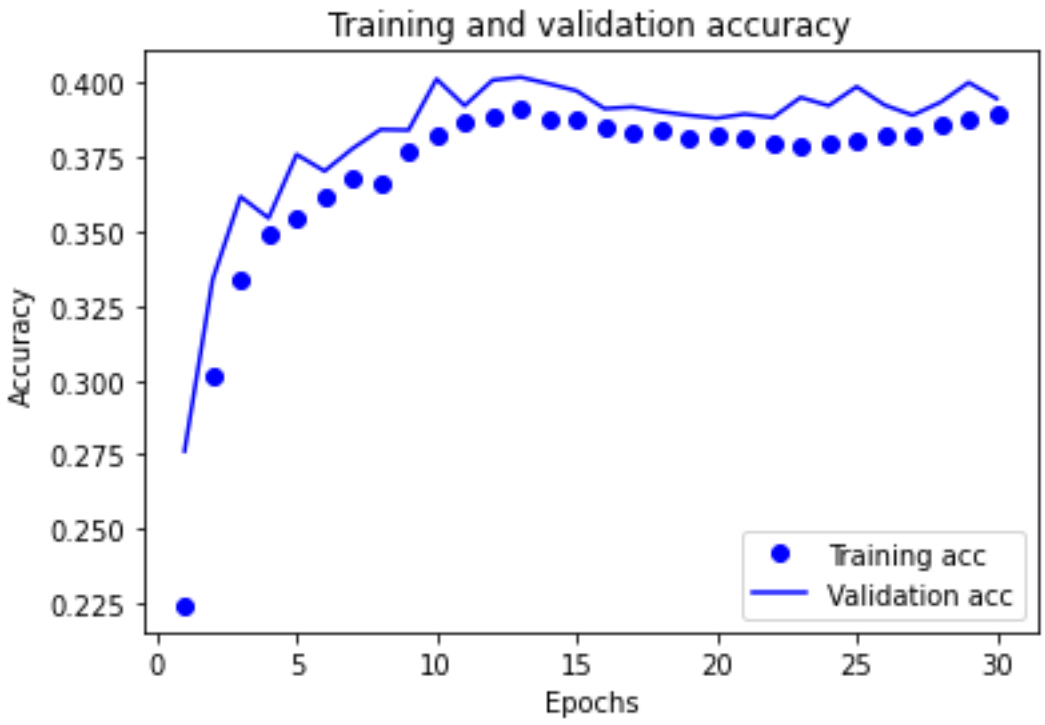
The Keras model compiler will be configured to use an RMSprop optimizer, calculate cross entropy loss using the SparseCategorialCrossentropy loss class, and model performance will be based on accuracy metrics. The model fit process will run though a number of given epochs, saving the best version of the model before the training data overfits it in later epochs.

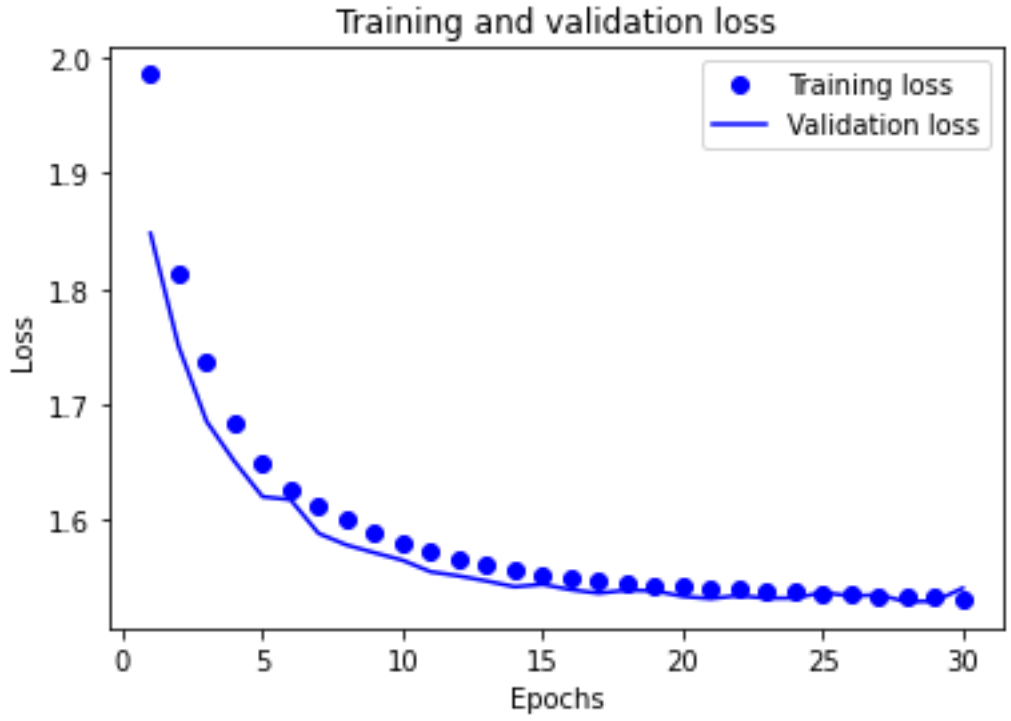
For reproducibility and stabilize the random nature of the model training process, we set a system seed of 43.

Results

Experiment 1: Single unit in the hidden layer

Our first experiment started with a very simple neural network containing only a single nonlinear processing unit in the hidden layer. We ran 30 epochs where both the training and validation accuracy and cross entropy loss were consistently improving over the course of the epochs.





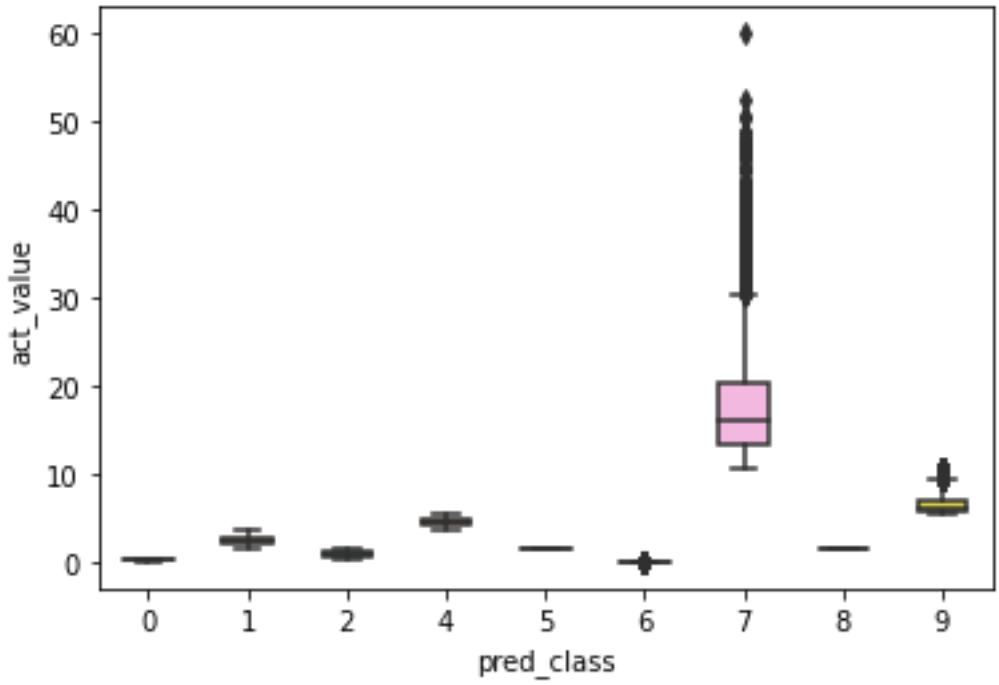
As we can see in the graphs above, we don't see the validation accuracy or the loss diverge from training accuracy or loss over the 30 epochs of model fitting. But the resulting model only gives us a test data accuracy of 37.8%. While not great, just having a single unit to process all 784 pixels from the greyscale image from the MNIST dataset is still better than the 10% random chance of classifying the image to one of the ten numerical values.

This experiment gives us the following confusion matrix:

```
array([[ 168,  118,  557,    0,    2,   27, 4537,    0,   35,    0],
       [   5, 5712,  133,    0,  154,   52,   45,    0,   76,    2],
       [ 207,  623, 1208,    0,   55,  104, 3161,    2,  102,    8],
       [  47, 3664,  698,    0,  388,  167,  314,   21,  240,   99],
       [   4,  792,   77,    0, 3127,   12,   31,   35,   16, 1213],
       [  74, 2702, 1084,    0,  205,  244,  275,   35,  314,   54],
       [  92,   62,  362,    0,    2,   29, 4851,    0,   19,    0],
       [   0,  261,   20,    0, 1072,    6,    5, 3386,    2,  963],
       [  32, 3522,  814,    0,  338,  233,   76,    9,  339,   26],
       [   7,  299,   21,    0, 1629,    4,   11, 1226,    7, 2250]], dtype=int64)
```

We see that the model is not able to classify any image as 3 as its whole column is 0. Ideally the confusion matrix should only have values in a diagonal from the top left to the bottom right, but we see the model makes many misclassifications.

