IDENTIFYING HANDWRITTEN DIGITS USING A SINGLE LAYER DENSE NEURAL NETWORK

Michael Mistarz

Northwestern University, MSDS458: Artificial Intelligence & Deep Learning

GitHub: https://github.com/mistmr7/MSDS458_Single-Layer-DNN-on-MNIST-Data

January 26, 2024

1. Abstract

This study utilized a dense neural network with a single hidden layer to identify handwritten numerical digits from the Modified National Institute of Standards and Technology (MNIST) dataset. The MNIST dataset consists of 60,000 training images and 10,000 test images of digits zero through nine that are grayscale, centered, and converted to a grid of 28 x 28 pixels. A convolutional neural network (CNN) was used to analyze the handwritten digits, with modification of activation functions, number of input nodes, and with and without analytically reducing the input data of the model.

The study started with a single node relu activation function layer, then added a second node to see how that altered the results. Each of the 23 activation nodes was run with 10 hidden nodes over 10 epochs to determine which activation functions to pursue for building the best model. After the top four activation functions were selected, 11 different node sizes were compared with each of the four best activation functions and from there a best model was selected. This architecture of this model was used to analyze how pre-processing the input data using both primary component analysis (PCA) and random forest decision tree modeling. The best model selected utilized a mish activation function and 256 nodes over 10 epochs. This model was able to predict the test images with nearly 98 percent accuracy, and the PCA-reduced model produced nearly identical results with more than a fifty percent reduction in processing time. The RF-reduced input data also reduced the processing time by more than fifty percent, but the test accuracy of the best model on this pre-processed data closer to 94 % accuracy.

2. Introduction

Using a computer to classify handwritten images is a form of optical character recognition (OCR). OCR is an automatic identification technique that was first attempted in 1870 with the invention of the retinal scanner by C.R. Carey (Eikvil 1993). Attempts to further the use of OCR were explored through the early twentieth century, and in 1954, Reader's Digest employed the first OCR machine converting typewritten reports to a computer system (Eikvil 1993). By the MNIST dataset was made publicly available for training digit recognition, OCR was a burgeoning field that had many applications throughout a variety of industries.

A simple example of a use case would be by the United States Postal Service (USPS) utilizing OCR to identify handwritten addresses on handwritten envelopes and packages. OCR can aid in automating identifying addresses and sorting mail and packages for easier processing and shipping. It is used in camera systems to identify license plates, used by self-driving cars to read road signs and features of other cars, scanning documents for text-to-speech recognition, and has many other use cases as well (Boesch 2023).

OCRs often utilize CNNs due to their ability to utilize backpropagation, a technique that takes the error rates from the loss function and adjusts the weights of the neural network to improve accuracy of the output of the model (Agrawal et al 2024). CNNs are a powerful machine learning tool, but they do come with several drawbacks. CNNs utilize backpropagation which is a time-consuming and resource-intensive technique. CNNs need large processors to be able to efficiently run the models, forcing many users to purchase cloud server space which is costly, time-consuming and comes with data privacy concerns (De Gruyter 2023). CNNs require a large, labeled datasets which can often be difficult to aggregate, and they are also prone to overfitting, where the model focuses too intently on the training data and the results in testing

have an increase in noise (Tamanna 2023). As shown in this paper, these can be mitigated through data preprocessing by using techniques like Primary Component Analysis (PCA) and Random Forest decision trees to lessen the input parameters to the most important features to lower the time and energy constraints needed to run the model. To mitigate noise issues and overfitting, adding a monitor to stop the model when the results become noisy can lead to better results (TensorFlow n.d.).

3. Literature Review

The MNIST dataset has been used for decades to work on optical character recognition systems and there are many examples of deep learning techniques used to classify the images (LeCun n.d.). From how-to guides written on data science websites like *Medium* showing step-by-step how to set up a simple neural network using only NumPy to guides for TensorFlow and Keras or PyTorch deep learning libraries (Ombaval 2024; Khan 2024; Gopalakrishnan 2022). Several books have also been written on the subject to help users get into entry level deep learning, with step-by-step guides alongside in-depth discussions of deep learning and computer vision (Shanmugamani 2018). MNIST provides a perfect entry level dataset to start exploring what can be done with OCR and deep learning frameworks.

More in-depth studies show the power of how different neural network architectures perform on the MNIST dataset, with several methods exploring image recognition models that are available for use publicly. One such study compared four different publicly available image recognition models, including a CNN, ResNet, CapsNet, and DenseNet (Chen et al 2024). In the study in this paper a single layer CNN was analyzed. ResNet was created to address degradation issues from CNNs without using dense layers, DenseNet was modeled after ResNet but with dense connections, and CapsNet is a model consisting of both a convolutional layer and two

capsule layers (Chen et al 2024). All four of the models in the study were able to achieve accuracy results greater than 98.3% (Chen et al 2024).

4. Methods

The input data used in this study was loaded from the MNIST dataset (LeCun n.d.). This dataset is provided in Keras's TensorFlow built-in datasets and was accessed using Python. The dataset provides 60,000 grayscale and centered images of handwritten digits transformed into a 28 by 28-pixel grid, along with 10000 separate test images to test the model. An example of the input data can be seen in Figure 1.

4.1 Exploratory Data Analysis

Exploratory data analysis (EDA) was initiated by creating a bar plot to visualize the distribution of the numbers in the training dataset, shown in Figure 2. One-hot encoding was used to convert each of the image labels to an array of size 10 to help pair with the expected output of the model. A sample of an input image can be seen in Figure 2, where each pixel from the input image is weighted from 0 to 255, where 255 corresponds to the black ink of the digit and 0 corresponds to white with no ink present. The images were reshaped to two dimensions using NumPy and then normalized by dividing by 255, the maximum value of the black pixels, to give each pixel a value from zero to one.

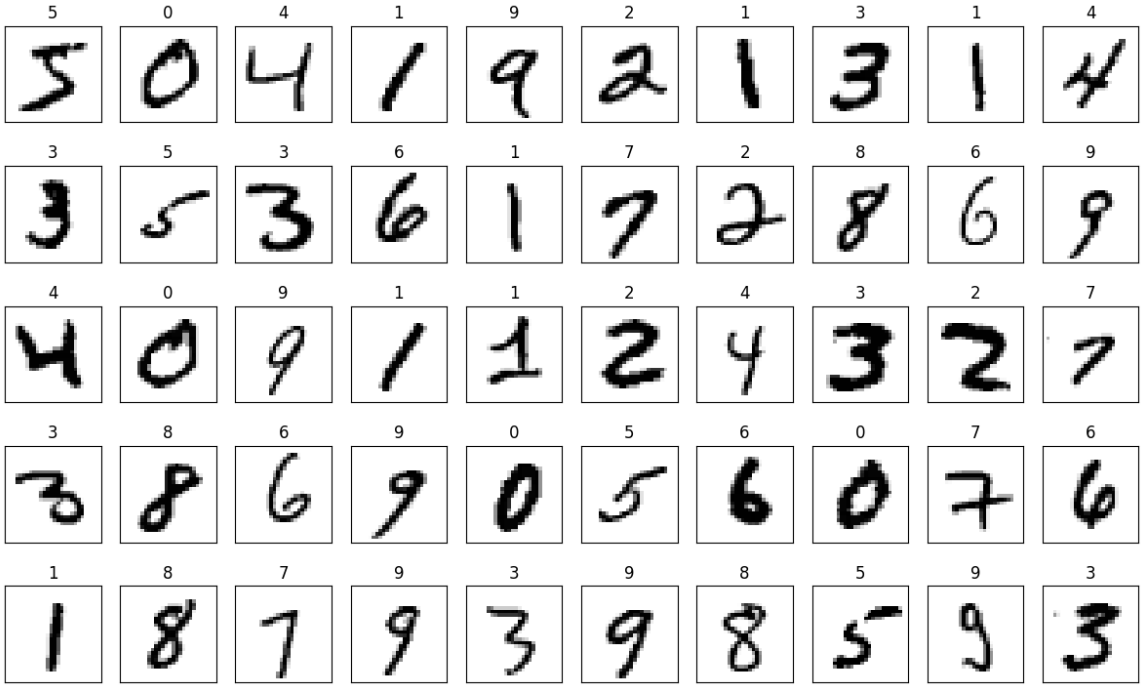

Figure 1: Example images from the first 50 images of the MNIST dataset in 28 x 28-pixel format

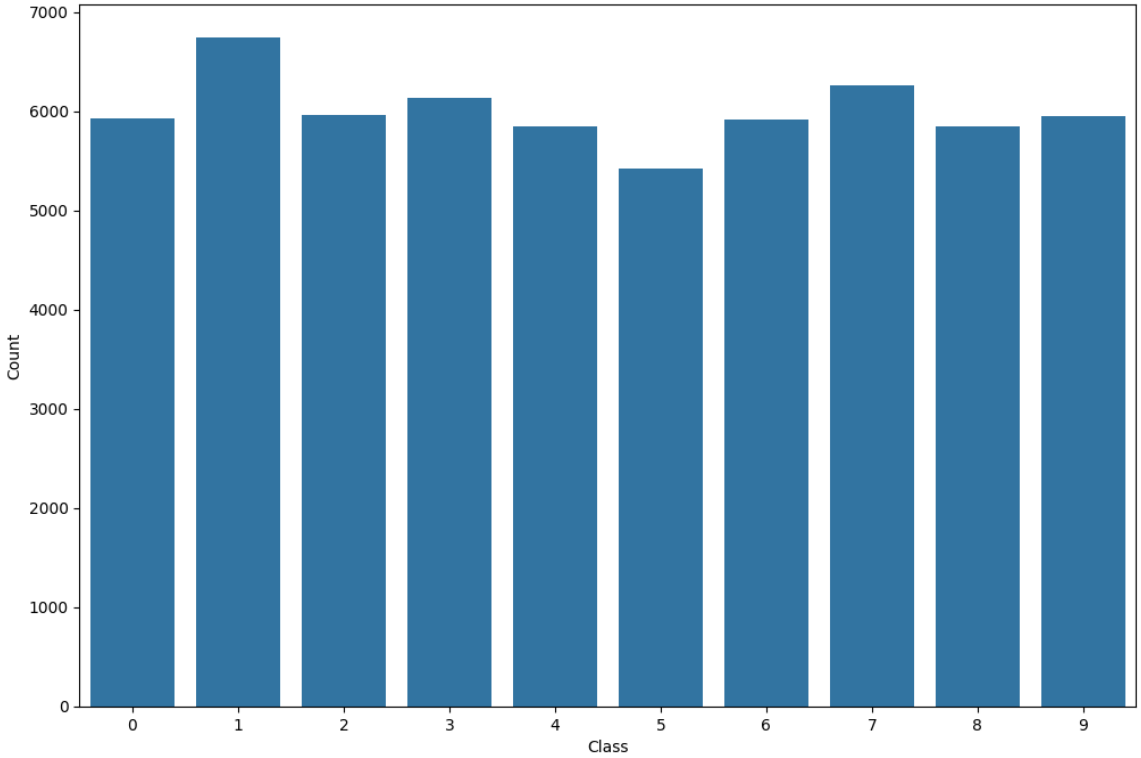

Figure 2: Distribution of values for the training data of the MNIST dataset

```
[[  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  83  91 143 255 190  91  50   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   9  49 180 246 253 253 253 253 253 220 154  17   3   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  46 107 178 253 253 253 253 253 253 253 253 253 253 253 126  45   0   0   0   0   0]
 [  0   0   0   0   0   0   0 107 253 253 253 253 223 220 220 220 220 245 253 253 253 253 106   0   0   0   0   0]
 [  0   0   0   0   0   0   0 107 173 253 229 129  12   0   0   0   0 110 253 253 253 253 106   0   0   0   0   0]
 [  0   0   0   0   0   0   0  17  14  40  32   0   0   0   0   0   0  57 253 253 253 242  85   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   5 139 224 253 253 253 105   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  65 178 253 253 253 253 219  24   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  97 250 253 253 253 253 127  47   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  46 125 250 253 253 253 245 171  33   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   5  41 217 253 253 250 245 245 115   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 124 253 253 253 192 105   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  11  47 220 253 253 188  25   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 107 253 253 253 189  13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  41 225 253 253 186  22   0   0   0   0   0  31  42 174 205 205 205 193  58   0   0   0]
 [  0   0   0   0   0  48 218 253 253 253 150  59   0   0 128 131 131 222 253 253 253 253 253  94   0   0   0   0]
 [  0   0   0   0   0  12 152 253 253 253 253 236 222 222 252 253 253 253 253 253 253 253 253 122   0   0   0   0]
 [  0   0   0   0   0   0   7 167 253 253 253 253 253 253 253 253 253 253 253 253 253 124 106   7   0   0   0   0]
 [  0   0   0   0   0   0   0  76 188 253 253 253 253 253 253 253 224  57  15  15  15   2   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  12  89 121 253 253 151  89  89  55   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]]
```

Figure 3: Training image viewed as pixel values from 0 to 255, on a scale from white (0) to black (255)

4.2 Building the First Model

Now that the data has been reshaped and normalized, the model was created as a sequential model from Keras TensorFlow. For the first experiment, an input layer was added with a single node and an input shape of 784, the number of pixels in an image. The activation for this layer was set to a rectified linear unit activation function, or relu(keras n.d.). An output layer was added with a softmax activation function that will map a probability function to an array of length ten in order to pair with the one-hot encoded labels. This will allow the model to determine the predicted value of the image. For validation of our training model, the first 55000 images and labels were set to training input data and the last 5000 from the MNIST training set were labeled as validation input data. A visual representation of the model can be seen in Figure 4.

The model was compiled using rmsprop as an optimizer with categorical cross entropy as the loss function while outputting accuracy. The model was fit to the training data and run for

thirty epochs. The training data was used to fully train the model, with the model outputting accuracy and loss for each epoch in the run. This was repeated for the validation data for each epoch to see how the model fared with non-training data to predict. The accuracy and loss were plotted over the epochs to see how the model progressed.

A confusion matrix was plotted to determine the true value for each input data point and the predicted value from the first model to visually view where the model was having difficulties in predictions. The values that the model consistently predicted incorrectly were visualized in their original images to understand where the model was having difficulty. An activation model was created to find the outputs of the activation value of the hidden layer, and this was plotted using a box plot grouped by predicted class.

