Peter Looi

Thien Nguyen

CIS 472

12 June 2019

Handwritten Character Recognition with Nearest Neighbor

Abstract

This document describes an experiment that tests the viability of using nearest neighbor to classify handwritten characters. Typically, a convolutional neural network would be used for this task. However, nearest neighbor, if optimized properly, may still hold some advantages over convolutional neural networks. The most difficult part of this experiment is optimizing the hyperparameters. There are many hyperparameters, including the distance metric, the K value, and the number of training examples. Finding the right combination of hyperparameters is key to the success of nearest neighbor character recognition.

Introduction

Handwritten character recognition is an essential component of many software applications such as image to text conversion systems and hand-written character input systems. Thus, due to the importance of handwritten character recognition, different approaches to the problem must be explored in order to determine the most desirable approach. The most well-known approach to handwritten character recognition is to use neural networks or convolutional neural networks. However, one approach that is commonly overlooked is nearest neighbor. Nearest neighbor is typically not thought of as an algorithm designed for image recognition. For handwritten character recognition, however, nearest neighbor is a very promising model because the Euclidian distance between two images of handwritten characters of the same class is likely smaller than the Euclidian distance to images of other character classes. This is due to handwritten characters having limited variation. The goal of this project is to determine the viability of nearest neighbor at the task of handwritten alphabetical character classification (A to Z) with greyscale images of size 25 by 25.

The first step of this project is to build a nearest neighbor model from scratch. The next step is to gather training and validation data from the EMNIST dataset. The final phase involves experimenting with different combinations of hyperparameters to determine the optimal hyperparameter configuration.

Background

The EMNIST dataset is a publicly available dataset that contains large quantities of labeled handwritten characters. The dimensions of the images in the EMNIST dataset are 28 by 28 pixels. For this project, the CSV version of the EMNIST dataset (can be found at https://www.kaggle.com/crawford/emnist) will be used. The images will need to be scaled down to 25 by 25 pixels, as that is what was chosen as our standard size.

The model that will process this data is nearest neighbor. The nearest neighbor model is a model that uses a distance metric to determine similarity between data points. With the standard Nearest Neighbor model, the training phase just involves memorizing each training example. The inference phase involves iterating through each training example to find the one with the least distance to the input data point. The label of the training example that has the least distance to the input data point will be returned as the final result. If the “K” hyperparameter is set to an integer greater than 1,

the nearest neighbor algorithm will instead search for the K training examples with the least distance to the input data point. The majority label out of the top K training data points will be returned as the final result.

The experiments described in the following sections of the document involve various distance metrics. These distance metrics are Euclidian distance, l1 distance, weighted Euclidian distance, and Minkowski distance. Euclidian distance is defined as

$D(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

L1 distance is defined as

$D(p, q) = \sum_{i=1}^{n}|q_i - p_i|$

Weighted Euclidian distance is defined as

$D(p, q) = \sqrt{\sum_{i=1}^{n} w_i(q_i - p_i)^2}$

Minkowski distance is defined as

$D_u(p, q) = (\sum_{i=1}^{n}(q_i - p_i)^u)^{\frac{1}{u}}$

The experiments described in the following sections also involve various image transformations. These image transformations are downscaling and image gradient. Downscaling is a transformation that involves resizing an image to smaller dimensions. This is done by passing a filter, with appropriate size and stride, over the original image. With "max downscaling," at each step the resulting pixel in the downscaled image is given the same value as the highest valued pixel in the filter's region. With "average downscaling," at each step the resulting pixel in the downscaled image is given the average value of all the pixels in the filter's region ("average downscaling" is also just referred to as "downscaling"). With "min downscaling," at each step the resulting pixel in the downscaled image is given the same value as the lowest valued pixel in the filter's region. Image gradient is a transformation that involves transforming the original image to a matrix (of smaller size), where each pixel conveys the angle at which the greyscale values are increasing.

