Hand Digit Recognition through Statistical Features

Bart Joseph J. Nadala
Seattle University, Master of Science in Computer Science
nadalab@seattleu.edu

Abstract

Hand writing is difficult enough to read depending on the writer but poses a more difficult problem when being digitalized. In our current age, we now strive to automate everything. Hand digit recognition is a crucial aspect of digitalization that requires the utmost accuracy and consistency due to its critical nature. This technology is most sought after by banks for recording exact amounts from checks, postal offices for sorting mail by postal code, and any office seeking to digitalize their hand-written records. This paper seeks to shed light in ways statistical features can be extracted, manipulated, and used to determine proper classification. The steps involve pre-processing the image, statistical features extraction, then finally classification.

Introduction

Writing by hand has become a necessary component of human communication, documentation, and learning. It is, however, being progressively replaced by contemporary technologies like keyboards and typed media because writing by hand is oftentimes difficult and incompatible with complex computations and processing. It seems as though hand writing's purpose has been reduced to a type of technological buffer between those adept at digital communication and those favoring the literal feeling of writing by hand. Simply put, digital records are better in most aspects. There is a clear need to quantify script into its digital equivalent in order to merge both worlds without linear repetition. To address this problem, it would be useful to convert digits in script into its digital equivalent. Areas where this would be useful are bank checks to digitize the amount, postal codes in mail for automated sorting, and forms/answer sheets/questionnaires digitalization to name a few. This is necessary to streamline document control and handling in relevant businesses more efficiently. The goal of this paper is to demonstrate the use of statistical features as a means of recognizing hand written numeral digits. Specifically, it seeks to explain the process of processing the image, identifying features, and classifying them into digits or others.

The data used for this classification method is the MNIST handwritten digit database comprised of 60,000 training set examples and 10,000 test set examples as grey scale 28x28 pixel images. The MNIST database is a subset of the NIST database which have been normalized into a 20x20 pixel image fit into a 28x28 image and centered by its center mass. Using this data, the image is subjected to a pre-processing phase of conversion into a binary image, noise removal, spurring, and thinning; it then moves to the statistical feature extraction phase. Statistical features are features that can be identified using simple image manipulation. Examples of these statistical features are number of foreground pixels, width-height ratio, feature points, end points, branch points, and cross points to name a few. Once it is finished with the features extraction phase, it moves on to the classification phase where it classifies the image using K-Nearest Neighbor classifier or a more deterministic approach where the relevance of the features classify the image. By this point, the image is classified then compared with the data's actual nature then used for statistical accuracy.

Related Works

Three papers were researched to find an implementation for recognizing hand written digits. "Handwritten Digit Recognition Using Structural, Statistical Features and K-Nearest Neighbor Classifier" by U. Babu, A. Chintha, and Y. Venkateswarlu is the primary inspiration for this paper. It talks about using statistical features taken from pre-processed images to classify the image via KNN. Another paper, "A Statistical Approach For Latin Handwritten Digit Recognition" by I. Zaqout, also uses statistical features but directly indicates which features determine a digit.

Although both papers generally used different approaches, they demonstrated the same process of pre-processing, features extraction, and classification. Both selected papers demonstrate some differences with this pro-

cess. However, their main difference is their means of classification. Where one paper used KNN to classify the statistical features as vectors, the other used a more deterministic approach where the features were used to identify possibilities and converged into a result. The KNN approach showed clear dominance in terms of accuracy with a successful identification of 98.42% versus 92.90% for the deterministic approach. The downside demonstrated by both papers was that they did not explicitly note their process for classification. The KNN paper went as far as to note that classifiers are more of an art rather than a science which is understandable but difficult to replicate.

They both moved towards the same goal of using statistical features to identify hand written numerals rather than the more successful method of Neural Networks as proposed by this paper "Simplified Neural Network Design for Hand Written Digit Recognition" by Z. Asghar, H, Ahmad, S. Ahmad, S. Saqib, B. Ahmad, and J. Asghar which, in the same manner as the first paper mentioned, classifies images with a training set to classify test images. This paper uses neural networks which is a system of gates to determine a score for an image which best classifies the test image. This particular implementation does not use statistical features analysis but rather the state of the image itself.

