# Handwritten Digit Recognition

ZHONGHuan
Software College, SDU

## *Abstract:*

In test 1, we train and test KNN and CNN classiers for pattern analysis in solving handwritten digit recognition problems, using MNIST data set. We will use Kth Nearest Neighbor and Convolutional Neural Networks to recognize these data, and we'll see the result using these two methods. In test 2, we add a new method called dropout method in NN and SAE to reduce the over fitting of variable MNIST.
**Key words: KNN, CNN, MNIST, NN, Dropout, SAE.**

## Introduction

The handwritten digits recognition is an old and interesting problem, there have been lots of people done some research about it[1]. KNN is the old and simple method, although it is easy to implement, the result of KNN is not bad, especially for the normal MNIST data. But facing to the variations of MNIST, KNN is not a good method to figure out it. The CNN is a kind of neural network of deep learning, which is a good algorithm to solve the recognition problem with so many advantages. But the recognition of variations of MNIST is really a tough problem, so we use NN architecture with dropout method and SAE (Stacked Auto-Encoders, a implement of auto-encoders in deeplearntoolbox.). And the result shows that it is really good to avoid over fittings.

## Data Description

The MNIST[1] database contains 60,000 digits ranging from 0 to 9 for training the digit recognition system, and another 10,000 digits as test data. Each digit is normalized and centered in a gray-level image with size 28 × 28, or with 784 pixel in total as the features. Some examples are shown in Figure 1.1.

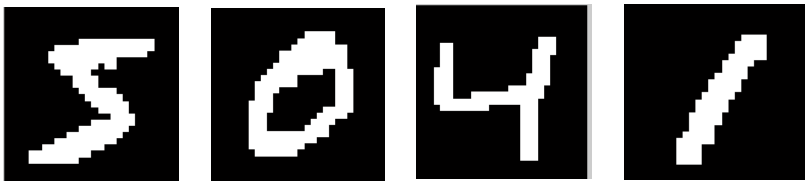

***Figure 1.1: Examples of MNIST data set***

Variations on the MNIST digits[15], contains different kinds of variations of MNIST, each digit is also centered in a gray-level image with size 28×28. But there are some changes to MNIST. Here introducing multiple factors of variation leads to the following benchmarks:

1) mnist-rot: the digits were rotated by an angle generated uniformly between 0 and $2\pi$ radians. Thus the factors of variation are the rotation angle and the factors of variation already contained in MNIST, such as handwriting style(figure 1.2):

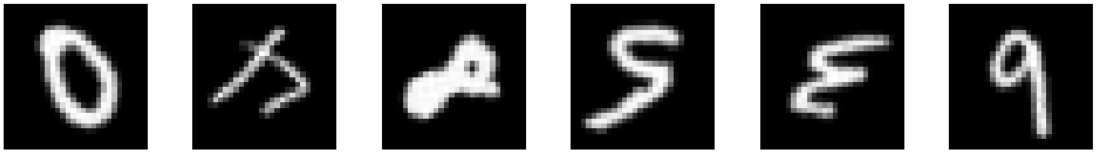

***Figure 1.2: Examples of MNIST Variation***

2) mnist-back-rand: a random background was inserted in the digit image. Each pixel value of the background was generated uniformly between 0 and 255(figure 1.3);

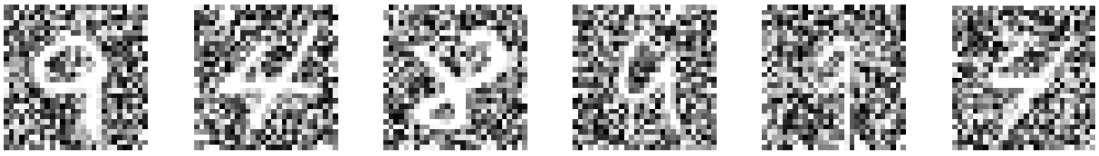

***Figure 1.3: Examples of MNIST Variation***

3) mnist-back-image: a patch from a black and white image was used as the background for the digit image. The patches were extracted randomly from a set of 20 images downloaded from the internet. Patches which had low pixel variance (i.e. contained little texture) were ignored(figure 1.4);

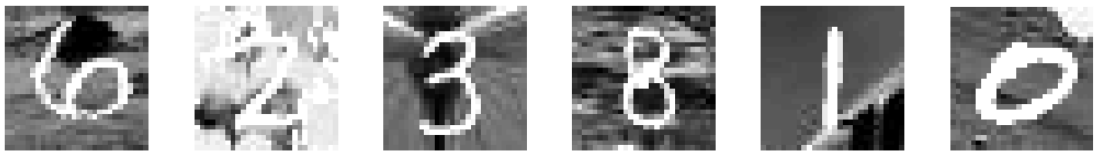

***Figure 1.4: Examples of MNIST Variation***

4) mnist-rot-back-image: the perturbations used in mnist-rot and mnist-back-image were combined(figure 1.5).