Methods

Responsibilities

The first two tasks are to and collect the handwritten character data and build the nearest neighbor model. I took responsibility for building the model, and my partner, Daniel Leef, took responsibility for collecting the data. Then, we both each made separate optimizations to the nearest neighbor model. I focused on optimizing the distance metric, the K value, the image transformations, other hyperparameters, and the inference speed, while Daniel focused on achieving higher accuracy using edited nearest neighbor.

Approach

The approach I took to building the nearest neighbor model was to build it from scratch, without any machine learning libraries. This approach was chosen because the nearest neighbor algorithm is simple enough that building it from scratch is a relatively simple task. In addition, building the model from scratch gives us total control over the model's functionality, giving us the option to optimize any aspect we wish.

The nearest neighbor model was constructed as a class. The class just serves as a container for the nearest neighbor functionality, and contains no data members. The most important method in the NearestNeighbor class is the "classify" method, which takes an array of training examples, an array of test data points, a K value, and a distance metric. This method returns the list of classifications for each test data point made by the nearest neighbor algorithm.

Optimizing performance involved converting PIL images to numpy matrices as well as resizing the images. The handwritten character images begin in the form of a PIL.Image object. However, it is expensive to find the distance between two PIL.Image objects, as this requires an iteration over every pixel in both images. Instead if the images are converted to numpy matrices, the initial loading time will be increased due to the time spent converting from PIL.Image to numpy matrices, but once the conversion is complete, each individual comparison will be much faster, as numpy matrix operations yield a 30x speed up over iterating through each pixel. Also, resizing the numpy matrices to 12x12, instead of 25x25, allowed for the distance calculations to be significantly faster. This is because comparing images with less pixels requires fewer operations. In addition, resizing the images to 12x12 actually provides an increase in accuracy as well. This will be discussed in the "Experiments" section.

Determining the viability of nearest neighbor with handwritten character recognition involves finding the optimal set of hyperparameters that yields the maximum accuracy. If the optimal set of hyperparameters yields an acceptable accuracy, then it can be concluded that nearest neighbor is viable for handwritten character recognition.

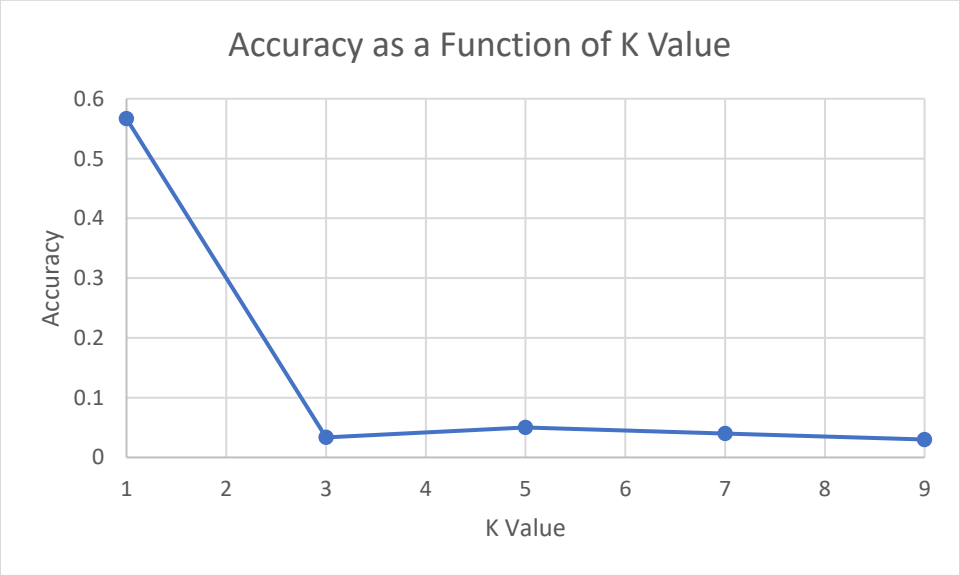
The full list of hyperparameters that I will focus on optimizing is number of training examples, K value, image transformations, and distance metrics. The method that will be used to optimize these hyperparameters involves first testing each hyperparameter individually. Testing hyperparameters individually will help determine any obvious trends, such as if certain hyperparameters clearly do not have any effect, or if certain hyperparameters have an obvious effect. Testing hyperparameters individually will also provide a general sense of which hyperparameters and settings will yield the highest accuracies. Then, the settings that yielded the highest accuracies will be combined and tested further.