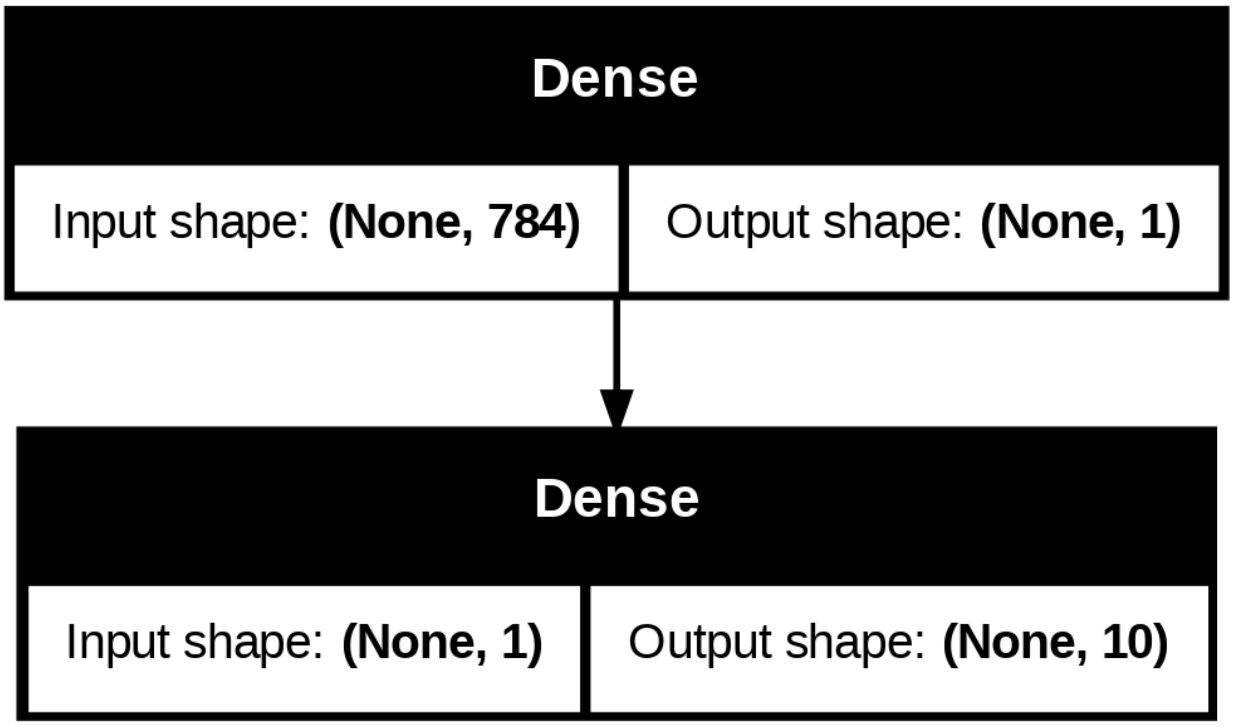

Figure 4: Visual representation of the model from Experiment 1

4.3 Building the Second Model

For the second model, all data preparation steps were repeated and a second node was added to the hidden layer. The model was built and compiled as discussed in experiment 1, and the resulting accuracy and loss functions for the training and validation data were plotted. A confusion matrix was built and an activation model was created to see the hidden node activations for each of the two nodes in the hidden layer. These values were combined with the predicted classes and plotted using a scatterplot to visualize how the plot of activation node 1 and activation node 2 clustered into predicted classes.

4.4 Exploring Activation Functions

To explore how the activation functions affect the outcome of otherwise identical models, each of the activation functions listed in the TensorFlow Keras library was tried as the activation function of the hidden layer, for a total of 23 activation functions (TensorFlow n.d.). A model was built for each activation layer with 10 nodes for the hidden layer and run for 10 epochs. Each working model's training accuracy, validation accuracy, training loss, and validation loss was plotted in small batches with the best models being added to the subsequent plots while the worst models were removed to determine a final set of four activation functions to explore more deeply.

4.5 Exploring Node Size

Eleven different node sizes were tried with each of the four best activation functions from section 4.4. The training loss, training accuracy, validation loss, and validation accuracy were plotted for each of the different node sizes to visualize how each of the four best activation

functions performed at each of the eleven node sizes. From these experiments, a best model was chosen and a confusion matrix was plotted for that model.

4.6 Utilizing PCA on Input Data

For this experiment, primary component analysis (PCA) was run on the input data to determine the components in images that corresponded to those that contain 95 percent of the variance in the training images. This selects the pixels that made up the majority of decision making for the model. This reduces the image size from 784 pixels to the 154 most important pixels. These reduced images were transformed and fit to the best model from section 4.5 and were preprocessed identically to earlier experiments. Data was seeded using NumPy and TensorFlow to get consistent results, and the best model from section 4.5 was compared to the PCA adjusted model by plotting the training and validation loss and training and validation accuracy for each.

4.7 Utilizing Random Forests on Input Data

Similarly to section 4.6, random forest decision tree classification (RF) was used on the input data to gather the 70 most important pixels in the images. This reduced image was fit to the best model from section 4.5. The data was seeded using NumPy and TensorFlow to get consistent results, and the best model from section 4.5 was compared to the output model of the RF reduced images by plotting the training and validation loss and the training and validation accuracy for each.

5. Results

5.1 Experiment 1

The model with a single node and a relu activation layer resulted a training accuracy of 0.3669 and a validation accuracy of 0.3946. The training loss was calculated as 1.6049 and the

validation loss was calculated as 1.5841. Plots of the loss and accuracy of the training and validation data over each epoch can be seen in Figure 5 and Figure 6 respectively.

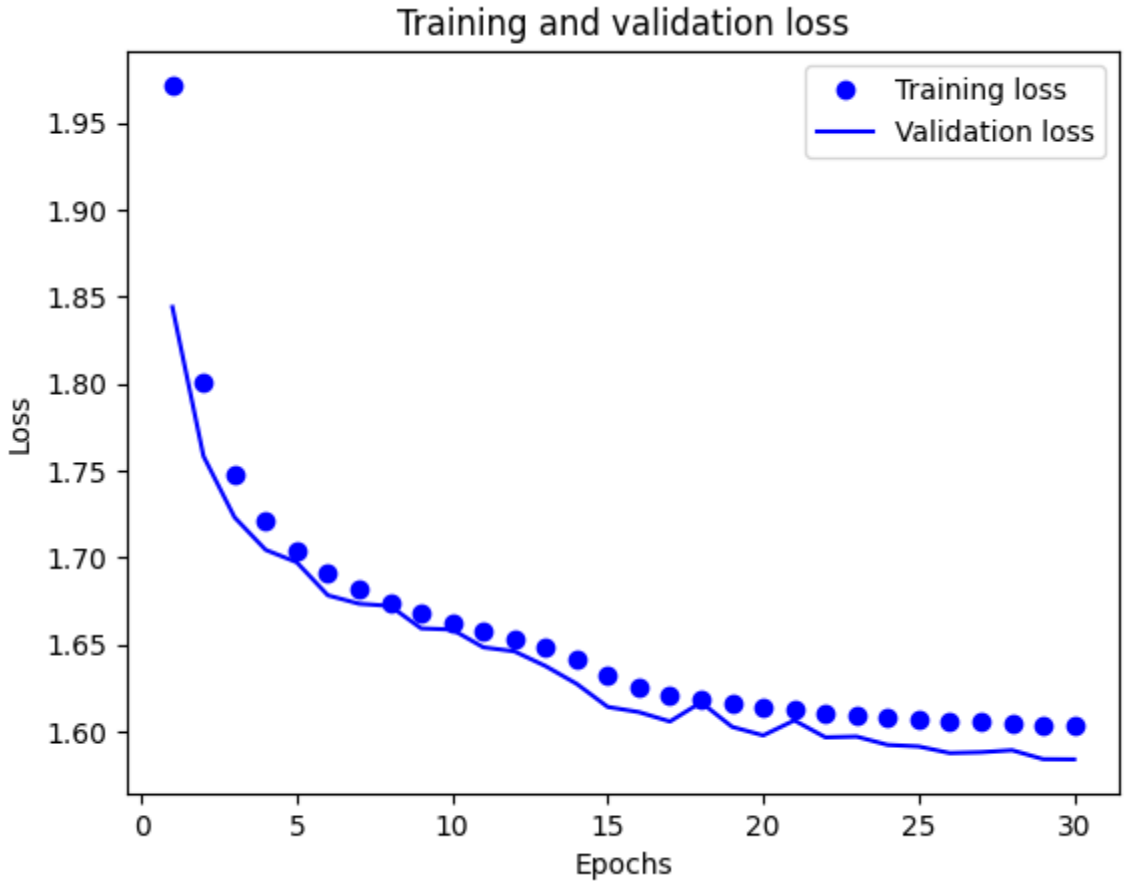

Figure 5: Training and validation loss for a hidden layer with a single node

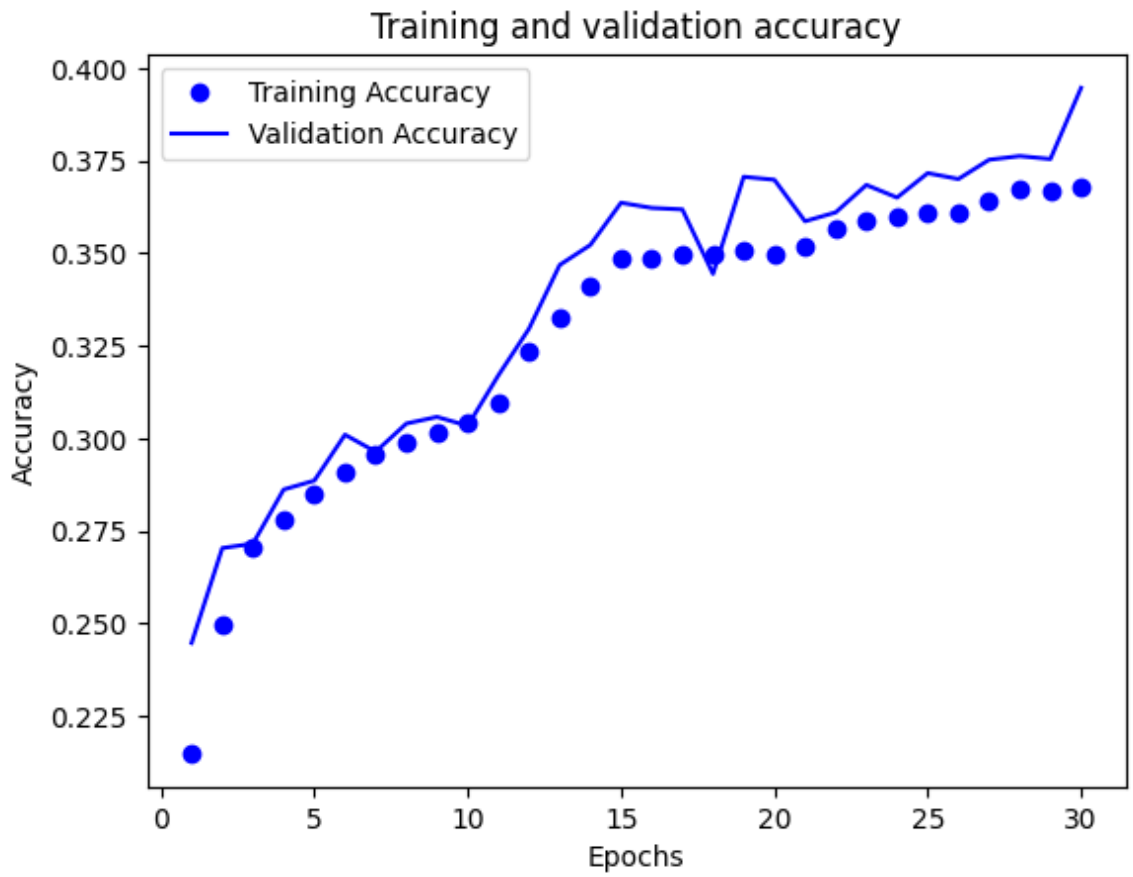

Figure 6: Training and validation accuracy for a hidden layer with a single node

The test data was fit to the model with a predicted accuracy of 0.3732 and a final loss value of 1.6490. A confusion matrix was generated to help make sense of the data, showing a plot of the true value of the image against the predicted value of the image, which can be seen in Figure 7. To visualize where the model was having the most issues, a hot spot of the confusion matrix that did not correspond to an accurate prediction was chosen and visualized, shown in Figure 8. The boxplot used to visualize the spread of the data can be seen in Figure 9. It should be noted that the model did not predict any of the images were a 5, 8, or 9 handwritten digit.

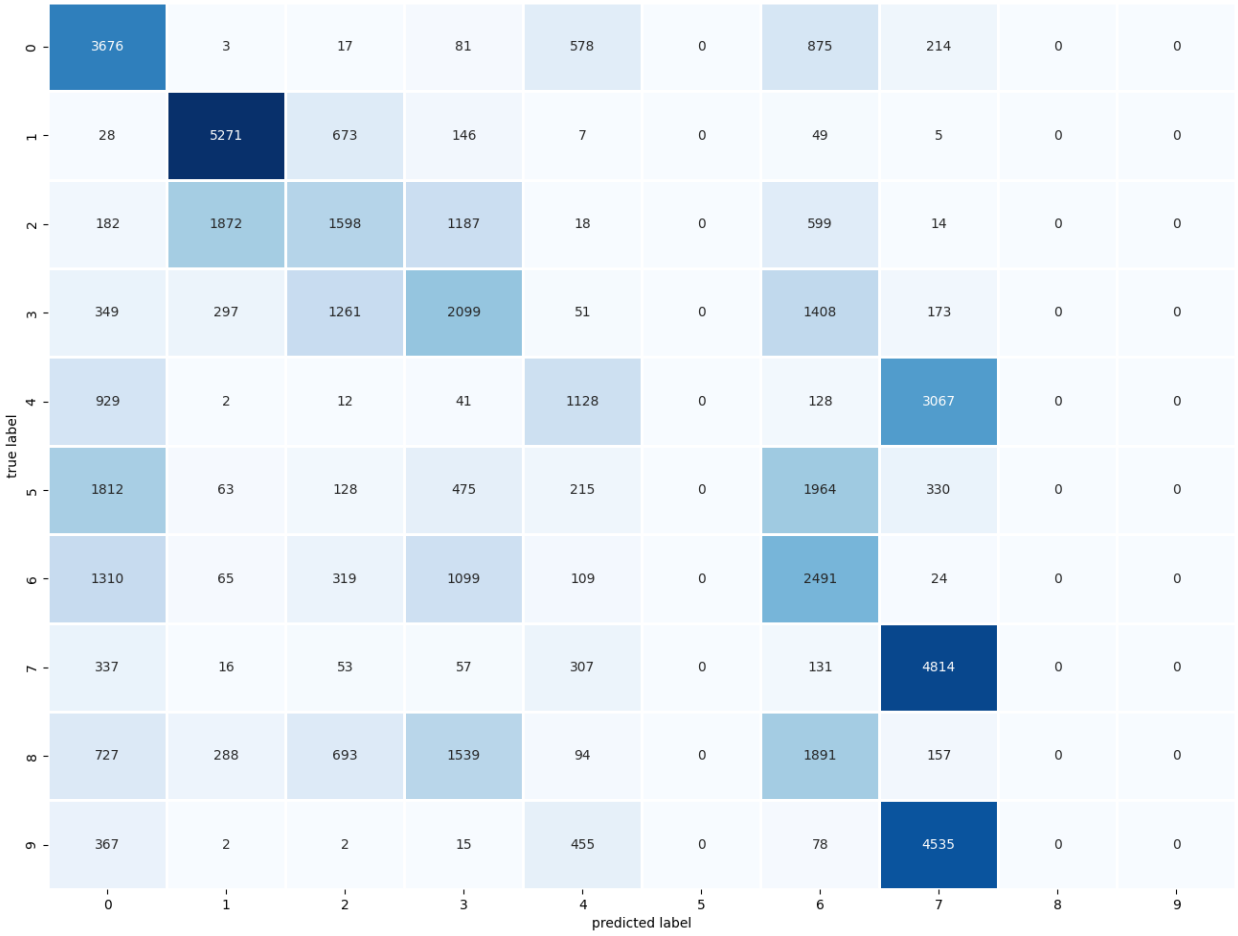

Figure 7: Confusion matrix for a hidden layer with a single node, plotting true value vs predicted value