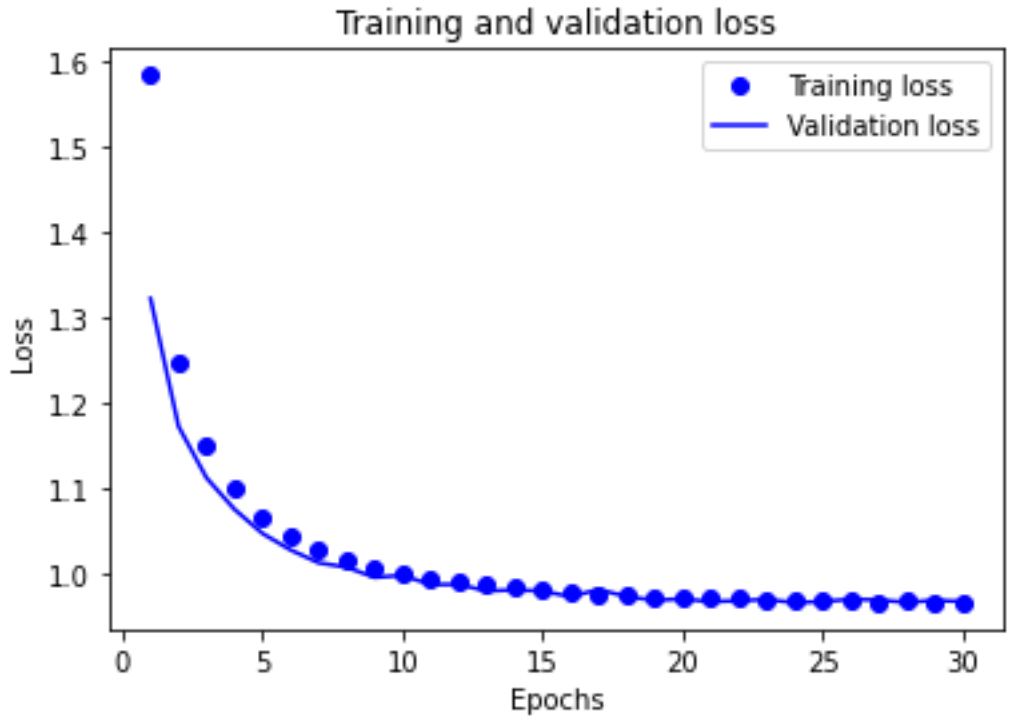
We can observe the overlap of the classification using a boxplot of the classification data:

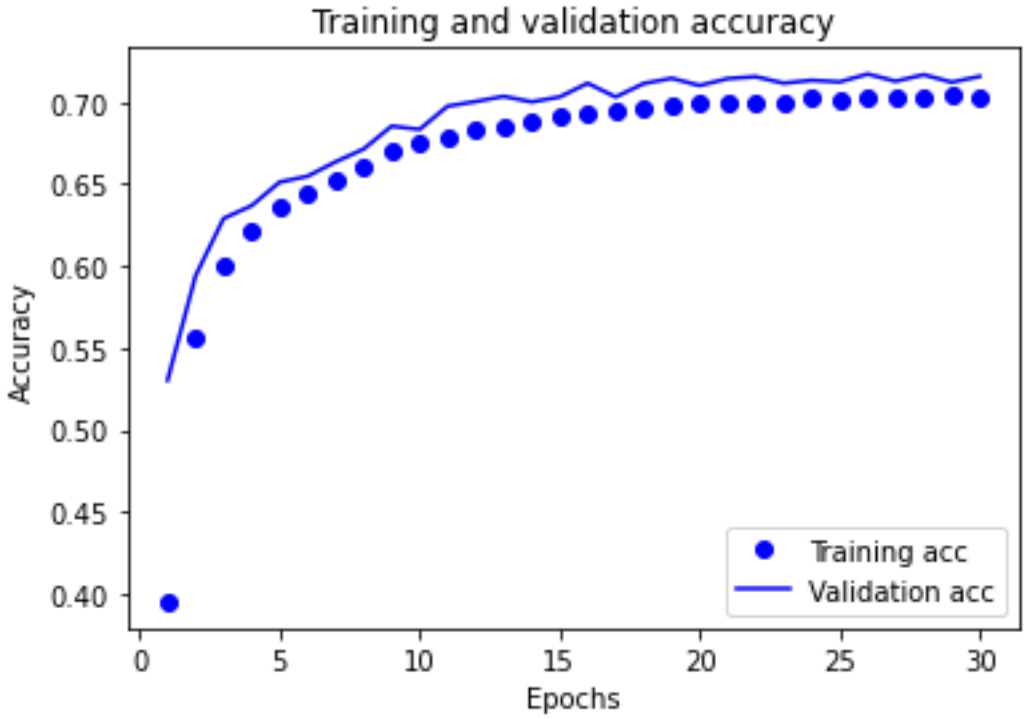


With the boxplot we can see some predicted classes overlap with other classes. For example, 1 overlaps with classes 3, 5, and 8. It shows that the model classifies an image as ‘1’ but is really a 3, 5, or 8. We can wee that with the high number of misclassifications for the 1 column (3665, 2702, and 3522 respectively) in the confusion matrix above.

Experiment 2: Two units in the hidden layer

Experiment 2 simply adds another unit to the hidden layer of our neural network, which makes it 2 units total. Again, we ran 30 epochs where both the training and validation accuracy and cross entropy loss were consistently improving over the course of the epochs.



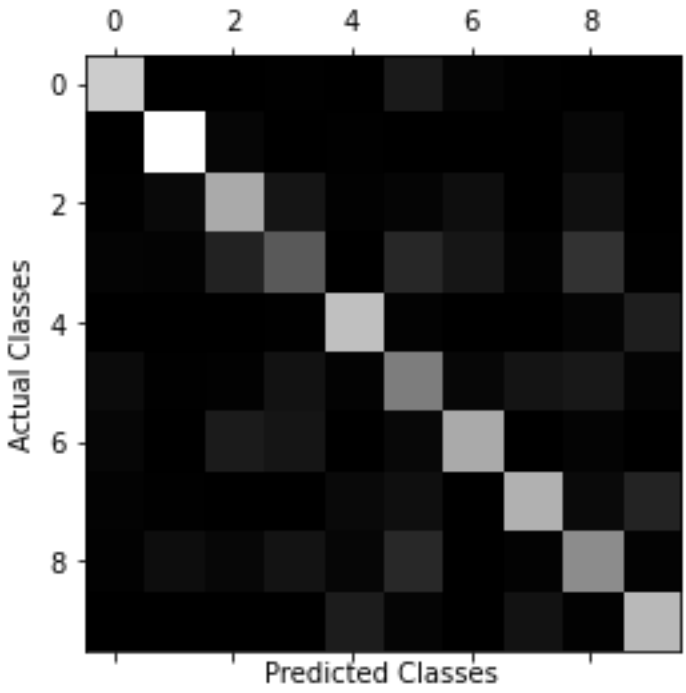


We get a significant improvement with test set accuracy at 70.61%. Still not great, but we see how much of a difference we get from just adding one additional unit to the hidden layer.

Our confusion matrix is the following:

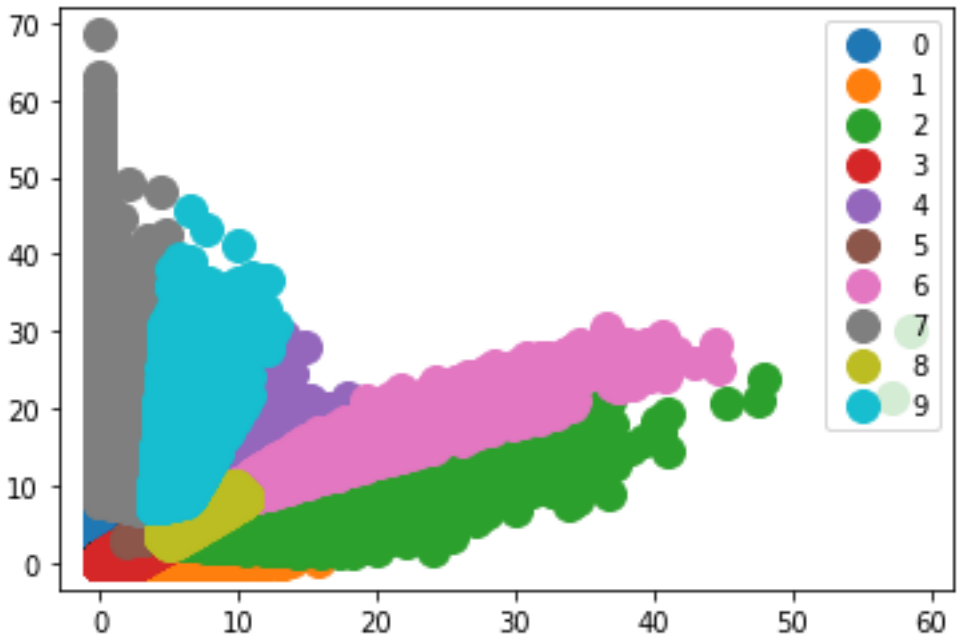
```
array([[4617,    2,    0,   39,    7,  594,  115,   42,   21,    7],
       [   0, 5775,  138,   16,   37,   10,   21,    5,  173,    4],
       [  23,  204, 3840,  485,   63,  131,  334,   14,  365,   11],
       [ 104,   75,  774, 2045,   19,  894,  500,   70, 1128,   29],
       [   1,   14,    6,   31, 4339,   70,   14,   21,  122,  689],
       [ 268,   29,   54,  415,   75, 2842,  177,  465,  553,  109],
       [ 141,   17,  623,  483,   12,  188, 3843,    4,   98,    8],
       [  69,   35,    8,   15,  220,  339,    2, 3997,  234,  796],
       [  46,  305,  177,  447,  145,  924,   22,   73, 3171,   79],
       [  22,    4,    2,    3,  639,  128,    1,  423,   56, 4176]],
      dtype=int64)
```

We can see the larger classification values lie along the diagonal line of the matrix. We can visualize the number of classifications with the following chart:



Again, we can see the intensity of classifications lie along the diagonal of the chart.