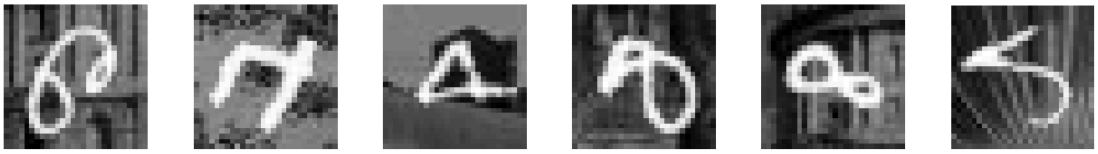

***Figure 1.5: Examples of MNIST Variation***

| Link to dataset | Size | File description |
|---|---|---|
| MNIST basic | 23M packed; 599M + 144M unpacked | 12000 train, 50000 test |
| MNIST + background images | 88M packed; 599M + 144M unpacked | 12000 train, 50000 test |
| MNIST + random background | 219M packed; 599M + 144M unpacked | 12000 train, 50000 test |
| Rotated MNIST digits | 56M packed; 338M + 56M unpacked | 12000 train, 50000 test |
| Old version: ~~Rotated MNIST digits~~ | ~~284M packed; 599M + 144M unpacked~~ | ~~12000 train, 50000 test~~ |
| Rotated MNIST digits + background images | 115M packed; 599M + 144M unpacked | 12000 train, 50000 test |
| Old version: ~~Rotated MNIST digits + background images~~ | ~~124M packed; 599M + 144M unpacked~~ | ~~12000 train, 50000 test~~ |

***Figure 1.6: MNIST Variations***

As we can see from figure 1.6, MNIST has 5 types, and we'll use these 5 kinds of data set.

## KNN

This classifier is one of the simplest classifier to implement because it is a brute force method [2]. The algorithm is used to find the nearest match for a given case to the known cases stored in memory. The K is used to signify how many votes are used for decision making. It is most optimal to choose a value of K that is odd so it eliminates a tie between two sets. [3]

Compare known samples with the test sample using a distance metric. The distance metric is used to find the nearest neighbor. The most widely used distance metric is the Euclidean distance[5], because it gives a normalized value. Other distance metrics include Cityblock[6], and Chebychev[7]. Here we use the Euclidean distance. The Euclidean distance is:

$$\vec{D} = \sum_i (known_i - testcase_i)^2$$

Use the distance of the test case between known data, and then reorder it by distance. The value K denotes the number of ranks to use. Choose the top K classes, and the answer is the class appears most.

We use K-nearest neighbor classier on the three feature spaces (the original 784 pixels). The results are summarized in Figure 2, which shows the error rate with different K. And we can see that the best result is 0.0283 error rate with value of K is

3.

We will use the “knnclassify” method to test the accuracy of the KNN algorithm, “knnclassify” is a function belongs to the Bioinformatics Toolbox. The run time takes too long, to reduce the run time, you can use the PCA(Principal component analysis)[17] to reduce dimensions of the data, which will make the program faster.

