MNIST HANDWRITTEN DIGIT RECOGNITION USING SINGLE HIDDEN LAYER NEURAL NETWORKS

Betzalel Moskowitz
MSDS458-DL: Artificial Intelligence and Deep Learning
July 10, 2023

Abstract

Handwritten digit recognition has been an ongoing problem that remained unsolved for decades until the release of the MNIST dataset. The MNIST dataset provided quality data with sufficient volume to train predictive models with near human-level performance. This paper examines previous attempts at solving the problem, the efficacy of training single layer multi-layer perceptron (MLP) models on MNIST, and the effect that the quantity of hidden units, optimizers, dropout layers, and dimensionality reduction techniques had on performance. After many training experiments, a 98.48% test accuracy was achieved, beating out a previous benchmark set out using a similar MLP architecture.

Introduction

Since the inception of the United States Postal Service (USPS) in 1775, its excellence of service has been credited to its innovative operations that have allowed it to deliver mail quickly and efficiently. Despite its success, its task has never been easy – America is massive country and the USPS is tasked with delivering an average of 472.1 million postal items daily (SRI International). While the delivery of mail from one place to another would in theory take a fixed amount of time, the actual delivery times would fluctuate based on how quickly the postal items could be sorted. Sorting is necessary to make delivery of mail efficient by grouping together pieces of mail that could be delivered to areas close in proximity to one another. This process was conducted by manually reading addresses. However, as the volume of mail grew, it became evident that automating sorting would be the only way to maintain an efficient postal service (SRI International).

In 1957, the USPS began using a semiautomatic sorting machine, which was followed in the 1960s by another sorting machine powered by an optical character recognition (OCR) software (SRI International). This OCR software worked by running a scanner across the envelope or parcel to locate the bottom row and measuring the height of the characters that appeared on that line. The OCR software then measured the length of the characters and attempted to identify the character one by one using hardcoded defining features. For instance, a ‘6’ may be identified by looking for a circular shape and a curve similar to the direction a hand on a clock may travel from nine to twelve (Hanson 1970).

The OCR software was programmed with rules for every digit or letter that enables the scanner to identify the character based on the defining features (Hanson 1970). While this performed decently for mono-spaced printed text, it was much less effective for handwriting where the characters may be written in cursive or are hardly legible. Over the next several decades, USPS continued to adapt new systems of recognizing addresses based on OCR, but these systems were only marginally better – sorting mail was still slow as errors were common (Saldarini 1999). However, this all changed in 1998 with the release of the Modified National Institute of Science and Technology’s (NIST) MNIST dataset.

The MNIST is a dataset with 60,000 training images plus 10,000 test images of handwritten digits. (LeCun et al. 2010). At the time of release, MNIST was easily the largest labeled dataset of handwritten digits to date, and it was preprocessed using normalization and centering to

produce a clean dataset ripe for machine learning experiments. Due to advances in machine learning algorithms and access to this new data, researchers were able to produce models capable of quickly and accurately classifying digits, enabling zip codes on addresses to be read more accurately and efficiently. Thanks to better methods of handwritten digit recognition, 400 million pieces of mail were automatically routed using models likely trained on MNIST during the Christmas season of 1998. In its first year alone, the handwritten addresses recognition software saved the USPS $90 million (Saldarini 1999).

While handwritten digit technology has benefitted the USPS immensely, it has many other use cases. In particular, the technology was beneficial in helping the Internal Revenue Service (IRS) and Social Security Administration (SSA) to automate tax forms and W2 forms respectively (Saldarini 1999).

For all of these business cases, accuracy and speed are integral for effective and efficient operations. In many models, there is often a trade-off between accuracy and speed– often times the most accurate models have complex architectures that take up more space and are slower at making predictions. Using MNIST, a dataset of representative and labeled samples, the paper will investigate producing accurate models for handwritten digit classification using smaller architectures of neural networks (multi-layer perceptrons). In the process, the paper will uncover key insights as to how neural networks learn to produce representations to assist in making accurate classifications on the MNIST handwritten digits. The paper will also discuss the efficacy of additional strategies to improve model performance.

Literature Review

Due to its large sample size, completeness, structured nature, and minimal preprocessing required to start modeling, the MNIST dataset has been researched extensively throughout literature. At the time of writing, 1,597 teams have submitted their models to compete in Kaggle's "Digit Recognizer" competition, with the best submission achieving near 100% accuracy on the test set (Kaggle). Papers with Code, a website that tracks performance from different research papers on machine learning tasks, lists over ninety papers written on MNIST's classification task, with many achieving test accuracies over 99% (Papers with Code).