Between these three papers, one can never really tell which is the perfect solution despite knowing their accuracy and performance. This is primarily due to there being too much disparity between the different ways to write. It is not as if they are all unacceptable, as a matter of fact they all perform well enough for all practical uses.

Database and Implementation

The dataset most commonly used for hand digit recognition software is the MNIST database which is a sub-collection of the much larger NIST database. The MNIST dataset is comprised of 60,000 training images and 10,000 test images. These images come in 28x28 grayscale format as an IDX file. They are also accompanied by another IDX file that contains the intended classification of the images.

The implementation is a three-step process of image pre-processing, statistical features extraction, and classification. Pre-processing is done via image filtering. The original grayscale image is converted into a binary image then filtered again for noise removal. The image also goes through thinning later in the process but not before extracting certain features that can only be extracted with a clean binary image. Features extraction is the step in the process where statistical features of the image are saved for classification. These features are image measurements such as area, height, width, width-to-height ratio, distribution features, and skeletal features. The last step of the process is classification. This paper will use the linear implementation of K-Nearest Neighbor for classification. The statistical features are used to compute for Euclidean images between the training dataset and the test image. The k-closest Euclidean comparison from the training dataset to the test image will be used to classify the image. Trial and error will determine K and whichever K can produce the highest accuracy will be selected for further classification. The overall accuracy of this implementation will be determined by comparing the correctly classified images to the total number of test images. The sequential list of steps is as follows:

Step by Step Process of the Implementation

1. Extract all 60,000 training dataset and 10,000 test dataset and construct an appropriate 2D array paired with the accompanying labels file to determine intended classification;
2. Convert every image into binary;
3. Remove Noise;
4. Extract Area, Height, Width, and Width-Height Ratio;
5. Horizontally and Vertically divide the image in half then extract the area of the Upper Left Quadrant, Upper Right Quadrant, Lower Left Quadrant, Lower Right Quadrant, Upper Area, Lower Area, Left Area, and Right Area.;
6. Inverse Area, Height, and Width;
7. Perform Thinning on every image;
8. Extract Horizontal and Vertical Crossings, End Points, Branch Points, and Cross Points;
9. For every test image, take the Euclidean distance to every training dataset and save the K-Nearest Neighbors;
10. Count most common classification from the K-Nearest Neighbors then classify the test image;
11. Compile overall accuracy based on resulting classification and total test images;
12. Re-compute K-Nearest Neighbors by finding highest accuracy while incrementing K;
13. Resulting accuracy will determine K and best accuracy possible with indicated method.

Pre-Processing Stage

The initial state of the image is a grayscale 28x28 image provided by MNIST database. The image itself is a centered version of the original NIST image which is also grayscale but sized 20x20. Since the image is centered, it produces more useful and consistent features that are better for classification. These images also come with an intended classification to allow the user to test and compute for accuracy when selecting their classification process.

This paper uses three types of image filters for pre-processing. From the initial grayscale through a binary filter, noise filter, then finally thinning.

Conversion from Grayscale to Binary Image

This image filter reduces every pixel in the image to either 1 or 0. It does so by comparing the grayscale level pixel to a threshold to determine if it should be a 0 or a 1. The purpose of this filter is to make the image more deterministic when features are extracted. It makes for easier extraction since the images are clearer.

For the purpose of this paper, the threshold is set to exactly half so that any pixel with grayscale level between 0-128 is 0 and any pixel with grayscale level between 129-255 is 1. This clarifies the image and eliminates any blurring that may be present.

Noise Removal

The next step after converting the image into binary is to clear any excessive noise that may be present. This increases the consistency of the image but may also prove to be problematic when the image is not clear enough or too thin to begin with. This problem is the more common difficulty that is mentioned in other hand digit classification research.