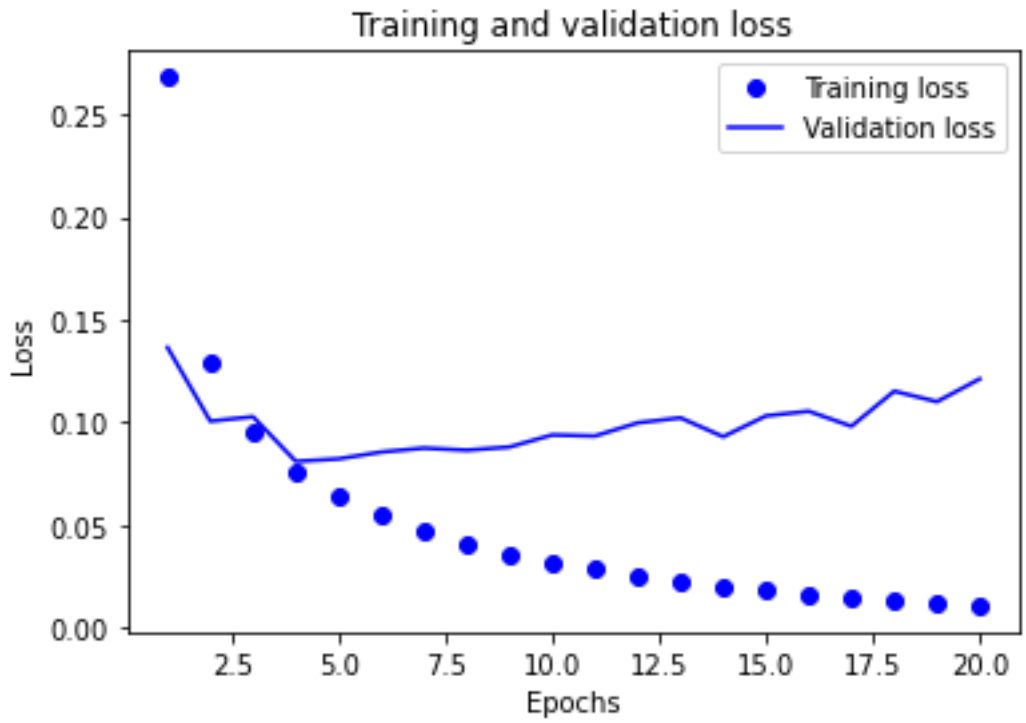
Using the combined activations of the units in the hidden layer, we can create a scatterplot of the classifications:

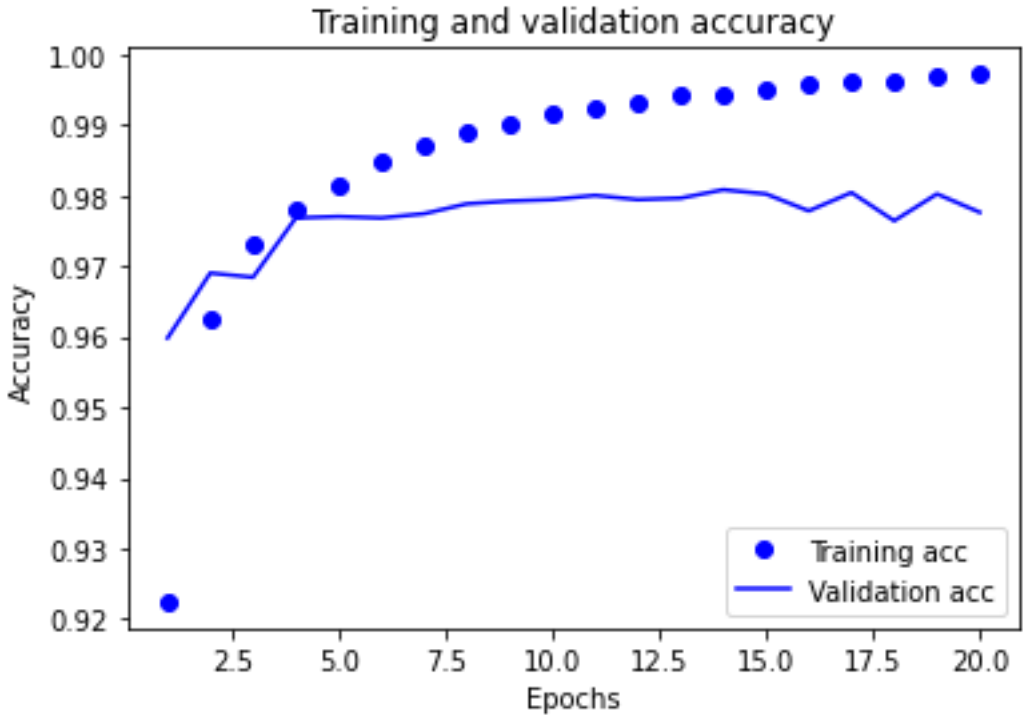


The scatterplot above shows that classifications 2, 6, 7, and 9 spread out and don't overlap the other classifications as much as the other values. 3, all the way at the lower right hand corner, overlaps with other classifications. With the confusion matrix above, we can see that the 3 classification has the least number of classifications to itself compared to the other numerical classifications.

Experiment 3: 128 units in the single hidden layer

Experiment 3 adds a significant number of units to the hidden layer, this time 128 units total. We ran 20 epochs to train the dataset with this model, but early epochs show the validation accuracy and loss diverging from the training accuracy and loss:

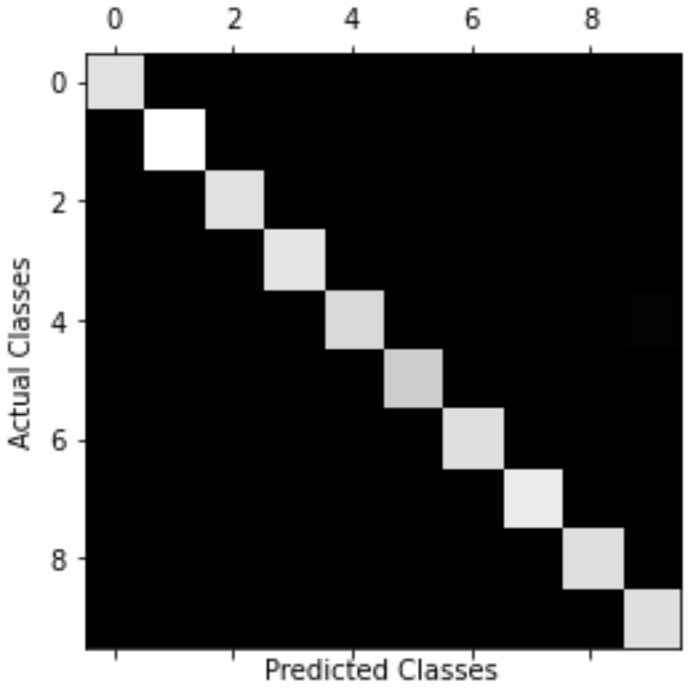




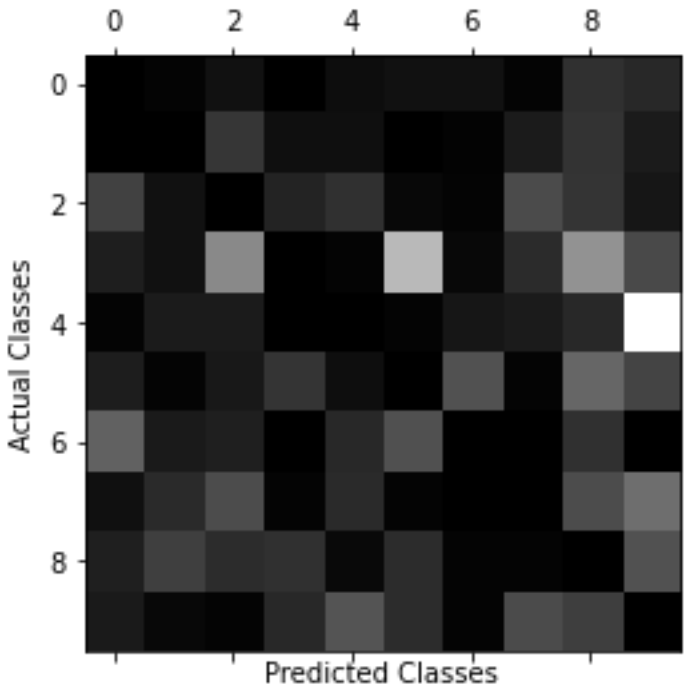
The validation results diverging from the training results indicate signs of the model overfitting to the training data. That being said, the resulting test set accuracy is now at 97.45%, which is highly accurate.