Virtually all of the highest performing models are variations of deep neural networks (DNNs) with multiple hidden layers or convolutional neural networks (CNNs). These ability to easily extract features from image data have allowed these architectures to provide breakthrough performance, helping to gain what had been an elusive extra percentage point of accuracy (Yahya et al. 2021, 1).

In addition, many of the top performing models utilize data augmentation techniques to generalize better to data outside of the training set and avoid overfitting. For example, Yahya et al. augmented the MNIST dataset via rotation and shifting transformations (2021, 11). The augmentation increased the size of the training dataset from 60,000 to 420,000 and 10,000 to 280,000 images in the test set, allowing the researchers to gain almost an extra half percentage point of accuracy on their CNN with an impressive 99.98% test accuracy (Yahya et al. 2021, 6-14).

While similar methods produced near-human-level performance, such models are very large with millions of parameters, rendering the model slower and harder to fit on devices with limited memory. LeCun et al. (2010) experimented with many different types of classical machine learning and deep learning methods to tackle the MNIST classification task and has achieved strong performance using variations of the k-nearest-neighbor (KNN) and support vector machine (SVM) algorithms, as well as non-linear classifiers (LeCun et al. 2010). LeCun also reached error rates under 1% using various DNN and CNN architectures. Most of these models utilized advanced preprocessing or distortion data augmentations to improve performance.

The best observed multi-layer perceptron (MLP) model in literature with just one hidden layer that doesn't utilize data augmentation or distortions and has no advanced data preprocessing methods on the MNIST dataset was an MLP with 800 hidden units and utilized cross-entropy loss. This model was developed by Simard et al. (2003) and produced an accuracy score of 98.4%. In this paper, we will attempt to produce another multi-layer perceptron model that can defeat this benchmark.

Methods

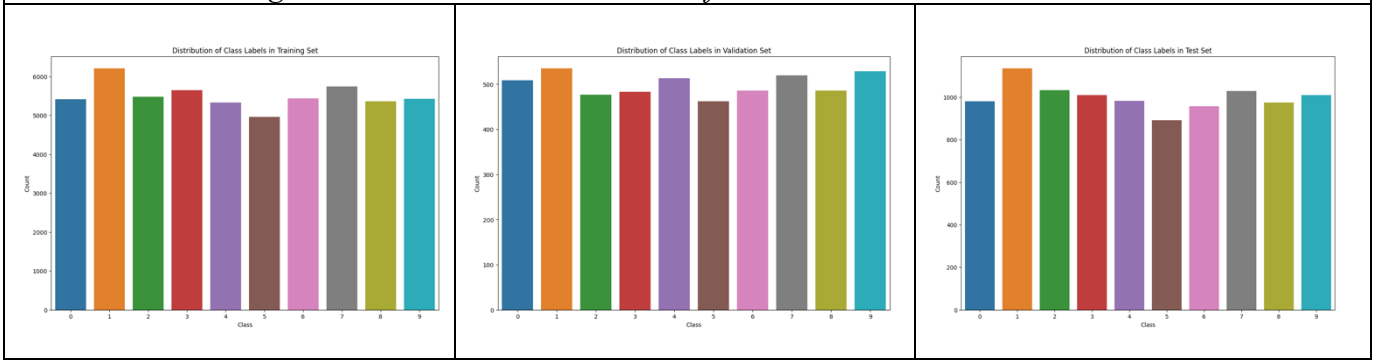
Data Exploration

We will be using the MNIST dataset for the classification task. The MNIST dataset consists of 60,000 labeled training images and 10,000 labeled test images. MNIST is a popular dataset for beginners to machine learning because it has no missing labels and has already been preprocessed. Each of the images in the dataset were size-normalized to fit into a 20x20 pixel box with their aspect ratios preserved. The images are visible in grey levels due to the anti-aliasing technique used by the normalization algorithms. The digits were then centered in a 28x28 image, having computed the center of mass of the pixels and translating the image to ensure that the center of mass is at the center of the new 28x28 image (LeCun et al. 2010).

The MNIST data itself was curated from two datasets – 30,000 of its training images and 5,000 of its test images are taken from NIST's Special Database 3 (SD-3), a dataset of handwritten digits by high schoolers. The other 30,000 of its training images and 5,000 of its test images are taken from Special Database 1 (SD-1), a database of handwritten digits from Census Bureau employees (LeCun et al. 2010). The combination of handwritten digits from high school children and Census Bureau employees covers a diverse set of writing styles and levels of legibility, providing a representative sample for our model.

The 60,000 image training set was split into 55,000 images for training, and 5,000 images for validation using the same random sampling state to allow for each of the models to be trained and validated with the same data. After this split, the number of class labels in each dataset can be seen below in *Table 1 – Distribution of Class Labels*. It can be observed that the distribution of class labels in each dataset is close to uniform, making for a balanced dataset. An exact frequency count of each class label can be found in the Appendix, *Figure A1*.

