Handwritten Digits: Predictive Models Using Neural Network Multi-Layer Perceptrons

W. Andy Holst

School of Professional Studies, Northwestern University

MSDS 458: Artificial Intelligence and Deep Learning

Dr. Syamala Srinivasan

July 11, 2021

Abstract

This paper addresses how simple (single hidden layer) multi-layer perceptron neural networks can process digital images of handwritten numbers with high acuracy. It is important to fundamentally understand how a neural network of as few as one or two nodes functions in order to truly understand the more complex models that are in use today inside of our iPhones and computers transcribing our handwritten numbers into equivalent digits with exceptional accuracy. In our experiments we were able to achieve **97.70% Accuracy** with relatively simple neural network techniques.

By implementing predictive models using neural networks, industries will be able to completely automate the reading of all the numeric digits on documents with nearly superhuman accuracy. Thanks to the advent and proliferation of computer-based optical character recognition (OCR) systems, the creation of the MNIST dataset, and groundbreaking work in advanced multilayer neural network techniques, the opportunities to deploy predictive models using neural network multi-layer perceptrons is now a reality and benefitting millions of people every single day.

Handwritten Digits: Predictive Models Using Neural Network Multi-Layer Perceptrons

The banking industry has historically relied heavily on transferring money between two parties via handwritten checks. The processing of handwritten checks obviously requires very high accuracy with clients expecting absolute perfection from the bank to transfer the correct amount of money. So as a result the check reading process used to be time-consuming, human-centric, and very expensive for banks. With the advent and proliferation of computer-based optical character recognition (OCR) systems which first emerged in the 1990's, researchers such as Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner recognized the potential business value of automating the check reading process. In a joint research paper presented in 1998 in the Proceedings of the IEEE (Institute of Electrical and Electronics Engineers), the researchers recognized and built upon the recent advancements of another major industry entity, the US Census, ultimately leading to both the creation of a dataset and the groundbreaking creation of advanced multilayer neural network techniques that completely automated the reading of all the numeric digits on checks with superhuman accuracy.

It is with this business opportunity in mind -- automation of check processing -- that I now explore how multi-layer perception neural networks can process digital images of handwritten numbers. In this paper, I explore neural networks with only a single hidden layer, or shallow neural networks. Further research is intended to explore deeper layered neural networks, along with techniques pioneered by Lecun, Bottou, Bengio and Haffner, called convolutional neural networks and graph transformer networks (GTN). But it is important to fundamentally understand how the neural network of as few as one or two nodes works in order to truly understand the more complex models that are in use today inside of our iPhones and computers transcribing our handwritten numbers into equivalent digits with exceptional accuracy.

Literature Review

Firstly, I reviewed the seminal paper which has made conducting handwritten number recognition research a possibility for the masses (including myself): in 1998 Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner published a paper entitled “Gradient-Based Learning Applied to Document Recognition” which outlined how they both built a database for their research and then compared performance of a multitude of different models that can recognize handwritten digits. The results of their work in convolutional neural network character recognizers are combined with graph transformer networks (GTN) to provide accuracy on business and personal checks. These techniques were deployed for commercial use in the 1990’s and now read millions of checks per day (LeCun et al., 1998)

Secondly, I reviewed “The First Census Optical Character Recognition Systems Conference” which provides insight into how and why the dataset for this research was initially developed. In order to improve OCR of census records, the NIST distributed multiple datasets called “Special Databases” (SP1 and SP3 are specifically relevant) to participants in the “First Census OCR Systems Conference” as training and test datasets so that improvements in OCR could be studies. The use of optical character recognition (OCR) is now commonplace today, but in 1992 the OCR techniques in use were still relatively immature and achieved only 95% accurate on numeric digits as compared to human accuracy of 98.5% (Wilkinson et al., 1992).

Lastly, Yann Lecun’s website yann.lecun.com provides unique and personal insight into the development of the dataset used for this research, specifically the MNIST or the “Modified National Institute National Institute of Standards and Technology” database of handwritten digits, which was developed “for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting” (LeCun et al., n.d.).