The model yields the following confusion matrix:

```
array([[5407,    1,    4,    0,    3,    4,    4,    1,   11,    9],
       [   0, 6129,   14,    4,    4,    0,    1,    7,   13,    7],
       [  15,    4, 5395,    8,   11,    2,    1,   17,   12,    5],
       [   7,    4,   32, 5488,    1,   43,    2,   10,   34,   17],
       [   1,    6,    6,    0, 5217,    1,    5,    6,    9,   56],
       [   6,    1,    5,   11,    3, 4908,   17,    1,   21,   14],
       [  22,    6,    7,    0,    9,   18, 5344,    0,   11,    0],
       [   4,   10,   18,    1,   10,    1,    0, 5627,   18,   26],
       [   7,   14,   10,   11,    2,   10,    1,    1, 5315,   18],
       [   6,    2,    1,    9,   19,   10,    1,   17,   14, 5375]],
      dtype=int64)
```



With a 97.45% test accuracy we see the majority of classifications lie along the diagonal of the confusion matrix, with few mismatches. Though the model is highly accurate, let us see what mismatches are occurring:

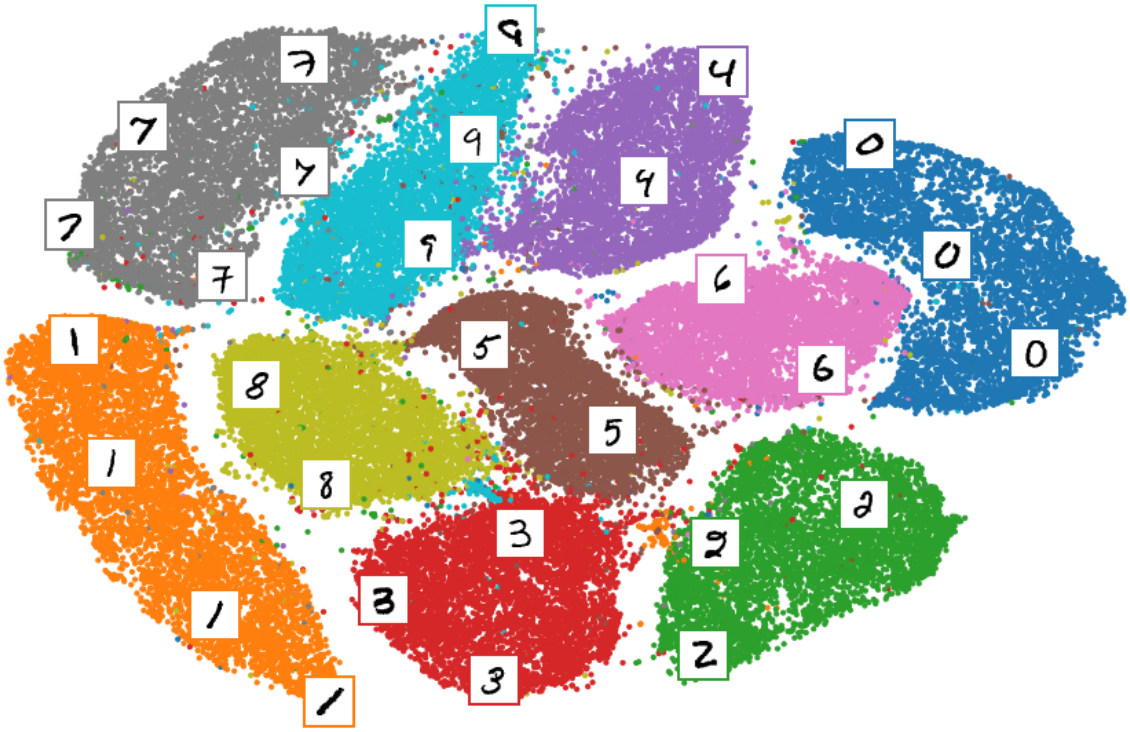


The intense white area of the chart above show that the most misclassifications are images of 4s being classified as 9s. Let us observe a sample of the raw data classification between the two numbers:

Some of the misclassifications between 4s and 9s could be due to the ‘closed’ or ‘curved’ handwritten versions of 4, which makes it similar to 9s. Handwritten 9s that are not fully closed could be classified as 4s.

Using 128 nodes in the hidden unit results a complex neural network containing 101,770 trainable parameters. A criticism of deep learning algorithms is that due to this overwhelming complexity of parameters that need to be adjusted for each backpropagation process these models are considered to be a ‘black box.’ But similar to experiment 2, we can combine the activation values of the units in the

hidden layer and visualize the classifications.

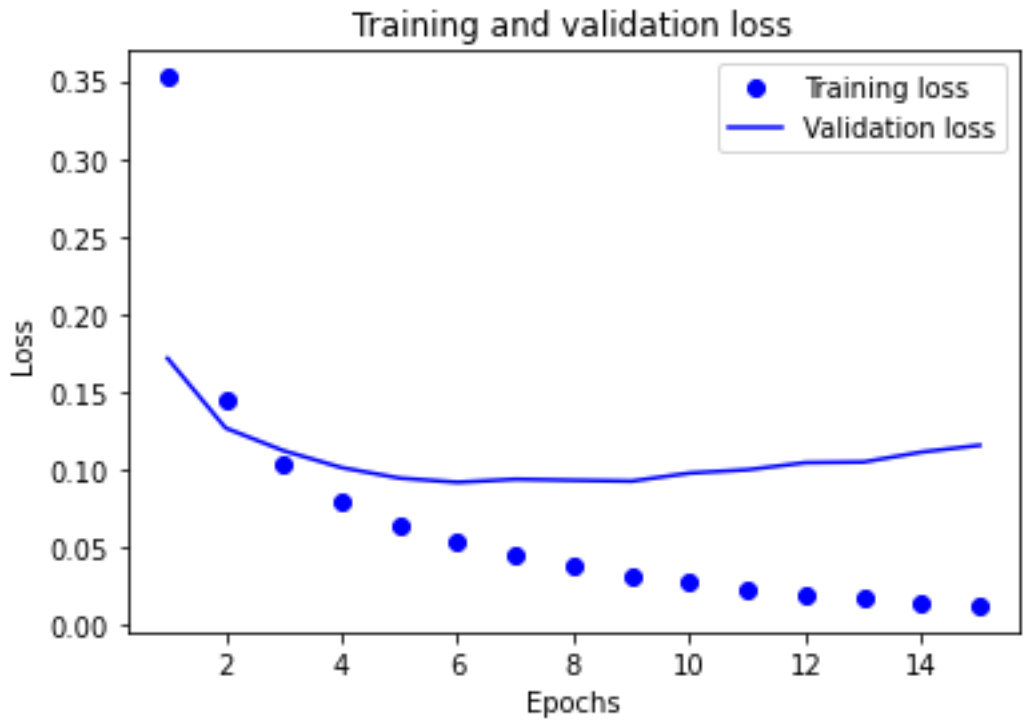


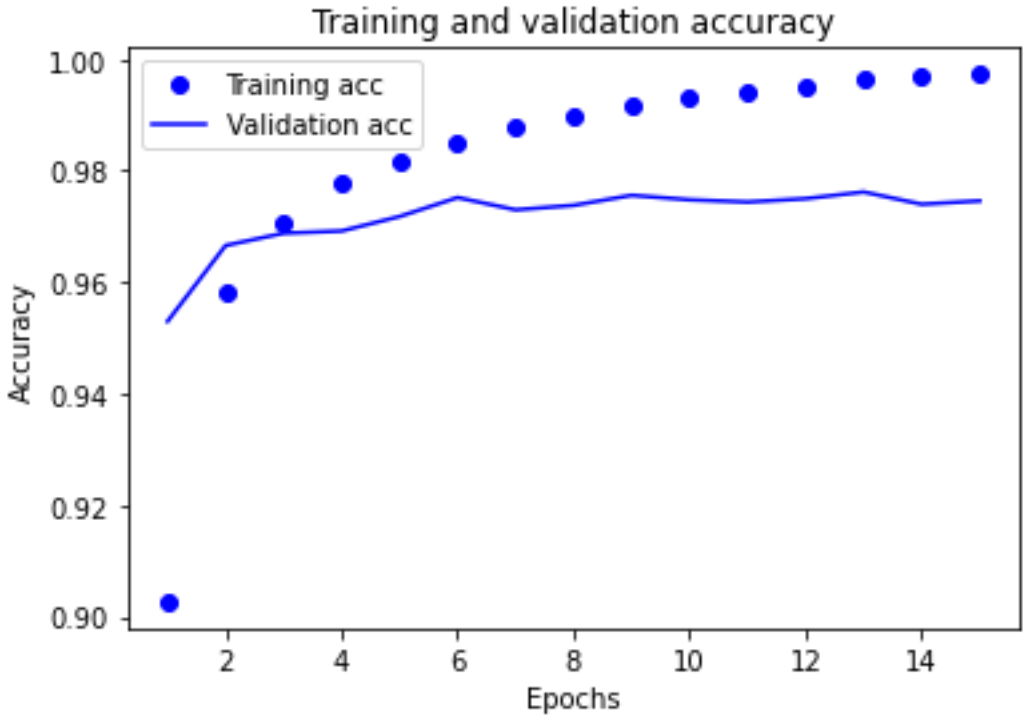
With such a high accuracy of 97.5%, we can see distinct classifications form in this reduced dimensionality scatterplot with very low overlap.

