Comparison of Classification Algorithms for Handwritten Digit Recognition

Jincheng Zhang
Department of Information Engineering
The Chinese University of Hong Kong
zj012@ie.cuhk.edu.hk

Abstract

Handwritten digit recognition is one of the fundamental problems in practical pattern recognition system design. In this work, I try a set of classifiers to tackle the handwritten digit recognition problem and compare the classifiers' performance on the MNIST data set. The gray-scale pixel values of the digit images are directly used as the features. I conduct extensive experiments to evaluate the performance of different classifiers. Experimental results show that the convolutional neural network (CNN) achieves the best accuracy of 98.85%. Polynomial kernel SVM with degree 2 can obtain a comparable accuracy of 98.08% in much less training time compared with CNN. KNN is the worst classifier in terms of either the recognition accuracy or the computation time. Detailed analysis and potential improvements are discussed. [1]

1. Introduction

Handwritten digit recognition is to use computer to interpret intelligible handwritten digits automatically, which is an important topic in OCR (Optical Character Recognition) applications and the pattern recognition community. Handwritten digit recognition is fundamental to many practical applications, such as bank check processing, postal mail sorting, automatic number plate recognition, etc. For these applications, the recognition accuracy and speed is crucial to the overall performance.

For handwritten digit recognition, the performance (*i.e.,* accuracy and speed) largely relies on the features and classification methods. For the features, many prior works [4] tried to propose feature extraction approaches to improve the performance, such as template matching, projection histograms, Fourier descriptors, etc. [8] proposed a method to extract features directly from gray-scale character images by Gabor filters, which has excellent performance on low-quality machine-printed character recognition and cursive handwritten digit recognition. For the classification methods, a comprehensive experimental study [6] by Yann LeCun et al. compared the performance of linear and polynomial classifiers, K-nearest neighbor classifier, fully connected multi-layer neural network and three types of convolutional neural networks, including LeNet 1, LeNet4 and LeNet 5. A best test error rate of 0.7% is achieved on the MNIST data set by Boosted LeNet 4, a combination of three LeNet 4.

In this work, I directly use the gray-scale pixel values of the handwritten digit images as the features and mainly focus on the classification methods. K-Nearest Neighbor Classifier (KNN), Support Vector Machine (SVM), fully connected artificial neural network (ANN) and convolutional neural network (CNN) are adopted in my experiments. I compare the performance of these classifiers by considering the recognition accuracy, and the training and testing time on the MNIST data set.

2. MNIST Data Set

The MNIST data set [1] was constructed by Yann LeCun et al. from the NIST's special databases, which is a very representative data set for testing the classification performance on handwritten digit recognition. This data set contains 60000 training samples and 10000 testing samples. Each sample represents a digit ranging from 0 to 9, and is normalized and centered in a gray-scale image with size 28 $\times$ 28. I directly use the $28 \times 28 = 784$ gray-scale pixel values as the features of each sample. The first 100 sample handwritten digit images are shown in Figure 1.

3. Classifiers Specification

In this section, I only briefly describe the four classifiers used in my experiments. For more detailed explanations, readers can refer to the references.

[1] The data set and code of this work can be found on the project webpage https://github.com/jincheng9/MNIST_digit_recognition.

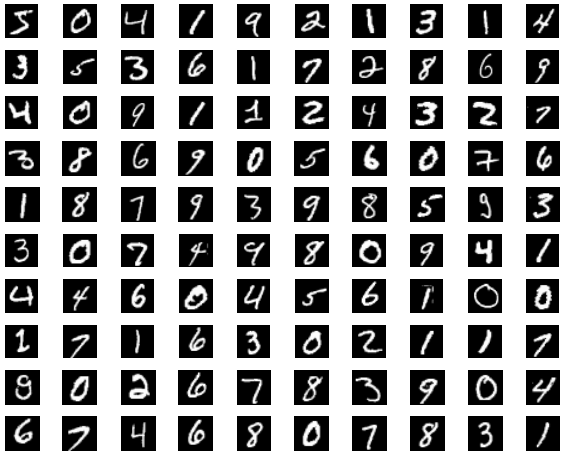

Figure 1. Sample images of the MNIST data set

3.1. K-Nearest Neighbor Classifier

K-nearest neighbor classifier (KNN) is a very simple non-parametric classifier. To classify a new sample, we first calculate its K nearest neighbors and then assign the new sample to the most common class of the K nearest neighbors. We can also weight the neighbors when determining the class of new samples. An advantage of KNN is no training time. However, the testing is time-consuming and has high requirements on memory for large data sets. The choice of distance measure and value of K is critical to the performance.