Figure 1. Class Label Distribution of Train, Validation, and Test Sets



Data Preprocessing

When downloading and splitting the datasets, the inputs (images) were flattened by reshaping each image of size (28, 28) 2D arrays to shape (784,) vectors (1D arrays), allowing for each of the 784 pixels in the image to be fed into the input layer of the neural networks. Each of the pixel values in the image were represented by an integer value from 0 (white) to 255 (black), but these were scaled by dividing each value in each image by 255.

The original labels for the MNIST dataset are integer digits from 0-9. Because this is a multi-classification problem, the labels were one-hot encoded into an array of length 10. For instance, the label 5 would be represented as [0 0 0 0 0 1 0 0 0 0], where the 1 is located at the index of the label it represents (5), with the rest filled with zeroes.

Implementation and Programming

The experiments were carried out via Google Colab notebooks, utilizing Google's T4 GPUs on a hosted Python runtime. All modeling was conducted using scikit-learn and Keras within Tensorflow version 2.12. All visualizations were created using Matplotlib or Seaborn. Additional Python libraries used included the Pandas and NumPy libraries for data manipulation.

The models built to classify handwritten digits were multi-layer perceptrons, a type of neural network with at least three layers. In order to develop an intuition of how neural networks learn representations of the features within the data, only one hidden layer was used in the neural network. The goal was to experiment with different numbers of neurons (hidden units) to gain insight into how the number of neurons affects the fit of the model to the test data and how the number of neurons affects the quality of the representations learned by the model. Under the described methods, training experiments were conducted with the goal of achieving a higher accuracy than the 98.4% test accuracy reached by Simard et. al using a single hidden layer MLP (2003).

Each model was trained using the ReLU activation function, with a learning rate of 0.001, and used the cross-entropy loss function.

Each experiment was allowed to train on up to 200 epochs, but with an early stopping patience threshold of ten. The early stopping occurred if the validation accuracy did not improve after ten

epochs and preserving the best model and avoiding additional overfitting. The experiments conducted focused primarily on testing different numbers of hidden units in the hidden layer, starting with 1 hidden unit and doubling the number of hidden units every experiment until 2048 hidden units.

Two experiments were run to test the efficacy of using classical machine learning techniques for reducing dimensions and extracting features from the input data. These two techniques were PCA and random forest.

In the PCA experiment, an appropriate number of principal components were taken to explain 95% of the variation in the data, resulting in 154 principal components. These components were used to transform the input data and reduce the number of input dimensions from 784 to 154.

Additionally, a random forest classifier was trained producing an accuracy of 94%. The 70 most important pixel values according to the model's feature importance ranking were used to reduce the dimensionality of the image data from 784 to 70.

The transformed data from both the PCA and Random Forest models were used as input data when used to train a model with the same architecture as the best model trained on the original input of size 784.

Figure 2 provides a list of all experiments. As more experiments were run and higher performances were reached, additional tweaks were run to see whether changing the optimizer from RMSProp to Adam and adding a 50% dropout layer would help to achieve superior performance. The following is a list of experiments that were run to try to achieve an accuracy greater than 98.4%:

Figure 2: A Table of Experiments Run

MLP- 1 Hidden Layer Node
MLP - 2 Hidden Layer Nodes
MLP - 4 Hidden Layer Nodes
MLP- 8 Hidden Layer Nodes
MLP- 16 Hidden Layer Nodes
MLP - 32 Hidden Layer Nodes
MLP 64 Hidden Layer Nodes
MLP - 128 Hidden Layer Nodes
MLP - 256 Hidden Layer Nodes
MLP- 256 Hidden Layer Nodes - Adam Optimizer
MLP - 256 Hidden Layer Nodes with 50% Dropout Layer
MLP - 256 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer
MLP - 512 Hidden Layer Nodes
MLP - 512 Hidden Layer Nodes Adam Optimizer

MLP - 512 Hidden Layer Nodes with 50% Dropout Layer
MLP - 512 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer
MLP - 1024 Hidden Layer Nodes
MLP - 1024 Hidden Layer Nodes Adam Optimizer
MLP - 1024 Hidden Layer Nodes with 50% Dropout Layer
MLP - 1024 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer
MLP - 2048 Hidden Layer Nodes
MLP - 2048 Hidden Layer Nodes - Adam Optimizer
MLP - 2048 Hidden Layer Nodes with 50% Dropout Layer
MLP - 2048 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer
MLP with 1024 Hidden Layer Nodes with 154 Input Nodes via PCA
MLP with 1024 Hidden Layer Nodes with 70 Input Nodes via Random Forest