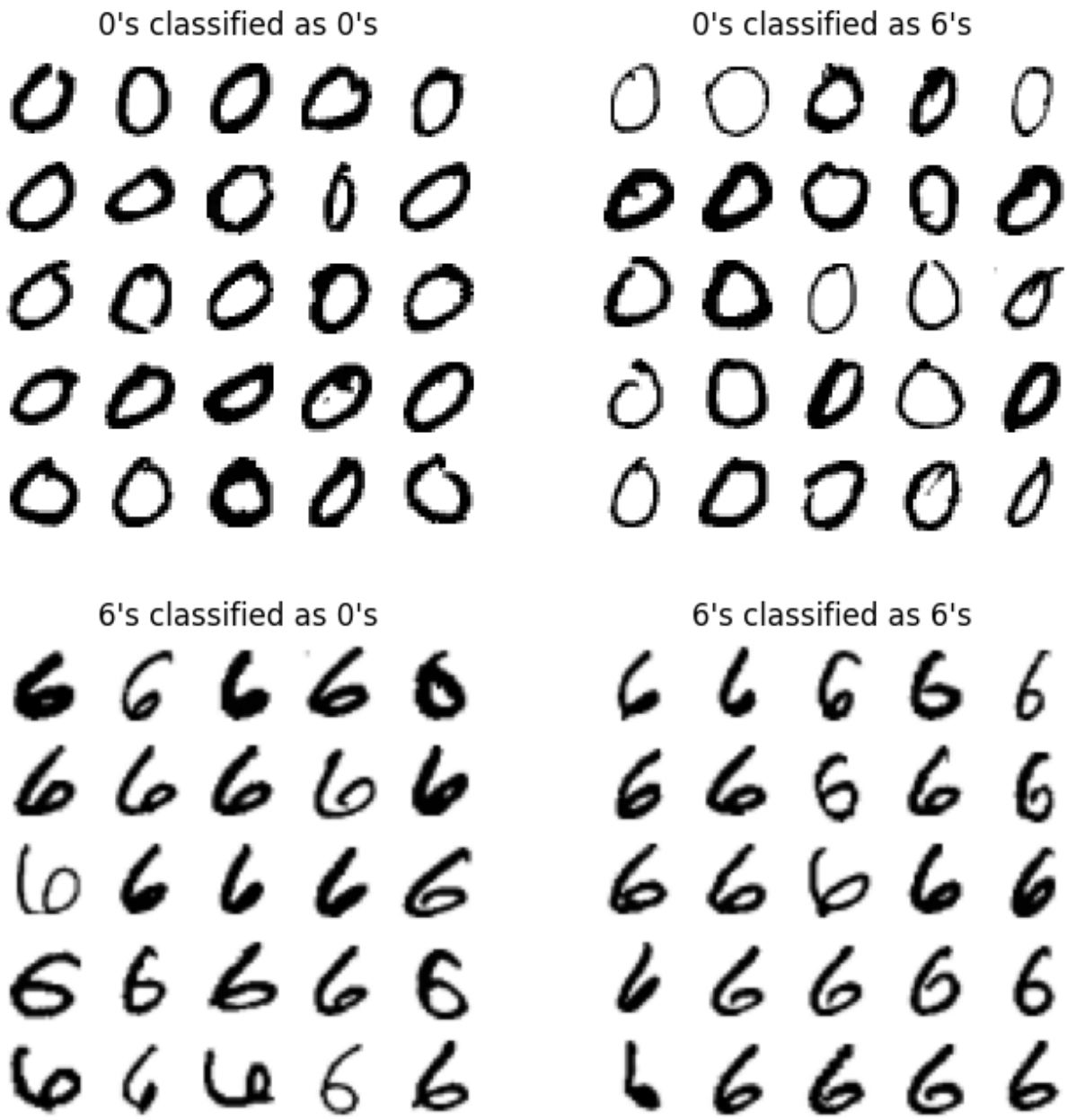

Figure 8: Visualization of incorrect classification of digits from Model 1

5.2 Experiment 2

The second trained model with two nodes in the hidden layer resulted in a training accuracy of 0.6987 and a validation accuracy of 0.7094. The calculated loss for the training and validation runs were 0.9245 and 0.9216 respectively. When the test data was fitted to the model, the model predicted the test images with an accuracy of 0.6782 and a loss of 1.0155. Comparing

the training, validation and testing data with that from the first model from Experiment 1 shows a marked improvement in both accuracy and loss for the model. The resulting scatterplot of the two activation nodes of the hidden layer can be seen in Figure 9, labeled by predicted class. Notice how there is very little overlap in the individual numbers, allowing the model to separate the numbers into expected results. The resulting plots of the data along with the confusion matrix can be seen within Appendix A.

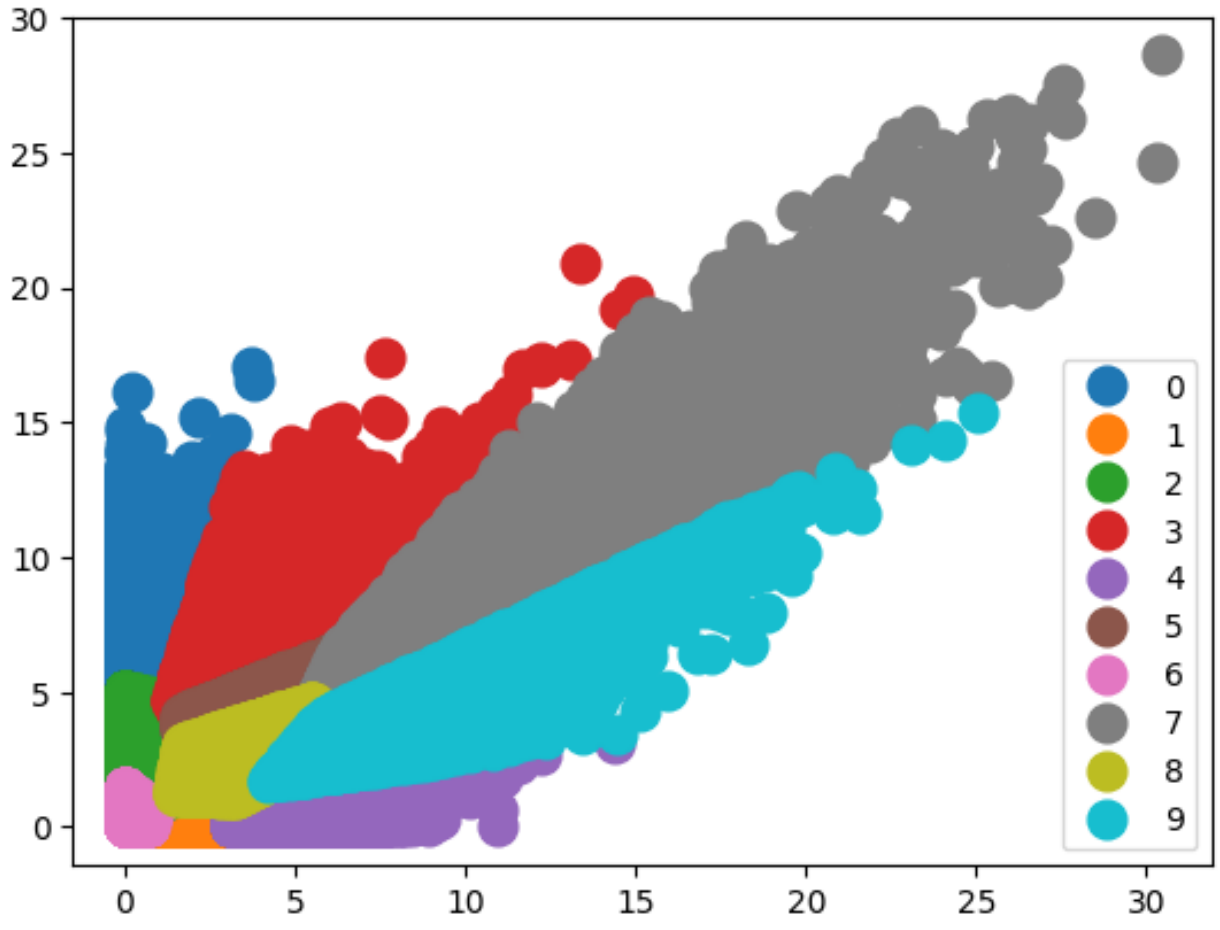

Figure 9: Scatterplot of activation value of hidden node 1 plotted against the activation value of hidden node 2 labeled by predicted value.

5.3 Exploring Activation Functions

Of the 23 activation functions listed in the TensorFlow Keras backend, 19 activation functions were able to be used to successfully train the model. These 19 models were each run with 10 nodes in the hidden layer for 10 epochs each. The accuracy and loss validation data were each plotted and the best 4 activation functions were selected for further analysis. These functions were hard swish, mish, relu and selu. An example loss and accuracy plot from this analysis can be seen in Figure 10 and Figure 11 respectively, while accuracy and loss plots from the other two runs can be found in Appendix B.

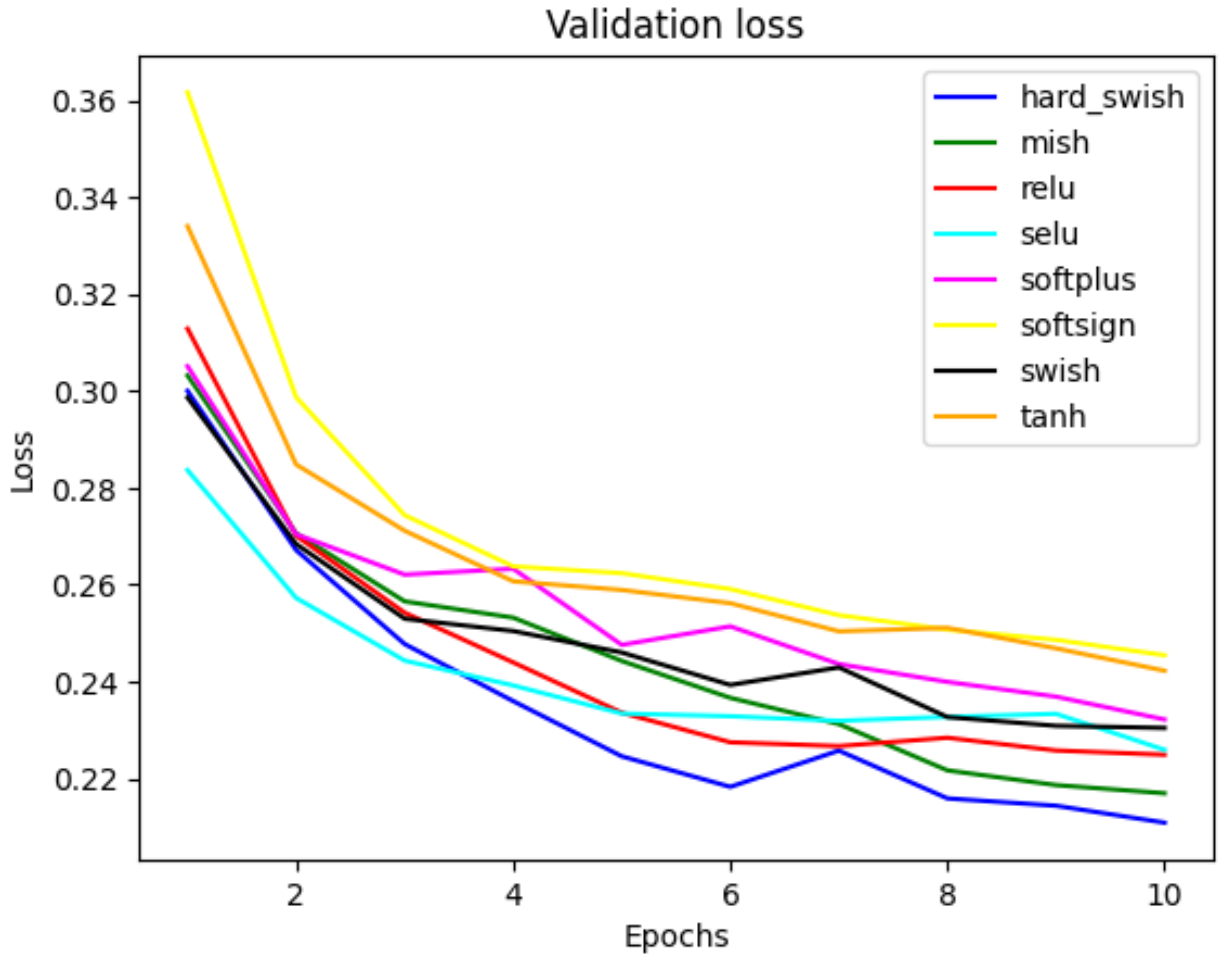

Figure 10: Validation loss plots for 8 of the 19 activation functions

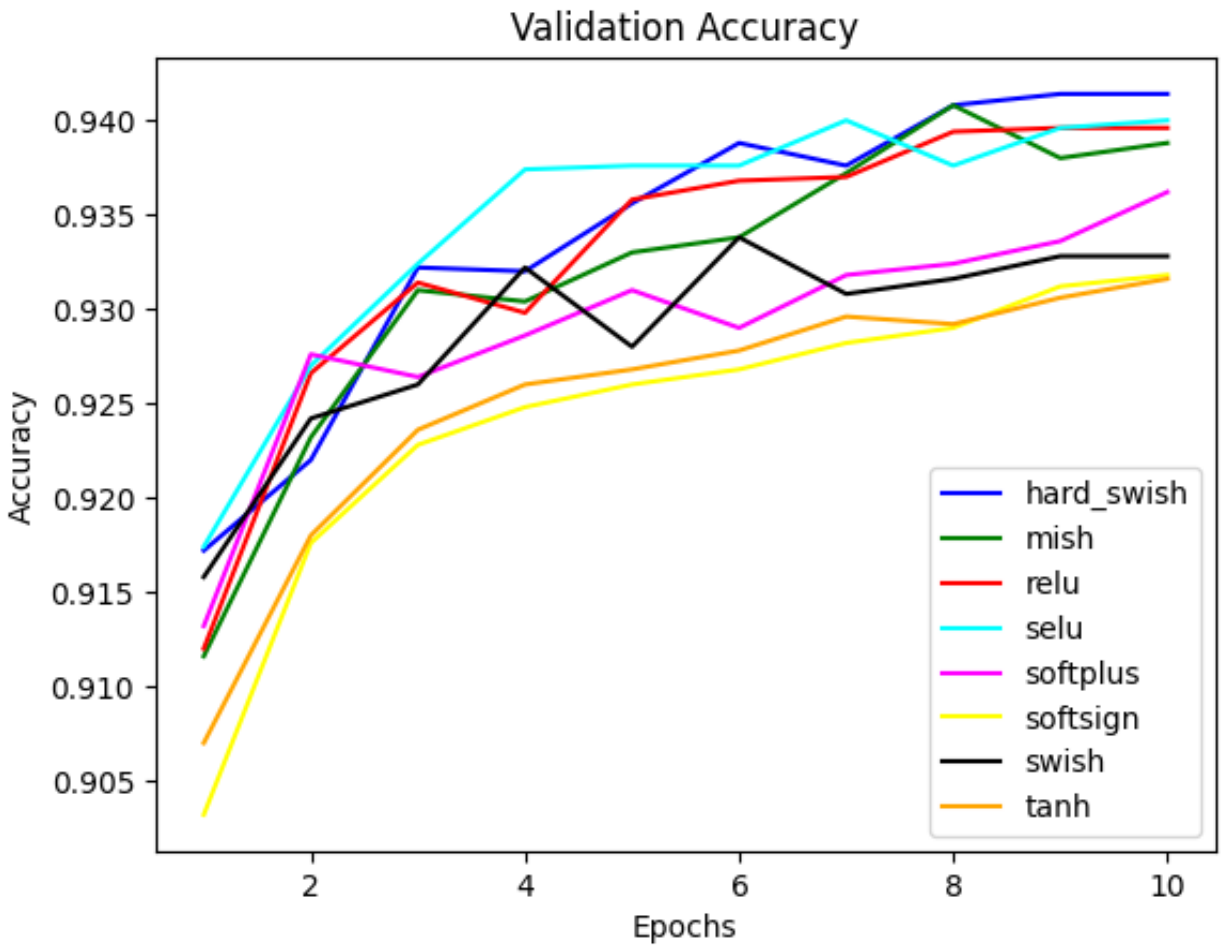

Figure 11: Validation accuracy plots for 8 of the 19 activation functions

5.4 Exploring Node Size

Eleven different node sizes were tested for each of the four activation functions chosen in Section 5.3. Node sizes of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024 were tested with hard swish, mish, relu and selu activation functions and training. Validation loss and accuracy were plotted for each of the different node sizes to pick the best model to go forward with. Example plots from the 256 node plots for validation loss and accuracy can be seen in Figure 12 and Figure 13 respectively.