Data

The dataset used for this research -- the MNIST database of handwritten digits -- was originally compiled by Yann Lecun, Corinna Cortes and Christopher Burges. It consists of a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized to 28 pixels by 28 pixels square and have been centered in a fixed-size image. The data preparation, exploratory data analysis and visualization steps undertaken to prepare the dataset for subsequent research, analysis, and modeling were as follows:

- A Google Colaboratory Notebook (Python 3.7.10) was created, with GPU enabled.
- Python Packages imported as necessary for pre-processing and modeling, including: Numpy (1.19.5), Pandas (2.5.0), Tensorflow (2.5.0), Sci-kit Learn (0.22.2) (*for Confusion Matrix, PCA, TSNE, and RF*), Matplolib (3.2.2). Seaborn (0.11.1), and Time.
- Importing the MNIST database of handwritten digits (60,000 training images and 10,000 test images), which is available using the Tensorflow API: `tensorflow.keras.datasets.mnist.load_data()`
- "Flattening" the training dataset Numpy arrays `(60000, 28, 28)` to a two-dimensional shape of `(60000, 784)`. "Flattening" the test dataset as well, to `(10000, 784)`.
- Splitting the training dataset into 55,000 training images and a 5,000 validation images. So the shapes of all six datasets ready for research and modeling are as follows:
    - `x.train` shape `(55000, 784)`: Used as input variables for training the models
    - `y.train` shape `(55000, 10)`: Used for output labels for training the models
    - `x.val` shape `(5000, 784)`: Used as input validation variables during training
    - `y.val` shape `(5000, 10)`: Used as output validation labels during training
    - `x.test` shape `(10000, 784)` : Used as test input variables for prediction

- `y.test` shape `(10000, 10)` : Used as test output labels to measure accuracy

Methods: Research Design and Modeling

Over the course of the first three (3) weeks of the Summer Term 2021, I conducted ten (10) experiments on the MNIST dataset, which is a multiclass classification problem. The neural network models for these experiments were intentionally restricted in complexity to a single hidden-layer perceptron because it is important to fundamentally understand how the neural network of as few as one or two nodes works in order to truly understand the more complex models in use today. The single hidden-layer models were constructed in a `Sequential` layer-fashion using Keras (Python library was built by and described eloquently by Francois Chollet in “Deep Learning with Python”) (Chollet, 2018). The input layer consists of `784` input nodes. The single hidden layer is a fully connected (`Dense`) layer with `relu` activation function, and various nodes (ranging from `1` to `4,096`) are explored throughout the Experiments. The last layer uses a `softmax` activation with `10` nodes, outputting a probability distribution over the 10 different output classes. The loss function used in these experiments was `sparse_categorical_crossentropy`, and the optimization algorithm used was `RMSProp`. A validation set was created by setting apart `5,000` samples from the original training data, to minimize overfitting.

Because this problem involves identifying handwritten numerals, and the evolution of this type of data is expected to be almost never change over time (i.e. we will likely be using Arabic numerals for quite some time), it is expected that this model will not require any periodic updating once it has been placed into production. One caveat to this assumption is that the model will be implemented in the United States. Different countries and cultures likely write their numbers differently than Americans do, and this dataset is based entirely on US handwriting.

Experiment 1

Our very first dense neural network consisted of `784` input nodes, a hidden layer with `1` node and `10` output nodes (corresponding to the 10 digits). This model consists of a total of `805` parameters (including weights and bias). After training the model using `30` Epochs, I group the 60,000 activation values of the hidden node for the (original) set of training images by the 10 predicted classes and visualize the activation variables using a boxplot. Other results and visualizations were explored: using the `confusion_matrix`; creating visual lineplots of accuracy and loss over time (ie. each epoch) for the training and validation datasets; and by visually inspecting examples of digits which were misclassified by the model.