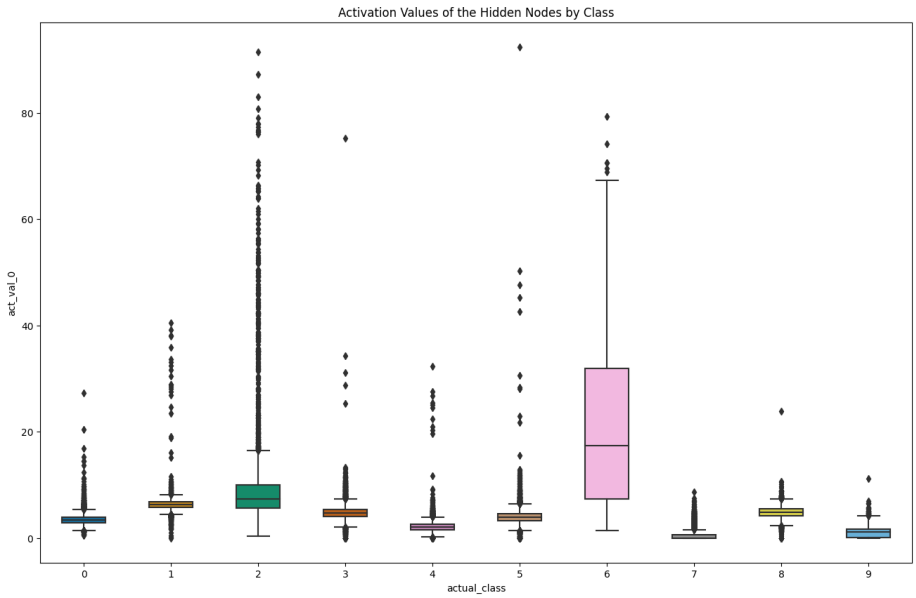
For each of the experiments run, performance metrics, and a confusion matrix were recorded. For some of the smaller models, additional visualizations were created to show how activation values portray representations learned in that layer.

Results

Hidden Unit Representations

Figure A2 in the appendix displays the results of the experiments. What is immediately evident is how the one hidden unit model severely underfit and achieved a dismal 36% on the test set. This is due to the fact that the one hidden unit was not sufficient to produce the representation necessary to discriminate between the classes. To illustrate this point, take a look at Figure 3.

Figure 3: Boxplot of Activation Values by Class



As can be seen, the range of values that the neuron's activation function produces do not seem to be useful in discriminating between the task. For instance, the activation value of 5 falls neatly within the distributions of classes 0, 1, 3, 4, 5, and 8, making these values largely useless in discriminating between classes.

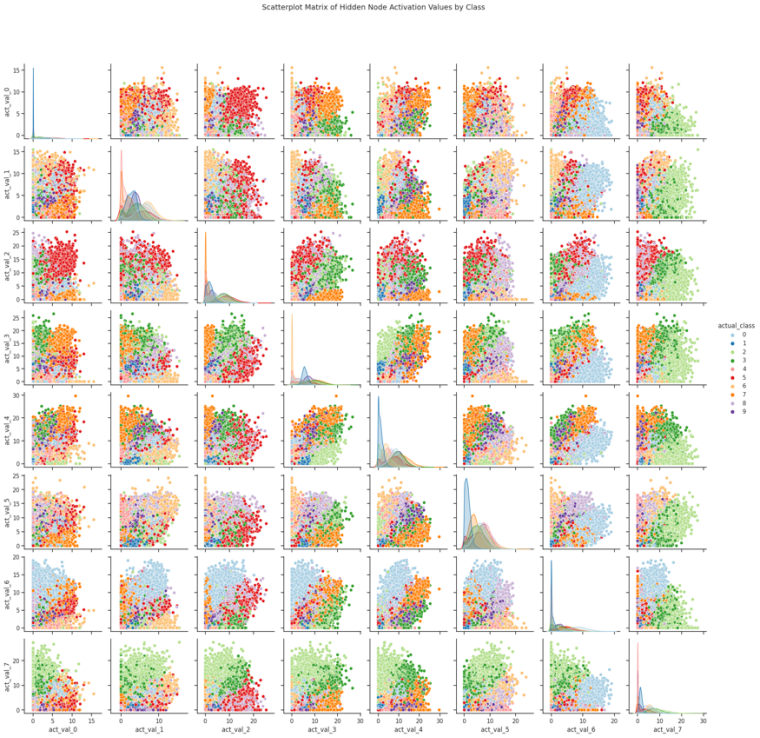
However, performance seems to increase as the number of hidden units are increased in the hidden layer. Jumping from one to two hidden units bring the test accuracy up to almost 70%. Figure 4 (pictured on the next page) shows a scatterplot of activation values from neurons one and two. Looking at this figure, one might notice that while there is still a large amount of overlap between activation values in some classes, the activation values show proof of a slightly more useful representation – one can follow "the clusters of activation values" and have a better chance at correctly classifying a given point using this representation.