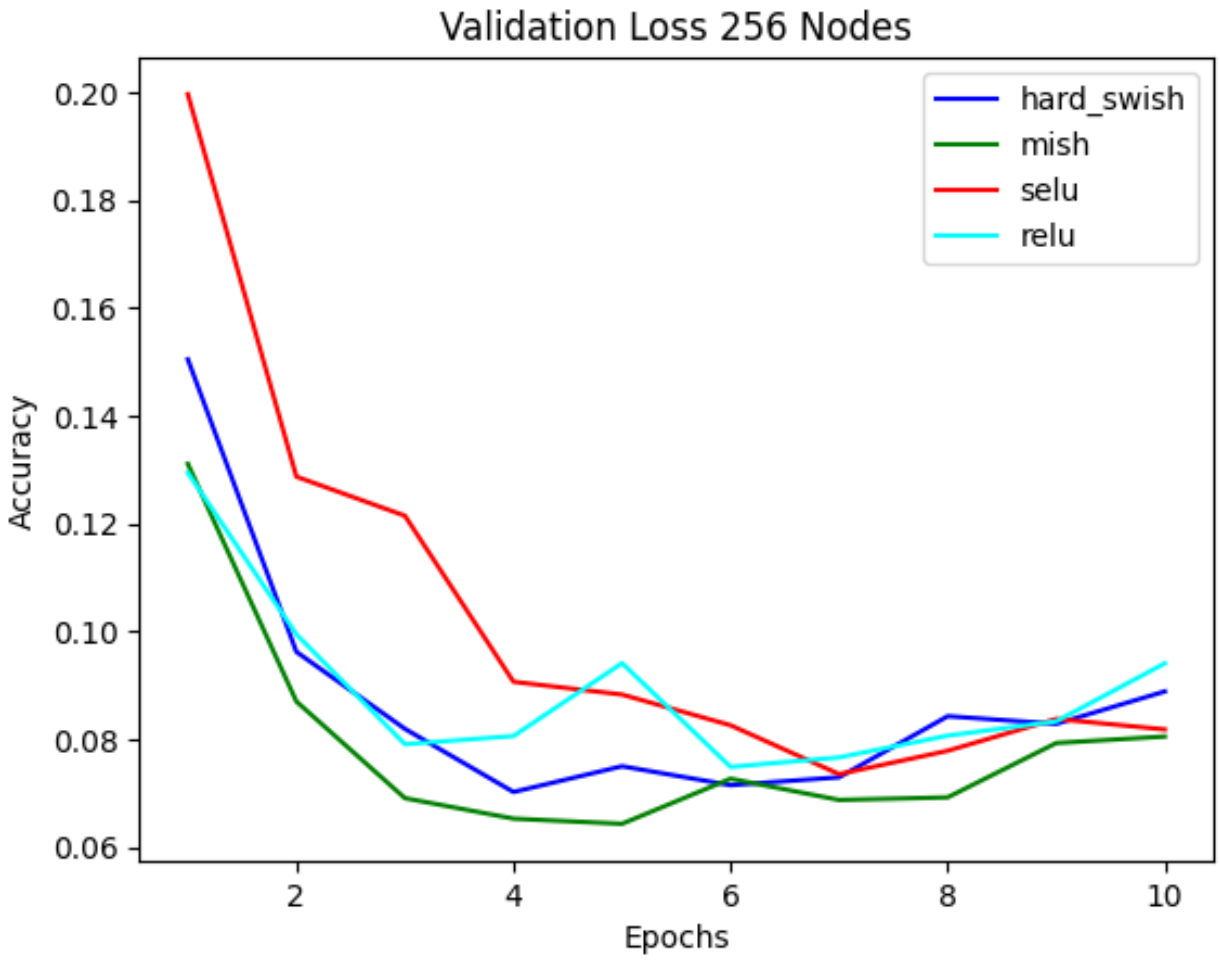

Figure 12: Validation loss plot from the model with 256 nodes comparing the four best activation functions

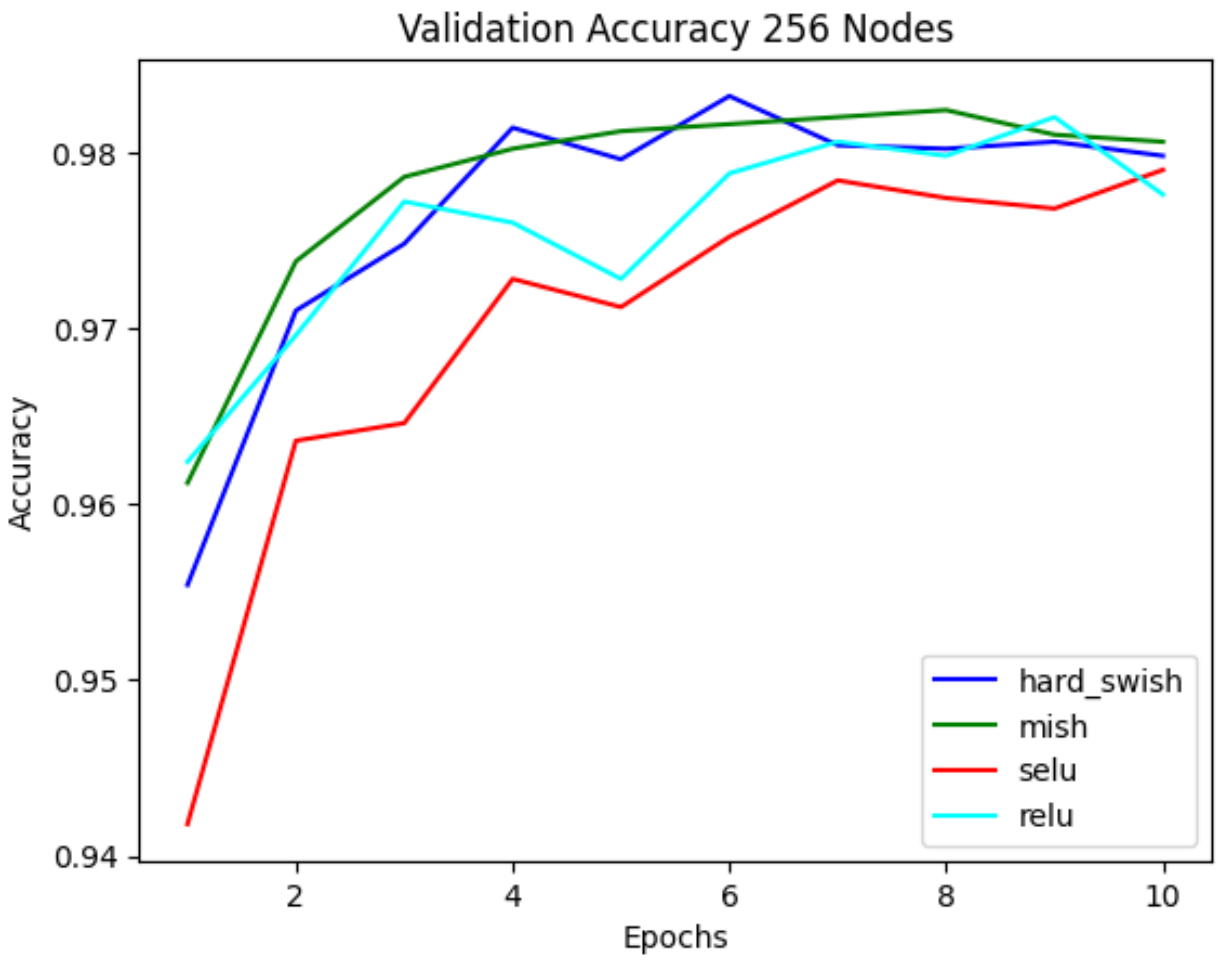

Figure 13: Validation accuracy plot from the model with 256 nodes comparing the four best activation functions

After analyzing all of the data and plots, the best model moving forward was chosen with a mish activation function and 256 nodes in the hidden node. The training and validation accuracy for this model were 0.9969 and 0.9806 respectively, and the training and validation loss for the model were 0.0103 and 0.0805 respectively. A confusion matrix was plotted for the model which can be seen in Figure 14. The confusion matrix shows very little deviation from the from the expected values, and one of the situations that deviated the most from the expected values is shown in Figure 15.

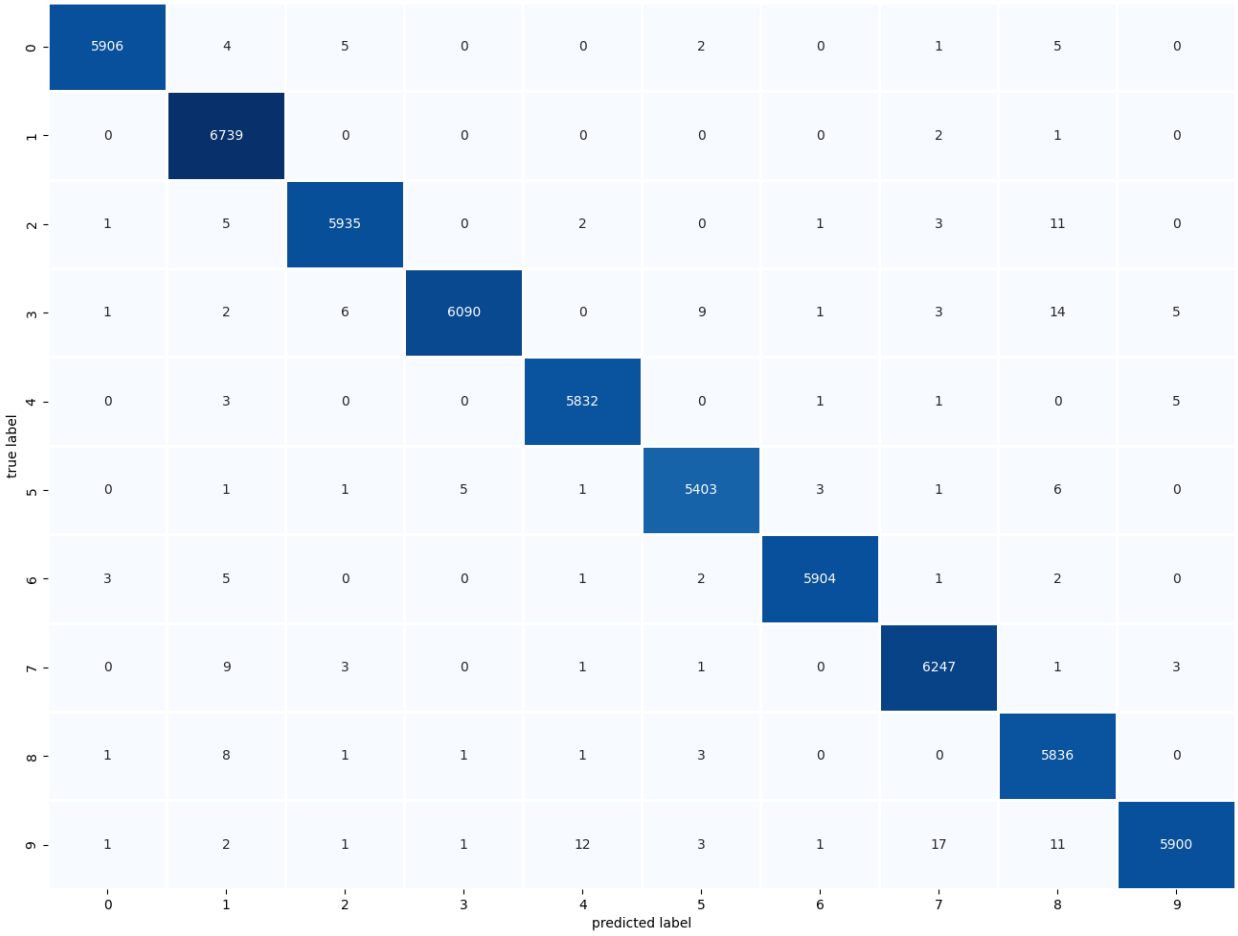

Figure 14: Confusion matrix for the model using mish activation and 256 nodes, chosen as the final mode before input data processing

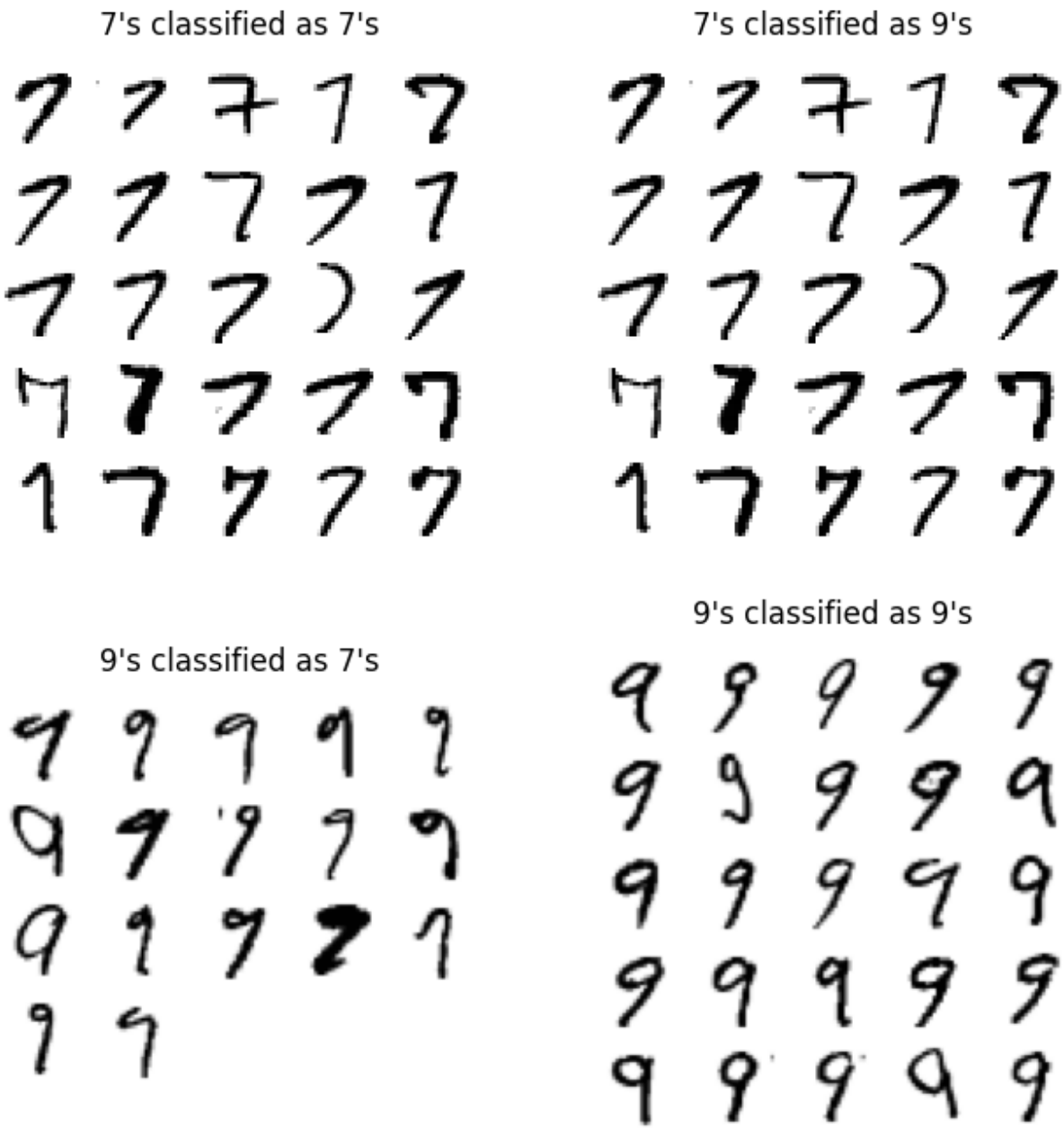

Figure 15: Examples of times the model properly and improperly assigned values for the training images from the final model before input data processing

5.5 Input Data Reduction Using PCA

In choosing the pixels containing 95 percent of the variance in the training images, the input data was reduced from 784 to 154 pixels to reduce the processing needed to run the model. The randomizers of both NumPy and TensorFlow were

seeded with the same inputs to be able to directly compare the models. The model reduced with PCA took 41 seconds to run through ten epochs, while the model without reduction took 102 seconds to run. The results from each of the two runs compared with each other can be seen in the Table 1. The training and validation loss plots for each method can be seen in Figure 16 and the training and validation accuracy can be seen in Figure 17.