Experiments

This section contains accuracy data resulting from various modifications of hyperparameters. Each trial, trial 1, 2, and 3, was done using a different validation dataset. The dataset of each trial is of size 100.

Modify K Value

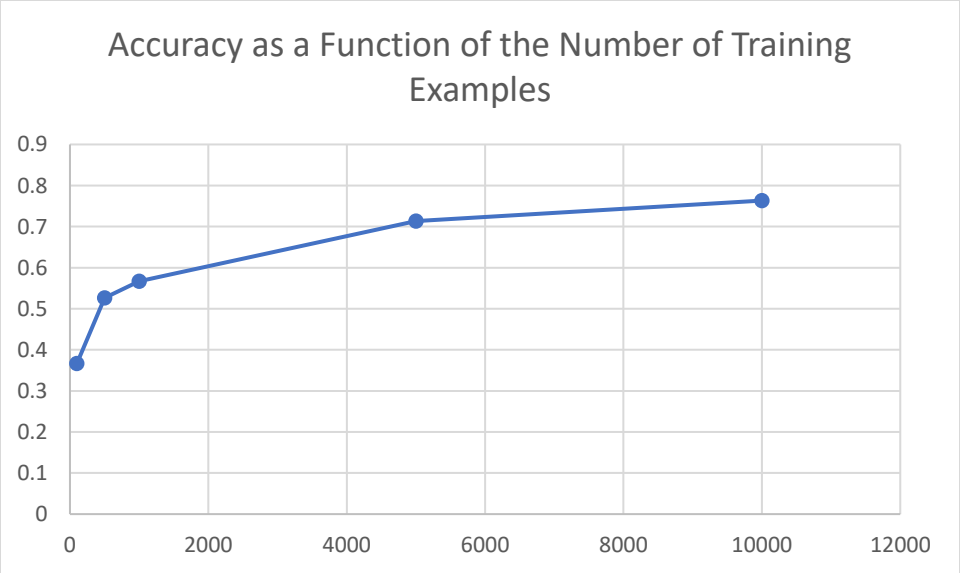
trial	train examples	distance metric	transformation	**k**	accuracy
1	1000	euclidian	None	**1**	0.5
1	1000	euclidian	None	**3**	0.02
1	1000	euclidian	None	**5**	0.05
1	1000	euclidian	None	**7**	0.04
1	1000	euclidian	None	**9**	0.02
2	1000	euclidian	None	**1**	0.62
2	1000	euclidian	None	**3**	0.05
2	1000	euclidian	None	**5**	0.07
2	1000	euclidian	None	**7**	0.05
2	1000	euclidian	None	**9**	0.04
3	1000	euclidian	None	**1**	0.58
3	1000	euclidian	None	**3**	0.03
3	1000	euclidian	None	**5**	0.03
3	1000	euclidian	None	**7**	0.03
3	1000	euclidian	None	**9**	0.03



A K value of 1 seems to yield the highest accuracy. This may be due to some images in the validation data matching well with only one training example in their class, and not matching well with any other training example. If this is the case, having a K value other than 1 will likely cause a decrease in accuracy.