3.2. Support Vector Machine

For binary classification problems, Support Vector Machine (SVM) tries to find a hyperplane to separate the two classes with the maximum margin. Assuming that we have n training samples and each sample is represented by x_i with class label y_i, where $y_i \in \{-1, +1\}$. If we allow non-linear decision boundary and misclassified samples, the SVM can be formulated as the following optimization problem:

$$\max \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j [\phi(x_i) \cdot \phi(x_j)] \quad (1)$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, i = 1, ..., n \quad (2)$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad (3)$$

where $\phi(x)$ is the feature vectors of the input x, and C is the slack penalty. By solving the above optimization problem, we can obtain the optimal hyperplane and use it to classify new samples.

$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ is the kernel function. We have three common kernel functions, which are listed as follows:

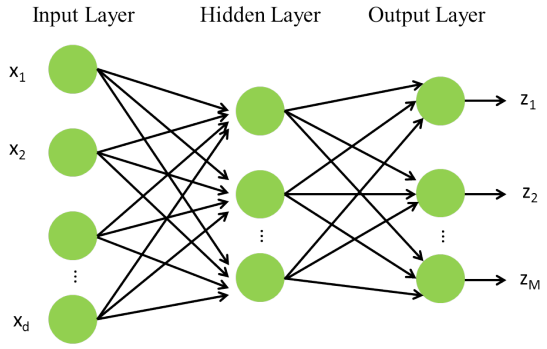

Figure 2. Architecture of fully connected multi-layer neural network

- Linear Kernel: $K(x, x') = x \cdot x'$
- Polynomial Kernel: $K(x, x') = (1 + x \cdot x')^p$
- Radial Basis Kernel: $K(x, x') = exp(-\gamma ||x - x'||^2)$

Different kernels may have different performance. In the experiments, I will try these three kernels to compare their performance on handwritten digit recognition problem.

For multi-class classification, we can extend the two-class SVM with one-versus-all or one-versus-one approach.

3.3. Fully Connected Multi-Layer Neural Network

Fully connected multi-layer neural network is a typical neural network model, which consists of an input layer, one or several hidden layers and an output layer. For M-class classification, we have M output units in the output layer. We can use back propagation algorithm to train the neural network. Figure 2 shows the architecture of fully connected multi-layer neural network.

For the handwritten digit recognition problem, we have 10 classes from digit 0 to digit 9. Therefore, we have 10 output units in the output layer. Each output unit represents a class label. The choice of initialization weights, activation function, learning rate and the number of iterations can affect the performance of neural networks.

3.4. Convolutional Neural Network

Convolutional neural network (CNN) [5] is a kind of locally connected neural network inspired by biological systems, where a cell is only sensitive to a small subregion of the input space, called a receptive field. Many cells are tiled to cover the entire visual field. Figure 3 [2] shows the typical architecture of CNN.

At each stage, we have a convolutional layer, local contrast normalization and a pooling layer. Convolutional layer increases the number of feature maps. Pooling layer decreases the spatial resolution. After multiple stages, the output is connected to a fully connected neural network. For CNN, appropriate model architecture, input normalization,

Table 1. Summary for the experimental results of SVM. The total time includes both the training and testing time.

Type	Accuracy	Training time (min)	Testing time (min)	Total time (min)	# of Total Support Vectors
Linear Kernel	93.98%	6.6580	2.7603	9.4183	10347
Polynomial Kernel (degree=2)	98.08%	3.9027	2.4145	6.3172	8729
Radial Basis Kernel ($\gamma = 1/784$)	94.46%	10.4340	4.8396	15.2736	19625

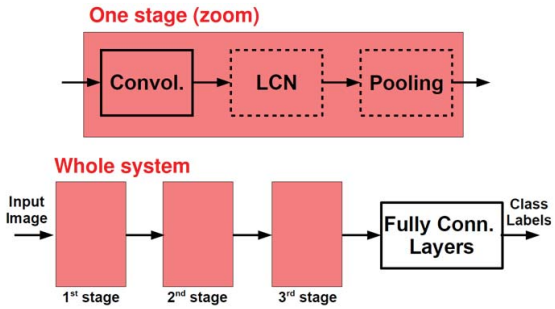

Figure 3. Typical architecture of CNN

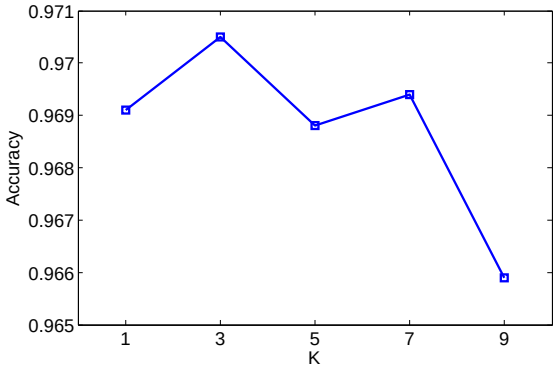

Figure 4. Accuracy with different K

suitable activation function, data augmentation and dropout can help achieve a better performance on the classification tasks.

