Digit Recognition Project Report

Delafrouz Mirfendereski
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
dmirfenderes@cs.stonybrook.edu

Ashkan Banitalebi Dehkordi
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
abanitalebid@cs.stonybrook.edu

Abstract

The problem for this project is to recognize images of handwritten digits. We have tried methods such as k-means and SVM and reached different accuracies of more than 93%. There are still challenges to be solved and improvements to be made, some of them include the big size of data and increasing the achieved accuracy.

1 Introduction

This project is about recognition of handwritten digits. Every data instance is a black-and-white 28×28 picture. Each pixel is an integer from 0 to 255, with 0 meaning black pixels and 255 meaning white pixels.

1.1 Data

The training dataset is a 42000×785 matrix. Each row represents one data point, the first number is the label, an integer from 0 to 9; and the rest of 784 numbers show the amount of each picture's pixels, from the top-left to the bottom-right corner.

The test set is a 28000×784 matrix. Each row represent one picture and there are no labels for the test images.

1.2 Accuracy

We get the accuracy of the estimated test labels through Kaggle evaluations, by submitting a csv file consisting of image IDs and their labels.

1.3 Challenges

We faced some challenges in this project, including:

- Big train data set: when using SVM, $n \times n$ kernel matrix exceeds RAM space.
- Shift invariant: our algorithm must not depend on the place of the digit in the picture, Figure 1-a.
- Rotation invariant: our algorithm should also be rotation invariant but not too much so that 6 and 9 are mistaken, Figure 1-b.
- Recognition difficulty: some pictures are hardly recognized with human eye, Figure 1-c.
- Low quality of pictures: makes it hard to detect lines or corners which we will discuss later.
- Long run-times: some parts of the algorithm needed long run-times, which in some

cases took longer than a day.

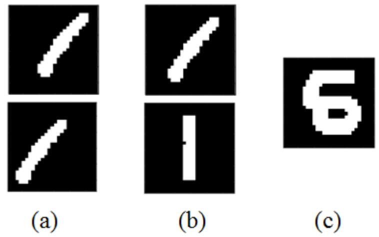


Figure 1: (a) A shifted digit. (b) A rotated digit. (c) A hard number to recognize.

2 Approach

Lots of machine learning algorithms and image processing methods can be used for this problem. Our goal was to apply some methods and compare the results. Suggested methods are k-means and k-NN[1], SVM, PCA, cross validation, edge detector, corner detector, line detector, etc.

2.1 K-means and k-NN

The first implemented method was a combination of k-NN and k-means. We did this to get a better perception of different aspects of this problem, such as the data itself and the accuracy to expect.

First we measure the centroids for each class of the train data points (centroids are average of the points in one class) and then assign each test point to the nearest centroid based on Euclidean distance. This approach had an 80.614% accuracy for estimated test labels.

Although the data points are 784-dimentional, this algorithms was fast and did not need lots of memory.

2.2 SVM

Next we used libsvm with linear kernel. There were two big challenges here, the first one was the big volume of the data and the second one was long run-times.

The first problem was that there are 42000 train points, therefore we needed a 42000×42000 kernel. This matrix exceeded the 8 GB RAM (the one we ran the program on) and therefore we were forced to sample the training data. Another solution would be hard example mining, which we could not use here because of the long run-times that is mentioned next.

To get the best performance of SVM, we had to tune the parameters C and γ of the linear kernel for every set of train data. This means that when we wanted to change the number of data samples, or use some kind of features instead of the exact amount of pixels as a train data point, we had to tune them again. There were two problems here:

1. Wide ranges for C and γ: for different types of features, we found wide ranges for the best C and γ ($[2^{-5}, 2^{15}]$ and $[2^{-15}, 2^{-1}]$ respectively). This made it impossible for us to estimate a rough amount for C and γ and we had to search in the range every time.
2. 5-fold cross validation: in order to get an accuracy for different amounts of C and γ, we used 5-fold cross validation on the train data.

These two difficulties caused long run-times for tuning C and γ. Therefore we were able to tune these parameters for only a few sets of features we used, and for others we used the same C and γ, this caused in lower accuracy in some cases.