Experiment 4: Dimensionality reduction via PCA (95% of variance) over 85 units in the hidden layer

In experiment 4 we attempt to reduce the number of features for each image from 784 to 154 using PCA, resulting in a smaller data to process in the neural network. While the reduction is significant, the data still represents 95% of the variance from the original image. We apply the reduction to the training, validation, and test sets, and run 15 epochs on a neural network containing 85 units.

Our training model loss and accuracy is the following:



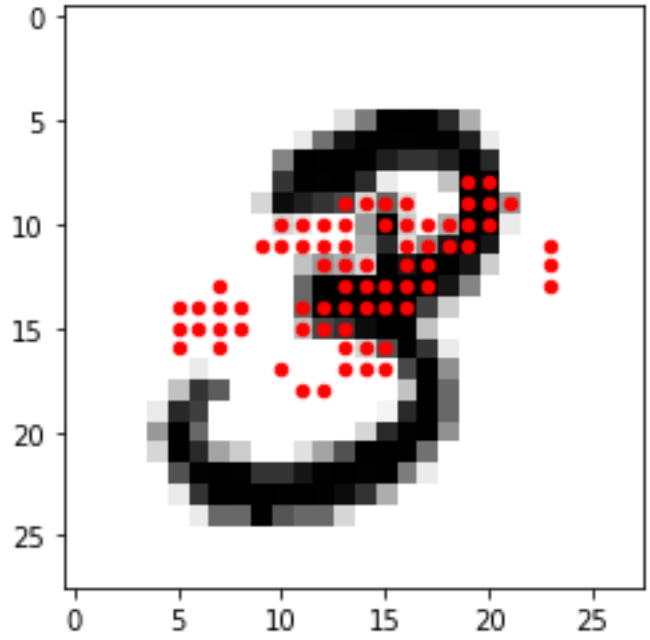


Again we see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The test accuracy resulted in a 97.71% which is a better result than experiment 3, despite the data reduction and a lower number of units in the hidden layer.

Experiment 5: Data dimensionality reduction using Random Forests and 85 units in the hidden layer

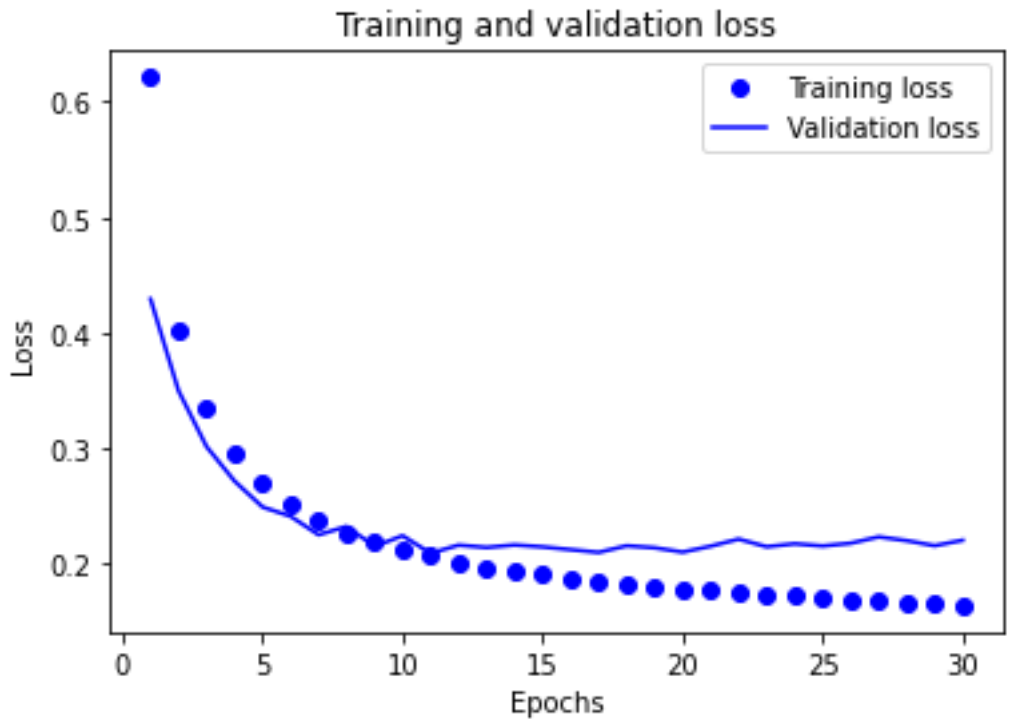
In Experiment 5 we attempted to reduce the number of features for each image from 784 to the 70 'most important features' using random forests. We then apply the reduction to the training, validation, and test sets, and run 30 epochs on a neural network containing 85 units in the single hidden layer.

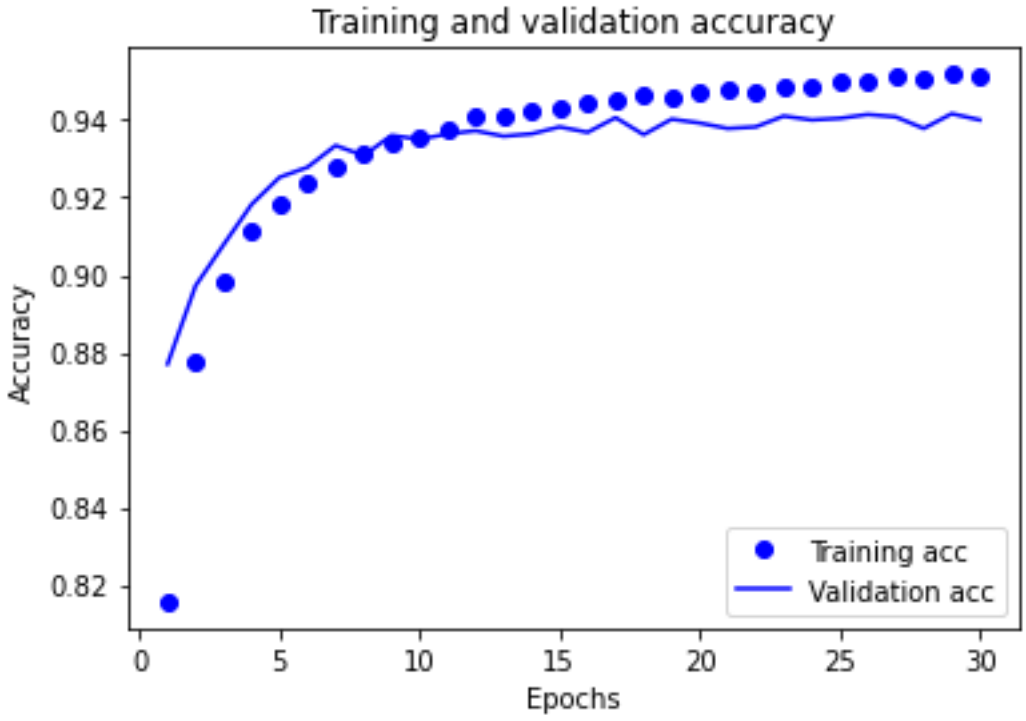
For example, the reduction would use the following pixels marked in red in the image below to use in the neural network:



While we might observe that a questionable area of the image is being used, we will run our experiment and observe its results.