Method	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**	**Testing Accuracy**	**Testing Loss**
Non-Reduced	0.9973	0.0116	0.9806	0.0987	0.9779	0.1086
PCA Adjusted	0.9987	0.0073	0.9788	0.09218	0.9781	0.9613

Table 1: Model comparison between PCA reduced model and non-reduced model using mish activation function and 256 nodes

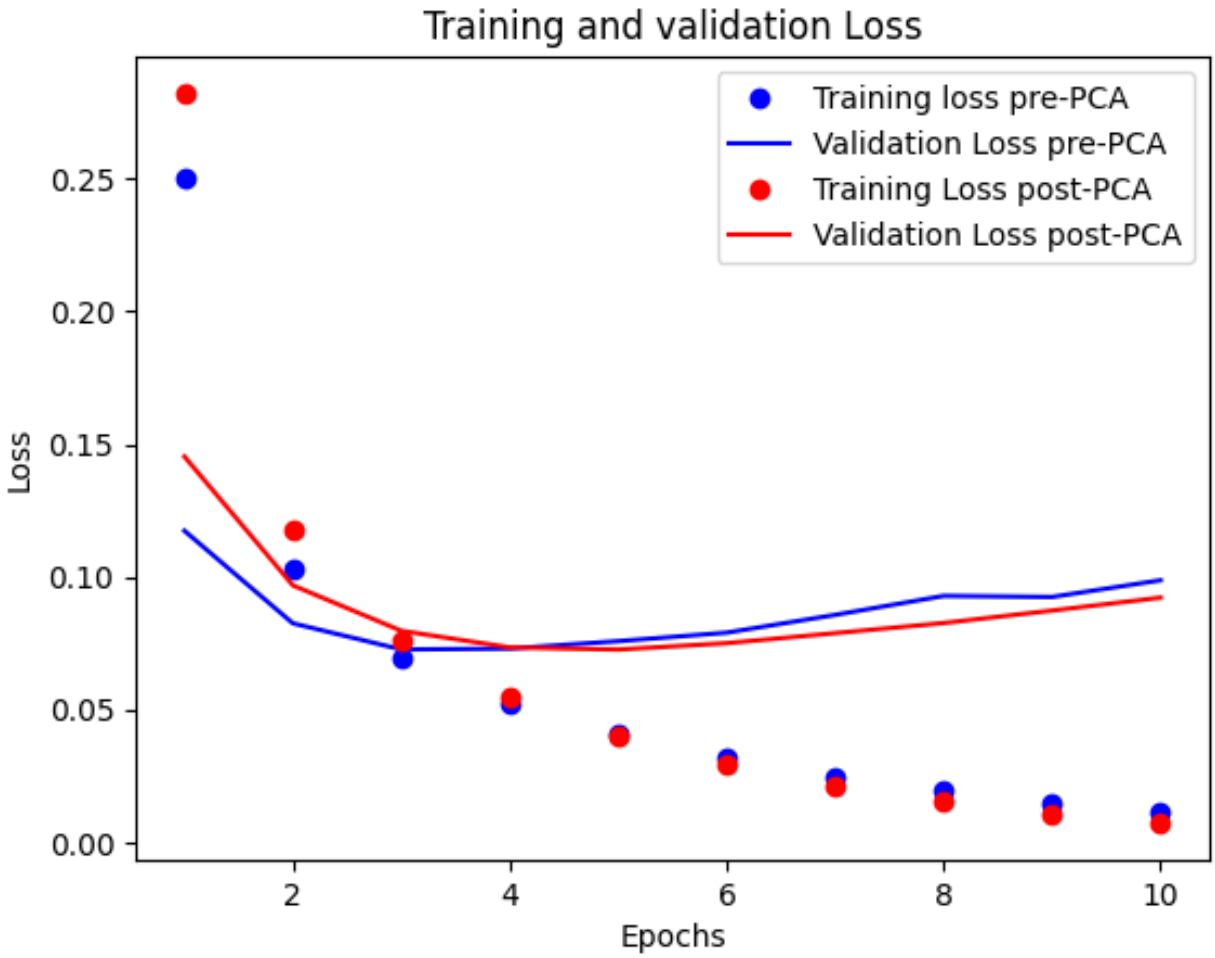

Figure 16: Training and validation loss before and after reduction using PCA for the mish activation function model with 256 nodes

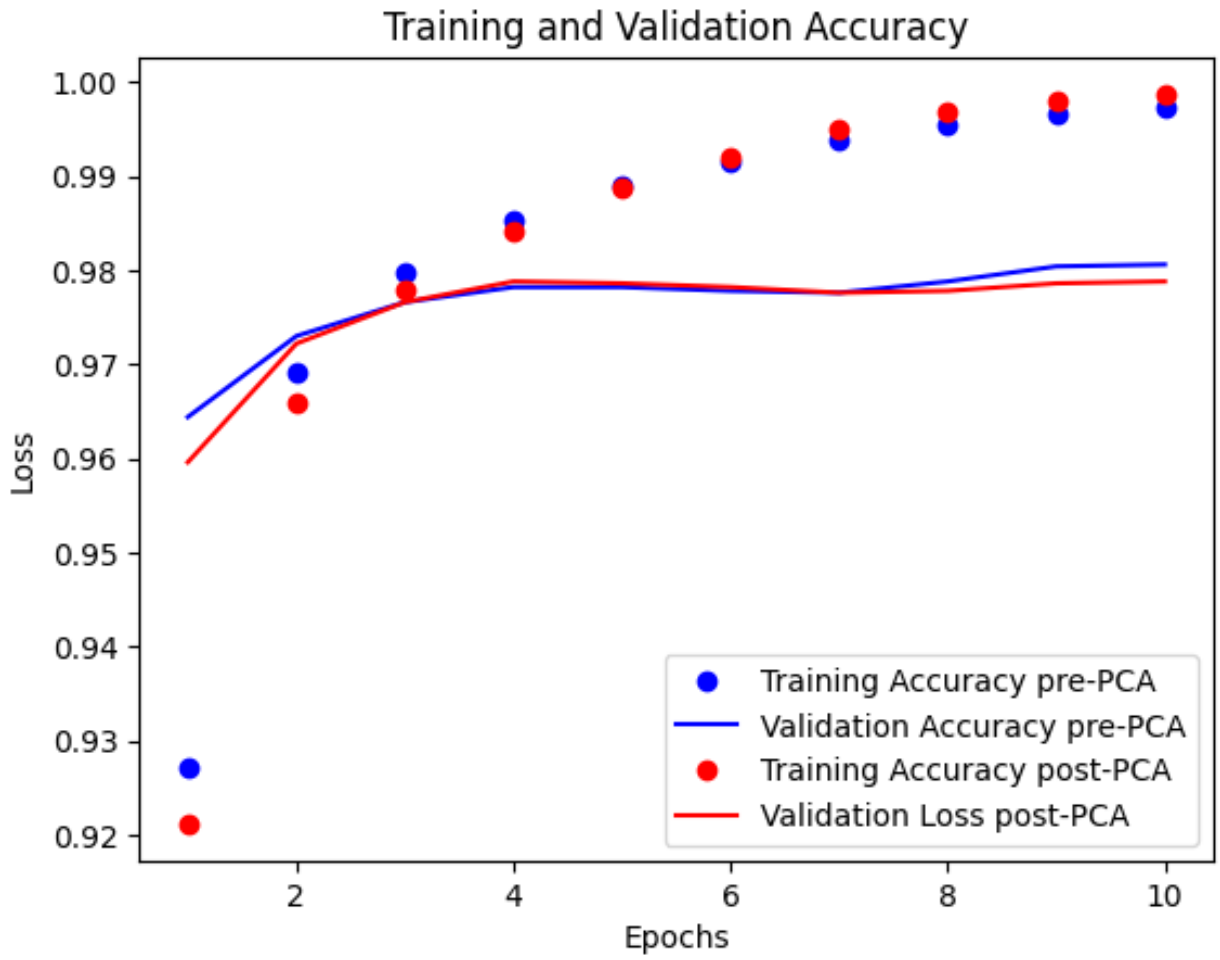

Figure 17: Training and validation accuracy before and after using PCA reduction for the mish activation function model with 256 nodes

5.6 Input Data Reduction Using Random Forest Decision Trees

Similarly to section 5.5, this section explored using random forest decision trees. Random forest decision trees were used to select the 70 most important pixels from the training images and used those pixels as the input data for the model. The processing time of the random forest model took 46 seconds to run 10 epochs while the non-reduced model took 110 seconds to run 10 epochs. A hot spot image of the most important pixels can be seen in Figure 18 and an example of the chosen pixel locations overlaid on an example training image is shown in Figure 19. The model was compared

to the best non-reduced mish activation function model with 256 nodes. The data comparing the two models can be seen in Table 2 and the training and validation loss and accuracy can be seen in Figures 20 and 21 respectively.

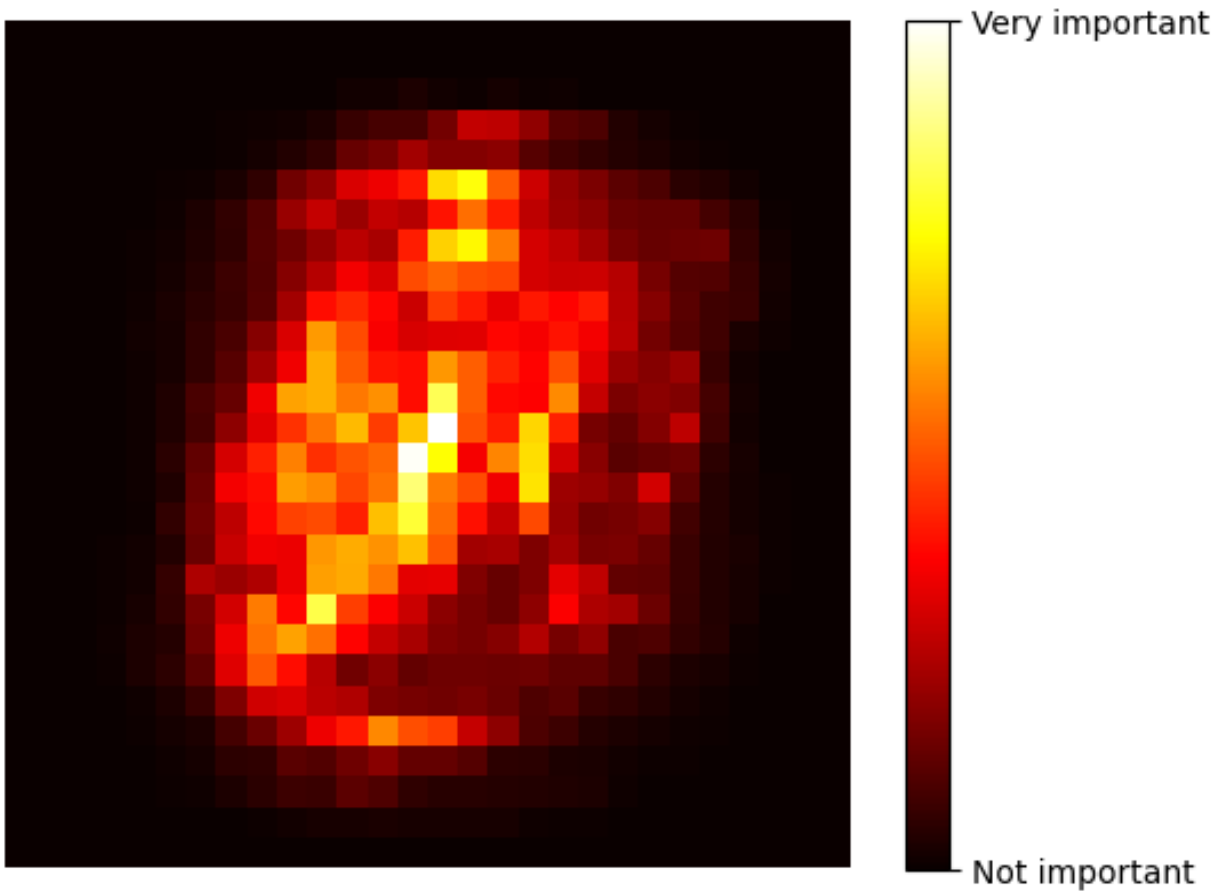

Figure 18: A hot spot map of the most important pixels from the training images using random forest decision trees