We used SVM for different experiments, where we used the picture itself as the data point or a vector of feature or a combination of features instead of the picture, each is explained below:

[1] k nearest neighbors

2.2.1 SVM on the exact data

At first we chose 5000 pictures at random, 500 of each digit. We did not apply any feature extraction method at first, and each data point was a 1×784 vector of the pixel amounts. Using these samples, we tuned the amounts of C and γ for linear kernel with cross validation. The result is shown in Figure 2. Using the best parameters, we reach an accuracy of 83.78% in train points and 82.543% in test points. We used this set of parameters for many other experiments (different amount of samples or different feature extraction methods). The same amounts for C and γ resulted in 83.614% and 74.571% accuracy on 10000 and 20000 samples, respectively, with no feature extraction method applied.

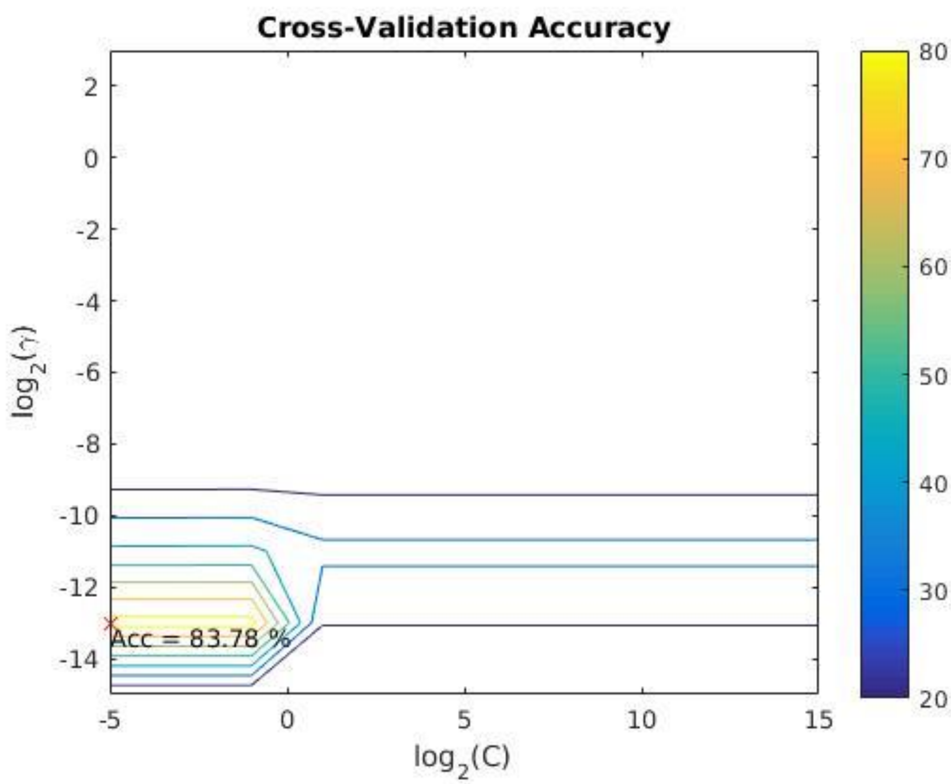


Figure 2: contours of cross validation accuracy over a variety of parameters C and γ

2.2.2 Feature extraction: number of white pixels

As a simple feature extraction method, we counted the number of white pixels in each row, each column, 49 squares of 4×4 pixels, and 24 15°-angles with vertex at the center of the picture (Figure 3 shows how we roughly divided the 28×28 picture into 24 different zones). We did not tune parameters for this set of train data and using $C = 2^{-5}$ and $\gamma = 2^{-13}$, we reached a accuracy of 67%.

2.2.3 Feature extraction: HOG

As we mentioned earlier, the orientation of the image is not important and we should be able to recognize the same digit from different angles. Histogram of Oriented Gradient (HOG) is both good for being rotation invariant and object detection. We used hog_feature_vector library.

We used this feature extraction method both in combination with other features and not, with different sets of amounts for C and γ. The results are shown in Table 1.

