Classifying Images of Numbers Using First Nearest Neighbor

Chase Perry

October 28, 2019

1 Problem Definition

Automating various processes has recently become a point of focus for many industries, as there are countless jobs and tasks that could be replaced by a machine that frees up a persons time to focus on more high level tasks. Sorting items, data retrieval, organization are all such activities that while important, don't necessarily require a great deal of thought to manage. What they do require however is time, and when a person is left to deal with these tasks they are being taken away from focusing on more intensive and beneficial tasks that benefit their business or company more.

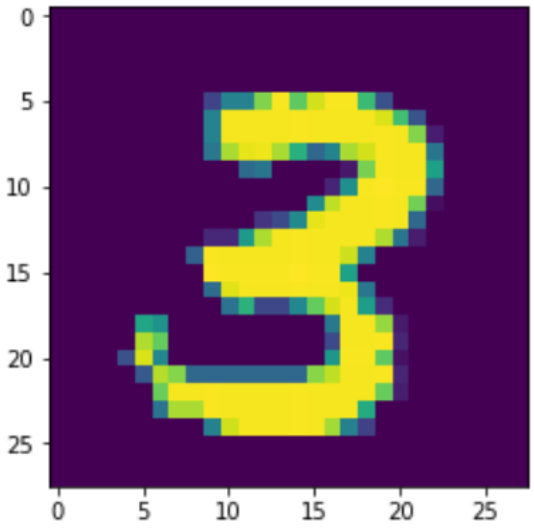
One such instance of this is sorting mail. Organizing mail based on where it needs to go is not a particularly difficult task in the sense of complexity of the issue, but it does require enough thought to understand the information on each package or letter to know

where it's going. In the past we've reserved this task for people, as it was necessary to have an individual look at each item and recognize the characters on it to decide where it would be going and to sort it accordingly. However, in this project I will attempt to show that this process is outdated and could be improved using a computer that can recognize the characters on a letter, specifically the numbers. This will be done using K-Nearest Neighbor implementing different distances metrics, such that calculation costs can be minimized while maintaining a high accuracy.

2 Data Set Source

The data set used for this project is the Modified National Institute of Standards and Technology database of handwritten figures, using sixty thousand training images and ten thousand testing images. This database was built by 'remixing' data from the National Institute of Standards and Technology, which was done in an effort to improve the variations and patterns seen in the images. The original NIST database was built using a training set of images written by U.S. Census Bureau employees while the testing set consisted of images written by U.S. high school students, as such there were likely to be some key differences between the figures written in training versus those tested against.

While the training and testing sets were in no particular order, as in all images of zero were first and all images of nine were last, they were shuffled during each run to ensure that there was minimal chance of creating a bias towards a specific class of digit. The most frequently occurring number in the training set was the number '1', and the least occurring



Example Image Of The Number '3' As Stored By MNIST

figure was '5' though they were still relatively close in overall samples. Because of this difference however, there could be a potential bias to the figures though this remains to be seen after actually attempting to classify each image.

The original dataset obtained from MNIST also contains each image as a 784 value vector, which when reshaped into a 28 x 28 matrix yields each number similar to that in Figure 1. Since K-Nearest Neighbors isn't dependent on the data being in a specific shape, other than having the same number of attributes, each image was left as the 784 value vector for classification purposes and only reshaped when it was necessary to visualize the image stored.

3 Proposed Solution

Our attempt to classify each number will use K-Nearest Neighbor, where we will focus on only using the first nearest neighbor from the testing set to classify each test image. The

general idea here being that after using some metric by which to judge how 'close' another sample is to our test image, we can assume that our test resides within the same class as that sample. In our attempts to classify each image we would look for the training image that has the closest 784 values to the 784 values of our testing image, and whichever class this closest training image belongs to is our predicted class for our training image. Each of these 784 values represents a pixel of the original image, by looking for a similar image we are saying that the two are so similar that the unknown test image should be another instance of whatever class the known training image belongs to.

The other main effort of this project is by what distance metric can we best classify each testing image. As there is no automatic or instant way to identify which metric we can use to identify the nearest neighbor we must also investigate this as well. This project will mainly focus on identify the differences in using a Euclidean Distance metric and a Cosine Difference metric in their accuracy. The Euclidean Distance is given by the equation found in Equation 1, where 'P' is the test image we are attempting to classify, X_n is one such training image by which we are comparing the test image, and 'i' is one feature of the 784 both images contain. For this metric the training image with the lowest euclidean distance is understood to be the nearest neighbor to the test image, and thus are predicted class.

$$d(P, X_n) = \sqrt{\sum_{i=1}^{784} (P_i - X_{n_i})^2} \tag{1}$$

The cosine difference is given by Equation 2 where 'P', 'X_n', and 'i' all indicate the same values as what was described up above. However, unlike the euclidean distance the cosine difference describes the most similar value as that which has the greatest result. Thus, after

computing this value between the testing image and the training set whichever image has the greatest cosine difference is chosen as the nearest neighbor, and used as the class prediction of the image.

$$d(P, X_n) = \frac{\sum_{i=1}^{784} P_i X_{n_i}}{\sqrt{\sum_{i=1}^{784} P_i^2} * \sqrt{\sum_{i=1}^{784} X_{n_i}^2}} \tag{2}$$

4 Solution

As stated before, while the data seems to be in no particular order when obtained from the source, meaning the classes of numbers aren't grouped together for example, the first step to classifying the images was to shuffle both the training and testing sets such that the chance of bias is minimized. After this, two main functions were defined and used to compute both the euclidean distance and cosine difference between the training image vectors and a test image vector. The two functions can be seen in Figure 4, where train and test are expected to be the matrix of training images and the test vector respectively. Both functions return a vector of values where each value is the euclidean distance or cosine difference between each training image and the test image.

```
1 def euclidean(train, test):
2   return np.sqrt(np.sum(np.square(train - test), axis=1))
3
4 def cosine(train, test):
5   num = np.sum(train * test, axis=1)
```

```
    first = np.sqrt(np.sum(np.square(train), axis=1))
    second = np.sqrt(np.sum(np.square(test), axis=0))
    denom = first * second
    return num / denom
```

After these functions were defined the next step was to decide which of the training and testing sets would be used for analysis, as both sets are so large that using every image would not be realistic. In addition, it was also necessary to define some manner in which the accuracy of each prediction could be visualized and quantified. These two goals resulted in the functions define in Figure 4, where test_euclidean and test_cosine use the euclidean distance and cosine difference metrics respectively. Both functions return a confusion matrix where the main diagonal is the accurate classifications, the row is the actual class if the image, and the column is the predicted class. The expected inputs for both functions are the training images matrix, the training class vector, the testing image matrix, the testing class vector, and two integers representing how many images should be used for training and testing in that order.

```
def test_euclidean(trainX, trainY, testX, testY, train_size, test_size):
    trainX = trainX[:train_size, :]
    trainY = trainY[:train_size, :]
    testX = testX[:test_size, :]
    testY = testY[:test_size, :]
    acc = np.zeros((10, 10))
    for i in range(0, test_size):
```

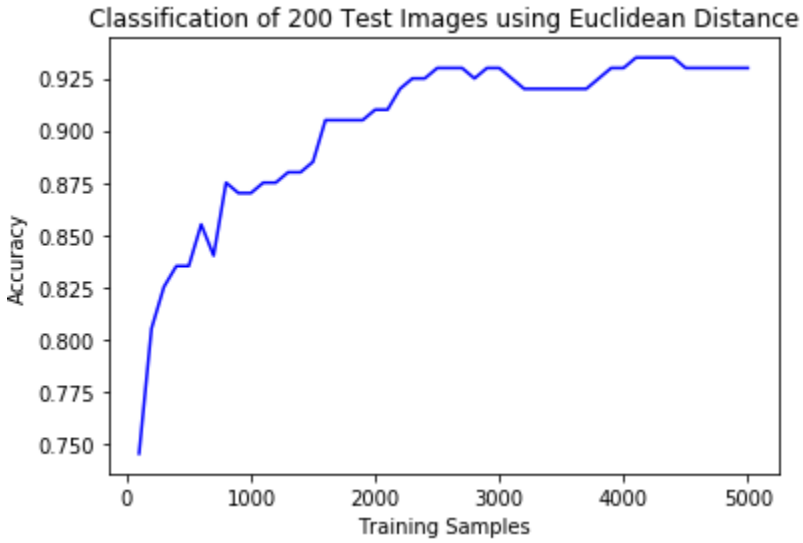
```
8       distances = euclidean(trainX, testX[i])
9       loc = np.argmin(distances)
10      pred = int(trainY[loc, 0])
11      actual = int(testY[i, 0])
12      acc[actual, pred] += 1
13    return acc
14
15  def test_cosine(trainX, trainY, testX, testY, train_size, test_size):
16    trainX = trainX[:train_size, :]
17    trainY = trainY[:train_size, :]
18    testX = testX[:test_size, :]
19    testY = testY[:test_size, :]
20    acc = np.zeros((10, 10))
21    for i in range(0, test_size):
22      distances = cosine(trainX, testX[i])
23      loc = np.argmax(distances)
24      pred = int(trainY[loc, 0])
25      actual = int(testY[i, 0])
26      acc[actual, pred] += 1
27    return acc
```

Using the four functions described above and some simple loops it becomes easy to adjust the main factors we care about, those being how many training images we wish to use and how many tests we are attempting to make. It is by this process that tests were run attempting to classify 200 images using 100-5000 training images, skipping every 100, using both the euclidean distance and cosine distance. Similarly, this was also how data

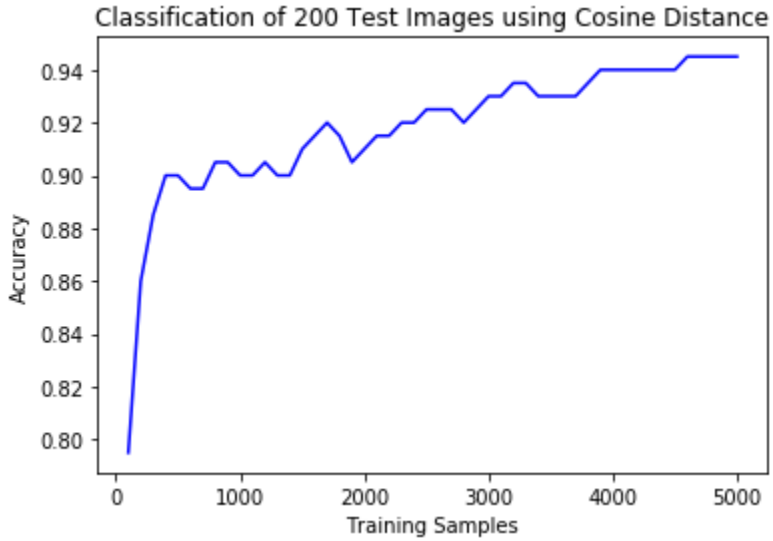
was retrieved for tests in which 500 training images were used to classify 100-5000 testing images, again skipping every 100. The first sets of testing in which the number of training images was varied was done in effort to test the change in accuracy as a result in the change in the training size, while the second sets of testing in which the number of testing images was changed was done in effort to see how both models stood up to more variations in the testing images. The results of all of these tests however, will be discussed below.

5 Evaluation

Overall, I would say that classifying images of numbers using K-Nearest Neighbor with a euclidean distance or cosine difference metric is a successful strategy. While errors are still present, results of all tests indicate that it could a useful method of classification for items like sorting mail or something to that degree. As can be seen in Figure 2 or Figure 3 both tests had a decent accuracy, though neither were perfect with its predictions. The largest difference between the two tests is that using the cosine difference metric seems to yield better accuracy overall, though it also takes more training samples for it plateau. With only 100 samples the test using euclidean distance has just below 75% accuracy, while the test using cosine difference has just below 80% accuracy. Similarly, the euclidean distance test plateaus around 2500 training samples while the cosine difference test does so around 4000 training samples. This indicates that while neither is perfect, using the cosine difference metrix yields a better accuracy but requires more training samples to base the prediction off of.



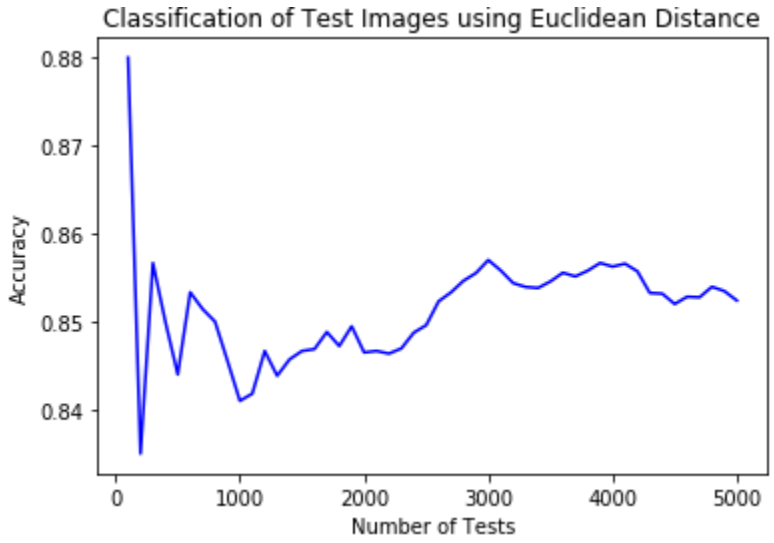
Classifying 200 Test Images Using Euclidean Distance



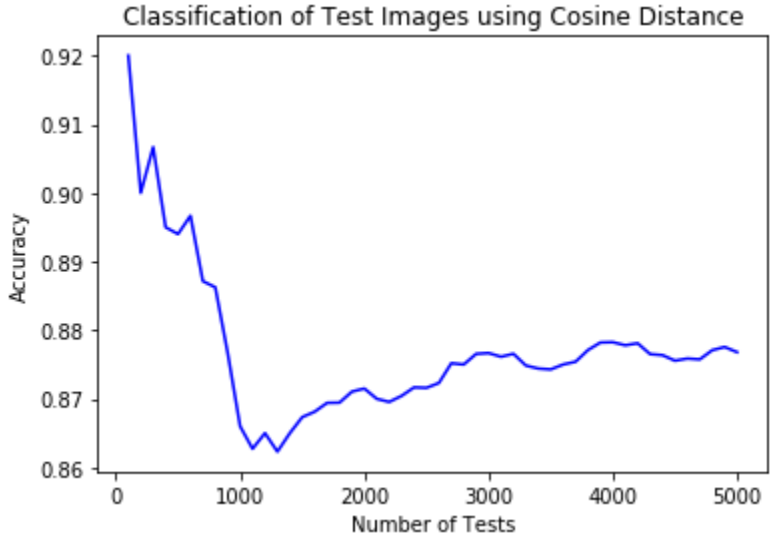
Classifying 200 Test Images Using Cosine Difference

Additionally, using the cosine difference metric maintained a better accuracy as the number of overall tests was increased while keeping the testing samples at 500. This effect can be seen in Figures 4 and 5, where the accuracy steeply decreases at first before plateauing as well. Similar to the data presented in 2 and 3, the cosine difference metric had a better accuracy when compared to the euclidean distance metric with the same number of testing

samples however in these tests both metric plateaued at around 3000 testing images. As these tests were meant to be a judge of how either metric would hold up to more testing classifications they show that overall the cosine difference maintains a better reliability as that number of tests increase, as the number of possible variations in the tests increase.



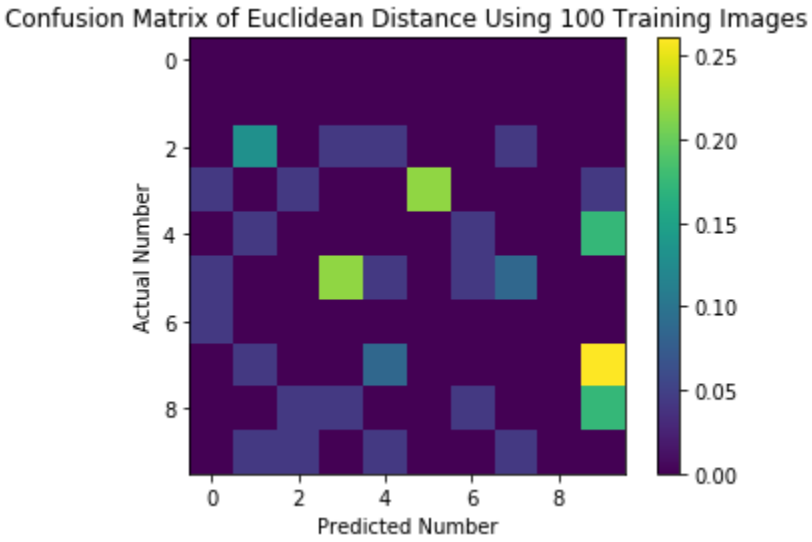
Classifying Images Using Euclidean Distance with 500 Training Samples



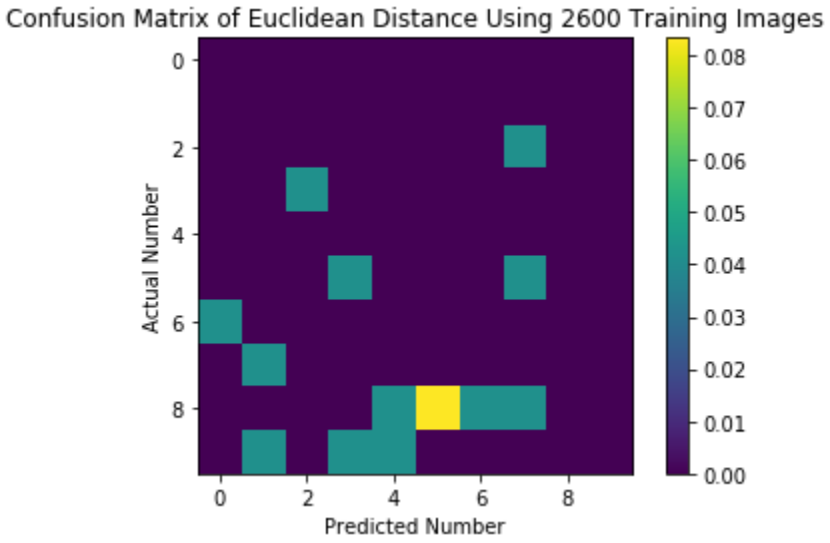
Classifying Images Using Cosine Difference with 500 Training Samples

As stated above, as the number of training samples increased both metrics had noticeable

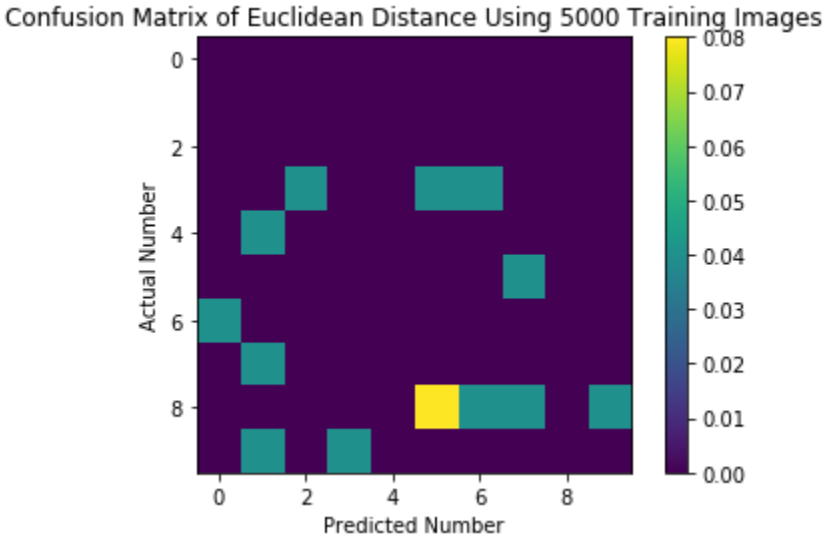
increases in accuracy nearing a 20% improvement. However, while the accuracy was improved for nearly all classes there were noticeable differences in this change. Seen in the differences between Figures 6, 7, and 8 where error occurred in the test with 100 training samples there was improvement observed when 2600 or 5000 training images were used for the euclidean tests. However where error occurred in the test with 2600 samples, there was improvement in the test with 5000 samples but this error was less likely to be fully removed as before. For instance, '6' was most often confused with 4-7 in the tests using 2600 samples versus in the test with 5000 samples there was minor or no change in this confusion.



Confusion Matrix of Classification with 100 Samples using Euclidean Distance



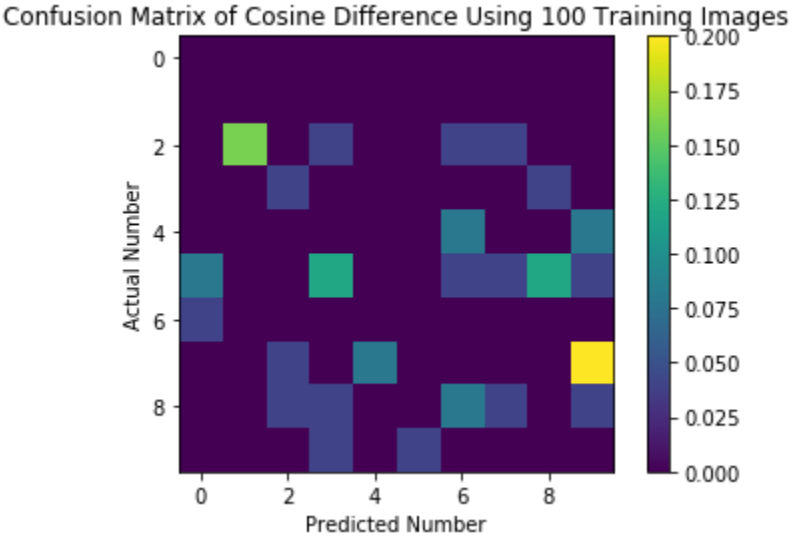
Confusion Matrix of Classification with 2600 Samples using Euclidean Distance



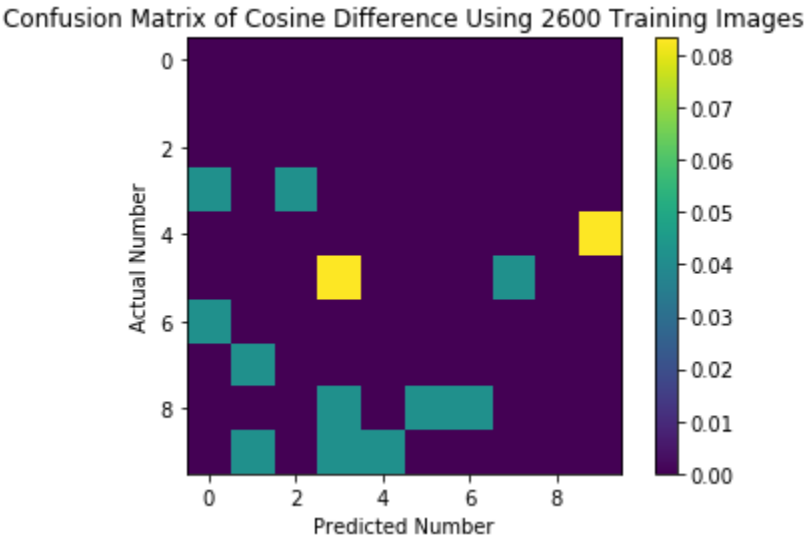
Confusion Matrix of Classification with 5000 Samples using Euclidean Distance

Similarly, in Figures 9, 10, and 11 it can be observed that there was a great deal of improvement between the test with 100 training images and both the tests with 2600 and 5000. Unlike the tests with euclidean distances however there was a fair amount of error elimination between the cosine difference test with 2600 samples and 5000 samples although there were certain areas where the error persisted such as where 7 or 9 was confused with 1

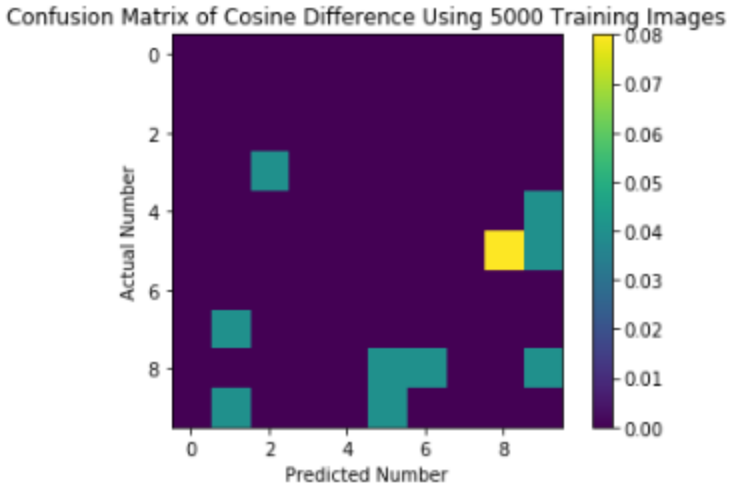
or where 8 was confused with 5 or 6.



Confusion Matrix of Classification with 100 Samples using Cosine Difference

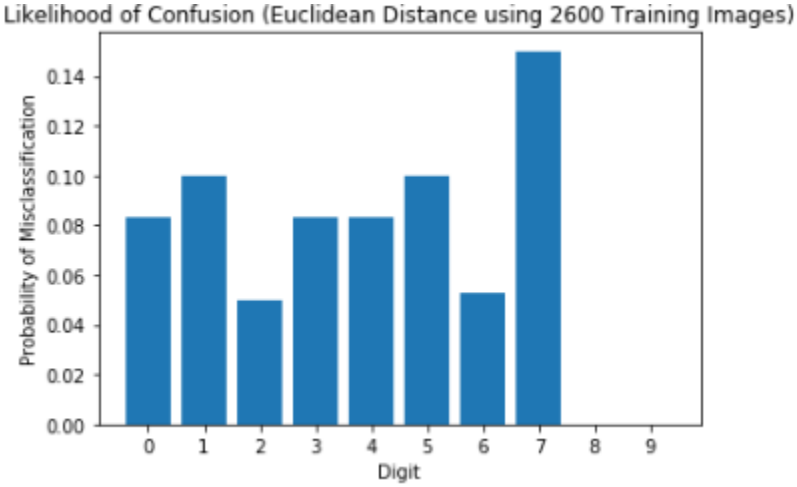


Confusion Matrix of Classification with 2600 Samples using Cosine Difference

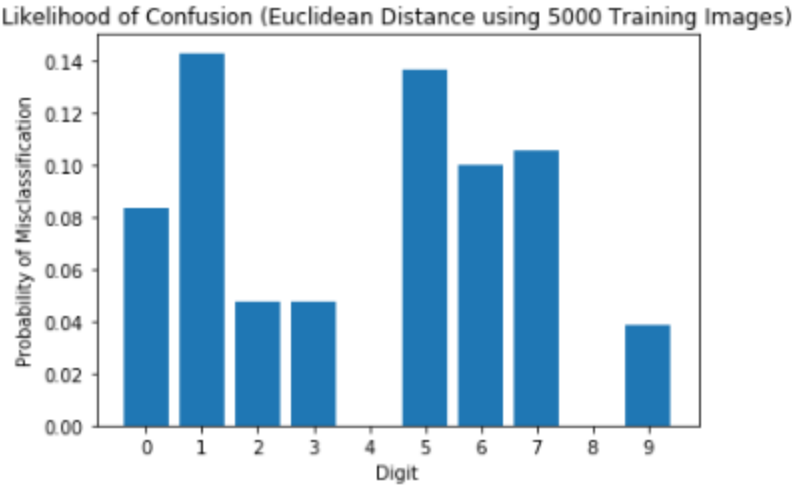


Confusion Matrix of Classification with 5000 Samples using Cosine Difference

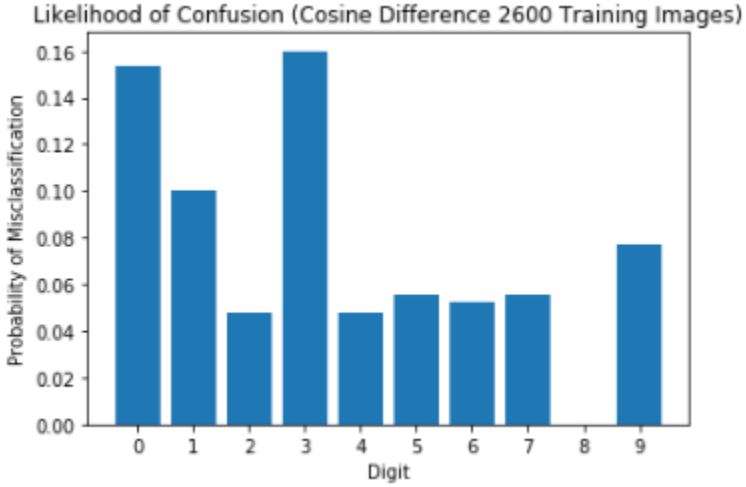
This consistency in error can be seen in Figures 12 and 13 for the tests with euclidean distance and Figures 10 and 11 for the tests using cosine difference. While these figures do not show which numbers they were confused with, it does show that as the number of training samples was increased the errors were consolidated in specific figures rather than being error prone around the board. This is especially observable for the cosine difference where errors moved from being everywhere but in class 8 to only classes 1, 2, 5, 6, 8, 9.



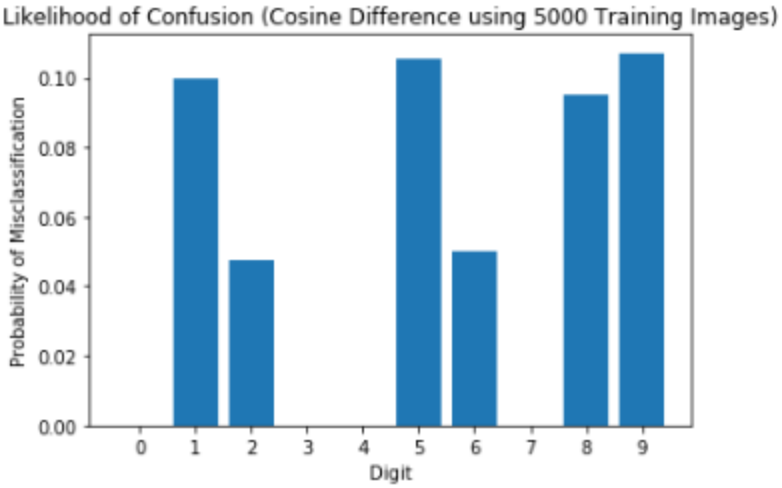
Error Histogram of Euclidean Distance Using 2600 Training Images



Error Histogram of Euclidean Distance Using 5000 Training Images



Error Histogram of Cosine Difference Using 2600 Training Images



Error Histogram of Cosine Difference Using 5000 Training Images

6 Future Work

The next step to take this research, in my opinion, would be to test out difference metrics by which images can be classified or to use various neighbors for classification. In this project the Euclidean Distance and Cosine Difference metrics were used to determine the similarity between the test image and the training images, but these are not the only metrics available. As was seen with the difference in accuracy between the Euclidean Distance and Cosine Difference there could potentially be another metric that has an even better accuracy than either of the two used here. Another avenue could be to use various numbers of the nearest neighbors, rather than just the first, and to do so efficiently. For instance, here we used the first nearest neighbor however there could be potential accuracy improvements (or losses) if the top 2 nearest neighbors were used for classification, or top 5, or 100. Attempts were made to do this for this project however it could not be completed efficiently so it was left out of the project.

7 What I've Learned

Through this project I feel that I've learned a fair deal about how to find similarities between objects or instances of various classes of information. Prior to this project I never would have thought it would be possible to identify a image of a number with just some images of other number and comparing their pixels. Similarly, I learned a bit about optimization in Python to improve the kind and number of comparisons that needed to be made. While a couple of for loops and some if statements would've been the easy route, it's also very much possible to avoid a great deal of this so that the overall testing becomes much more efficient.