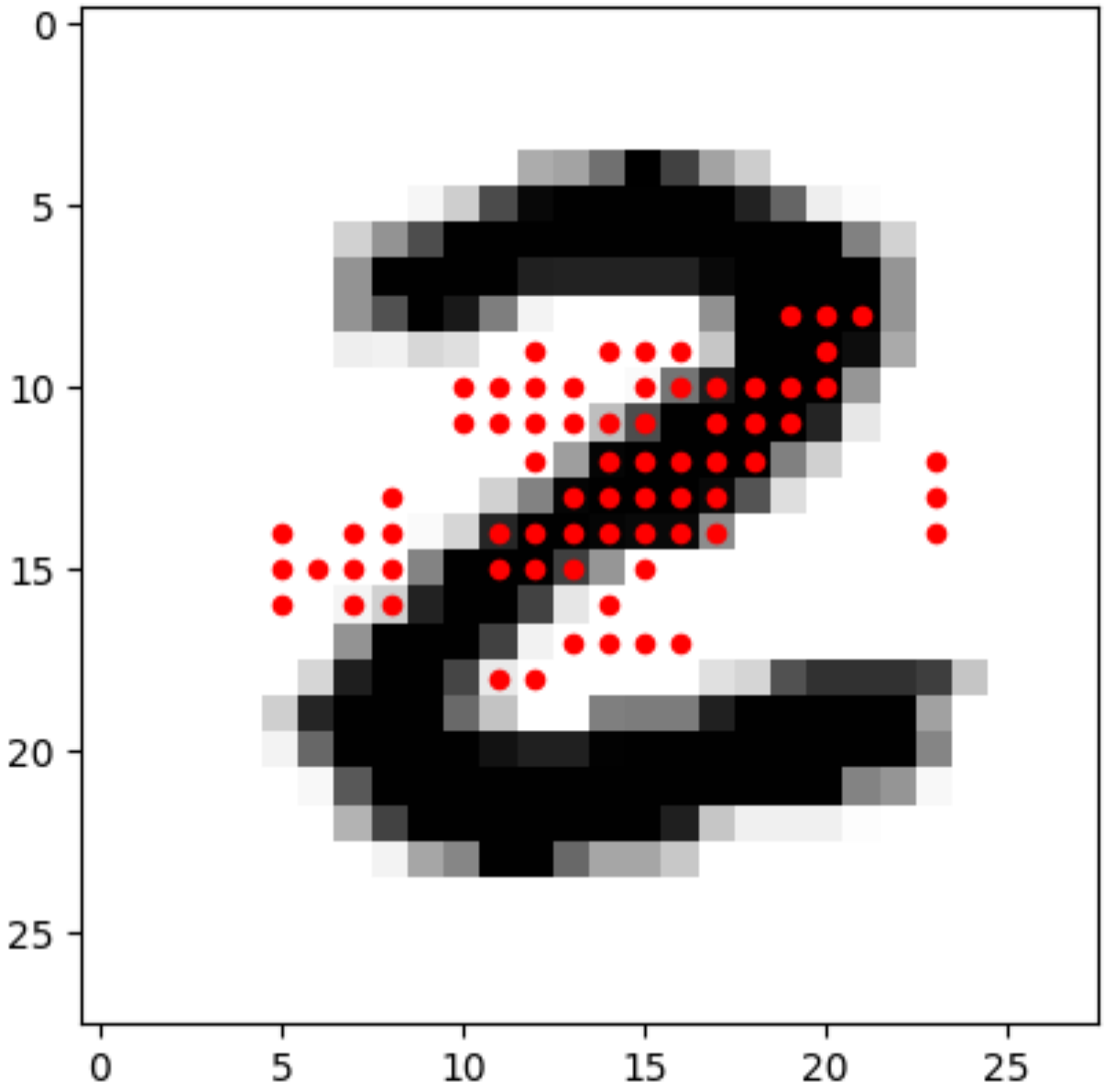

Figure 19: The 70 most important pixels chosen by a random forest decision tree overlaid on a training image

Method	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Testing Accuracy	Testing Loss
Non-Reduced	0.9973	0.0116	0.9794	0.1046	0.9785	0.0978
RF Adjusted	0.9497	0.1665	0.9428	0.1931	0.9418	0.1920

Table 2: Model comparison between PCA reduced model and non-reduced model using mish activation function and 256 nodes

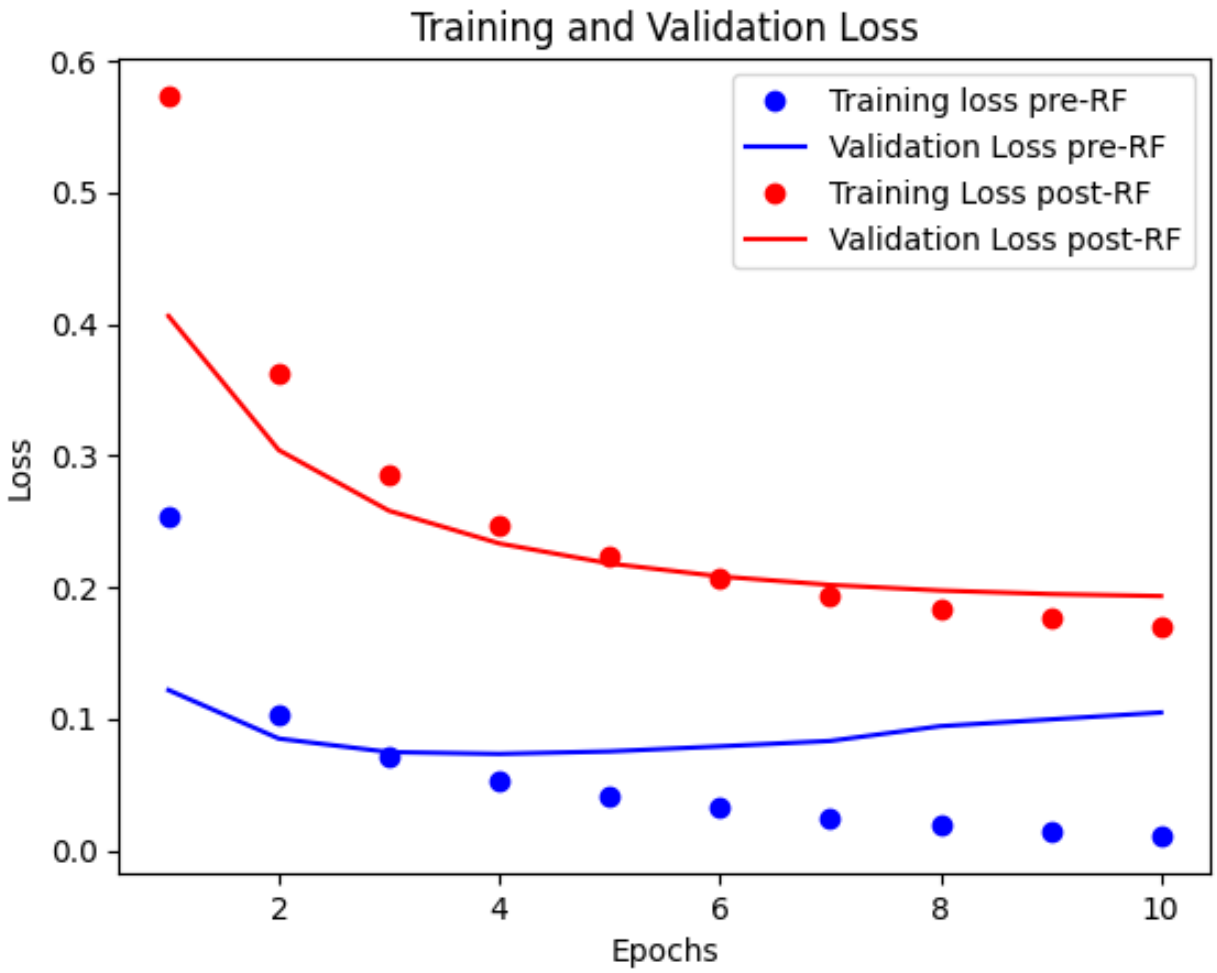

Figure 20: Training and validation loss for the mish activation model with 256 hidden nodes with and without random forest pre-processing

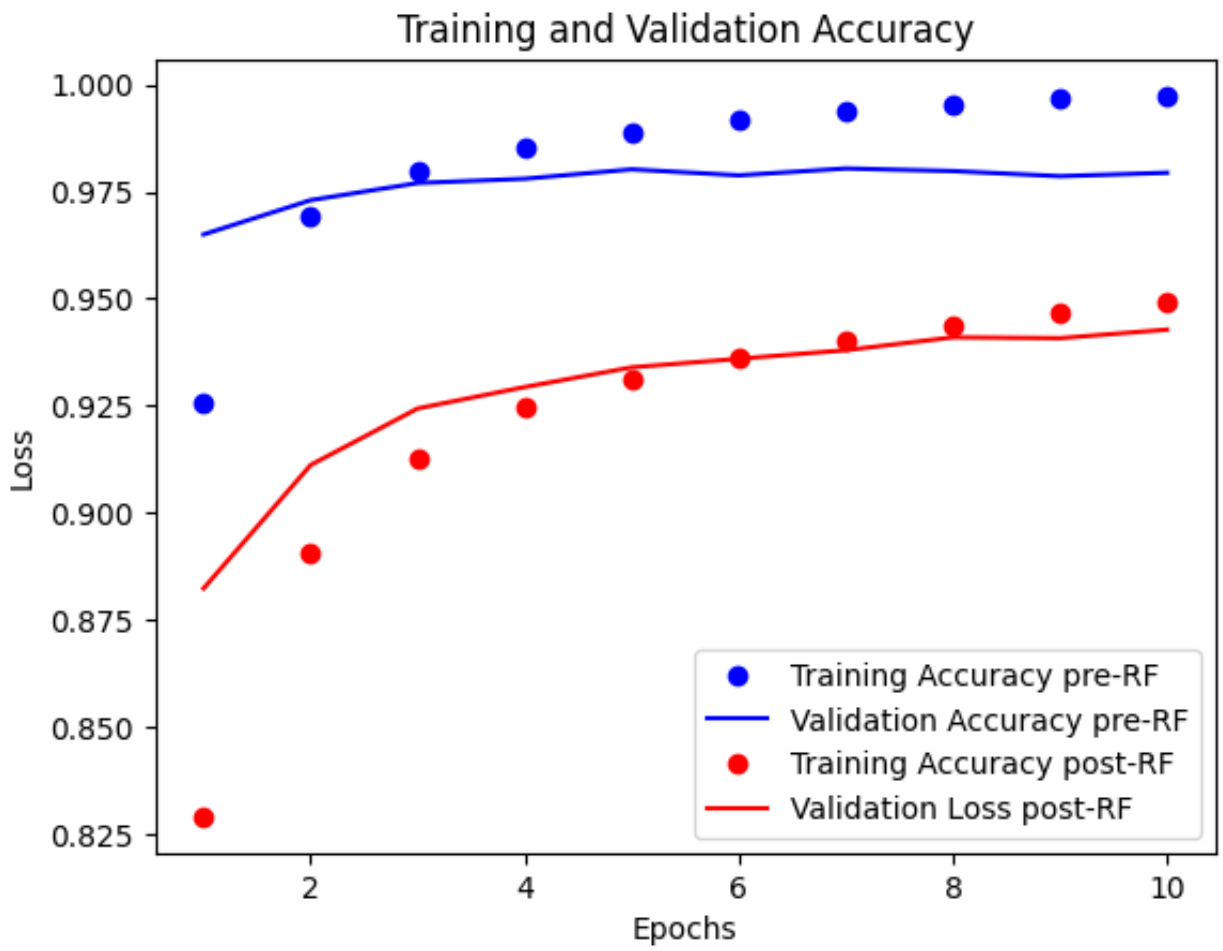

Figure 21: Training and validation accuracy for the mish activation model with 256 hidden nodes with and without random forest pre-processing

6. Conclusions

Dense neural networks provide a powerful tool for non-linear problem solving and producing models using labeled input data to make output predictions. For the given problem of identifying handwritten digits, with ten categories of output and a pretty evenly distributed population, random guessing of the true label of the image would result in a probability of around 0.10 in selecting images. Using a dense neural network optimized through repeated experimentation was able to achieve a testing accuracy probability of 0.978 with the best models. Several different activation functions within the

hidden layer provided similar results among otherwise identical models, and with 23 different available activation functions to work with convolutional neural networks using TensorFlow and Keras can help model a myriad of different scenarios.

Both PCA and RF decision trees reduced the processing time while still performing well, though the PCA reduction using 154 of the pixels from the image performed better, produced near-identical results to the model utilizing input data from all 784 image pixels. The reduced processing time from lowering the size of the input data show the importance of optimizing input to produce more time and energy efficient models.

One issue consistently seen with increasing either the node size or epochs for a run is the issue of overfitting. An example can be seen in Figure 19, where the training and validation curves diverge as the model fits too specifically to the training data and becomes less robust for measuring the validation data. This should ideally be optimized with an early stopping monitor, and further study with this could slightly improve these models. This project showed that even a single layer dense neural network is accurate in predicting handwritten digits, and further work should be done studying multiple layers to see if a more optimal model can be efficiently created.