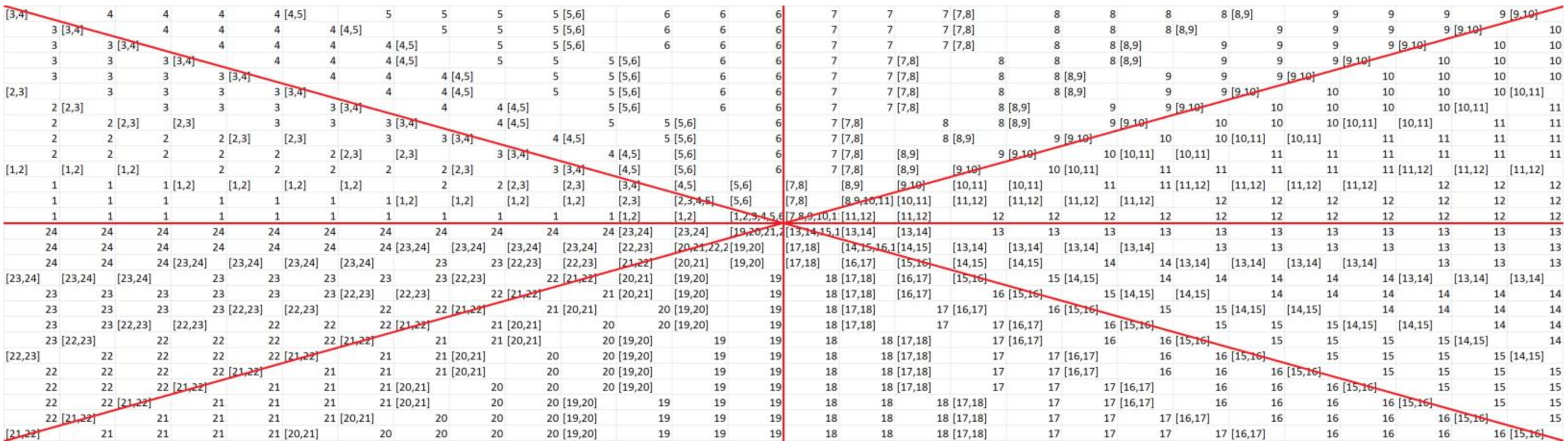
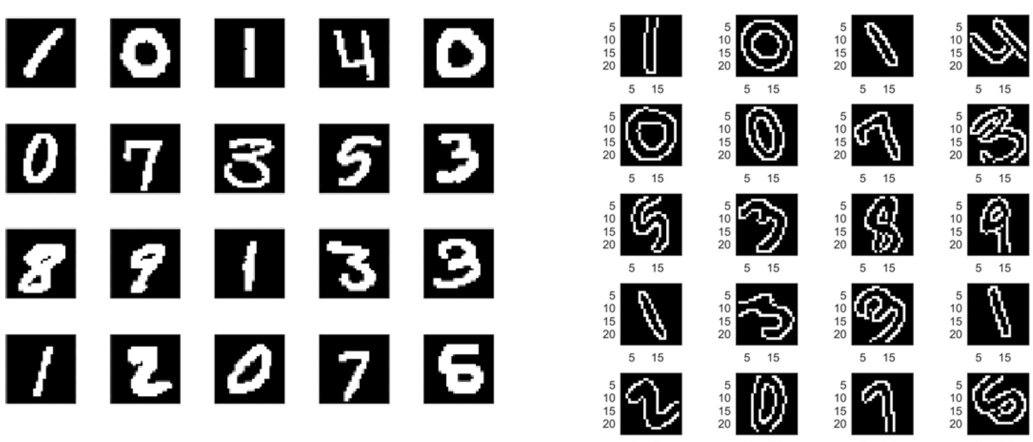


Figure 3: the division of a picture into 24 different zones. Each zone is a 15°-angle starting from the center of the picture. Each cell represents a pixel and the numbers written in it, roughly show what zones that pixel belongs to.

2.2.4 Feature extraction: Canny edge detector

Edges are the places in picture where a significant change in color or pattern happen. We can find edges using gradients. The most famous method for edge detection is Canny edge detector [1] which consists of 5 steps:

1. Smoothing: using a Gaussian filter to remove the noise.
2. Find gradients: edges are where image pixels have changed significantly. Gradients show the change in pixels.
3. Non-maximum suppression: edges are where there is a local maxima in gradient.
4. Double thresholding: to find the main edges, two thresholds are chosen, one for strong edges and one for weak ones.
5. Edge tracking by hysteresis: long edges connected by weak edges result in main edges, other weak edges are removed.