Our training model loss and accuracy is the following:





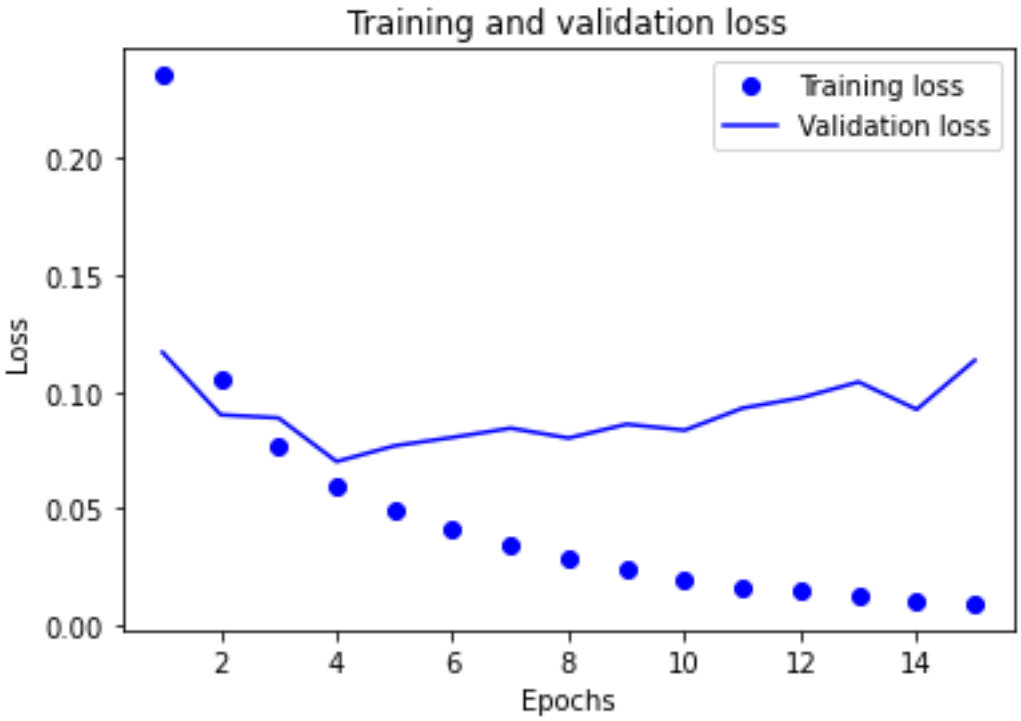
The validation loss and accuracy slightly diverge from the training loss and accuracy over the 30 epochs. The model fitting results in a 93.08% test accuracy. While not as good as experiments 3, and 4, it is a decent result given the model only uses 70 pixels to classify an image in its 85-unit hidden layer.

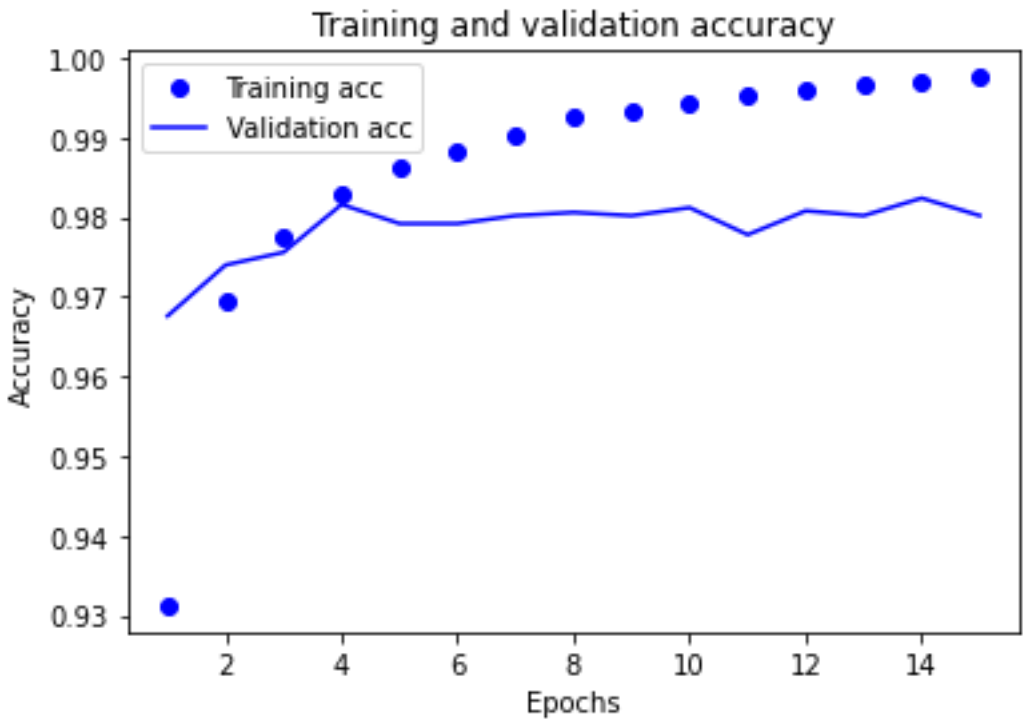
So far, we've seen how effective using PCA on a dataset can be for a neural network. Encouraged by these findings, experiments 6 to 11 will use combinations of increased data compress via PCA and increasing the number of units in the hidden layer.

Experiment 6: Dimensionality reduction via PCA (95% of variance) over 128 units in the hidden layer

In experiment 6 we use the reduced dataset of 154 features via PCA and run 15 epochs on a neural network containing 128 units.

Our training model loss and accuracy is the following:





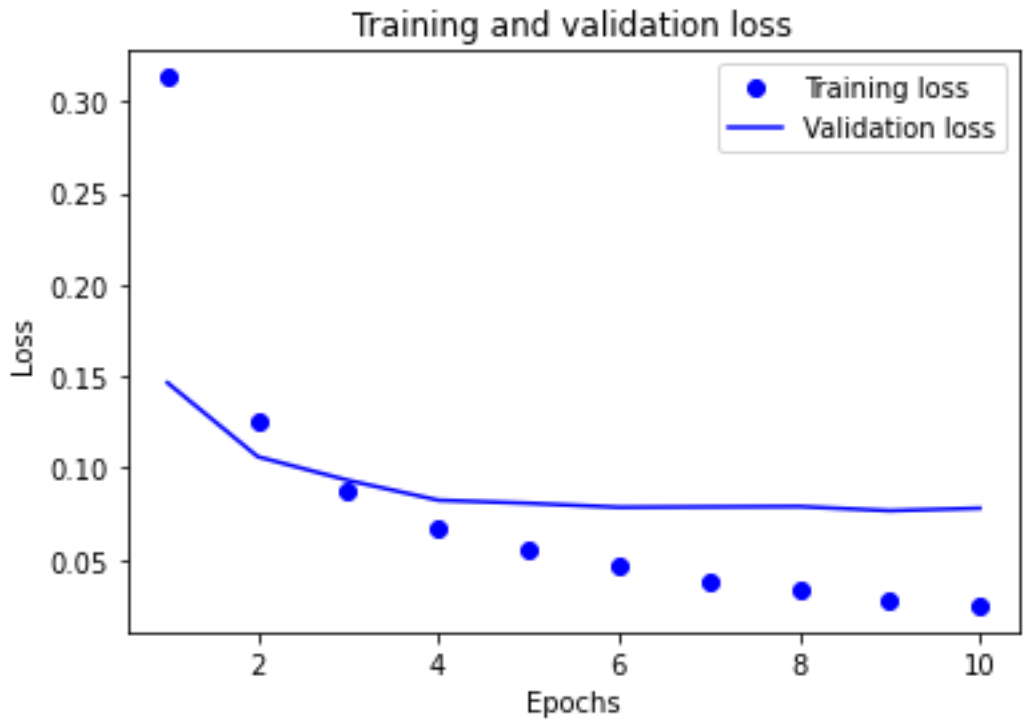
We see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The increase in units improves test accuracy over experiment 4 to 97.54%.

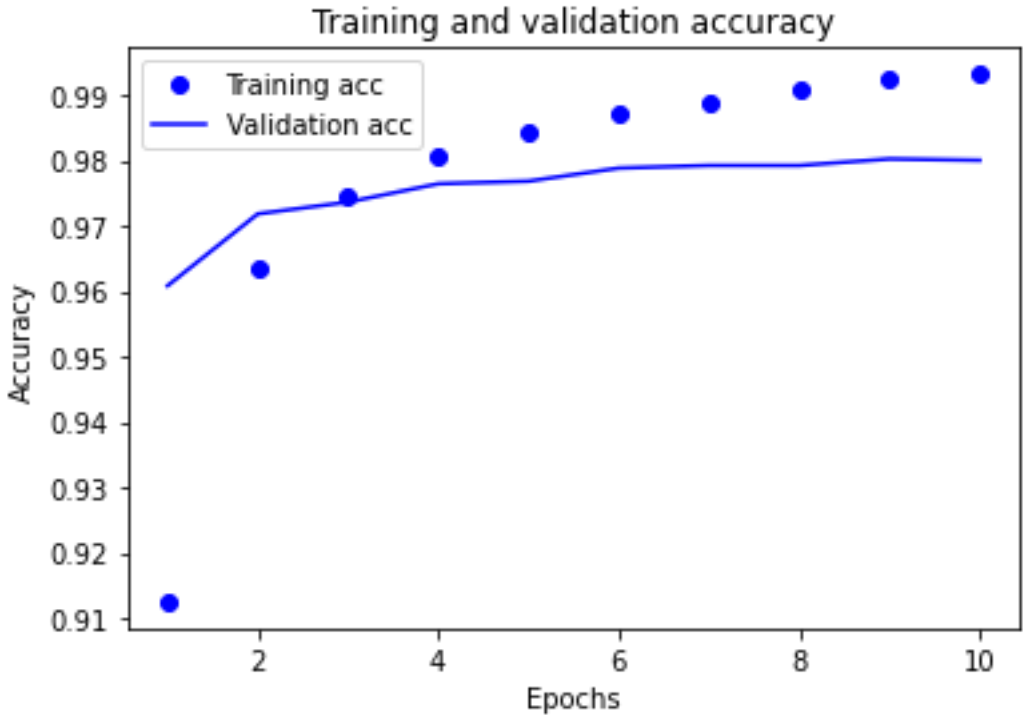
Going forward, we feel confident in reducing the number of epochs in future experiments to 10, cutting down model fitting time.

