# ECE 532 Final Project

Timothy Winfree

December 2020

## 1 Introduction

For this project, I will be classifying the MNIST dataset located here: `http://yann.lecun.com/exdb/mnist/`. The MNIST dataset is a commonly used benchmark dataset used for evaluating the performance of image classifiers. This dataset contains 70,000 20x20 binary images of handwritten digits 0-9. The authors of the dataset have segregated the images into a training bank of 60,000 images, and a test bank of 10,000 images. Each image is labeled with the integer corresponding to the handwritten digit. The classification task at hand is to create an algorithm that takes an image of a handwritten digit as an input, and outputs the corresponding integer as a label. In the following report, I document three methods of implementing such a classifier, and discuss each method's strengths and weaknesses.

Figure 1: Example images of handwritten digits from the MNIST dataset

## 2 Methods and Results

### 2.1 Pre-Processing

One approach to this problem is to do no pre processing and simply unwrap each column of each image and stack them into a feature vector. Since every digit of the same class is drawn differently, the relationship between the pixel values and the correct label will be weak. To provide a more meaningful set of features, I want to extract structures from the images, whos existence and location in the image have a strong connection to a subset of the classes. To accomplish this I convolved each image with 20 pattern matching filters. The output of each of these filters is subsampled to 8x8, then unwrapped and concatenated into a feature vector. A feature vector is produced in this manner for each image and then stored as a row in a matrix X. Thus, each image is described by a feature vector containing $8 * 8 * 20 = 1280$ elements. There is surely redundancy in using this many features. To trim redundant features, I first subtracted from each column its mean, then divided each column by the magnitude of its max value. Using the singular value decomposition, I projected X onto a 112 dimensional subspace defined by the 112 principal components of X with the highest singular values.

### 2.2 k-Nearest Neighbors

The K-nearest neighbors algorithm is simple to implement, and a natural multiclass classifier. To compute the label of a novel data point with feature vector $\hat{x}$, I compute the euclidean distance between $\hat{x}$ and each

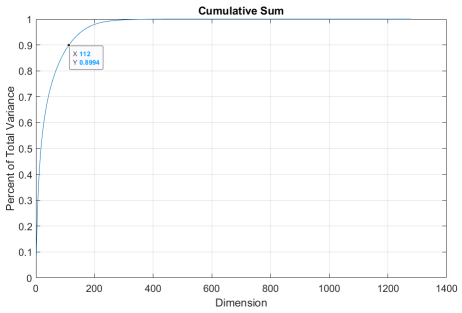


Figure 2: Cumulative sum of singular values normalized to sum of all singular values of X. Evidently, 90% of the variance in X can be captured by the first 112 principle components. A ten fold reduction in the size of X.

data point in the training data, then predict the label of $\hat{x}$ as the mode of the k nearest labels. If there is a tie I apply a weight function $\frac{1}{d}$ to each vote to break the tie, where $d$ is the distance of the voting data point to $\hat{x}$. I used cross validation to determine the best values for $k$ and the dimensionality $n$. Using $k = 3$ and $n = 50$, I classified the test bank of 10,000 images with an accuracy of 96.8%.

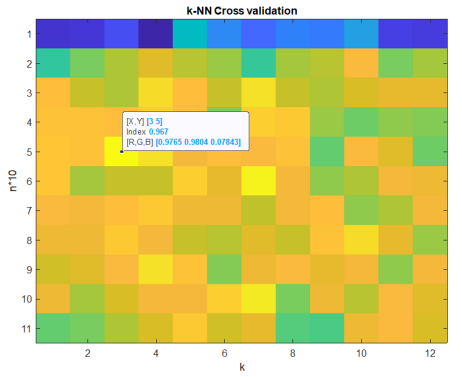


Figure 3: Randomly selecting 5 sets of 50,000 training and 10,000 validation images, I computed the average accuracy of the knn algorithm for the shown values of k and n.

