# Project Nature and Content - Computer Vision

*Sonny Desai*

January 2020

**Abstract**

This project deals with creating a neural network and five experiments which are conducted on the MNIST data set. Additionally, I examine alternative neural net structures with a simple, single-hidden layer network, and learn how to fit a neural network directly in Python. The five experiments are: 1) we group the 60,000 activation values of the hidden node for the (original) set of training images by the 10 predicted classes and visualize these sets of values, 2) For each of the 60,000 images, the output of the two hidden nodes are plotted using a scatterplot. We color code the points according to which of the 10 classes the output of the two nodes predicts, 3) explore with more hidden nodes, 4) Use PCA decomposition to reduce the number of dimensions of our training set of 28x28 dimensional MNIST images from 784 to 154, 5) use a Random Forest classifier to get the relative importance of the 784 features (pixels) of the 28x28 dimensional images in training set of MNIST images and select the top 70 features (pixels).

**Introduction**

Artificial neural networks have been around for nearly a century but have been further discussed lately as artificial intelligence and machine learning has become more and more prominent. Neural networks are based upon how human brains function and go through decisions and data similar to how a human brain would. A “neuron” in a neural network is a mathematical function that collects and classifies information according to a specific architecture. A neural network contains layers of interconnected nodes. Each node is a

perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. Hidden layers fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis. The main purpose of this research problem is to use the aforementioned five experiments to find the hidden layer in the MNIST data set. There is only one hidden layer, but through the five experiments, I will explore the designing, training, and assessing a neural network, and interpreting the impact of hyperparameters. The MNIST database or Modified National Institute of Standards and Technology database, is a large database of handwritten digits that is commonly used for training various image processing systems. It was created by "re-mixing" the samples from NIST's original datasets. The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

**Literature Review**

While researching I was able to find one other scholarly article related to our experiments. "Digit Identifcation using Deep Neural Networks in TensorFlow" by Pankaj Kumar covers the authors own efforts to create a DNN with the MNIST dataset. He simarlarily split his

data into test and training datasets with 10,000 and 60,000 images used respectively. His experiment had two hidden layers, compared to our 1 (even though we eventually did a prediction with two hidden layers. For his DNN he used a Mini-Batch Gradient Descent for our training which means instead of sending whole data, we will be sending mini batches of data and will learn or update our model weights on them. His predictions were similar to ours as well as they were very accurate. He doesn't provide his accuracy rate, but from viewing his results, they looked almost 90% accurate. He notes that there is future work to do on his model, an although he did not cover all the elements of our different experiments, he had a similar result to our model predictions!

**Methodology**

As mentioned previously, our data comes from the MNIST dataset which was imported from the keras open source library in python. Following the code provided by Paul, we then moved to the pre-processing of the data, which was done before the creation and testing of the models. We break out the data into test and training datasets. First we converted the labels (integers 0 to 9) to 1D numpy arrays of shape (10,) with elements 0s and 1s with all the elements set to 0 except the one which the label belongs to - which will be set to 1. We did this with one-hot encoding, which we use when there are categorical variables where no ordinal relationship exists. allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories). In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer

value. Next, we reshape the images from 2D arrays of shape (28,28) to 1D float32 arrays of shape (784,) and then rescale their elements to values between 0 and 1. Lastly, we scale the data so that they are both normalized.

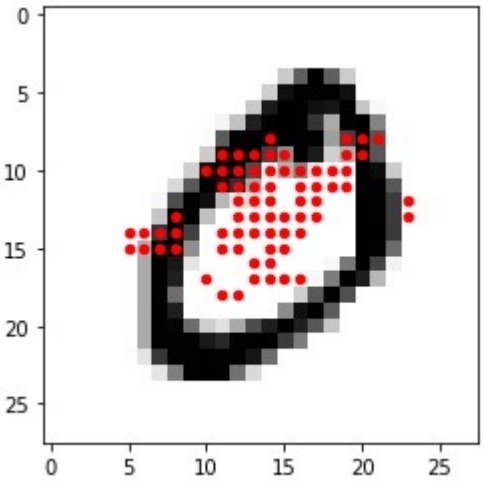
The models we use to experiment on the data and find the hidden layer are the DNN, a PCA (Principal Components Analysis), and a Random Forest Classifier. A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. We used a Sequential class defined in Keras to create our model. All the layers are Dense layers which means all the nodes of a layer would be connected to all the nodes of the preceding layer. We used stochastic gradient descent function and defined a loss faction. The loss needs to be minimized in order to have a higher model accuracy. We used the accuracy during training as a metric to keep track of as the model trains. In order to ensure that this is not a simple "memorization" by the machine, we evaluated the model on a test set.

Finally, we created a Random Forest Classifier (with the default 100 trees) and used it to find the relative importance of the 784 features (pixels) in the training set. We produced a heat map to visual the relative importance of the features. The last step was to select the 70 most important feature (pixels) from the training, validation and test images to test our 'best' model on.

**Computational Experiment and Results**

The DNN model was very accurate, especially after all of our data pre-processing, and ended up with an accuracy score of 97.65%. The confusion matrix that was created using the DNN predictions also showed positive results with only 28 fours that were misclassified as nines (and 10 nines were classified fours). After the PCA which successfully reduced dimensions

down from 128 to 3, we used the Random Forest classifier to find the top 70 pixels from the training set, and when we visualize the last 70 we get this image:




:

**Discussion and Conclusions**

The DNN model was very successful as mentioned above, likely due to the heavy pre-processing of the data. For experiment 1, we were successfully able to break out the 60,000 images to training leaving 10,000 for the test. When viewing the boxplot of the 60,000 values, we expected to see minimal overlap between the range of values in the boxes, and this was the case for the 10 nodes we had split out. There was some variance in the size of the boxes with 3 and 5 being particularly spread out, however, there was only minimal overlap. For experiment 2, we created a scatterplot of 784 input nodes, 2 hidden layer nodes and 10 output nodes. The scatterplot shows the ten nodes and the 2 hidden layers spread out pretty evenly with the vast majority of the inputs coming at zero. There is some overlap especially around 1 with node 6, but it is still minimal which is a good sign for the model! All the other nodes are evenly spread out. For experiment 3, we furthered the analysis from the previous two experiments and decided to use a PCA and random forest classifier to limit some of the dimensions. Experiment

4 showed that the PCA was successful in limiting the amount of dimensions and was able to make future predictions on one, two, and three hidden nodes. The difference wasn't too great for the results of the different predictions with one and 2 hidden nodes. However, the t-SNE prediction created quite a few clusters that were more separated than the previous two examples. Lastly, for experiment 5, we found the top 70 pixels after running the random forest classifier. We used the classifier to first find the importance of the 784 features and then picked the top 70 from both the training and test datasets. This was a successful operation of the five experiments and concluded with interesting results. We were finally left with different predictions and a successful DNN model. By using PCA we could better find the hidden layers and even use it to predict other hidden layers.