Modify Number of Training Examples

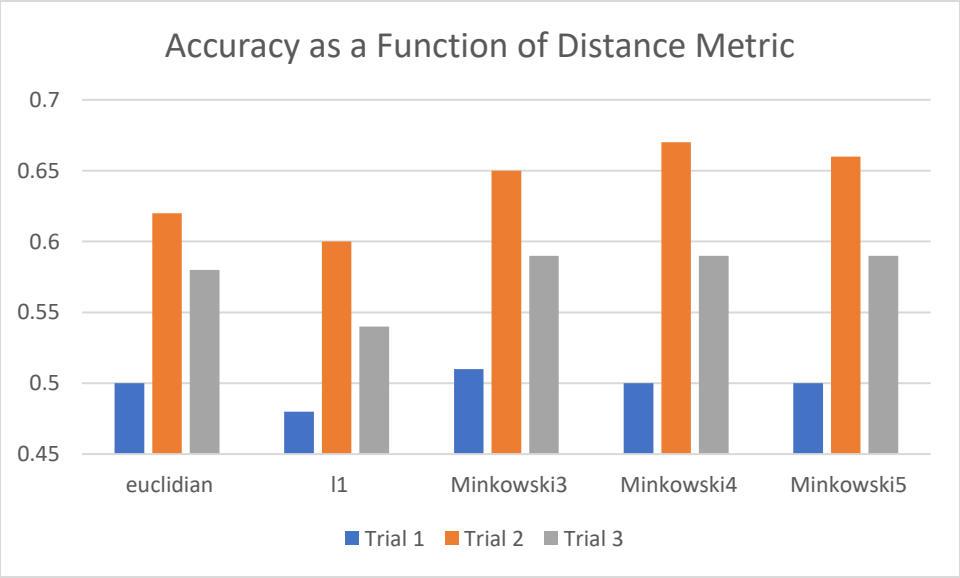
trial	**train examples**	distance metric	transformation	k	accuracy
1	**100**	euclidian	None	1	0.32
1	**500**	euclidian	None	1	0.5
1	**1000**	euclidian	None	1	0.5
1	**5000**	euclidian	None	1	0.66
1	**10000**	euclidian	None	1	0.73
2	**100**	euclidian	None	1	0.42
2	**500**	euclidian	None	1	0.59
2	**1000**	euclidian	None	1	0.62
2	**5000**	euclidian	None	1	0.75
2	**10000**	euclidian	None	1	0.78
3	**100**	euclidian	None	1	0.36
3	**500**	euclidian	None	1	0.49
3	**1000**	euclidian	None	1	0.58
3	**5000**	euclidian	None	1	0.73
3	**10000**	euclidian	None	1	0.78



Accuracy seems to increase, with diminishing returns, as the number of training examples increases. This is likely due to more training examples increasing the likelihood that each of the validation examples will match well with at least one training example.

Modify Distance Function

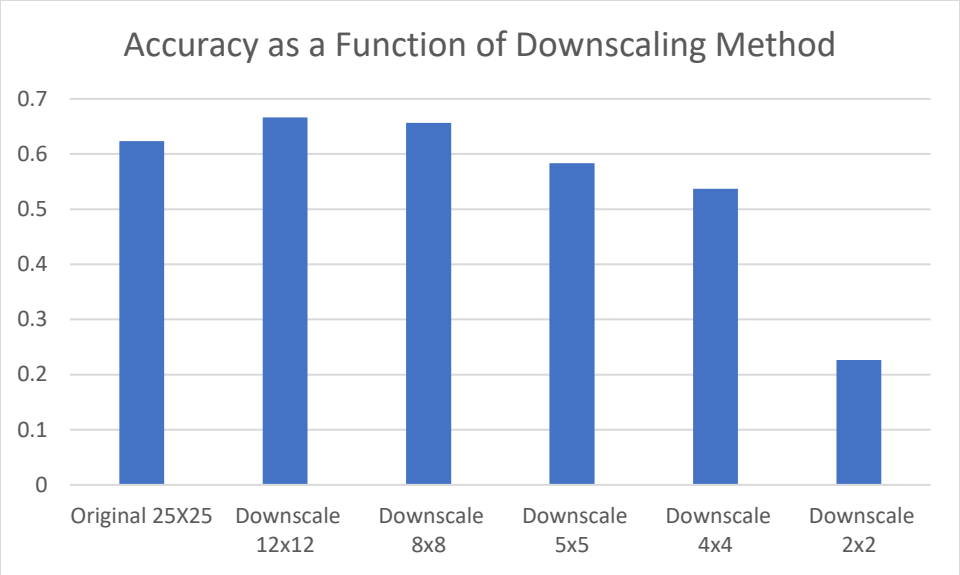
trial	train examples	**distance metric**	transformation	k	accuracy
1	1000	**euclidian**	None	1	0.5
1	1000	**l1**	None	1	0.48
1	1000	**Minkowski3**	None	1	0.51
1	1000	**Minkowski4**	None	1	0.5
1	1000	**Minkowski5**	None	1	0.5
2	1000	**euclidian**	None	1	0.62
2	1000	**l1**	None	1	0.6
2	1000	**Minkowski3**	None	1	0.65
2	1000	**Minkowski4**	None	1	0.67
2	1000	**Minkowski5**	None	1	0.66
3	1000	**euclidian**	None	1	0.58
3	1000	**l1**	None	1	0.54
3	1000	**Minkowski3**	None	1	0.59
3	1000	**Minkowski4**	None	1	0.59
3	1000	**Minkowski5**	None	1	0.59