Experiment 7: Dimensionality reduction via PCA (90% of variance) over 128 units in the hidden layer

In experiment 7 we use the reduced dataset of 87 features via PCA over on 10 epochs on a neural network containing 128 units.

Our training model loss and accuracy is the following:





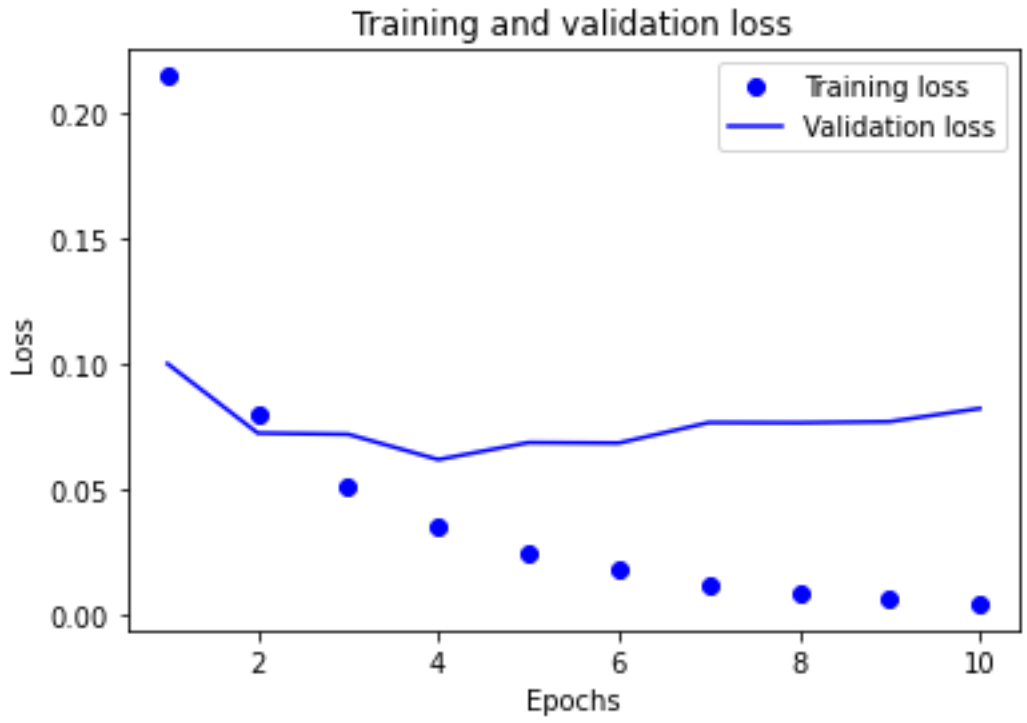
We see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The data reduction improves test accuracy over experiment 6 to 97.8% over the same number of units in the hidden layer.

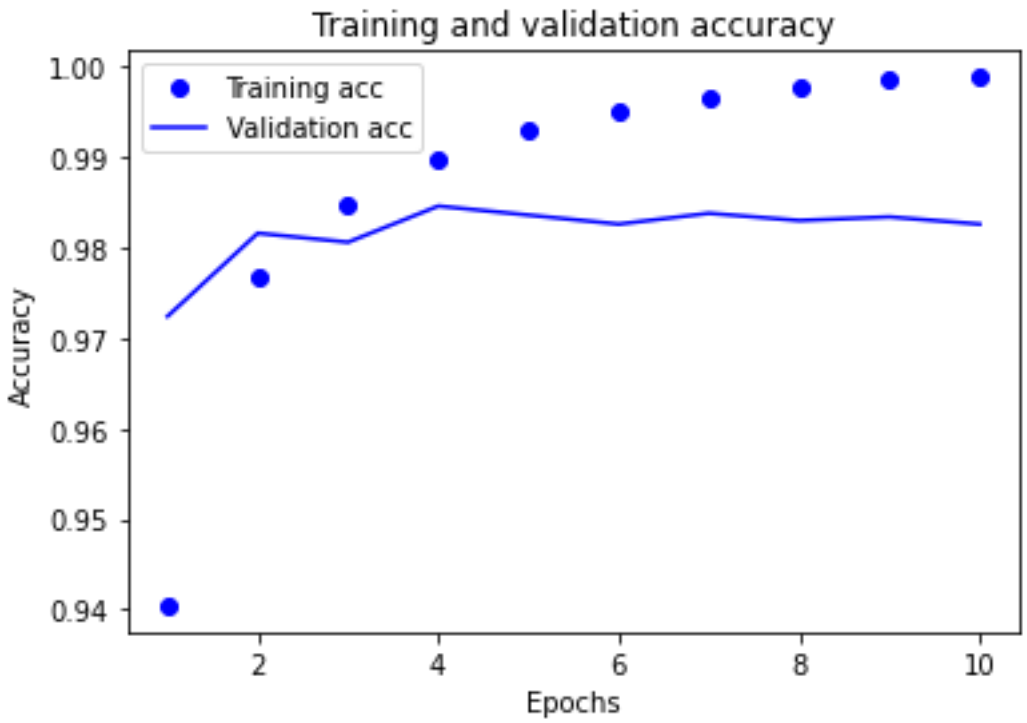
Going forward, we will continue with a reduced dataset to 90% as it is more effective in the model.

Experiment 8: Dimensionality reduction via PCA (90% of variance) over 512 units in the hidden layer

In experiment 8 we use the reduced dataset of 87 features via PCA over 10 epochs on a neural network containing 512 units.

Our training model loss and accuracy is the following:



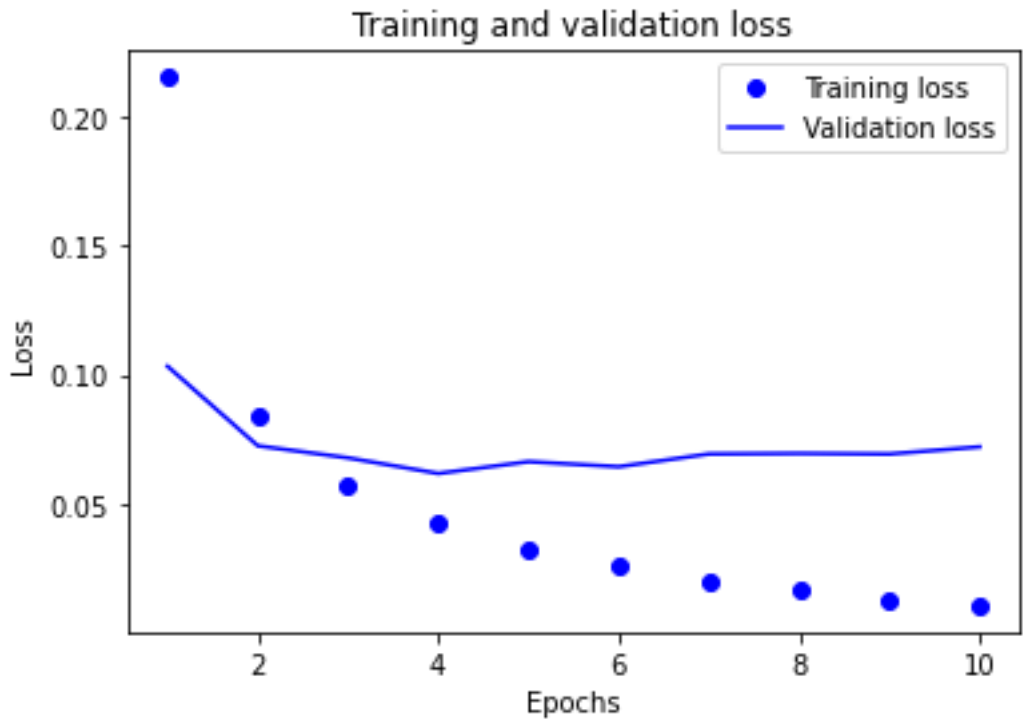


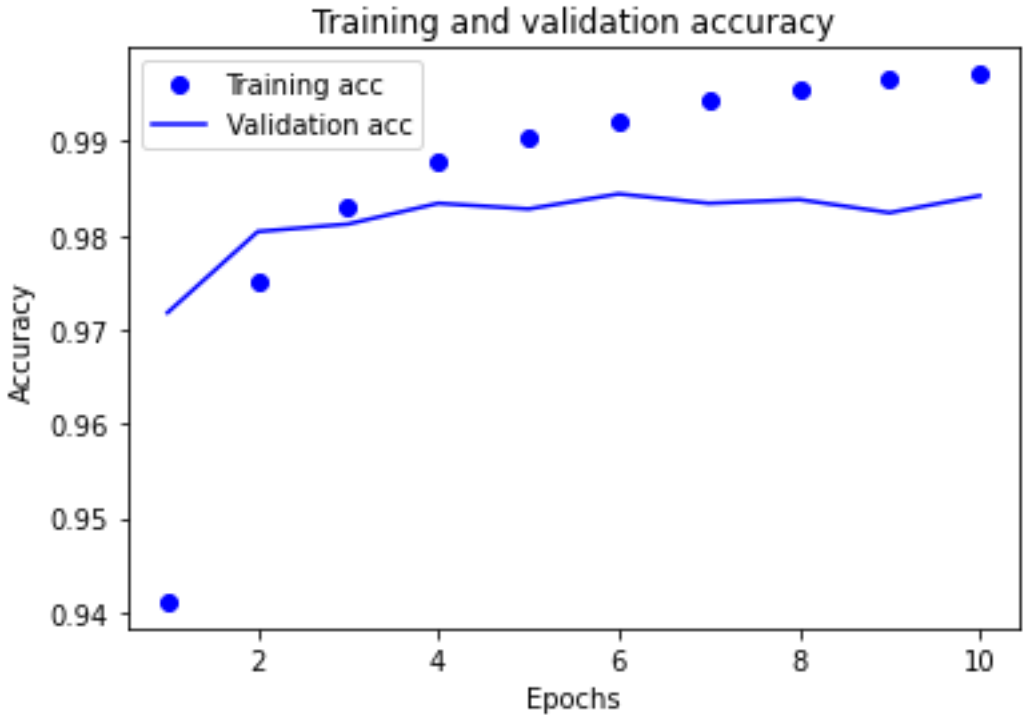
We see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The data reduction improves test accuracy over experiment 6 to 98.23% over the same number of units in the hidden layer.