Since human beings can better visualize higher dimensional spaces in 2D, looking at Figures 5 (four neurons) and 6 (eight neurons) help explain why more neurons produce better representations, as the scatterplot matrices display how an ensemble of representations makes it easier to find ways to discriminate between classes.

Figure 4: Scatterplot of Activation Values of Two Neurons

Figure 5: Scatterplot Matrix of Activation Values of Four Neurons

Figure 6: Scatterplot Matrix of Activation Values of Eight Neurons

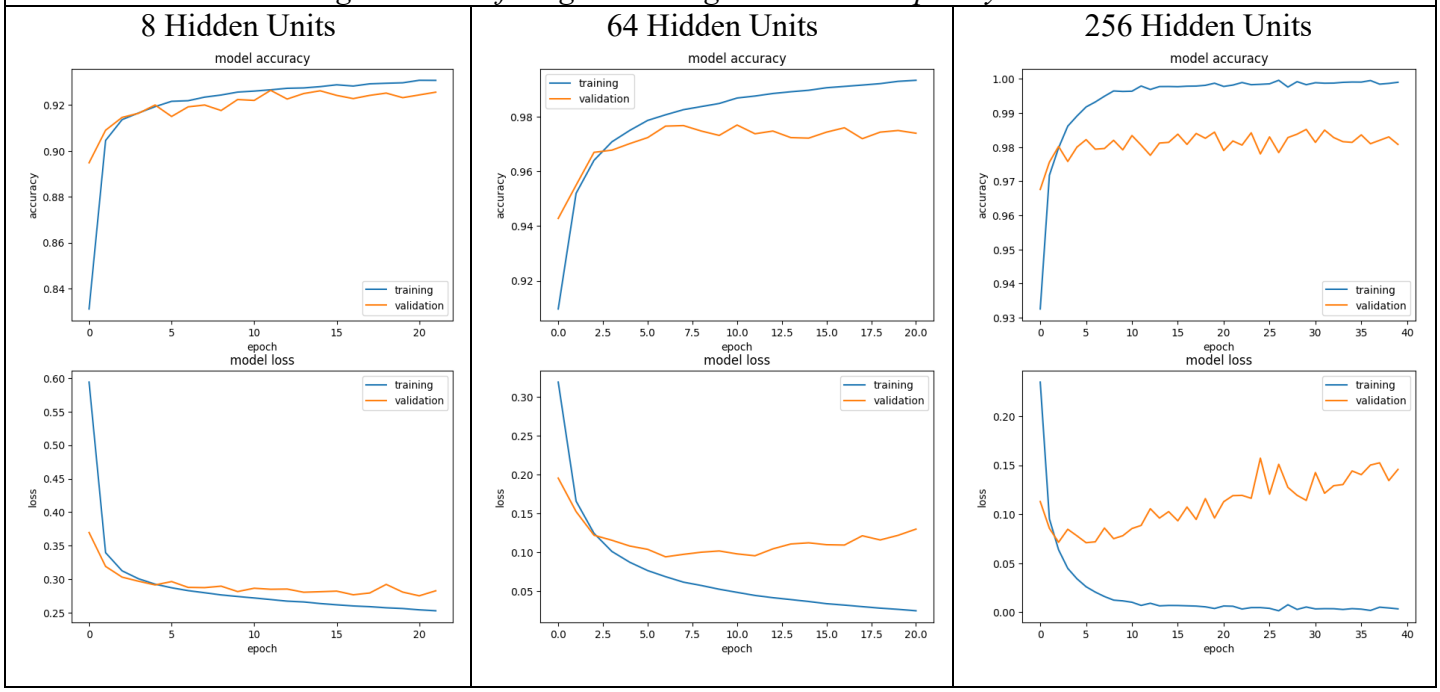


Overfitting

The performance of the models tended to increase as the number of hidden units increased, up until around 1024 hidden units where additional units do not seem to yield a significant increase in performance and often leads to overfitting. The overfitting can be witnessed during training by looking at plots of the training vs validation loss and accuracy.

Figure 7 below shows that as the number of hidden units increases, the gap between the training and validation loss and accuracy grows, indicating that the model will not translate as well in discriminating between handwritten digits in out of domain cases.