Appendix A

Experiment 2 Figures

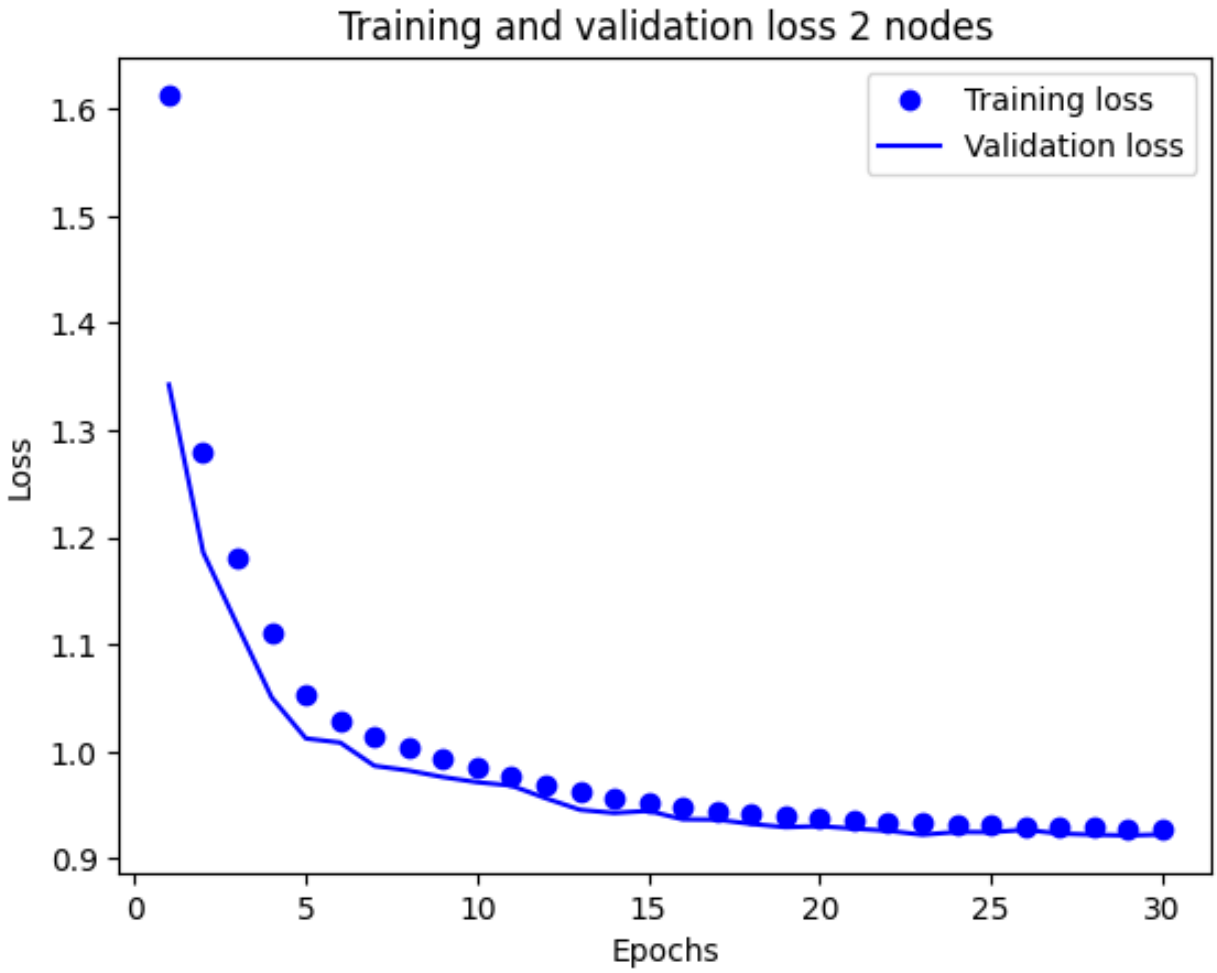

Figure A.1: Training and validation loss for the model with 2 hidden nodes

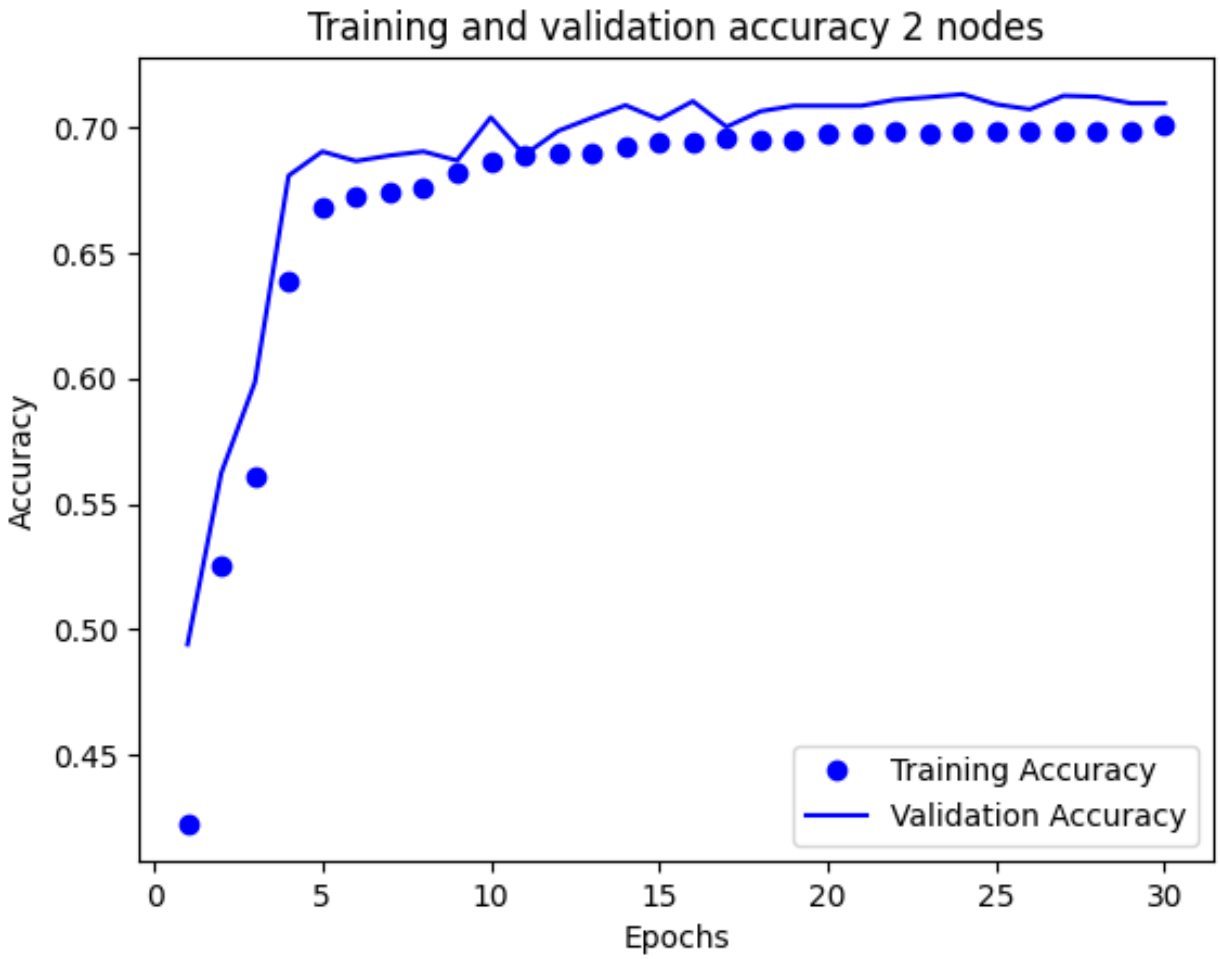

Figure A.2: Training and validation accuracy for the model with 2 hidden nodes

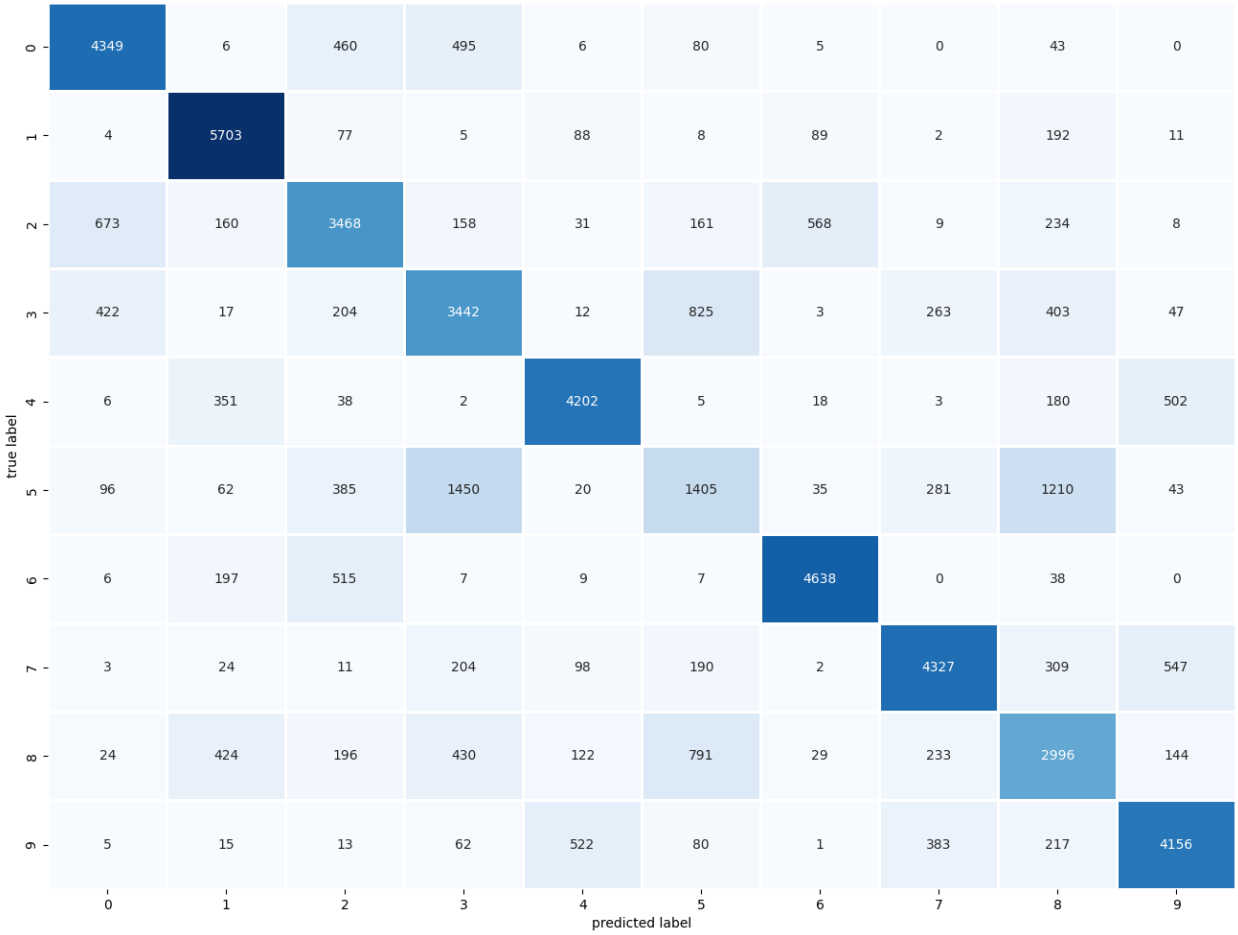

Figure A.3: Confusion matrix for model with two hidden nodes

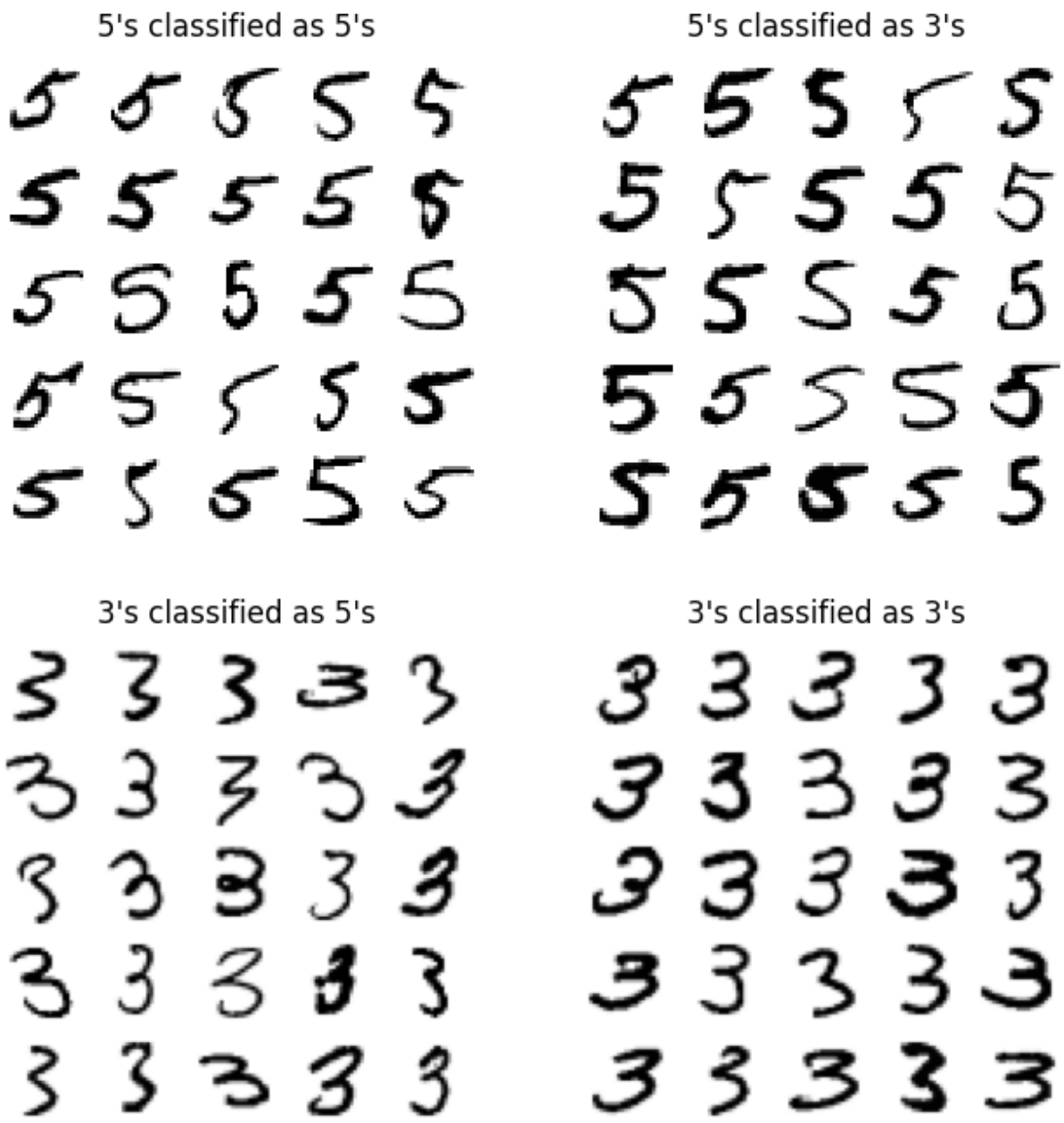

Figure A.4: Training images from model 2 where 3s and 5s are correctly and incorrectly classified