Experiment 9: Dimensionality reduction via PCA (85% of variance) over 512 units in the hidden layer

In experiment 6 we used the reduced dataset even more to 59 features via PCA. The reduction represents 85% of the variance in the data. We run the data on 10 epochs on a neural network containing 512 units.

Our training model loss and accuracy is the following:





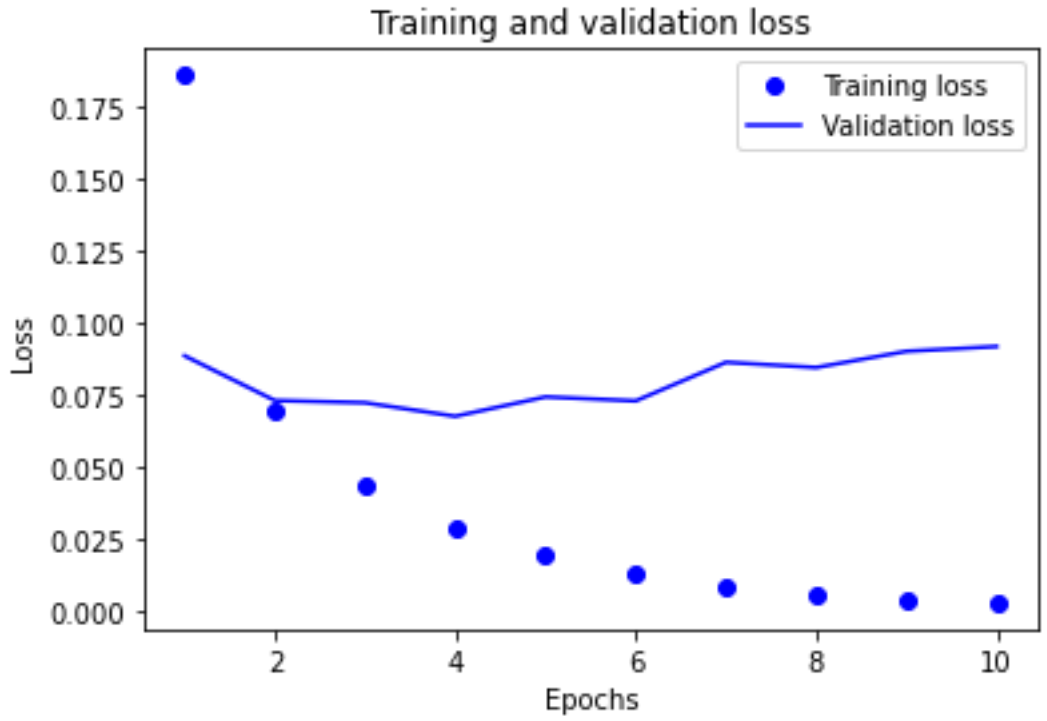
We see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The data reduction performs slightly worse than experiment 8 with a test accuracy of 98.13% over the same amount of units in the hidden layer.

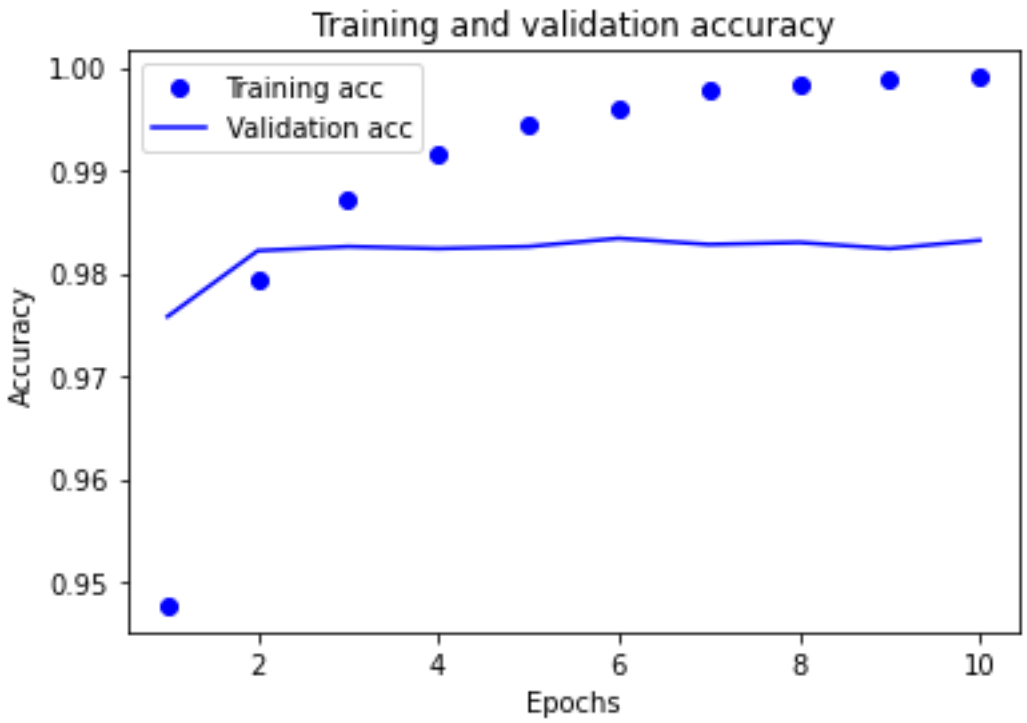
We start to see the limits on the amount of data reduction we can apply to the model.

Experiment 10: Dimensionality reduction via PCA (90% of variance) over 1024 units in the hidden layer

In experiment 10 we use the reduced dataset of 87 features via PCA over 10 epochs on a neural network containing 512 units.

Our training model loss and accuracy is the following:





We see the validation loss and accuracy diverge from the training loss and accuracy after a few epochs. The data reduction performs slightly worse than experiment 8 with a test accuracy of 98.07%.

At this point we start to see the limits on the number of units we can add in the hidden layer of the model.

The results of our experiments are summarized in the table below, with the best model highlighted.

Experiment	Hidden Layer Units	Compression/Dimensionality Reduction	Test Accuracy	Test Loss
1	1	None	0.3779	1.573
2	2	None	0.7061	0.9694
3	128	None	0.9745	0.0893
4	85	PCA (95% variance, 154 features)	0.9771	0.0888
5	85	Random Forests (70 features)	0.9308	0.2281
6	128	PCA (95% variance, 154 features)	0.9754	0.0869
7	128	PCA (90% variance, 87 features)	0.978	0.0734
8	512	PCA (90% variance, 87 features)	0.9823	0.0626
9	512	PCA (95% variance, 59 features)	0.9813	0.0661
10	1024	PCA (90% variance, 87 features)	0.9807	0.0681

Conclusions

This study ran experiments of different neural network models to process classify the handwritten numerical data of the MNIST image dataset. Using a combination of data dimensionality reduction via PCA and increasing the number of units in our single hidden layer we've consistently improved the test accuracy performance of the model but eventually found the limits on the amount of data reduction we can perform and the number units we can add to the neural network. The most optimal model we found was one which reduced the dataset to 87 features representing 90% of the variance on a neural network using 512 units in the single hidden layer. That being said, this model is just a marginal improvement over a model that uses a more reduced dataset of 59 features representing 85% of the variance on a neural network. The decision to use the model from experiments 8 or 9 can depend on the amount of memory or processing power that is available. The 0.1% difference between the models could be negligible but scaling to the number of times the model could be used could result in a significant difference of errors. If this could be used at a shipping company where reading zip codes is critical, I would encourage to use the best model available. If the model were used at a medical facility such as reading a doctor's orders for prescriptions, I would strive to find ways to further improve the model and aim for an even higher test accuracy, given doctors' typical penmanship and the difficulty reading it.

Resources

Palvanov, Akmaljon and Young Im Cho. 2018. "Comparisons of Deep Learning Algorithms for MNIST in Real-Time Environment." *International Journal of Fuzzy Logic and Intelligent Systems* 18, no. 2, pp. 126-134. http://doi.org/10.5391/IJFIS.2018.18.2.126