Noise removal is a filter that compares the pixel value to its neighbors. Let p0 = (x, y) be a pixel in the image. The filter takes into consideration p1 = (x-1, y), p2 = (x-1, y+1), p3=(x, y+1), p4=(x+1, y+1),p5=(x+1, y), p6=(x+1, y-1), p7=(x, y-1), p8=(x-1, y-1). The neighbors will determine whether the 1 in the center will be converted to 0 but never from 0 to 1. In this implementation, single to four pixel combinations are removed as noise. Single pixel is when p0=1 and all neighbors are 0. Two-pixel combination is where p0=1 and there exists exactly one more 1 within p1 to p8. Three-pixel combination is where p0=1 and there are exactly two more 1s within p1 to p8. Four-pixel combination like the others, also requires p0=1 and exactly three more 1s within p1-p8. One and two pixel combinations are removed as stated, however, for three and four pixel combinations. The exact pattern shown in Figure 1 is used.

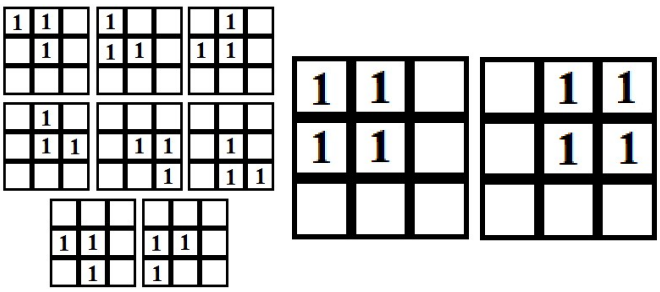

Figure 1*: Three and Four Pixel Combinations for Removal*

Thinning

This image filter is used to reduce the image to a 1-pixel thickness version of the binary image. It, like noise removal, is also a filter that removes pixels relative to its neighbor's combination. The resulting image is used to extract skeletal features explicitly.

The algorithm used to employ this filter is Zhang-Suen thinning algorithm. It is a simple two-pass filter that repeats when a change is made. This filter does not scale well because it requires multiple processing of the entire image sequentially. This paper implements it using the same neighbor definition as in noise removal. Its pseudocode is as follows:

- A(p0) = number of 0 to 1 transitions in the ordered sequence of p1, p2, p3, p4, p5, p6, p7, p8, p1.
- B(p0) = p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8 (number of black or 1 pixel, neighbors of p0).
- Condition 1: 2 <= B(p0) <=6;
- Condition 2: A(p0) = 1;
- Condition 3: p2 * p4 * p6 = 0;
- Condition 4: p4 * p6 * p8 = 0;
- Remove pixels that fulfill the condition from the image then continue (this is the first pass);
- Condition 1: 2 <= B(p0) <=6;
- Condition 2: A(p0) = 1;
- Condition 3: p2 * p4 * p8 = 0;
- Condition 4: p2 * p6 * p8 = 0;
- Remove pixels that fulfill the conditions from the image then continue (this is the second pass);
- Repeat first pass when a pixel was changed from 1 to 0 in either of the two passes;

This algorithm finishes when no change was made after two passes. Figure 2 shows the resulting image when thinning is applied to a noise reduced binary image.

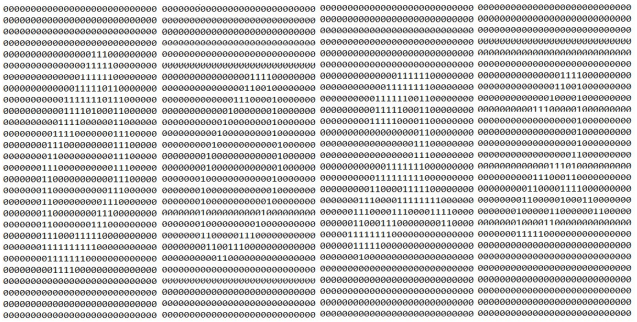

Figure 2: *Binary (left) to Thinned Version (immediate right).*

Statistical Features Extraction

The way a person usually identifies a number involves an instantaneous identification of multiple features. For example: the number one is easily identified by the vertical line with varying small marks at the top and bottom of the line. The number zero is also easily identified as a circle that may or may not be full due to obvious problems with handwriting. Statistical features work relatively the same way. They

are features that can be extracted from hand written digit images.

This paper utilizes 17 distinct features that can be identified after multiple image filtering and minor rearrangements. They are as follows: area, height, width, width-to-height ratio, distribution features, and skeletal features. These features separately make up the dimensions for Euclidean distances between test and training images. The first (4) features can be extracted after the image has been converted to binary and filtered for noise.