## 2.3 One vs. One Binary Least Squares Classifiers

In order to use least squares for multiclass classification, I must split the problem up into a series of 45 binary classification problems, one for each way of choosing two classes from 10. To do this, I split the training data into 45 sets, each containing only data from two classes (note: I added a bias to each sample for this method). For each of the 45 sets, I trained a binary least squares classifier (BLSC) to distinguish between the two classes. Prediction of a novel data point with feature vector $\hat{x}$ takes place as follows. For each of the 45 classifiers, I compute $sign(\hat{x}^T \cdot w)$, the class assigned to this output (either 1 or -1) receives a vote. The class with the most votes is chosen as the prediction. If there is a tie, classifiers that involve only classes involved in the tie are used to decide the vote, e.g., if there is a tie between 4 and 9, the winner of the 4 vs. 9 classifier is chosen as the prediction. I performed cross validation separately for each of the 45 classifiers using both lasso and ridge. As shown in figures 4-5, none of the classifiers benefit from the presence of either regularizer. As such, I trained each classifier on the 60,000 training images with no regularizer and classified the 10,000 test images with an accuracy of 94.2%.

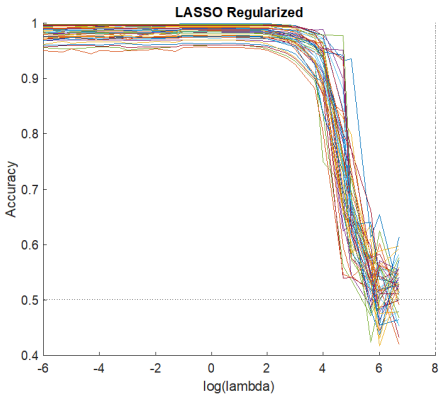


Figure 4: Accuracy vs log(lambda) using ridge regularization. Each line represents one of 45 binary classifiers

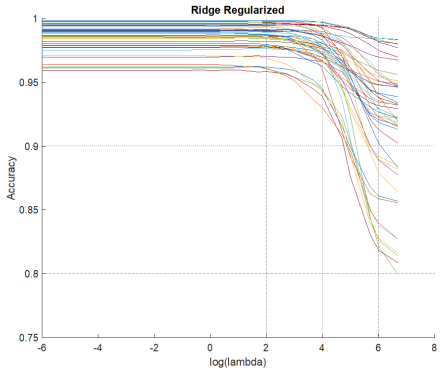


Figure 5: Accuracy vs log(lambda) using LASSO regularization. Each line represents one of 45 binary classifiers

## 2.4 Convolutional Neural Networks

### 2.4.1 Overview

Convolutional neural networks work similarly to traditional neural networks with a couple of key differences. In the standard multilayered perceptron (MLP), each output is connected to each input and each input-output connection has a unique weight. On the other hand, a convolutional neural network works by sliding a window of weights (aka filter) along an input image and computing the inner product between the window and the overlapping region of the input. This process is called the convolution, hence the name. Therefore, in contrast to the MLP, each output is only connected to a small portion of the input, called the receptive field, and each set of weights (filter) is shared between each receptive field. This design difference means that a CNN will have far fewer weights to optimize than a MLP. As such, the CNN functions as a form of dimensionality reduction for neural networks.

Machine learning algorithms benefit from dimensionality reduction when a significant portion of the input contains redundant or useless information. This is often the case for image classification tasks as we are only interested in a particular object in the image, all pixels forming the background are thus of no use. Put simply, the CNN allows us to efficiently train small filters to extract specific patterns (e.g. a line, curve etc.) from the image without wasting time processing useless information.

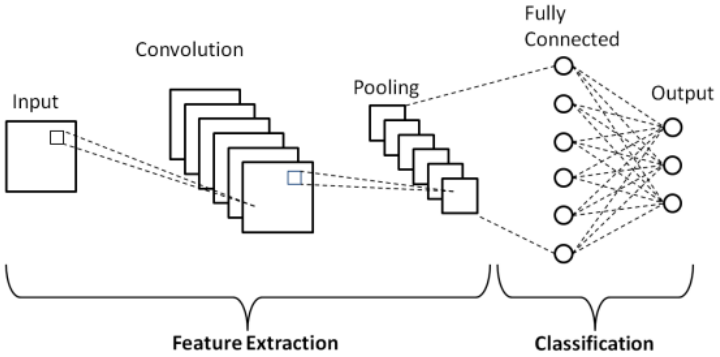


Figure 6: A diagram of a CNN with one C-Layer and one dense layer. In the C-layer, each filter is trained to extract a single pattern from the input. Information extracted via the C layers is synthesized into a prediction in the dense/classification layer. Citation:A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets

| Cross Validation | | | | | | |
|---|---|---|---|---|---|---|
| Num Layers | Filter Size | Num Filters (first Layer | Epochs | Learn Rate | Validation Accuracy | time |
| 3 | 5 | 20 | 4 | 0.01 | 99.11 | 4:30 |
| 3 | 5 | 20 | 20 | 0.001 | 98.92 | 22:51 |
| 3 | 5 | 20 | 20 | 0.01 | 99.17 | 24:45' |
| 3 | 5 | 20 | 20 | 0.1 | 98.92 | 24:09' |
| 3 | 5 | 16 | 4 | 0.01 | 98.97 | 3:33 |
| 3 | 3 | 15 | 4 | 0.01 | 98.73 | 2:29 |
| 3 | 3 | 20 | 4 | 0.01 | 98.86 | 3:16 |
| 3 | 3 | 10 | 4 | 0.01 | 98.73 | 1:38 |
| 2 | 3 | 30 | 4 | 0.01 | 98.72 | 4:16 |
| 2 | 3 | 15 | 4 | 0.01 | 98.45 | 2:03 |
| 2 | 5 | 10 | 4 | 0.01 | 98.6 | 1:40 |
| 2 | 5 | 20 | 4 | 0.01 | 98.5 | 3:13 |
| 2 | 5 | 30 | 4 | 0.01 | 98.66 | 5:00 |
| 1 | 5 | 60 | 4 | 0.01 | 98.2 | ?? |
| 1 | 5 | 40 | 4 | 0.01 | 98.2 | 4:32 |
| 1 | 5 | 20 | 4 | 0.01 | 98.31 | 1:51 |
| 1 | 5 | 30 | 4 | 0.01 | 98.16 | 2:42 |

Figure 7: Summary of cross validation experiments. Note only the number of filters from the first layer is shown. Because max pooling halves the size of the input, the number of filters always doubles for each subsequent C-Layer

```
>> layers

layers =

  16×1 Layer array with layers:

     1   ''   Image Input             20×20×1 images with 'zerocenter' normalization
     2   ''   Convolution             20 5×5 convolutions with stride [1  1] and padding 'same'
     3   ''   Batch Normalization     Batch normalization
     4   ''   ReLU                    ReLU
     5   ''   Max Pooling             2×2 max pooling with stride [2  2] and padding [0  0  0  0]
     6   ''   Convolution             40 5×5 convolutions with stride [1  1] and padding 'same'
     7   ''   Batch Normalization     Batch normalization
     8   ''   ReLU                    ReLU
     9   ''   Max Pooling             2×2 max pooling with stride [2  2] and padding [0  0  0  0]
    10   ''   Convolution             80 5×5 convolutions with stride [1  1] and padding 'same'
    11   ''   Batch Normalization     Batch normalization
    12   ''   ReLU                    ReLU
    13   ''   Max Pooling             2×2 max pooling with stride [2  2] and padding [0  0  0  0]
    14   ''   Fully Connected         10 fully connected layer
    15   ''   Softmax                 softmax
    16   ''   Classification Output   crossentropyex
; >>
```

Figure 8: CNN architecture with highest validation accuracy. 3 C-Layers, 5x5 filter size, 20-40-80 filters per layer. Predicted accuracy = 99.2%

### 2.4.2 CNN Architecture

**Convolution Layer** A convolutional layer convolves a user defined number of filters of user defined size with the input. Then, the output of this operation is normalized, ReLU activated, and downsampled via max pooling. The output of the max pooling operation is then passed on as the input to the next layer.

**Max Pooling** Convolution layers are always followed by max pooling operations. The max pooling operation segments the input image into 2x2 pools. Then, a downsampled output containing only the max value of each pool is constructed. The purpose of this downsampling operation is to provide a degree of shift/skew invariance to the system. To illustrate, consider two realizations of a hand-drawn '7'. Each realization will have the same basic structure, but, even if drawn by the same person, the two realizations will have subtle differences between them. Sufficient downsampling has the effect of smoothing over these small differences, allowing the CNN to more quickly learn the relationships between the values of certain pixels and each of the classes.