Experiment 2

Experiment 2 dense neural network consisted of `784` input nodes, a hidden layer with `2` nodes (increased from `1` in Experiment 1) and `10` output nodes (corresponding to the 10 digits). This model consists of `1,600` parameters (including weights and bias). After training the model using `30` Epochs, I group the 60,000 activation values of the hidden node for the (original) set of training images by the 10 predicted classes and visualize the activation variables (2 different nodes this time) using a Scatterplot (see Figure 1 in Results below). Other results and visualizations were explored (same as Experiment 1): using the `confusion_matrix`; creating visual lineplots of accuracy and loss over time (ie. each epoch) for the training and validation datasets.

Experiment 3

Experiment 3 dense neural network models (a total of eleven built) consisted of `784` input nodes, a hidden layer varying from `1` (ie. 2^0) node to `1,024` (ie. 2^{10}) nodes and `10` output nodes (corresponding to the 10 digits). These models consisted of a total number of parameters ranging from `805` to `814,090` (including weights and bias). After training each model using `30`

Epochs, I determine the “best” model to consist of `128` nodes. This “best” model consists of `101,770` parameters (including weights and bias).I visualize each model accuracy compared to each other and each model’s “build time”. Other results and visualizations were explored (same as Experiment 1): using the `confusion_matrix`; creating visual lineplots of accuracy and loss over time (ie. each epoch) for the “best” model’s training and validation datasets.

Experiment 4

Experiment 4 used PCA decomposition to reduce the number of dimensions of the training set of 28x28 dimensional MNIST images from `784` to `154` (with 95% of training images variance lying along these components). The 'best' model from Experiment 3 was reduced to `154` input nodes and trained on the new lower dimensional data. This model consists of `21,130` parameters (including weights and bias). To speed model build time, I introduced `EarlyStopping` to monitor validation accuracy, with a setting of `patience=5`. Training the model converged in `12` Epochs. Then I used t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the (activation) features from `128` (= num of hidden nodes) to `2` (see Figure 2 in Results below).

Experiment 5

Experiment 5 used a Random Forest classifier to get the relative importance of the `784` features (pixels) of the 28x28 dimensional images in the training set of MNIST images and selects the top `70` features (pixels). I train the 'best' dense neural network (from Experiment 3) using these 70 features and compare its performance to the dense neural network models with all others.

Experiment 5 dense neural network consisted of `70` input nodes, a hidden layer with `128` nodes and `10` output nodes (corresponding to the 10 digits). This model consists of `10,378` parameters (including weights and bias). The model is trained using `30` Epochs. Other

results and visualizations were explored (same as Experiment 1): using the `confusion_matrix`; creating visual lineplots of accuracy and loss over time (ie. each epoch) for the training and validation datasets.

Experiment 6

Experiment 6 utilizes the same models from Experiment 3, however revises the optimization algorithm from `RMSProp` to `Adam` for this experiment only. I visualize each model's accuracy compared to each other and each model's "build time", as well as compare with Experiment 3 accuracies and "build time".

Experiment 7

Experiment 7 utilizes the "best" model from Experiment 3 (`128` nodes), however explores how different activation functions may affect performance (if at all). The following alternative activation functions (besides `relu`) were explored: `Sigmoid`, `TanH`, `leaky_relu` (PReLu / Parametric Rectified Linear Unit), `ELU` and `softplus`. I visualized each model's accuracy compared to each other and each model's "build time", as well as compare with Experiment 3 (`relu`) accuracies and "build time".

Experiment 8

Experiment 8 is a deeper dive into Early Stopping to capitalize on model build time, after noticing that larger networks potentially converge faster (i.e. achieve a much higher accuracy in earlier Epochs). For Experiment 8, I tested my theory on 2^n Nodes, with n ranging from 0 to 12. This produced models with the following range of Nodes = `[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]`. The difference between this Experiment 8 and previous Experiment 3 is that I chose to enable EarlyStopping to monitor validation accuracy, with a setting of `patience=1` with a goal of reducing the total number of epochs.