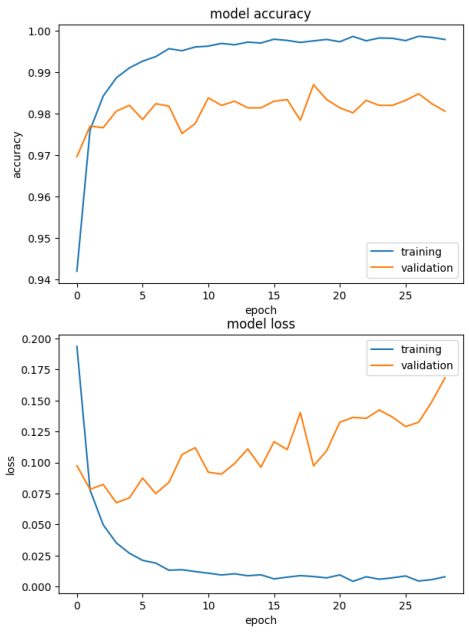
Figure 7. Overfitting Increasing as Model Complexity Increases



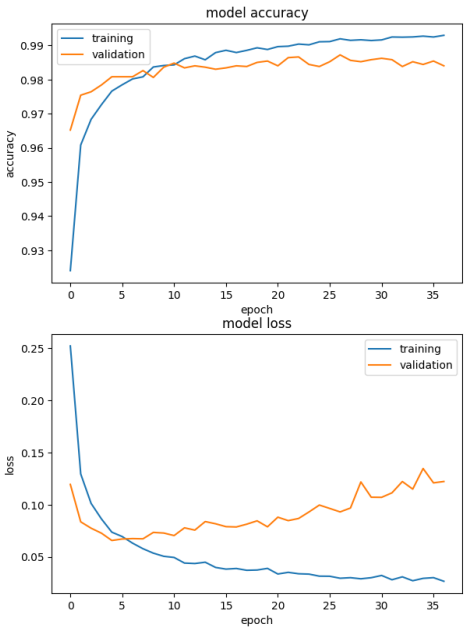
While some experiments attempted to mitigate overfitting through the use of a dropout layer, they did not result in a significant improvement in test accuracy, although those models did not overfit as severely, as seen in Figure 8.

Figure 8. Mitigating Overfitting During Training Using a 50% Dropout Layer

MLP Training with 1,024 Hidden Units



MLP Training with 1,024 Hidden Units with 50% Dropout Layer



Dimensionality Reduction

Dimensionality reduction is a technique often employed to reduce model complexity. In the context of neural networks, this results in a smaller input layer reducing excess noise and the number of parameters that need to be trained. There are multiple ways to perform dimensionality reduction.

The first method of dimensionality reduction performed was a Principal Component Analysis (PCA) which reduced the 784 inputs to 154, the number of components that explains 95% of the variability in the data. Finally, the input data in the train, validation, and test sets were transformed into the PCA components. This new input data was tested on the same architecture as the best model and produced an accuracy score of 98.02%. This was a lower accuracy score than the accuracy of the original best model.

The second method attempted at dimensionality reduction was performed using 70 of the most important features (pixels) according to a random forest model that itself achieved a test accuracy of 94%. These features were extracted from the input images and were fed into the same architecture as the best model, resulting in an accuracy score of 95%, but over three percentage points lower than our best model.

These dimensionality reduction methods are similar to convolutional layers of neural networks in that they attempt to extract the most important features and reduce the number of inputs into the dense neural network. However, the PCA and random forest techniques employed in this

experiment appear to be less performant than the convolutional neural networks in literature which have achieved close to human-level performance. It is clear that convolutional layers are a superior method of refining and extracting features than PCA and random forest for perceptual data.

Optimizers

Changing the optimizer from RMSProp to Adam did not result in a significant difference in performance. In some models, Adam performed marginally better than RMSProp, and vice versa in others. There did not seem to be any patterns to explain any performance gain in one versus the other.

Best Model

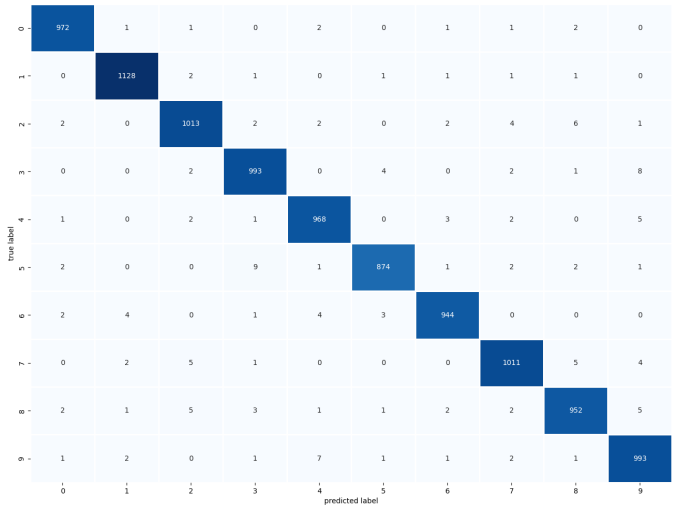
After conducting the experiments, the best model was the single hidden layer with 1024 hidden units, a ReLU activation function, learning rate of 0.001, and RMSProp optimizer. This model achieved a test accuracy of 98.48% beating out the model developed by Simard et al. (2003) by 0.08%. The model also boasted a 98% precision, recall, and F1 score. Figure 9 reports the precision, recall, and F1 score by class for the model. Figure 10 shows the confusion matrix for the best model.