Foreground Area (1)
Area is defined as the total number of foreground/black/1 pixels in the image.

Height (2)
The height is defined as the bottom most (larger value) foreground pixel subtracted by the top most (smaller value) foreground pixel.

Width (3)
Width is extracted in the same manner in that it is the right most foreground pixel subtracted by the left most foreground pixel.

Width-to-Height Ratio (4)
The resulting height and width are used to compute for the width-height ratio that is useful for identifying the number 1 for example.

Area, height, and width being larger numbers compared to the rest of the features are individually inversed to reduce its overwhelming effect on the Euclidean distance. This is due in part to the KNN classifier being implemented linearly. Via trial and error, this change to these features increases the accuracy by 10-20% on all numbers.

Distribution Features

Distribution features are features regarding the spatial distribution of the pixels relative to the total area. The features are as follows: upper left quadrant, upper right quadrant, lower left quadrant, lower right quadrant, upper area, lower area, left area, right area. These features totals to (8) individual features ranging from 0 to 1.

Quadrant (5-8)
The four quadrants are divided at the center of the height and width of the image. Every pixel within the quadrant will be counted towards the area it is in.

Halves (9-12)
The remaining four areas divide the space in half; horizontally between the upper area and lower area and vertically between the left area and right area.

Once computed, every one of these features are then divided by the original non-inverse area such that adding all quadrants totals to 1, adding upper and lower area totals to 1, and adding the left and right area also totals to 1. Like the first four features, these features are made extractable after converting the grayscale image to binary then removing noise. All twelve of these features must be extracted before thinning the image to better represent the features. However, the next set of features require thinning for them to be of use when extracted.

Skeletal Features

These statistical features represent the form of the image much more than the previous twelve. Like noise removal and thinning, these features also utilize the same definitions for neighbors. Neighbors are used to identify the type of point the pixel is in the image.
The type of point being a statistical feature that is counted in every image. These features are vertical and horizontal crossing, endpoint count, branch point count and cross point count.

Vertical and Horizontal Crossings (13-14)
Vertical crossing is a vertical line at the center of the image. Any pixel in contact with the line, assuming proper thinning was processed, will be counted. This would be helpful to differentiate between 7 and 9. Horizontal crossing is the same. However, the line crosses horizontally instead.

End Points (15)
The end points are pixels whose total number of foreground neighbors is exactly one. This indicates that there is an edge to the number. This would prove to be a very crucial feature if the image is perfectly thinned.

In practice, however, due to the differences in thickness, stroke, and style between the same numbers, this leads to gaps in the number. For example: the number 0 may have been written very thinly at the top and bottom, resulting in gaps after being converted to binary and cleaned of noise. The end point count of this number would result in four end points rather than the expected zero which is too large, an error gap to make endpoints reliable. Such problems are also common with the number 5 on the downward line, number 8, and to some extent number 4, due to the extra tip at the right size and the well-known gap at the top.

Branch Points (16)
Like end points, branch points rely on the neighbors of foreground pixels. In this case, branch point is a pixel with exactly three foreground neighbors. These points come in handy with numbers like 4, 6, and 9 that have sections with three lines connected. In terms of reliability however, the improper thinning on the image poses the same problem as end points.

Cross Points (17)
Cross points are foreground pixels with exactly four foreground neighbors. These points stand out well in numbers 4 and 8 with the same drawbacks as end points and branch points.

With this last feature, the images can now be classified via linear KNN Euclidean distance classifier.

Classification

Classification is the most important step of the three stages. It determines whether the statistical features will be of use or will further deter proper classification. This is also known as the most difficult part of the process to correctly implement due to the ambiguity of the relationships between the selected features, images, and unintentional classification of poor training samples. Many papers also claim this part to be the most difficult as it is simply a matter of trial and error to best fit the classifier to the dataset being tested and trained with.

This stage of the process classifies the test images based on the training set using k-nearest neighbors to identify the most appropriate training image to classify. The closest Euclidean distance between the test image and every training image will be used to compare the datasets.

K-Nearest Neighbor Classifier

This classifier takes the k-nearest images based on the Euclidean distance of the statistical features previously extracted and counts the most common classification to determine the best suited result. To determine the best K for the dataset, trial and error will be used to find the highest accuracy allowable by the selected statistical features and comparison. It has been noted in other papers however that the more accurate the statistical feature and comparison, the closer K goes to 1 for the best accuracy.