Experiment 9

Experiment 9 explores whether 100 models exhibit a normal / Gaussian Distribution with respect to `Test Accuracy`. Experiment 9 utilizes the Experiment 3 model with `784` input nodes, a hidden layer with `128` nodes and `10` output nodes (corresponding to the 10 digits). This model consists of `101,770` parameters (including weights and bias). One difference introduced was utilizing early stopping (`patience=2`)

Experiment 10

Experiment 10 revisits Experiment 4 (PCA decomposition), but with additional nodes. Experiment 10 utilizes a model with `154` input nodes, a hidden layer with `1024` nodes and `10` output nodes (corresponding to the 10 digits). This model consists of `10,378` parameters (including weights and bias). It utilizes early stopping (`patience=2`)

Results

Experiment 1

In Experiment 1, it was determined that the **Test Accuracy = 39.67%**. This is better than a “random” guessing (which has an expected accuracy of 10%). But this is certainly not good enough to place into production. The goal of this experiment was to verify that the overlap between the range of activation values in the "boxes" were minimal. This, in fact, proved to be case with Experiment 1 model producing the following activation range values:

- “7” having an activation value of `[0.0, 0.41]`
- “4” having an activation value of `[0.41, 1.12]`
- “1” having an activation value of `[1.13, 3.2]`
- “8” having an activation value of `[3.2, 4.39]`
- “5” having an activation value of `[4.39, 4.7]`
- “3” having an activation value of `[4.7, 7.08]`
- “0” having an activation value of `[7.08, 9.23]`
- “2” having an activation value of `[9.23, 14.27]`
- “6” having an activation value of `[14.27, 95.75]`

The only surprising result of this experiment was that the digit “9” did not have any activation values. Thus the model was not actually able to predict the number “9”. This was also confirmed by the Confusion Matrix, which was blank in the “9” Prediction Column.

Experiment 2

In Experiment 2, it was determined that the **Test Accuracy = 67.02%**. This is much better than a “random” guessing and also better than the Experiment 1 results. But this result is still certainly not good enough to place into production. The goal of this experiment was to verify that the overlap between the range of activation values in a scatterplot were minimal. This, in fact, proved to be the case with Figure 1 below visually depicting a very nice separation of digits, based upon the Activation Values present in the Experiment 2 model.

Figure 1 - Visual Scatterplot of Activation Values for Two Nodes

Experiment 3

It was expected and found that there is a varying time to build each model. As model complexity (number of parameters) increases exponentially in this Experiment, it was expected that a similar exponential growth would occur in the time required to build each model (`Total Epochs = 30`). Surprisingly, no correlation was found between "Build Time" and model complexity. It is unclear why this result occurred, but utilizing GPU Hardware Acceleration in Google Colaboratory may contribute. After review, a model with `128` nodes, with a **Test Accuracy = 97.75%**, built in 87.3 seconds, was selected as the "best" model:

```
Test set accuracy for 1 Node:    36.05% in 85.6 secs
Test set accuracy for 2 Node:    69.14% in 82.3 secs
Test set accuracy for 4 Node:    87.50% in 82.2 secs
Test set accuracy for 8 Node:    91.75% in 82.6 secs
Test set accuracy for 16 Node:   95.23% in 142.2 secs
Test set accuracy for 32 Node:   96.65% in 83.0 secs
Test set accuracy for 64 Node:   97.28% in 142.2 secs
Test set accuracy for 128 Node:  97.75% in 87.3 secs "Best"
Test set accuracy for 256 Node:  97.99% in 142.2 secs
Test set accuracy for 512 Node:  98.01% in 142.2 secs
Test set accuracy for 1024 Node: 98.12% in 87.6 secs
```

Experiment 4

This model consists of `21,130` parameters (including weights and bias), which is an ~80% reduction in parameter count (as compared with Experiment 3). Reducing model complexity and introducing the early stopping functionality led to an improvement of model "build time" down from 87.3 seconds previously down to 44.3 seconds, a nearly 50% reduction. With early stopping the model converged in 12 Epochs, and had **Test Accuracy = 97.56%** which is on par with Experiment 3 "best" model. Then I used t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the (activation) features from `128` to `2` (see Figure 2 in Results below). This visualization powerfully shows how each of the Activation Values are contributing to the model prediction results.