Appendix B

Activation Function Exploration Figures

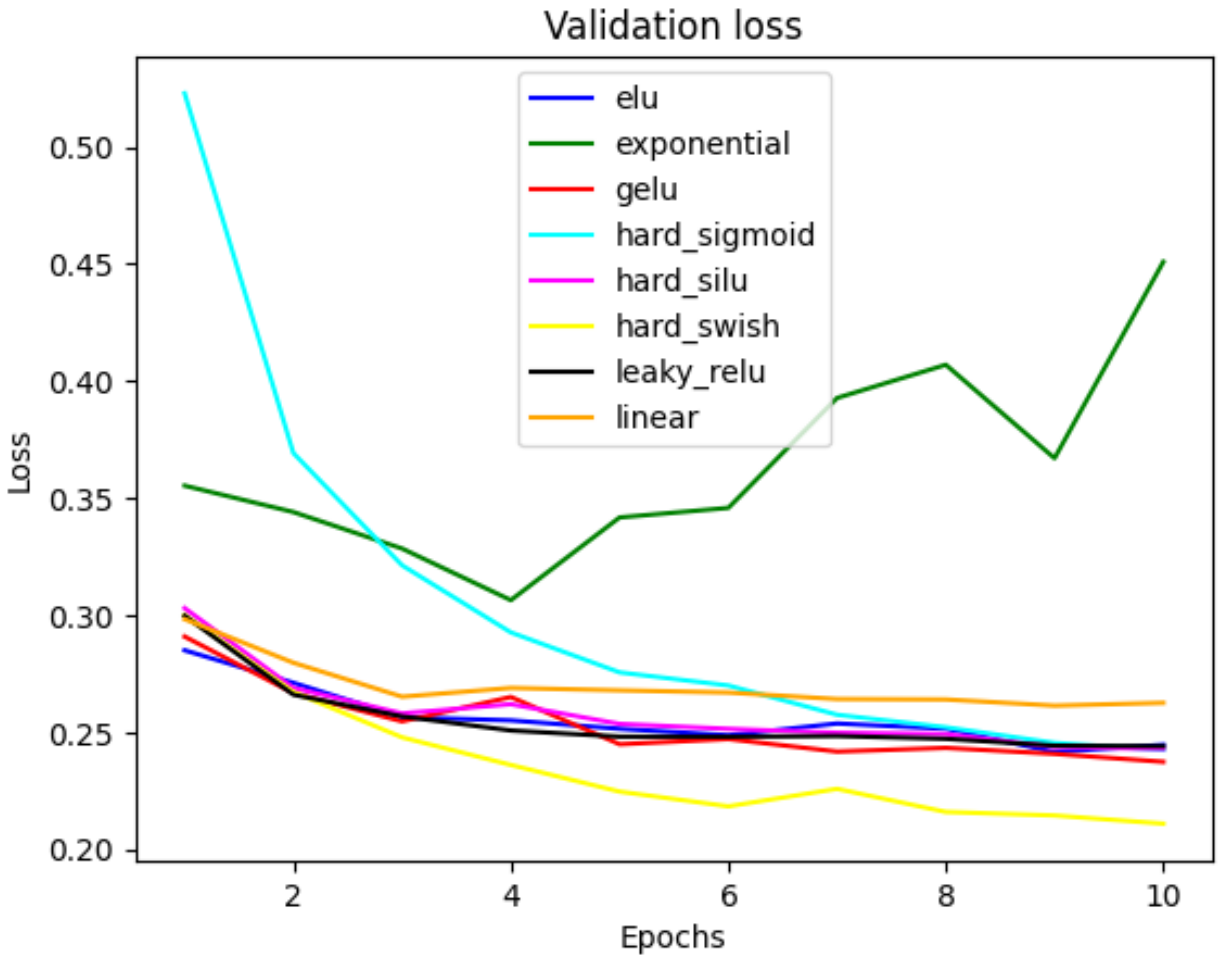

Figure B.1: Validation loss plot from the first set of activation functions

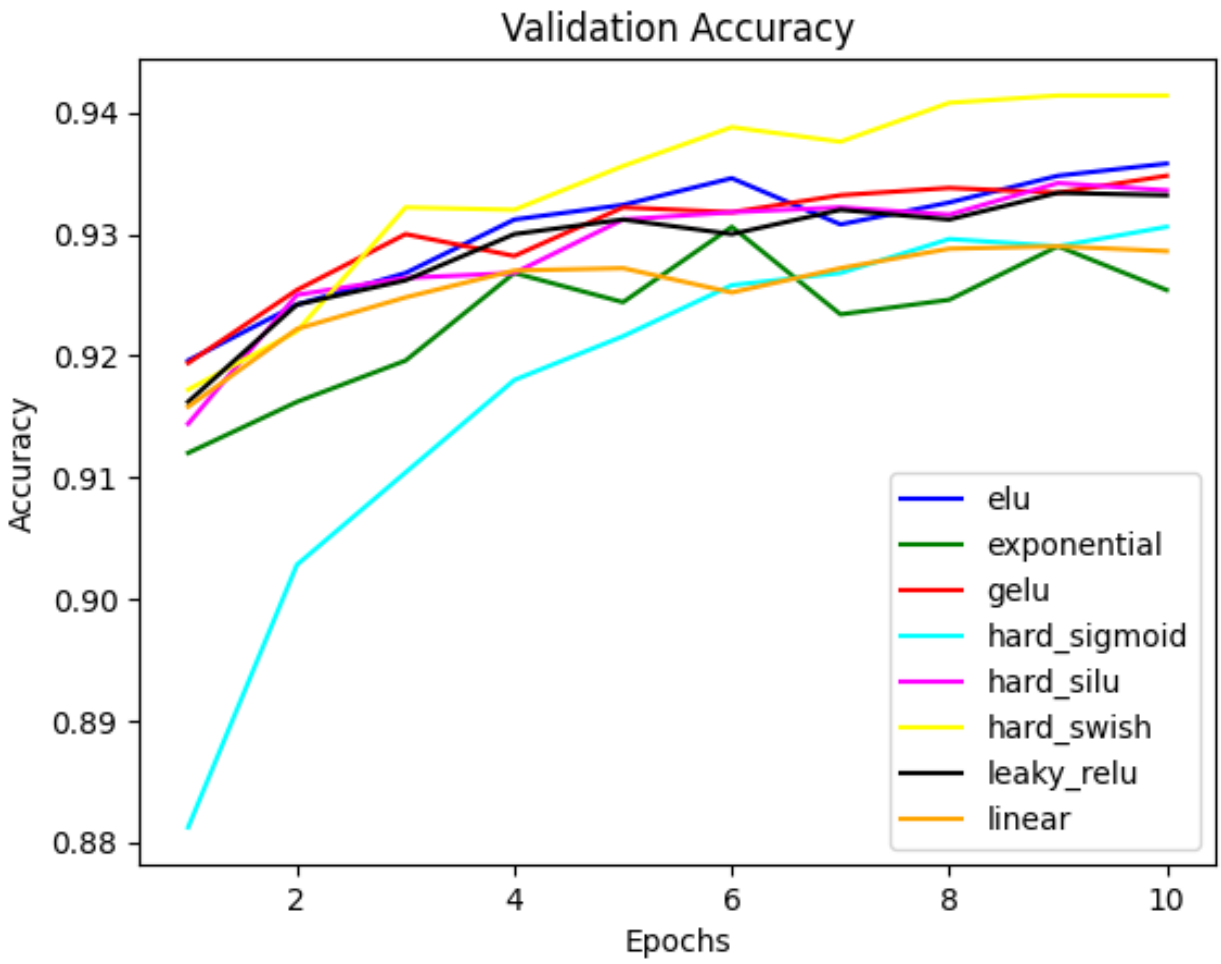

Figure B.2: Validation accuracy plot from the first set of activation functions

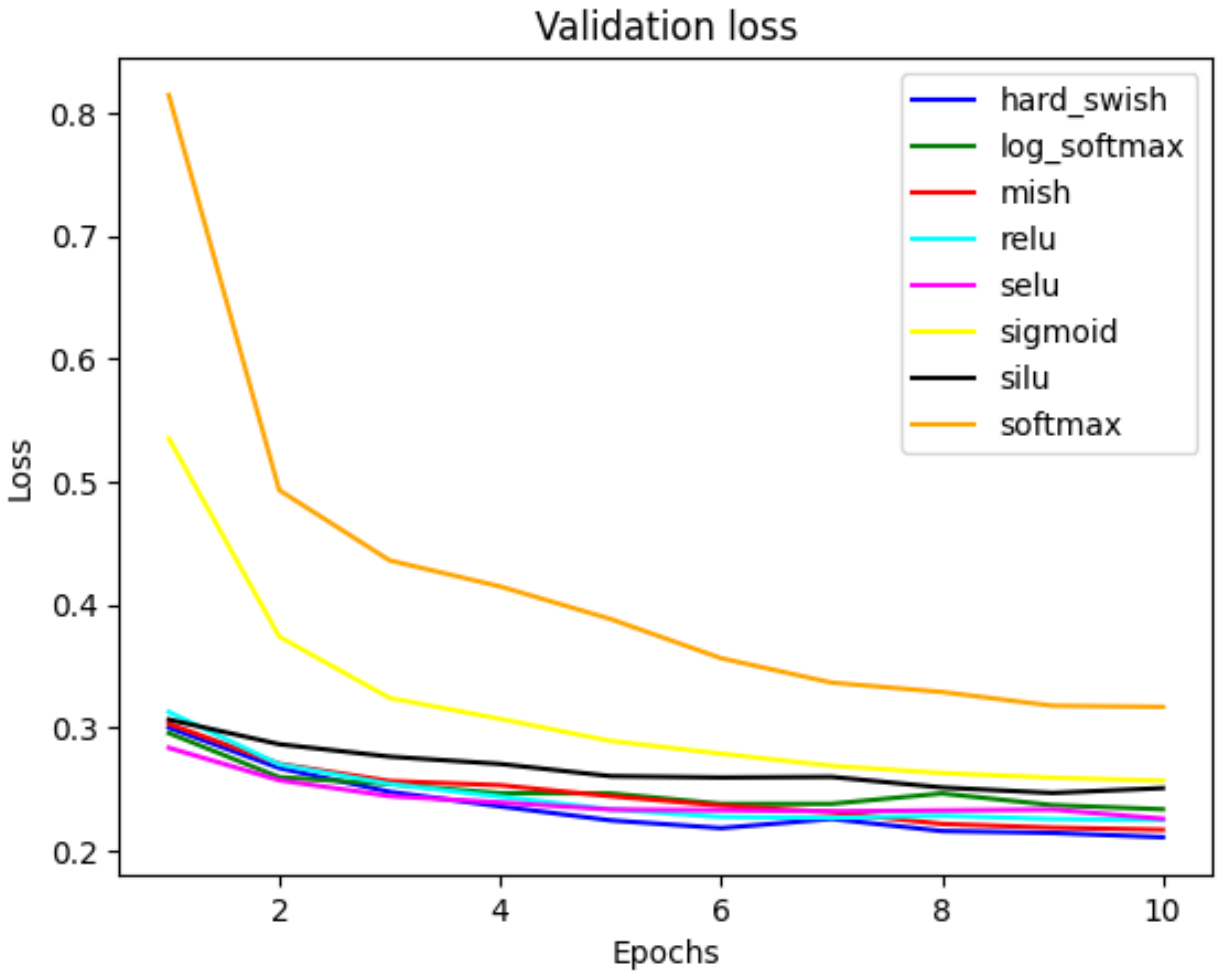

Figure B.3: Validation loss plot from the second set of activation functions

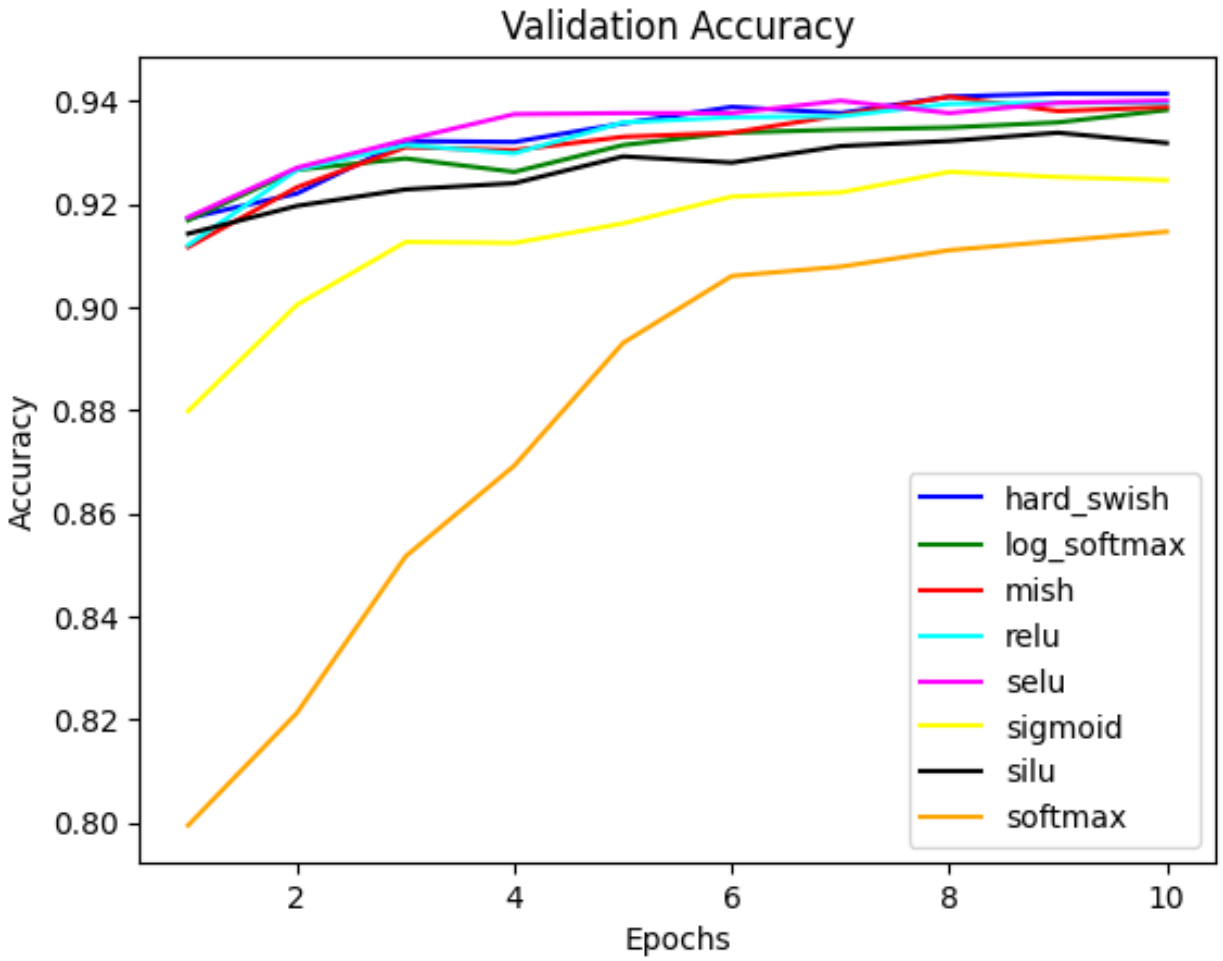

Figure B.4: Validation accuracy plot from the second set of activation functions

References

Agrawal, Yash, Rahul Meena, Himanshu Malviya, Srinidhi Balasubramanian, Rohail Alam, and Rohini P. 2024. "Optical Character Recognition using Convolutional Neural Networks for Ashokan Brahmi Inscriptions." https://arxiv.org/pdf/2501.01981.

Boesch, Gaudenz. 2023. "Optical Character Recognition (OCR) – The 2024 Guide." November 23, 2023. *Viso.ai.* https://viso.ai/computer-vision/optical-character-recognition-ocr/.

Chen, Feiyang, Ziqian Luo, Nan Chen, Hanyang Mao, Hanlin Hu, Ying Jiang, Xueting Pan, Huitao Zhang. 2024. "Assessing Four Neural Networks on Handwritten Digit Recognition Dataset (MNIST)." *Journal of Computer Science Research* 6 (3): 17-22.

De Gruyter, 2023. *Machine Learning Under Resource Constraints. Fundamentals.* Berlin; Boston: 1-12.

Eikvil, Line. 1993. "Optical Character Recognition." Norsk Regnesentral, P.B.: 5-10.

Gopalakrishnan, Ramamurthi. 2022. "A Simple Nueral Network on MNIST dataset using Pytorch." *Medium.* July 3, 2022. https://medium.com/@ramamurthi96/a-simple-neural-network-model-for-mnist-using-pytorch-4b8b148ecbdc.

Keras. n.d. "Layer Activation Functions." Accessed January 24, 2025. https://keras.io/api/layers/activations/.

Khan, Azim. 2024. "A Beginner's Guide to Deep Learning with MNIST Dataset." *Medium.* April 16, 2024. https://medium.com/@azimkhan8018/a-beginners-guide-to-deep-learning-with-mnist-dataset-0894f7183344.

LeCun, Yann. n.d. "The MNIST Database of Handwritten Digits." Archived March 31, 2022. https://web.archive.org/web/20220331130319/https://yann.lecun.com/exdb/mnist/.

Ombaval. 2024. "Building a Simple Neural Network from Scratch for MNIST Digit Recognition without using TensorFlow/PyTorch only using Numpy." *Medium.* May 14, 2024. https://medium.com/@ombaval/building-a-simple-neural-network-from-scratch-for-mnist-digit-recognition-without-using-7005a7733418.

Shanmugamani, Rajalingappaa. 2018. *Deep Learning for computer vision: expert techniquest to train advanced neural networks using TensorFlow and Keras.* Birmingham, England: Paths International Ltd. 2018 (1): Ch. 1. https://learning.oreilly.com/library/view/deep-learning-for/9781788295628/78521051-5153-4a75-b149-5219677cdd70.xhtml

Tamanna. 2023. “Exploring Convolutional Neural Networks: Architecture, Steps, Use Cases, and Pros and Cons.” *Medium*. April 24, 2023. https://medium.com/@tam.tamanna18/exploring-convolutional-neural-networks-architecture-steps-use-cases-and-pros-and-cons-b0d3b7d46c71.

TensorFlow. n.d. “Module: tf.keras.activations.” Accessed January 24, 2025. TensorFlow documentation v2.16.1. https://www.tensorflow.org/api_docs/python/tf/keras/activations.

TensorFlow. n.d. “tf.keras.callbacks.EarlyStopping.” Accessed January 24, 2025. TensorFlow documentation v2.16.1. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping.