Capstone Report

Analysis of Handwritten Digit Algorithms

Marco Sousa

University of Massachusetts Dartmouth
Dartmouth 2020

Contents

1 Introduction

1.1 Project Goals

The following paper is an investigation regarding classification of handwritten digits, which is a standard problem in pattern recognition. This investigation entails three project objectives. Most importantly, the goal is to classify a dataset of handwritten digits to the greatest accuracy possible. Additionally, this project serves as a method for me to acquire a better understanding of the tangent distance technique, and machine learning more broadly. Moving forward, I plan to work towards the latter with K-NN and neural networks to the best of my ability.

1.2 Classification

The essential idea of classification is to use a set of manually classified/identified objects (a training set) to construct a model that will further classify unknown objects (a test set). We are interested in the success or error rate, and also speed of the classification. I represent a success rate simply and intuitively as (correct classifications)/(total classifications).

2 The Data

There are two popular databases when classifying digits: US Post Office Zip Code Data and the MNIST DATABASE of handwritten digits. The latter has been classified with single convolutional neural network with an error rate as low as 0.25 percent.

2.1 Data Format

I chose to consider the US Postal service data first. This dataset contains 7291 training observations and 2007 test observations. Observations represent images, which can be represented in 3 formats in particular. Each row contains 256 cells that acts collectively as an observation. Such rows can be transposed to be columns, and vectors in R^{256}. Each observation can be reshaped into 16 by 16 pixel grayscale images that represent a handwritten digit. One can furthermore consider the 16 by 16 pixel image as a function of two variables, f(x,y).

2.2 Programming Observations

If the reshape command is simply used, in MATLAB, you may need to additionally perform a reflection and rotation. It is worth considering converting the values from [-1,1] to a grayscale [0,1]. To better display the image, one may consider calling imshow(1-image), so that black inverts with white. Alternatively, one can conserve their original values by reversing the color mapping in matlab settings.

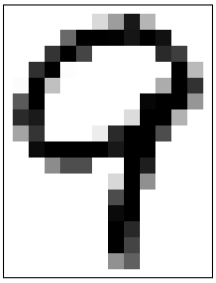


Figure 1: A representation of a formatted digit.

3 K-Means Classification

The classification of an image often revolves around the conception of minimizing differences. As such, conceptions of difference can be measured with forms of distances. The essential idea of a K-means clustering technique is to compare the euclidean distance of one image to each centroid representing a digit class. Note that euclidean distance can be calculated on an image regardless of whether it is a matrix of $R^{16\times16}$ or a vector in R^{256} without loss of generality because of the robustness of the norm operation. Such a difference for images p_1 and p_2 can be represented as follows:

$$\text{Euclidean Distance} := \|p_1 - p_2\|_2$$

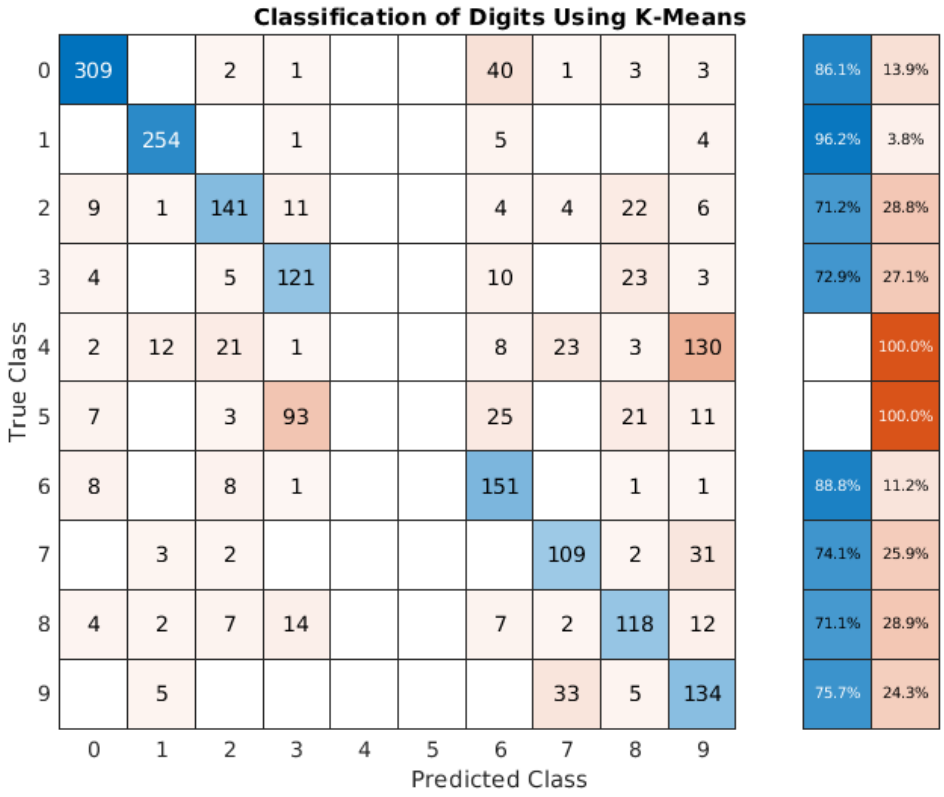
The Algorithm

The data is trained by performing a K-Means clustering on the training set. From such, 10 centroids represent each digit. This process is partially supervised at the start, as each cluster must be manually classified to represent a digit. In classification, for every digit p and each cluster c_k, the euclidean distance $\|p - c_k\|_2$ is calculated. The mean centroid with the minimum distance is chosen as the classifier for that test digit.

One common question for K-Means clustering is to consider how many k clusters are appropriate. However, with this data set, assuming the digits form distinct clusters, one must consider at least 10 clusters. However, since I found clusters were not sufficiently distinct from one another, there may be a repeat in considering a digit, and thus, by pigeon hole principle, all classifications of the given digit result in error. This was seen in the confusion matrix found on the next page, where the 4 and 5 were often classified as 9 and 3 wrongfully.

Because of the error with 4 and 5, this particular K-Means algorithm produced a success rate of 66.62 %. However, in other cases, this can by chance be as high as approximately 75 %. As such, I found a good deal of variability in my results for this algorithm. This

variability in the algorithm occurs when sampling the training set. Random sampling can improve the accuracy in this method case because it by chance can remove poorly written digits.



4 SVD Classification

Another classification algorithm involves the use of SVD. The essential idea of the SVD algorithm is to suggest all the digits of a single kind span a subspace. That subspace, represented in an orthogonal manner because of the structure of SVD, can be identified, and compared with each test digit, most simply. Rather than using euclidean distance, the difference is measured from the residual between the image and the computed subspace of the classes of a particular digit. The residual can be represented as follows:

$$\left\| z - U_k U_k^T z \right\|_2$$

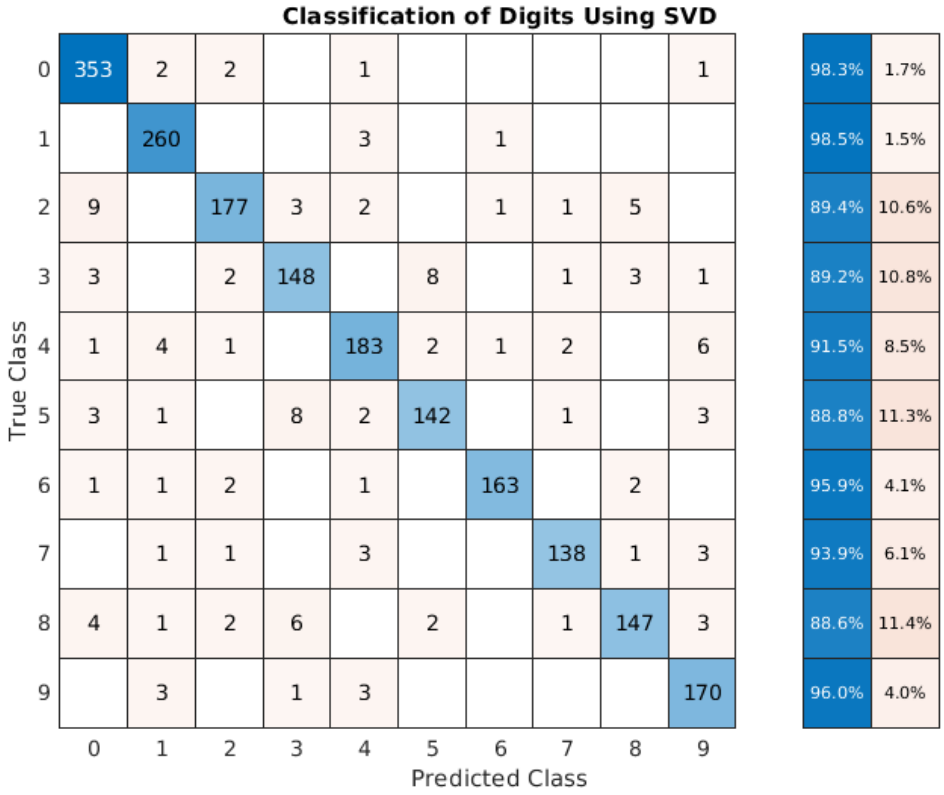
The relative residual can be represented as follows, similarly:

$$\left\|(I - U_k U_k^T)z\right\|_2 / \|z\|_2$$

The Algorithm

Separate the digits of the same kind in the training set and compute the SVD of of each to form a subspace of k singular values. For each test digit, compute the residual between the test digit and the 10 generated subspaces. Choose the minimum and classify the test in accordance to its respective subspace. One may also consider relative residual.

Such an algorithm holds some advantages. Since it is not necessary to perform a full SVD; one may choose k singular basis images to form the subspace, which saves a good deal of computation time. Furthermore, it's fairly accurate, with accuracy around 93 to 94 %, and low variability. One result is represented in the confusion matrix that follows below.



5 Tangent Distance Classification

5.1 Motivations for Tangent Distance

While the SVD method was a significant improvement from the K-means method, an upper bound of 94 % is not entirely accurate. One consideration we have yet to take into account is that the digits may be altered in some way. A digit can be rotated, translated, thickened, scaled, and so forth. A human eye can typically recognize these adjustments in writing and yet still classify an image appropriately. As such, the tangent distance procedure seeks to remedy this discrepancy in writing by conceptualizing a distance between two curves. The essential idea for the tangent distance method is to form a new conception of differences which approximates the distance between two curves by the distance produced between their tangent lines. These adjustments in writing are called 'invariances' because the transformation on the digit is invariant to the calculated tangent distance. In this manner, we may consider small transformations on digits, yet still conserve the same tangent distance, and thus, under small invariances, still retain an appropriate classification.

5.2 Tangent Distance

Considering a digit p, we can perform a transformation, s, on the image, with parameter a, in order to adjust the image. In such a manner, we can represent a transformation on an image as s(p,a), with s(p,0) = p. To approximate this nonlinear curve, or manifold, under a transformation s, we can construct a linear approximation. A Taylor expansion can be performed to approximate the curve as follows:

$$s(p_1, a) = s(p_1, 0) + \frac{ds}{da}(p_1, 0)a \approx p_1 + t_{p1}a$$

If we have a second point p_2, then we can define the transformation similarly:

$$s(p_2, a) = s(p_2, 0) + \frac{ds}{da}(p_2, 0)a \approx p_2 + t_{p2}a$$

Noticeably, the calculation of the tangent vector, $\frac{ds}{da}$(p,0), is required, and such tangent vector will be calculated in a later section. This approximation also satisfies our aforementioned essential idea. To approximate the curve s(p,a), we use a linear approximation represented by the tangent vector. To find the difference between curve one and curve two, we subtract the approximation of each curve. Then the difference can be taken as follows:

$$\begin{aligned}
&\min \|\text{(curve one) - (curve two)}\|_2 \\
&= \min \|s(p_1, a_{p1}) - s(p_2, a_{p2})\|_2 \\
&\approx \min \|p_1 + t_{p1}a_{p1} - (p_2 + t_{p2}a_{p2})\|_2 \\
&= \min \|(p_1 - p_2) + t_{p1}a - t_{p2}a\|_2
\end{aligned}$$

$$\min_{a_{p1},a_{p2}} \left\| (p_1 - p_2) - \begin{bmatrix} -t_{p1} & t_{p2} \end{bmatrix} \begin{bmatrix} a_{p1} \\ a_{p2} \end{bmatrix} \right\|_2$$