4. Experimental Results

In this section, I conduct extensive experiments to evaluate the performance of different classifiers. The computer I use has an Intel Core-i7 3770 3.4GHz Quad Cores CPU with Hyper-threading, 16GB of RAM. For the MNIST data set, each training and testing sample is represented by a vector with 784 normalized gray-scale pixel values. The class label is from 0 to 9.

4.1. Evaluation of KNN

For the KNN classifier, I choose the Euclidean distance measure to calculate the K nearest neighbors. The classification accuracy of KNN is affected by the value of K. If K is small, the noise will have a high influence on the classification result. However, if K is large, the result may not be consistent with the idea behind KNN since KNN focuses on the nearest neighbors.

To evaluate the performance of KNN, I try different K ($K = 1, 3, 5, 7, 9$) and obtain the corresponding accuracy as shown in Figure 4. The highest accuracy of 97.05% is achieved by $K = 3$. There is no training time for KNN, but the testing process of the 10000 testing samples takes about 77min for each K in the experiments, which is time-consuming.

4.2. Evaluation of SVM

I use libsvm [3] to train and test the SVM classifiers. Three different kernels, *i.e.,* linear kernel, polynomial kernel and radial basis kernel are evaluated. Table 1 shows the experimental results.

For the radial basis kernel, I try $\gamma = 1$ first and find that the objective value of SVM always oscillates. Therefore, I change $\gamma = 1/d$, where d is the number of features and $d = 784$ in our experiment settings. $\gamma = 1/d$ is the default value for γ in libsvm and I can get a good accuracy with much less training time compared with $\gamma = 1$.

From the experimental results shown in Table 1, we can find that polynomial kernel with degree 2 achieves the best accuracy of 98.08% and the minimum training and testing time among the three kernels. Linear kernel can get a reasonable training time and testing time with a good performance. Radial basis kernel has a slightly better accuracy compared with the linear kernel, but takes the longest training and testing time.

Usually, non-linear kernel (*i.e.,* polynomial kernel and radial basis kernel) has longer training time compared with linear kernel because of the need to evaluate the kernel products with each support vector, but can achieve a better accuracy for the non-linear classification tasks. In my experiments, the number of total support vectors for polynomial kernel with degree 2 is 8927, the minimum number among these three kernels. While the number of total support vectors for linear kernel and radial basis kernel are 10347 and 19625 respectively as shown in Table 1. Therefore, polynomial kernel is the fastest, while radial basis kernel takes the longest training and testing time among these three kernels.

4.3. Evaluation of ANN

ANN has many parameters to tune, such as the number of hidden units, initialization weights, learning rate, activation function and the number of iterations. In the experi-

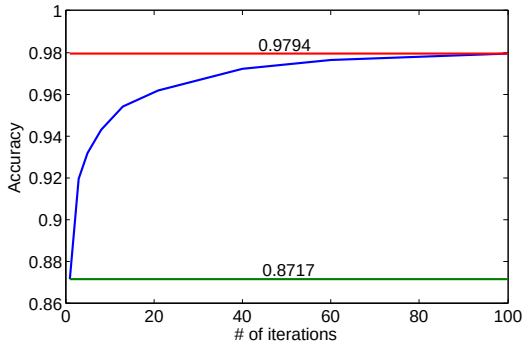

Figure 5. Accuracy with different number of iterations

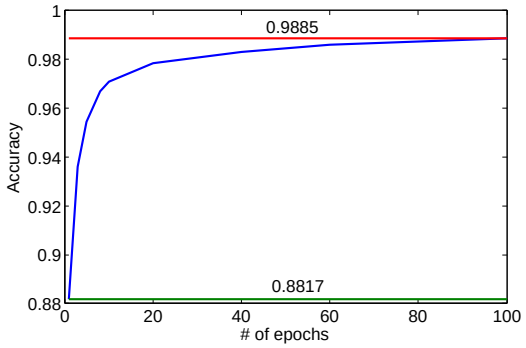

Figure 6. Accuracy with different number of epochs

Table 2. Training and testing time for ANN

# of iterations	training time (min)	testing time (s)
1	0.9232	0.5766
5	4.5557	0.5792
10	9.1266	0.5896
20	18.2523	0.5833

Table 3. Training and testing time for CNN

# of epochs	training time (min)	testing time (s)
1	1.4841	3.0588
5	4.4667	3.0403
10	14.8139	3.2490
20	29.6935	3.2030

ment, I use a 784-300-10 fully connected multi-layer neural network. The input-to-hidden and hidden-to-output weights are randomly initialized to [-1 1], the learning rate is set to 0.1. I use the sigmoid function $f(x) = \frac{1}{1+exp(-0.01x)}$ as the activation function for all hidden units and output units. I change the number of iterations, ranging from 1 to 100 and plot the corresponding accuracy as shown in Figure 5. We can find that the classification accuracy improves with the increase of the number of iterations and gradually becomes stable. The worst accuracy is 87.17% when the number of iterations is 1, and the best accuracy is 97.94% when the number of iterations goes to 100. I think the reason is that the estimated gradient is more reliable with more iterations.