(a) (b)

Figure 4: applying Canny edge detector on a set of 20 pictures. Each picture is rotated 33° rotated to the left due to a reason that is discussed in line detection section.

Together used with Hough transform, which is discussed later, we can also find the lines in one picture.

By using *edge* function in MATLAB with argument *'canny'*, one can extract the edges in an image. We have used this feature extraction method alone and in combination with other ones, with different sets of amounts for C and γ. The results are shown in Table 1.

2.2.5 Feature extraction: Hough transform

Hough transform [2] is mainly about detecting simple shapes, such as straight lines, circles or ellipses. One trivial way to do this is by grouping a set of edge features to an appropriate set of lines, circles or ellipses. In practice, this cannot be done due to imperfections in the picture or the edge detector, noise, missing pixels or spatial deviations between the ideal line/circle/ellipse and the edges. The goal of Hough transform is to solve this problem. [3]

The simplest case of Hough is line detection. Lines are represented as $y = mx + b$ or simply an ordered pair (m, b) but we face a problem in case of vertical lines. Therefore in Hough transform, we represent lines as $r = x\cos\theta + y\sin\theta$ where r is the distance from the origin to the closest point on the straight line, and θ is the angle between the x axis and the line connecting the origin with that closest point.

Hough function in MATLAB returns an $m \times t$ matrix, where t is the number of different amounts for θ and m is the number of different amounts of r which depends on θ. Here, in order to avoid a big matrix for result which complicates the computations, we used the range $[-60{:}\,1{:}\,60]$ for θ which resulted in range $[-33{:}\,1{:}\,33]$ for r.

2.2.6 Feature extraction: line detection using Canny edge detector and Hough transform

In order to find lines in an image, first we have to apply Canny edge detector on it and then Hough transform. Using *houghpeaks* function in MATLAB, we find the peaks in the result matrix of Hough transform and then apply *houghlines* function on the result, given two arguments of *FillGap* and *MinLength* which represent the maximum number of missed pixles and the minimum length of a line, respectively.

The main challenge here is caused by the low resolution of the images. As a result, we have to choose a small amount for *MinLength* which leads to finding straight lines in numbers such as 0 or 3 which we do not expect in most cases. Also because of the fact that we chose the range of θ from $-60°$ to $60°$, the algorithms could not find horizontal lines (which are important in case of images of numbers) so we rotated the images by an angle of $33°$ at the first place.

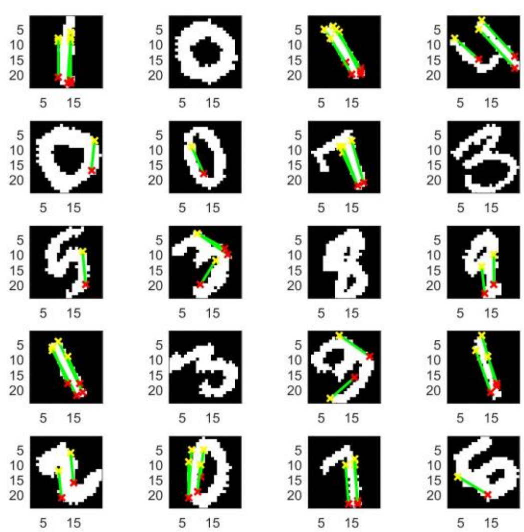


Figure 5: detected lines in some images using Canny edge detector and Hough transform.

In contrary to our expectation, line detection was not a good feature extraction for digit recognition. We used it in combination with HOG and corner detection feature, which both turned out to be really good for digit recognition, but reached an accuracy of only 10.14% for train data with $C = 2^{-5}$ and $\gamma = 2^{-1}$. This may be a result of low resolution, or an imperfect presentation of line (which is {first point, last point, θ, r}).

2.2.7 Feature extraction: corner detection

There are two main algorithms to find corners in a picture, Harris method [4] and Shi-Tomasi method [5]. Both methods follow almost the same procedure. At first, derivatives in both x and y directions are computed in a sliding window over the image. Then for each place, the eigenvalues of the derivative matrix in that window are computed. Three cases can happen:

1. Both eigenvalues are small: a flat area
2. One eigenvalue significantly larger than other: an edge
3. Both large: a corner

The only difference in the two methods mentioned above, is how they define large eigenvalues. In Harris method a threshold is used for the amount $\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$, where λ_1 and λ_2 are the eigenvalues, but in Shi-Tomasi method the used formula is: $\min(\lambda_1, \lambda_s) > threshold$.

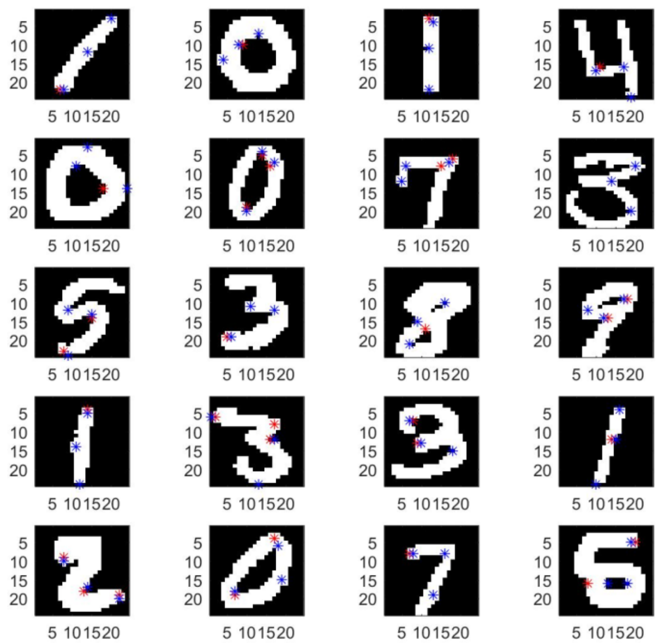


Figure 6: the result of Harris and Shi-Tomasi method on 20 images. The blue marks show corners found by Harris method and the red ones show corners found by Shi-Tomasi method.

As shown in Figure 6, these two methods result in different points chosen as corners. In order to emphasize on their similarities and weaken the differences, we used both methods here: 3 corners found by each method were put together in a vector (we used the [0,0] point in case an image had less than 3 corners). With a feature vector of length 12 for each train and test image, we reached an accuracy of 92.857%, which is our second best result.

3 Results

We used many methods and many types of feature extraction, on different number of samples with different values for C and γ. The complete results are shown in Table 1.

Number of train samples	Kernel	C and γ	Image itself	Sum of white pixels	HOG	Canny	Hough	Lines	Corners	accuracy
5000	Linear	$2^{-5}, 2^{-13}$	+							82.543
5000	Linear	$2^{-5}, 2^{-13}$		+						67
5000	Linear	$2^{-5}, 2^{-13}$			+					71.843
5000	Linear	$2^{-5}, 2^{-13}$				+				71.514
5000	Linear	$2^{-5}, 2^{-13}$			+	+				72.186
5000	Linear	$2^{-5}, 2^{-13}$	+		+	+				82.543
5000	Linear	$2^{-5}, 2^{-13}$					+			72.057
5000	Linear	$2^{-5}, 2^{-13}$	+		+		+			82.529
5000	Linear	$2^{-5}, 2^{-13}$			+		+			72.057
10000	Linear	$2^{-5}, 2^{-13}$	+							**83.614**
20000	Linear	$2^{-5}, 2^{-13}$	+							74.571
10000	Linear	$2^{-5}, 2^{-1}$						+		10.14
10000	Linear	$2^{15}, 2^{-15}$							+	**92.857**
10000	Linear	$2^{5}, 2^{-13}$			+	+	+			**93.771**

4 Programs

All our programs and results with used libraries (for SVM and HOG) can be found at https://github.com/ashkanb0/ML-Final-Project .

5 References

[1] J. Canny. “A computational approach to edge detection”. Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-8(6):679–698, November 1986.

[2] R. O Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures”, Comm. ACM, Vol. 15, pp. 11–15, January 1972.

[3] L. Shapiro and G. Stockman “Computer Vision”, Prentice-Hall, Inc. 2001

[4] C. Harris and M. Stephens. “A combined corner and edge detector”. Proceedings of the 4th Alvey Vision Conference. pp. 147–151, 1988.

[5] J. Shi and C. Tomasi. “Good Features to Track”. 9th IEEE Conference on Computer Vision and Pattern Recognition. Springer, June 1994.