The prior difference considers two images p_1 and p_2, and a single transformation parameter, a. However, we can imagine moving each digit along several curves transformed by many a. Thusly, we may consider a set of transformations parameterized by $\{ a_1 ... a_l \}$ with respective $T_p = (\frac{ds}{da_1} \frac{ds}{da_2} ... \frac{ds}{da_l})$ and continue as follows:

$$\min \left\| (p_1 - p_2) - \begin{bmatrix} -T_{p1} & T_{p2} \end{bmatrix} \begin{bmatrix} a_{p1} \\ a_{p2} \end{bmatrix} \right\|$$

Such a formulation is of course in the form of solving a classic least squares question:

$$\min \|b - Ax\|_2$$

Note that the solution for a minimum alpha is not what we are directly interested in, but rather, the residual of the solution, described by the norm. Recall that alpha is decided apriori as a paramaterization of the transformation of image p under s, in the form of s(p,a). An important consideration when solving such a system is whether matrix A is singular or not. Simard et. al.[1] marks that matrix A composed of tangent vectors may become singular when some of the tangent vectors for p_1 and p_2 become parallel. As such, to the best of my understanding, when constructing tangent vectors for p_1 and p_2, there may exist tangent vectors that are parallel, and furthermore linearly dependent, and thus singular. One solution is to use a QR method. Solving with a QR method must split into two cases of singular and nonsingular. In implementation, I chose to follow the method of controlled deformation, proposed Simard [1] and again used by Jose Israel Pacheco [2].

5.3 Transformations

Recall that we are considering slight perturbations of image p under s paramaterized by a. As such, different transformations, such as scaling, rotating, translating in x or y directions and so forth, act on image p, and produce respective tangent vectors t. It would be expected that a transformation by a nonzero alpha would change the curvature of image p, along with its tangent vectors. Furthermore, the change on the curve and tangent vector would naturally be different for different transformations. As such, it is necessary to describe and derive each transformation accordingly.

Transformation as Linear Combination

Most simply, in translation, we aim to move the image pixels horizontally or vertically. The direct mapping of the transformation is as follows for a horizontal translation:

$$t_a = \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + a \\ y \end{pmatrix}$$

If one considers p =p(x,y) as a function in two variables, then s becomes s(p,a_x)(x,y) = p(x+a_x,y). Finding the derivative will create our respective tangent vectors through our paramaterized operator L. $p_x = \frac{dp}{dx}$ can be derived with the chain rule:

$$L_x = \frac{d}{da_x}(s(p, a_x)(x, y))|_{a_x=0} = \frac{d}{da_x}p(x + a_x, y)|_{a_x=0} = p_x(x, y)$$

Such a mechanism constructs the derivative of the respective transformation. This produces a specific tangent vector for transformation s, of which we can paramateraize as operator L with respect to a in the form of $p + La$, analogous to the prior $p + t_p a$. The prior operator, L_x, is the derivative for a transformation s by translating in the x direction. Notice for any operator L, when considering a=0, the original p is retained: $p + 0L$ = p.

Table of Operators

The derivation of other transformations can be found in Analysis and Tests of Handwritten Digit Recognition Algorithms(Savas)[3] or Matrix Methods in Data Mining and Pattern Recognition (Eldén)[4]. It can be seen that the tangent vectors in the form of transformation operators can all be represented as linear combination of the x and y derivatives p_x and p_y. A table of the operators are as follows.

Table 1: Transformation Operators and their Derivative Representations

Transformation	Mapping from p(x, y)	Derivative
x-Translation	p$(x + a, y)$	p_x
y-Translation	p$(x, y + a)$	p_y
Rotation	p$(x\cos(a_r)+y\sin a_r, -x\sin a_r+y\cos a_r)$	$yp_x - xp_y$
Parallel Hyperbolic	p$(x + ax, y - ay)$	$xp_x - yp_y$
Diagonal Hyperbolic	p$(x + ay, y + ax)$	$yp_x + xp_y$
Scaling	p$(x + xa, y + ya)$	$xp_x + yp_y$
Thickening	N/A	$(p_x)^2+(p_y)^2$

Programming Observations

Regardless of the method in which you calculate the tangent vector, you will need to access each pixel at some point. As such, you are bounded by a minimum of kmn FLOPs, where k are the necessary flops to carry out your operator procedure for image of size m×n pixels. One should take notice, therefore, that the size of an image matters in respect to the FLOP count when considering many images. Thus, downscaling, as was done with the MNIST database, is worth considering for very large datasets.

If one chooses to implement tangent distance by means of mapping pixels directly, then some translations do not correspond in their pixel domains. When considering image $R^{16\times16}$ and P[x][y], x and y are elements of integers ranging from 1 to m and 1 to n respectively. However, for mappings such as rotation, x and y can map to noninteger values when

considering the output of trig functions. To adjudicate for this, an interpolation must be done for each rotation to map in the integer domain of x,y = 1...16. For many images, for the set of a_l parameters, considering each image of size m $\times$ n, an interpolation would be required, and can become quite expensive. As such, the next section details more appropriate methods of attaining the tangent vector, and operator similarly, by constructing a linear combination of p_x and p_y.

5.4 Tangent Vectors

In the calculation of the tangent distance, a necessary calculation is the construction of a tangent vector t_p, or $\frac{ds}{da}$. The appropriate derivation of the tangent vector requires differential geometry concepts I do not completely understand. However, I will briefly discuss the essential idea, and a practical approximation in the following section.

5.4.1 Derivation of Tangent Vector by Gaussian Convolution

In order to construct the tangent vector of the nonlinear manifold, we must convert an image from a discrete matrix P[i][j], for integer indices i and j, to a tensor function in two variables, p(x,y). An image P[i][j] can be written as a single discontinuous function using the 'delta' function (Savas,42)[3]:

$$P'(x,y) = \sum_{i,j} P[i,j]\delta(x-i)\delta(y-j)$$

The discontinuity of the function is obvious when expanded into summed piecewise functions:

$$P'(x,y) = P[i,j]\delta(x-1)\delta(y-1)+P[i,j]\delta(x-1)\delta(y-2)+P[i,j]\delta(x-2)\delta(y-1)+...+P[i,j]\delta(x-15)\delta(y-15)$$

Convolution with a Gaussian produces a continuous function f in the form of:

$$f(x,y) = \sum_{i,j} P[i,j]g_\sigma(x-i,y-j)$$

While I am somewhat familiar with the convolution of an image, the application of the convolution of the image with a Gaussian function, which later requires the derivative of such in x or y, is not distinctly clear to me. Furthermore, it's not exactly clear how the delta function constructs the derivative or why. As such, in implementation, I choose to approximate the tangent vector t_p for image p under transformation s by a method of finite differences.

5.4.2 Approximation of Tangent Vector by Finite Differences

The transformation of s on p necessitates the construction of tangent vector t_p. This tangent matrix, derived from the tensor p = p(x,y) under operation s, typically requires the derivative in terms of x or y, dependant on the specific operation. These derivatives $\frac{\partial p(x,y)}{\partial x}$ and $\frac{\partial p(x,y)}{\partial y}$, can be calculated with finite differences, such as central distance, as defined below:

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x+h,y) - f(x-h,y)}{2h}$$
$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+h) - f(x,y-h)}{2h}$$

Applying the central difference directly on the matrix, with h=1, is defined as below:

$$P_y(i,j) \approx \frac{P[i+1,j] - P[i-1,j]}{2h}$$
$$P_x(i,j) \approx \frac{P[i,j+1] - P[i,j-1]}{2h}$$

Notice that this runs contrary to our prior derivation for $\delta(x-i)$. In practice, the i in image P[i][j] corresponds to a row, and thus operates in the y dimension. Although this runs opposite of the notation described by Savas [3], I conjecture in implementation the derivatives are calculated in correct correspondence to ∂x and ∂y.

When considering the transformation operators in the prior section, x and y become somewhat less clear to me when we considering the image in respect to finite differences instead of a tensor function. x and y can no longer represent pixel domains, since a diagonal hyperbolic transformation would seem to imply a tangent vector in the form of 16px+16py for P[16][16], as an example. Yet this clearly cannot be the case, since the grayscale images range from [0...1], and would completely blacken an image. As such, my understanding is that x and y represent gradient matrices, in which the column or row values range from $[\frac{1}{n}, \frac{2}{n}, ...\frac{n}{n}]$ or $[\frac{1}{m}, \frac{2}{m}, ...\frac{m}{m}]$ respectively. In such a manner, we multiply p_x or p_y by a row or column gradient, which represents x or y. This intuition may be incorrect, however.

5.5 Pre-processing

When considering the classification of a digit, the representation of the digit is of significant importance. One method of improving classification performance is to employ a preprocessing technique, where images are adjusted in some way prior to training and testing. While there are several ways to preprocess grayscale images, I chose to blur (or smooth) the images with a gaussian kernal ($\sigma = 0.9$), so that the edges become more gradient. Berkant Savas [3] suggests a method of approximating the blurring process by cutting the tail of the kernal, which saves preproccessing time for large datasets. While

this is useful, in my implementation, I simply performed a full gaussian blur. This may become more important when considering the much larger MNIST dataset.

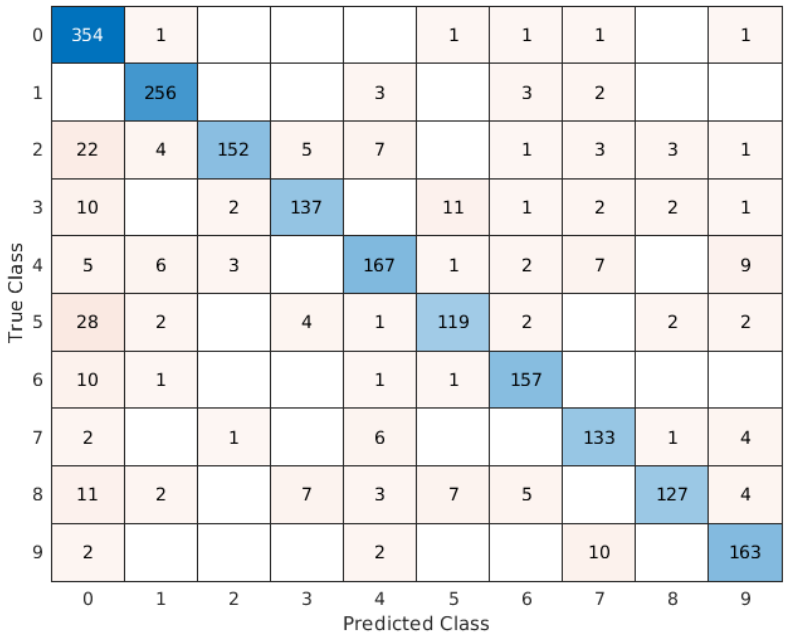
If you construct the tangent vector by the product of gaussian functions, then the blurring occurs locally and simultaneously with the construction of the tangent vectors. However, if constructed by finite differences, then blurring must be done ahead of time.

5.6 The Algorithm

Compute the Tangent Matrix T_p for each digit in the training set. For each test digit, compute its Tangent Matrix. Then compute the tangent distance to that of each training digit.

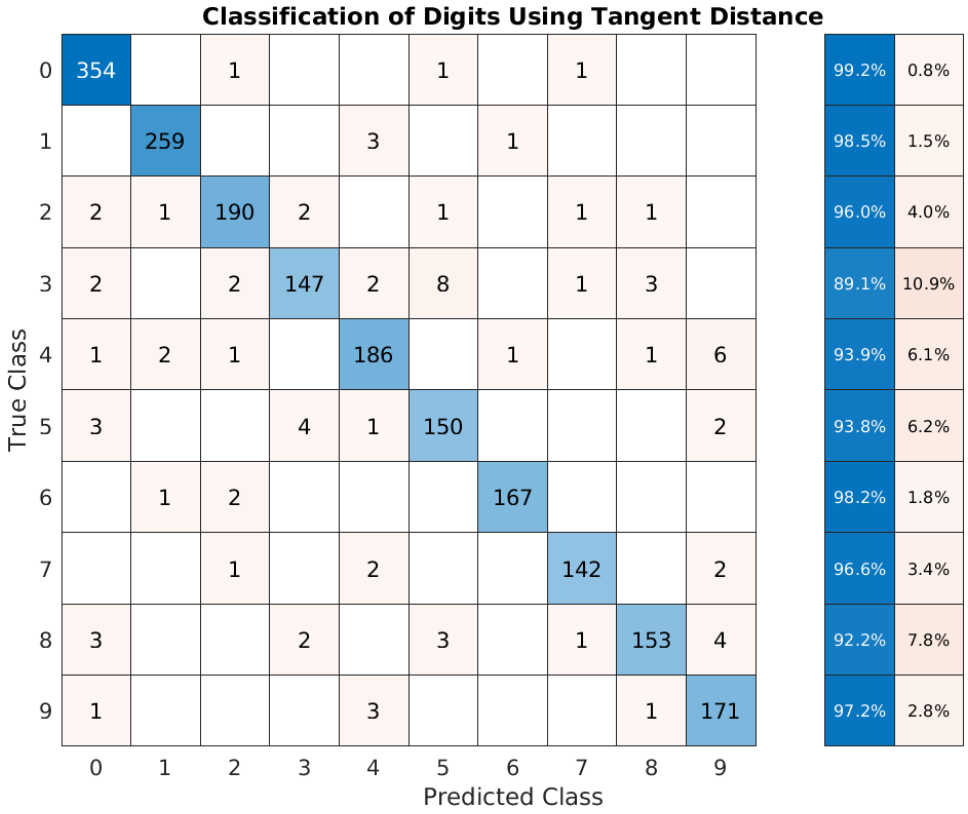
The clear disadvantage is of course the fact we must compare to every training digit. This can be very expensive, but will hopefully improve accuracy. It is also possible to instead consider a set of images that most distinctly represent the image, and use that training set. The following confusion matrix demonstrates my tangent distance algorithm with horizontal and vertical translation, rotation, and thickening, using the direct mapping and interpolation method on constructing tangent vectors and transformation operators. In this procedure, I did not blur the image set.

Tangent Distance Confusion Matrix - Pixel Mapping



The following is a confusion matrix with an success rate of 95.9%, using finite derivatives to approximate the and construct tangent vectors considering the same transformations. I implement a full gaussian blur before classification.

Tangent Distance Confusion Matrix - Finite Differences



6 Moving Forward

At the moment, my tests and their the results of my tests are not clear. Moving forward, I will create a formal results and conclusion section, containing a clear table and confusion matrix with discussion. Thus far, my tangent distance method was the most accurate. I hope to achieve better accuracy with K-NN or neural networks, as future prospects for this capstone study.

7 Citations

[1] Patrice Y. Simard, Yann A. Le Cun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. 2000.

[2] José Israel Pacheco.A comparative Study for the Handrwritten Digit Recognition Problem - Tangent Distance. California State University, May 2011, pp 17-20.

[3] Berkant Savas.Analysis and Tests of Handwritten Digit Recognition Algorithms. Linköping, 2003.

[4] Lars Eldén, Matrix Methods in Data Mining and Pattern Recognition - Classification of Handwritten Digits. SIAM, Society for Industrial and Applied Mathematics, 2019, pp. 113–128.