**Classification Layer** Information extracted by the convolutional layers is synthesized into a prediction in the classification layer. The classification layer feeds the input from the last convolution layer into a fully connected layer with ten output nodes (one for each digit). The dense layer uses the softmax activation function. The softmax function is the multiclass generalization of the logistic activation function. The $ith$ output of the softmax function can be interpreted as the probability of the $ith$ class being the correct label. As such, when we are classifying an input we choose the class with the maximum softmax output. When training the filters, the cross entropy loss function is used. Cross entropy is defined as $-log(y_c)$, where $y_c$ is the output of the softmax corresponding to the correct label. As such, we obtain zero loss when the softmax output corresponding to the correct class is 1.

## 2.5 Validation and Final Result

With the baseline architecture defined, the next task is to determine the optimal hyperparameters. To this end, I set aside a validation set of 10,000 images taken from the 60,000 training images. I used this validation set to determine the number of C-layers, number of filters, filter size, and learning rate that maximize the validation accuracy. A summary of my cross validation results can be found in fig 7.

Cross validation revealed that the best CNN architecture uses 3 C-layers with 5x5 filters, with 20,40, and 80 filters for the first, second, and third C-layer respectively. Using a learning rate of 0.01, I trained this network for 20 epochs using all 60,000 training samples, this process took 24 minutes. I then used the trained network to classify the 10,000 images in the test bank, and did so with an accuracy of 99.3%.

# 3 Discussion

With a 97% accuracy the KNN proves itself a simple and natural multiclass classifier. The least squares binary classifier performed worse, achieving an accuracy of 94.2%. This occurs because the BLS classifier relies on the data being linearly separable, which is usually not the case. Also, the least squares cost function does not minimize the number of misclassifications, but instead the average distance of each training sample from $\pm 1$ (depending on which side of the boundary the data point is on). Perhaps training 45 one vs. one SVM's would outperform BLS since SVM's minimize misclassifications, making them better suited to classification tasks such as this one whereas least squares is more appropriate for regression. One strength of the BLS classifier compared to KNN is computation time. Classifying the 10,000 test images took several minutes using the k-NN algorithm, whereas the BLSC performed this task in a few seconds, about a 100 fold time save. As such, the BLSC might be the preferred classifier for problems with large amounts of features and/or training samples.

The CNN was able to classify the test bank with an accuracy of 99.3%. This impressive result shows that the CNN is capable of near human levels of accuracy. The CNN is able to achieve higher accuracy than the previous two methods because of its ability to fit complex, non linear decision boundaries to the data. This is very powerful since real life data sets more often than not express complicated, non linear relationships. While the CNN is more complicated under the hood, its application was simpler in many respects compared to the other two classifiers. Very little pre processing is required when using a CNN because the feature extraction is learned via optimization of the filters. As such, no fancy handcrafting of features or principal components analysis is necessary.

While the CNN's ability to fit complex boundaries to data is the source of its strength, it can also be a source of weakness depending on the problem. Due to the CNN's ability to construct highly non linear boundaries, there is an inherent risk of overfitting a boundary to outliers in the data set, thus sacrificing generality. In general, you want the complexity of your decision boundary to match the complexity of the inherent trend. As such, if the underlying trend is simple, the BLSC or knn algorithm might be a better choice.

# 4 Closing Thoughts

Here we contrasted two simple parametric classifiers, the BLSC and the knn algorithm. We found that the knn was the more accurate classifier, achieving 97% accuracy compared to the 94% accuracy of the BLSC. The BLSC was, however, the much faster classifier. Prediction with the BLSC took a few seconds, while the lazier knn algorithm took several minutes. We then contrasted these simple classifiers with a non parametric classifier capable of constructing more complicated decision boundaries, the CNN. We found that the CNN was capable of near human levels of accuracy, achieving 99.3% accuracy on the test bank. While the results for the CNN are impressive in this case, it is important to realize that there is no "best" classifier. The performance of every method is problem dependant and you should always design your classifiers to suit the task at hand.

# 5 Link to Project

`https://github.com/twinfree/ECE532-Final-Project`