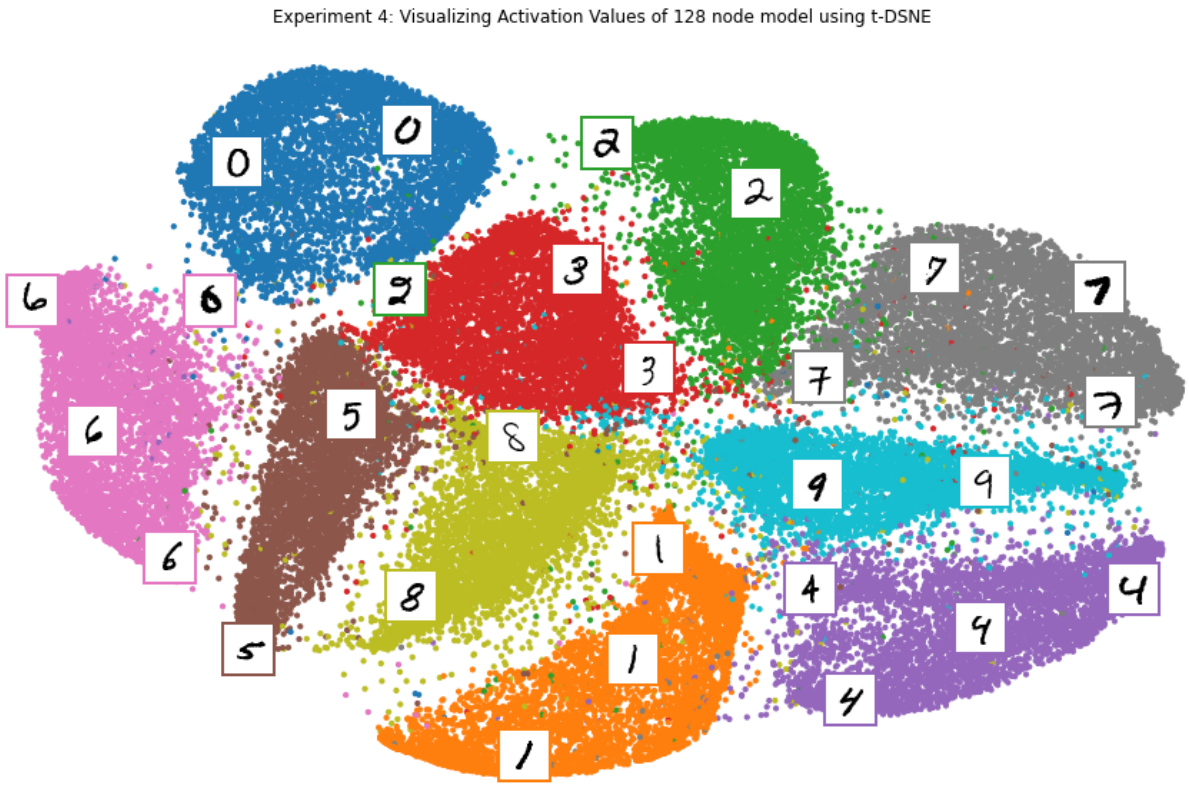

Figure 2 - Visualizing Activation Values of 128 node model using t-DSNE

Experiment 5

This model consists of `10,378` parameters (including weights and bias), which is nearly half of Experiment 4. Reducing model complexity even further and introducing feature reduction (from `784` pixels to `70` pixels) did not lead to any improvement of model “build time”, as it remained steady at 40.49 seconds. With early stopping the model converged in 11 Epochs, but had a **Test Accuracy = 93.03%** which is a reduction of ~4.75% from the Experiment 3 “best” model.

Experiment 6

Experiment 6 explored changing the optimization algorithm from `RMSProp` to `Adam` for this experiment only. For the “best” model using `128` nodes there was no appreciable difference

in Execution Time for `Adam` (82.12 secs) versus `RMSProp` (87.33 secs). So, I left the Optimizer as `RMSProp` for all future experiments.This model had **Test Accuracy = 97.49%.**