As regard to the computation time, ANN has a very short testing time, but the training time is long if we want to obtain a good performance according to my experimental results. I think one reason is that we have many parameters to adjust in order to achieve the local optima in the error surface. On average, we need about 1min to train all training samples per iteration and the testing process for the 10000 testing samples only need 0.6s or so. Table 2 shows part of the training time and testing time.

4.4. Evaluation of CNN

To evaluate the performance of CNN, I use the DeepLearnToolbox [7], a MATLAB toolbox which has implemented various deep learning models. For the architecture of CNN we use in the experiment, the input layer is a 28×28 digit image. The first convolutional layer has 6 feature maps connected to the input layer through 6 5×5 kernels. The second layer is a 2×2 mean-pooling layer. The third layer has 12 feature maps connected to each feature map of the second layer through 12 5×5 kernels. The fourth layer is another 2×2 mean-pooling layer. We concatenate the feature maps of the fourth layer into a feature vector which is connected to a 2-layer fully connected neural network. There are 10 output units in the final layer, corresponding to the 10 class labels. A learning rate of 1 and a batch size of 50 is used in the experiments. I increase the number of epochs from 1 to 100. Figure 6 shows the corresponding accuracy. The lowest accuracy is 88.17%, and the highest accuracy is 98.85%. In general, the accuracy improves with the increase of the number of epochs and finally stays stable.

The training time for CNN is very long. In our experiments, each epoch takes 1.5min around. To achieve the accuracy of 98.85%, the training process lasts about 148min. The testing process is very fast, we only need about 3s to test all the testing samples. [2]

4.5. Comparison of different classifiers

To compare the performance of different classifiers, I first plot the best accuracy of each classifier and the corresponding total running time (including training time and testing time) as shown in Figure 7.

Among all classifiers, CNN with 100 epochs obtains the highest accuracy of 98.85%. The accuracy of polynomial kernel SVM with degree 2 is very close to CNN, which is 98.08%. 3NN has the lowest accuracy. With respect to the computation complexity, SVM is the best since it only needs 6.3172min to complete the training and testing, which is

[2] I use CNN to participate in the handwritten digit recognition contest held on Kaggle and achieve 23rd among 354 teams. http://www.kaggle.com/c/digit-recognizer/leaderboard

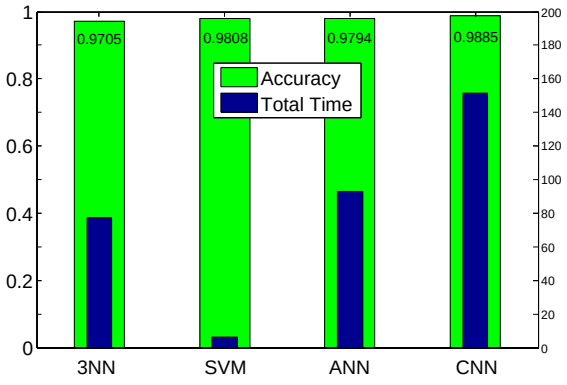

Figure 7. Comparison of the four classifiers

far less than the other three classifiers. 3NN is the worst because ANN and CNN can achieve the same accuracy with much less time. In my experiments, ANN and CNN need about 30min and 15min to surpass the accuracy of 3NN. ANN and CNN are both very time-consuming. Although CNN with 100 epochs has the best accuracy, the training time is too long, lasting about 148min.

5. Conclusions

In this work, I try four different classifiers to solve the handwritten digit recognition problem. CNN with 100 epochs achieves the highest accuracy of 98.85% and 3NN has the lowest accuracy of 97.05%. Polynomial kernel SVM with degree 2 can obtain a comparable accuracy of 98.08% in much less time compared to CNN. Among all these classifiers, SVM is the fastest, while KNN, ANN and CNN are all time-consuming if we want to obtain a good performance.

There are several interesting directions to be explored in the future. First, we can try the Boosting method. By combining the power of multiple weak learning algorithms, Boosting can achieve very good performance. Second, we can use some feature extraction methods to improve the recognition accuracy while reducing the computation time simultaneously.

References

[1] MNIST Data Set. `http://yann.lecun.com/exdb/mnist/index.html`.

[2] Slides from course ENGG5202 instructed by Prof. Xiaogang Wang.

[3] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[4] Ø. Due Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, 1995.

[7] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data, 2012.

[8] X. Wang, X. Ding, and C. Liu. Gabor filters-based feature extraction for character recognition. *Pattern recognition*, 38(3):369–379, 2005.