Results

The results ultimately show an accuracy of 66.75% using k=8. The resulting accuracy plainly shows that some combination between the three implemented steps were not cohesive in terms of implementation or selection.

During implementation, certain features showed better accuracy than others. Skeletal features showed an average of 50% accuracy by themselves. Inverting the value for the first three features also made a huge difference as it increased the accuracy of the classification from 39% to its current accuracy of 66.75%. This just goes to show how important the relationship is between every statistical feature used to the classifier that uses them.

Refer to Figures 3 and 4 for more statistical results. Figure 3 displays the results when computing for the best K to find the best accuracy with respect to the method and dataset. Figure 4 displays the results computed utilizing the best K and therefore displays the best overall result for each number.

K	Result	Intended	Accuracy
1	6514	10000	65.14
2	6417	10000	64.17
3	6596	10000	65.96
4	6663	10000	66.63
5	6665	10000	66.65
6	6674	10000	66.74
7	6672	10000	66.72
8	6675	10000	66.75
9	6660	10000	66.6
10	6654	10000	66.54

Figure 3*: Trial and Error Result on Best Value for K.*

Number	Result	Intended	Accuracy
0	858	980	87.55 %
1	1060	1135	93.39 %
2	545	1032	52.81 %
3	637	1010	63.07 %
4	744	982	75.76 %
5	479	892	53.7 %
6	597	958	62.32 %
7	678	1028	65.95 %
8	541	974	55.54 %
9	536	1009	53.12 %
K: 8	6675	10000	66.75 %

Figure 4: *Computed Accuracy Per Number Classification and Total Result with Best K Selected.*

Conclusion

The result clearly shows that this implementation of the k-nearest neighbor is inaccurate but by proof of concept and other research it is possible to find better accuracy. The first paper mentioned shows exemplary accuracy but an argument can be made that it performs well due to its constraints on the data that was used. The paper used only 5,000 training images and 5,000 test images from the NIST dataset. It also mentions having used specific image sets that properly demonstrate the image of the digits.

This shows that this implementation's weakness may lie with overtraining. The training set used is 60,000 images of varying writers that range from professionals to students. This forces the classifier to over-estimate its classification by relying too much on images with rather low probability of accuracy, much like how false information is given by those without proper knowledge to those who are eager to

listen. This problem however cannot be easily fixed because the classifier is also required to identify hand writing of poor nature.

In this implementation, it can also be surmised that there are many faults with the utilization of the statistical features. Perhaps they should be made more proportionate by dealing with percentages rather than raw value. The key here would be to present the statistical features relative to how well they work as a classifier. This can be observed by simply testing every feature and combining them systematically until they show improvement of accuracy. This method however can only be suited for the current data at hand and may not perform as well with images of different writing. Hence, appropriate identification of statistical features is necessary.

This is the case with the inversion of the statistical features: area, height, and width whose raw values make too big of a difference with the Euclidean function. They do not demonstrate the proper accuracy when identifying classifications. Its is not the case that they cannot and should not be used but instead need to be manipulated better to improve classification.

In terms of improving the implementation, different classifiers may also be tested using the same statistical features or either add or reduce them. In basic terms, playing around with how they are used. Trial and error always plays a large role in any scientific field and that proves true for this study as well.

References

[1] U. Babu, A. Chintha, and Y. Venkateswarlu, "Handwritten Digit Recognition Using Structural, Statistical Features and K-Nearest Neighbor Classifier", *International Journal of information Engineering and Electronic Business*, Vol. 6, No. 1, pp 62-68, February 2014

[2] I. Zaqout, "A Statistical Approach For Latin Handwritten Digit Recognition", *International Journal of Advanced Computer Science and Applications*, Vol. 2, No. 10, pp 37-40, October 2011.

[3] Z. Asghar, H, Ahmad, S. Ahmad, S. Saqib, B. Ahmad, and J. Asghar, "Simplified Neural Network Design for Hand Written Digit Recognition", *International Journal of Computer Science and Information Security*, Vol. 9, No. 6, pp. 319-322, June 2011