From the experiments, it appears that having more hidden nodes resulted in better representations and thus better performance. While the best model managed to beat the 98.4% benchmark (Simard et al. 2003), it still did not manage to eclipse 98.5%. The struggle to eclipse this mark seems to result from overfitting. As can be seen in Figure A2 in the appendix, many of the models with greater number of hidden units achieve close to perfect accuracy on the training set but result in 1-2% drops in performance on the test set. Literature shows that additional data augmentation and regularization techniques may prove effective in limiting overfitting (LeCun et al. 2010). Additionally, superior performance may be obtained from using more hidden layers and convolutional architectures to extract and refine features. These strategies should be explored further to improve performance.

Figure 9. Performance Metrics by Class of Best Model

Class	Precision	Recall	F1-score	Support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.98	0.98	0.98	1010
4	0.98	0.99	0.98	982
5	0.99	0.98	0.98	892
6	0.99	0.99	0.99	958
7	0.98	0.98	0.98	1028
8	0.98	0.98	0.98	974
9	0.98	0.98	0.98	1009
Accuracy	0.9848			10000
Macro Avg	0.98	0.98	0.98	10000
Weighted Average	0.98	0.98	0.98	10000

Figure 10. Confusion Matrix of Best Model



Conclusion

The best model proposed in this paper is effective at classifying handwritten digits with a test accuracy of 98.48% and provides a boost in performance over the slow and error prone OCR software used previously by the USPS in the 1990s. Deploying the model developed in this paper to read zip codes would result in a more efficient and accurate sorting of mail, saving a lot of time and money.

Additionally, the experiments highlighted the importance of having a sufficient number of hidden units to produce strong representations while exposing the adverse overfitting results of too many hidden units. The experiments also demonstrated that when applied to perceptual data, dimensionality reduction methods are less effective for reducing and extracting the most important features than convolutional layers. The use of data augmentation, convolutional layers, and more hidden layers have been proven in literature to improve performance on the MNIST dataset and should be investigated further to build a more performant model. The new model should be used by USPS to perform handwritten digit recognition.

References

Hanson, M., & CREATIVE ART STUDIO, INC. (1970). *Reading and Sorting Mail Automatically*. U.S. Post Office Dept. Retrieved July 9, 2023, from https://www.youtube.com/watch?v=V4LJs2ZoDR4&t=422s&ab_channel=USNationalArchives.

Kaggle. (n.d.). *Digit recognizer*. Kaggle. https://www.kaggle.com/competitions/digit-recognizer/leaderboard

LeCun, Y., Cortes, C., & Burges, C. (2010). MNIST handwritten digit database.

Papers with Code. (n.d.). *MNIST Benchmark (Image Classification) | Papers With Code*. Papers with Code: The Latest in Machine Learning. https://paperswithcode.com/sota/image-classification-on-mnist

Saldarini, K. (1999, February 1). *Postal Service Tests Handwriting Recognition System*. Government Executive. https://www.govexec.com/federal-news/1999/02/postal-service-tests-handwriting-recognition-system/1746/

Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003, August). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar* (Vol. 3, No. 2003).

SRI International. (2022, December 14). *75 years of Innovation: Advanced Postal Address Recognition*. 75 Years of Innovation: Advanced postal address recognition - SRI International. https://www.sri.com/press/story/75-years-of-innovation-advanced-postal-address-recognition/

Yahya, A. A., Tan, J., & Hu, M. (2021). A novel handwritten digit classification system based on convolutional neural network approach. *Sensors*, *21*(18), 6273.

Appendix

Figure A1: Frequency of Class Labels

Frequency of Class Labels			
Class	**Y_Train**	**Y_Val**	**Y_Test**
0	5414	509	980
1	6207	535	1135
2	5481	477	1032
3	5648	483	1010
4	5329	513	982
5	4959	462	892
6	5432	486	958
7	5745	520	1028
8	5365	486	974
9	5420	529	1009
Total	55000	5000	10000