Experiment 7

Experiment 7 utilized the "best" model from Experiment 3 (`128` nodes) and explored how changing activation functions may affect performance (if at all). The following activation functions (besides `relu`) are explored: `Sigmoid`, `TanH`, `leaky_relu`, `ELU` and `softplus`:

```
Test set accuracy for relu:          97.78% "Best"
Test set accuracy for sigmoid:       97.80%
Test set accuracy for tanh:          97.59%
Test set accuracy for leaky_relu:    97.54%
Test set accuracy for elu:           97.57%
Test set accuracy for softplus:      97.41%
```

There was no appreciable difference between the activation functions, so I left the Activation as `relu` for all future experiments. This model had **Test Accuracy = 97.78%.**

Experiment 8

As demonstrated in Experiment 8, Early Stopping does appear to successfully capitalize on the hypothesis and observation that larger networks (more nodes) potentially converge faster (i.e. achieve a much higher accuracy in earlier Epochs). Choosing to enable EarlyStopping by monitoring validation accuracy, with a setting of `patience=1` both reduces the number of epochs performed during each model build, and likely also avoids overfitting (not explored in depth in this paper). As Figure 3 below shows, even as model complexity (nodes) increases, there is a decrease in model "build time" resulting from this faster convergence on the Loss Function. The `128` node model in this Experiment had a **Test Accuracy = 97.53%**.

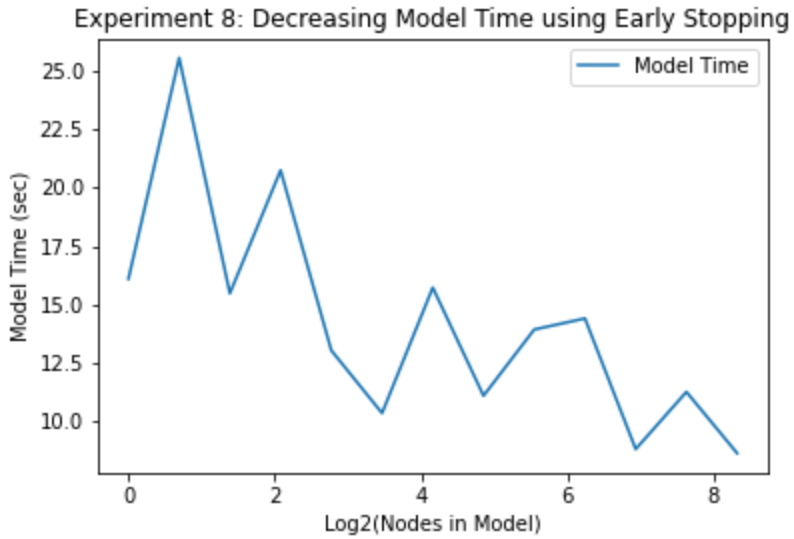

Figure 3 - Decreasing Model Time using Early Stopping

Experiment 9

By repeatedly testing Experiment 3 results (with added early stopping), the consistency of the model is considered high, with a **Sample Mean Test Accuracy = 97.70%** with a **Sample Standard Deviation = 0.14%**. Visually the histograms produced during this Experiment appeared to be normally distributed, with a slight skew to the left.

The model “build time” exhibited more variation, likely due to the early stopping functionality, with a **Sample Mean Model Build Time = 23.0 seconds** with **Sample Standard Deviation = 5.3 seconds.**

Experiment 10

By repeatedly testing Experiment 4 results (with PCA decomposition), the consistency of the model is considered high, with a **Sample Mean Test Accuracy = 97.74%** with a **Sample Standard Deviation = 0.17%**. Visually the histograms produced during this Experiment appeared to be normally distributed, with a slight skew to the left.

The model “build time” exhibited more variation, likely due to the early stopping functionality, with a **Sample Mean Model Build Time = 11.2 seconds** with **Sample Standard Deviation = 3.1 seconds.**

Analysis and Interpretation

The following Table produces final results for all ten (10) Experiments:

Experiment	Nodes	Epochs	Build Time (sec)	Test Accuracy (%)	Test Loss
1	1	30	*Not Tested*	39.67	1.550
2	2	30	*Not Tested*	67.02	0.993
3 “Best”	128	30	87.3	97.75	0.161
4 “PCA”	128	12	44.3	97.56	0.976
5	128	11	40.5	93.03	0.280
6 “RMS”	128	30	82.1	97.49	0.169
7 “relu”	128	30	142.2	97.78	0.162
8 “Best”	128	4	11.1	97.53	0.077
9	128	*Varies*	Mean: 23 STD: 5.3	Mean: 97.70 STD: 0.14	Mean: 0.082 STD: 0.006
10 “PCA”	**1024**	***Varies***	**Mean: 11.2 STD: 3.1**	**Mean: 97.74 STD: 0.17**	**Mean: 97.70 STD: 0.14**

Reducing model complexity and introducing the early stopping functionality led to an improvement of model “build time”. With early stopping, models converged in fewer Epochs, with no impact to the model accuracy. Early stopping appears to successfully capitalize on the hypothesis that larger networks (more nodes) converge faster (i.e. achieve a much higher accuracy in earlier Epochs). Choosing to enable early stopping by monitoring validation accuracy, both reduces the number of epochs performed during each model build, and likely

also avoids overfitting (not explored in depth in this paper). In fact, even as model complexity (nodes) increases, there is actually an unexpected decrease in model “build time” resulting from this faster convergence on the Loss Function.

There was no appreciable difference in Execution Time for different Optimizers (`RMSProp` was initially implemented, `Adam` was not). So, models utilized `RMSProp` for all experiments. There also was no appreciable difference in performance by changing activation functions, so the use of the `relu` activation function was kept unchanged for all experiments.

Lastly, by repeatedly testing results the consistency of the models are considered high, Visually the histograms produced during this study appeared to be normally distributed.

For all of these reasons, the **model developed in Experiment 10** provides the most powerful model, and performed exceptionally well at **97.70% Test Accuracy**, excellent performance even for a single hidden layer MLP. The chosen model has a very wide net of `1024` nodes which means that it learns very quickly and it also leverages the benefit of early stopping, thus resulting in the fastest model “build times”. Lastly, it utilizes the benefits of PCA decomposition, thus reducing the overall size of the model from 784 pixels down to 154 features. It accomplishes all of this without compromise on Test Accuracy, providing nearly the highest accuracy model tested. For all of these reasons, I would recommend putting the the model from Experiment 10 into production in order to predict handwritten digits with a 97.7% accuracy.

Conclusions

This paper addressed how simple (single hidden layer) multi-layer perceptron neural networks can process digital images of handwritten numbers with high accuracy. It is important to fundamentally understand how the neural network of as few as one or two nodes functions in order to truly understand the more complex models that are in use today inside of our iPhones

and computers transcribing our handwritten numbers into equivalent digits with exceptional accuracy. In our experiments we were able to achieve **97.70% Test Accuracy** with relatively simple neural network techniques, getting within 1% of the abilities of humans (according to the US Census at least, see Wilkinson, et. al., 1992).

By building and deploying a predictive model using neural networks, industries such as the banking industry, which we introduced at the beginning of this paper, are now able to completely automate the reading of all the numeric digits on documents with nearly superhuman accuracy. Manual methods, such as the check reading process, were once a time-consuming, human-centric, and very expensive process for banks. With the advent and proliferation of computer-based optical character recognition (OCR) systems, the creation of the MNIST dataset, and the groundbreaking creation of advanced multilayer neural network techniques, the opportunities for advancement to deploy predictive models using neural network multi-layer perceptrons is now a reality and benefitting millions of people every single day.

References

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications Co.

LeCun, Y., Burges, C., & Cortes, C. (n.d.). *The MNIST Database of Handwritten Digits*. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Christopher Burges. Retrieved from http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, November 11). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, vol. 86, no. 11. doi: 10.1109/5.726791. Retrieved from: https://ieeexplore.ieee.org/document/726791.

Wilkinson, R. A., Geist, J., Janet, S., Grother, P. J., Burges, C. J. C., Creecy, R., Hammond, B., Hull, J. J., Larsen, N. W., Vogl, T. P., & Wilson, C. L. (1992). *The First Census Optical Character Recognition Systems Conference, NIST Interagency/Internal Report (NISTIR)*. National Institute of Standards and Technology. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=905664.