It appears that the accuracy is not affected significantly by the distance metric. Note that the accuracy axis on the bar graph is already offset by 0.45. However, there still are a few trends that can be inferred. The clearest trend is that is that l1 distance performed the worst for all three trials. Also, it seems that the Minkowski distance metrics performed better than Euclidian distance and l1 distance. One explanation for this is that for effective character classification, smaller distances need to be forgiven and large distances need to be amplified, because often times two instances of the same character class have only very small differences. In such cases, the small distances should be forgiven in order for the two characters, which are of the same class, to have a smaller distance to each other. Minkowski distance has just this effect: It forgives smaller distances and it amplifies large distances. If a distance metric can cause characters of the same class to all have a small distance to each other, nearest neighbor classification will be very accurate.

Modify Image Downscaling Size

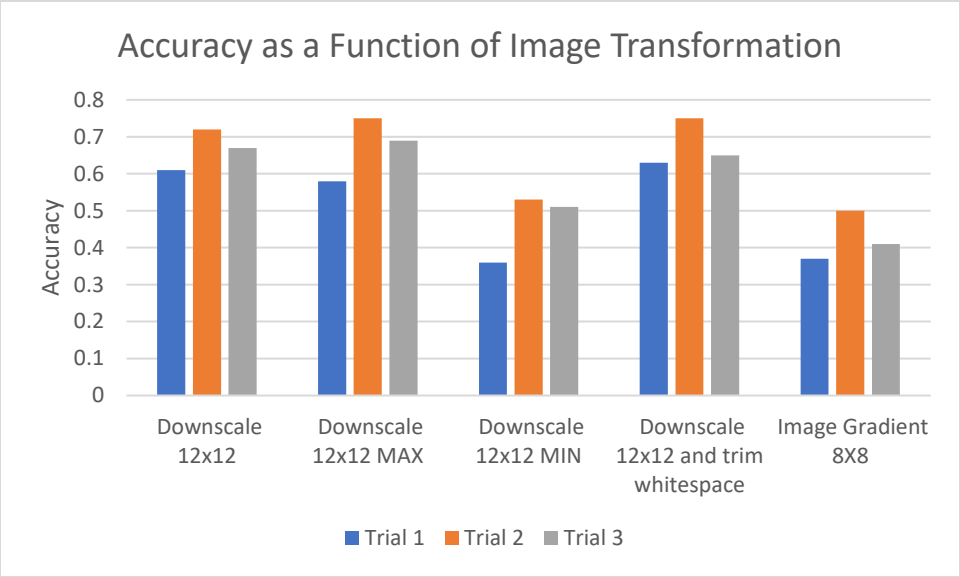
trial	train examples	distance metric	**transformation**	k	accuracy
1	2000	Euclidian distance	**Original 25x25**	1	0.53
1	2000	Euclidian distance	**Downscale 12x12**	1	0.61
1	2000	Euclidian distance	**Downscale 8x8**	1	0.61
1	2000	Euclidian distance	**Downscale 5x5**	1	0.51
1	2000	Euclidian distance	**Downscale 4x4**	1	0.48
1	2000	Euclidian distance	**Downscale 2x2**	1	0.15
2	2000	Euclidian distance	**Original 25x25**	1	0.71
2	2000	Euclidian distance	**Downscale 12x12**	1	0.72
2	2000	Euclidian distance	**Downscale 8x8**	1	0.69
2	2000	Euclidian distance	**Downscale 5x5**	1	0.64
2	2000	Euclidian distance	**Downscale 4x4**	1	0.54
2	2000	Euclidian distance	**Downscale 2x2**	1	0.24
3	2000	Euclidian distance	**Original 25x25**	1	0.63
3	2000	Euclidian distance	**Downscale 12x12**	1	0.67
3	2000	Euclidian distance	**Downscale 8x8**	1	0.67
3	2000	Euclidian distance	**Downscale 5x5**	1	0.6
3	2000	Euclidian distance	**Downscale 4x4**	1	0.59
3	2000	Euclidian distance	**Downscale 2x2**	1	0.29



Accuracy seems to be the highest using a 12x12 image. This is likely because a 12x12 image is just large enough to convey important information about which character class is present, but also small enough to allow ample overlap between characters of similar classes.

Modify Downscaling Method

Trial	Train examples	Distance metric	**Transformation**	K Value	Accuracy
1	2000	Euclidian distance	**Downscale 12x12**	1	0.61
1	2000	Euclidian distance	**Downscale 12x12 MAX**	1	0.58
1	2000	Euclidian distance	**Downscale 12x12 MIN**	1	0.36
1	2000	Euclidian distance	**Downscale 12x12 and trim whitespace**	1	0.63
1	2000	Euclidian distance	**Image Gradient 8X8**	1	0.37
2	2000	Euclidian distance	**Downscale 12x12**	1	0.72
2	2000	Euclidian distance	**Downscale 12x12 MAX**	1	0.75
2	2000	Euclidian distance	**Downscale 12x12 MIN**	1	0.53
2	2000	Euclidian distance	**Downscale 12x12 and trim whitespace**	1	0.75
2	2000	Euclidian distance	**Image Gradient 8X8**	1	0.5
3	2000	Euclidian distance	**Downscale 12x12**	1	0.67
3	2000	Euclidian distance	**Downscale 12x12 MAX**	1	0.69
3	2000	Euclidian distance	**Downscale 12x12 MIN**	1	0.51
3	2000	Euclidian distance	**Downscale 12x12 and trim whitespace**	1	0.65
3	2000	Euclidian distance	**Image Gradient 8X8**	1	0.41



The two image transformations that clearly decreased accuracy are downscale 12x12 MIN and Image Gradient 8x8. The downscale 12x12 MIN likely did not work because using the minimum pixel value at each filter likely causes a significant amount of information to be lost. The image gradient likely did not work because if the character in the image is shifted to the right or left slightly by a couple pixels, the gradients may undergo a disproportionately drastic change, due to each gradient calculation only focusing on a very small portion of the image. Downscale 12x12, Downscale MAX, and Downscale 12x12 and trim whitespace all performed about the same. Using MAX and trim whitespace did not cause an increase in accuracy likely because there is no information that is actually gained by doing these extra steps.

Based on the results of these tests, the most optimal combination of hyperparameters selected is

10,000 training examples

k=1

Downscale to 12x12

Minkowski distance u=4

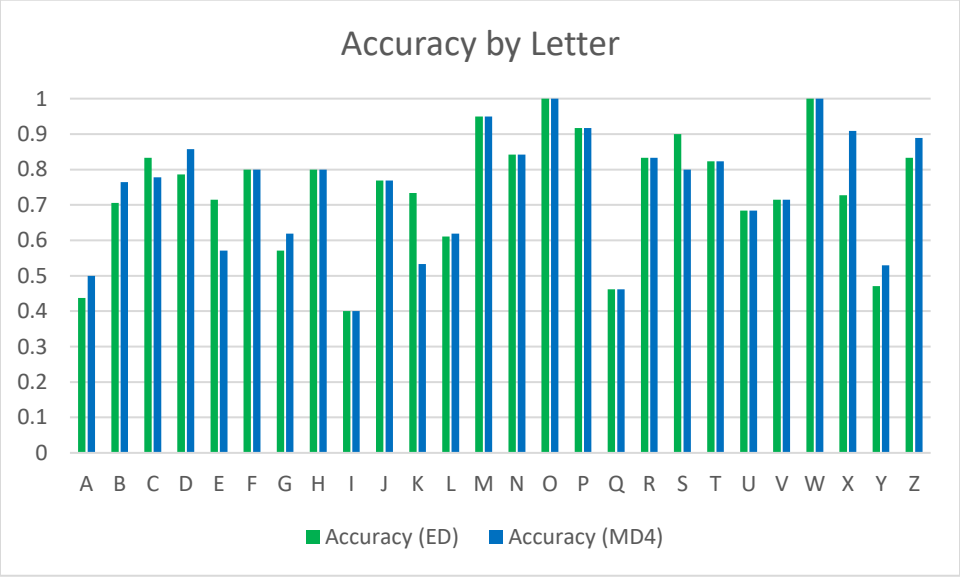
With this combination, the average accuracy is 0.7833333333333333. With even higher numbers of training data, the accuracy increases further, but at the cost of performance.

Other Experiments

Other experiments which yielded poor or unclear results were:

- Weighted Euclidian distance
- Modifying images so that 0 or 255 are the only greyscale values present
- Converting each white pixel into a pixel with a random value
- Finding the total distance by summing each x-y-grid distance, for each dark pixel, to the closest dark pixel in the opposing image

Accuracy of Each Letter



This graph shows the accuracy (on the vertical axis) when classifying each different letter. One interesting pattern is that "Q" has a very low accuracy, while "O" has a very high accuracy. It is possible that this is due to the fact that "Q" looks very much like "O," and that causes the system to misclassify many characters as "O" when they should have been classified as "Q." Also, "I" has an exceptionally low accuracy, while "L" has a mediocre accuracy. This is likely due to many Ls and I's being misclassified as the other. Finally, an interesting trend is that Minkowski Distance significantly improved the accuracy for "X." Minkowski distance is different from Euclidian distance because Minkowski distance (of 4^{th} degree) over-punishes high distances, and under-punishes low distances. It may be the case that Minkowski

distance increases the accuracy for letters, such as “X,” that contain a central point where many lines meet. The Minkowski distance function may be using the central point to ignore small differences created by small shifts or rotations in the test image.

Conclusion

With an accuracy of almost 80%, the nearest neighbor algorithm could be considered a viable option for handwritten character classification. However, for character classification, the nearest neighbor algorithm will have difficulty competing against other machine learning models such as the convolutional neural network, which is specifically designed for images. However, KNN could still be advantageous in some situations. For example, a KNN algorithm could be very effective at converting an image of a computer-printed document to words, as computer-printed characters are always identical, which means that nearest neighbor will be very effective.

My contributions to this project are in constructing the KNN classifier as well as finding the optimal training data size, K value, image transformation, and distance metric. Also, analysis was done by me on the accuracies of each letter.

Considering the results of this study, a possible future study could involve exploring the viability of using nearest neighbor for processing images of computer-printed documents. For this task, nearest neighbor would likely be more efficient than other types of models, due to the monotony of computer-printed characters. If this future study is successful, it could result in a significant reduction in required computer processing time for scanning in printed documents compared to doing the same task with a convolutional neural network.