Figure A2: Performance Results from Experiments

Experiment	Description	Input Nodes	Hidden Layer Nodes	Hidden Activation	Optimizer	Epochs	Train Accuracy	Validation Accuracy	Test Accuracy	Precision	Recall	F1 Score
1	MLP- 1 Hidden Layer Node	784	1	ReLU	RMSProp	10	0.4569	0.451	0.3592	0.29	0.36	0.3
2	MLP - 2 Hidden Layer Nodes	784	2	ReLU	RMSProp	33	0.7089	0.7106	0.6959	0.7	0.7	0.69
3a	MLP - 4 Hidden Layer Nodes	784	4	ReLU	RMSProp	43	0.8667	0.865	0.8602	0.86	0.86	0.86
3b	MLP- 8 Hidden Layer Nodes	784	8	ReLU	RMSProp	12	0.9273	0.9264	0.9256	0.93	0.93	0.93
3c	MLP- 16 Hidden Layer Nodes	784	16	ReLU	RMSProp	26	0.9639	0.9554	0.953	0.95	0.95	0.95
3d	MLP - 32 Hidden Layer Nodes	784	32	ReLU	RMSProp	10	0.976	0.969	0.9634	0.96	0.96	0.96
3e	MLP 64 Hidden Layer Nodes	784	64	ReLU	RMSProp	23	0.9869	0.977	0.9726	0.97	0.97	0.97
3f	MLP - 128 Hidden Layer Nodes	784	128	ReLU	RMSProp	11	0.9945	0.981	0.9779	**0.98**	**0.98**	**0.98**
3g	MLP - 256 Hidden Layer Nodes	784	256	ReLU	RMSProp	25	1	0.9848	0.9803	**0.98**	**0.98**	**0.98**
3h	MLP- 256 Hidden Layer Nodes - Adam Optimizer	784	256	ReLU	Adam	30	0.9983	0.9852	0.9796	**0.98**	**0.98**	**0.98**
3i	MLP - 256 Hidden Layer Nodes with 50% Dropout Layer	784	256	ReLU	RMSProp	23	0.9805	0.984	0.9797	**0.98**	**0.98**	**0.98**
3j	MLP - 256 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer	784	256	ReLU	Adam	34	0.9869	0.9852	0.9816	**0.98**	**0.98**	**0.98**
3k	MLP - 512 Hidden Layer Nodes	784	512	ReLU	RMSProp	23	1	0.986	0.9801	**0.98**	**0.98**	**0.98**
3l	MLP - 512 Hidden Layer Nodes Adam Optimizer	784	512	ReLU	Adam	18	0.9974	0.9846	0.9788	**0.98**	**0.98**	**0.98**
3m	MLP - 512 Hidden Layer Nodes with 50% Dropout Layer	784	512	ReLU	RMSProp	23	0.9881	0.9848	0.9811	**0.98**	**0.98**	**0.98**
3n	MLP - 512 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer	784	512	ReLU	Adam	34	0.9912	0.9876	0.9803	**0.98**	**0.98**	**0.98**
3o	MLP - 1024 Hidden Layer Nodes	784	1024	ReLU	RMSProp	8	1	0.986	**0.9848**	**0.98**	**0.98**	**0.98**
3p	MLP - 1024 Hidden Layer Nodes Adam Optimizer	784	1024	ReLU	Adam	19	0.9976	0.987	0.979	**0.98**	**0.98**	**0.98**
3v	MLP - 1024 Hidden Layer Nodes with 50% Dropout Layer	784	1024	ReLU	RMSProp	23	0.9924	0.9862	0.9833	**0.98**	**0.98**	**0.98**
3q	MLP - 1024 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer	784	1024	ReLU	Adam	27	0.9919	0.9872	0.98	**0.98**	**0.98**	**0.98**
3r	MLP - 2048 Hidden Layer Nodes	784	2048	ReLU	RMSProp	25	1	0.9862	0.979	**0.98**	**0.98**	**0.98**
3s	MLP - 2048 Hidden Layer Nodes - Adam Optimizer	784	2048	ReLU	Adam	37	0.9983	0.9812	0.9792	**0.98**	**0.98**	**0.98**
3t	MLP - 2048 Hidden Layer Nodes with 50% Dropout Layer	784	2048	ReLU	RMSProp	26	0.9974	0.985	0.9835	**0.98**	**0.98**	**0.98**

3u	MLP - 2048 Hidden Layer Nodes with 50% Dropout Layer - Adam Optimizer	784	2048	ReLU	Adam	21	0.991	0.986	0.9809	**0.98**	**0.98**	**0.98**
4	MLP with 1024 Hidden Layer Nodes with 154 Input Nodes via PCA	154	1024	ReLU	RMSProp	35	0.9879	0.9838	0.9802	**0.98**	**0.98**	**0.98**
5	MLP with 1024 Hidden Layer Nodes with 70 Input Nodes via Random Forest	70	1024	ReLU	RMSProp	74	0.9429	0.9452	0.951	0.95	0.95	0.95