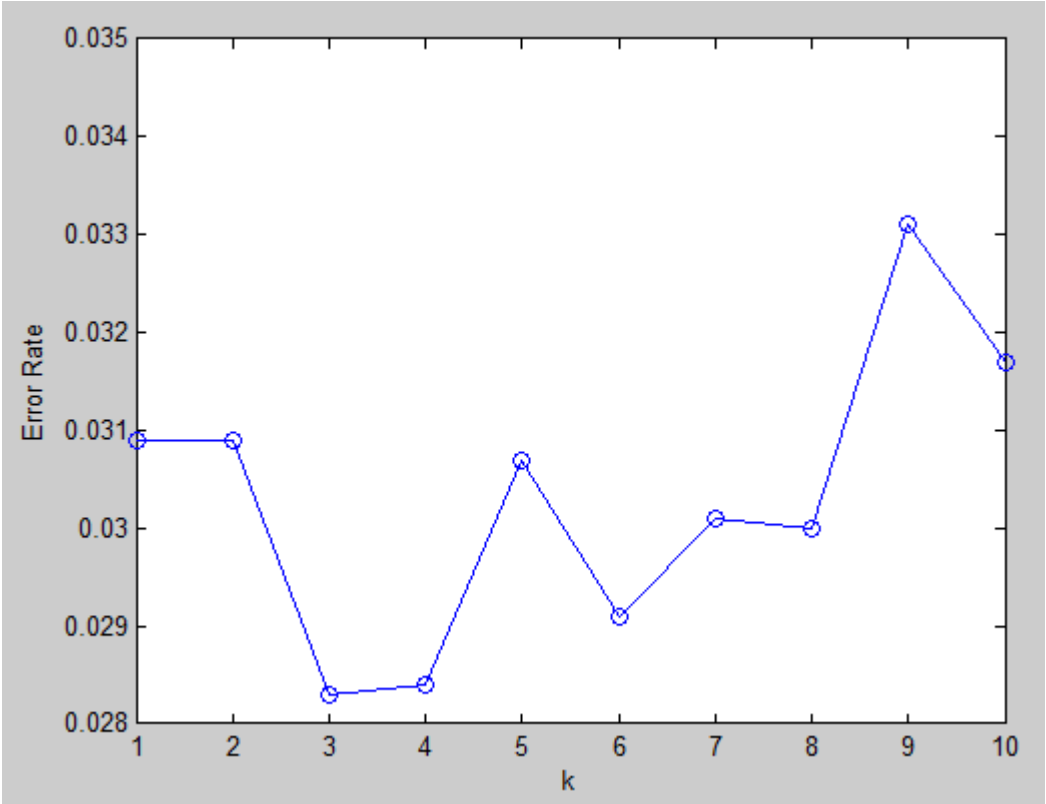


***Figure 2: Error rate of k-NN classier versus different choices of k***

## CNN Description

The classification task is performed using a Convolutional Neural Network (CNN). CNN is a special type of multi-layer neural network, being trained with an optimized version of the back-propagation learning algorithm. CNN is designed to recognize visual patterns directly from pixel images with minimal preprocessing, being capable to recognize patterns with extreme variability (such as handwritten characters), and with robustness to distortions and simple geometric transformations.

Before we begin to introduce convolution neural network，We should know about some concepts about CNN.

***Convolution:***

Let’s start from the feature extraction using convolution. It is one simple solution to this problem is to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units. Specifically, each hidden unit will connect to only a small contiguous region of pixels in the input. It involves “Sparse Connectivity” (Figure 3) and “Shared Weights”.[10]

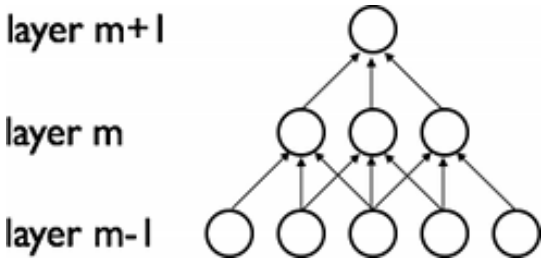


***Figure 3: Sparse Connectivity***

We can see from Figure 3 clearly that the node in layer m are connected to 3 node in layer m-1 but all nodes. We say that their receptive field with respect to the layer below is also 3. The architecture thus confines the learnt "filters" (corresponding to the input producing the strongest response) to be a spatially local pattern (since each unit is unresponsive to variations outside of its receptive field with respect to the retina).

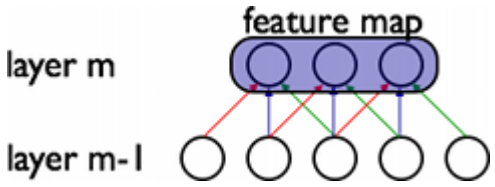


***Figure 4: Shared Weights in a feature map.***

As we can see from Figure 4, [10] each sparse filter $h_i$ is additionally replicated across the entire visual field. These "replicated" units form a feature map, which share the same parameters, i.e. the same weight vector and the same bias. Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing offers a very efficient way to do this, since it greatly reduces the number of free parameters to learn.

***Pooling:***

After obtaining features using convolution, we would next like to use them for classification. At this time, we can use some aggregation operation called operation pooling to reduce the computation complexity and avoid over fitting.

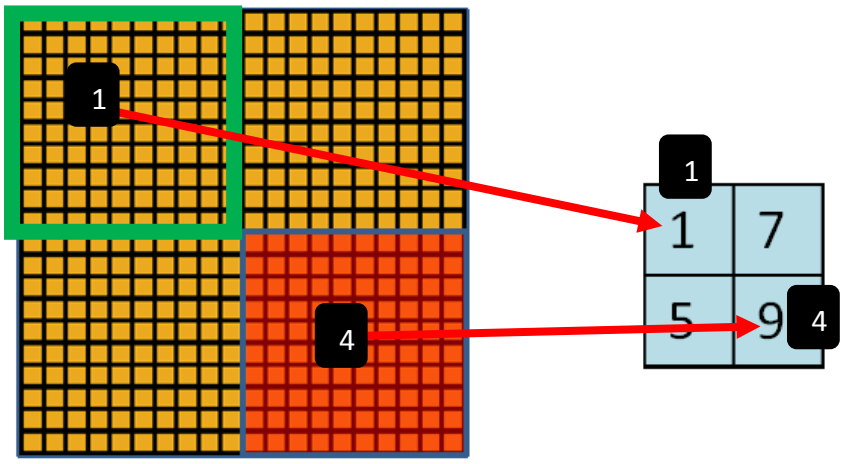

***Figure 5: Pooling***

As we can see that the region 1 of left image are change to the one node to the right image. The pooling sometimes is called mean pooling, max pooling or others, depending on the pooling operation applied. After we get to know about these concepts, we'll see the CNN architecture.

***CNN Architecture:***

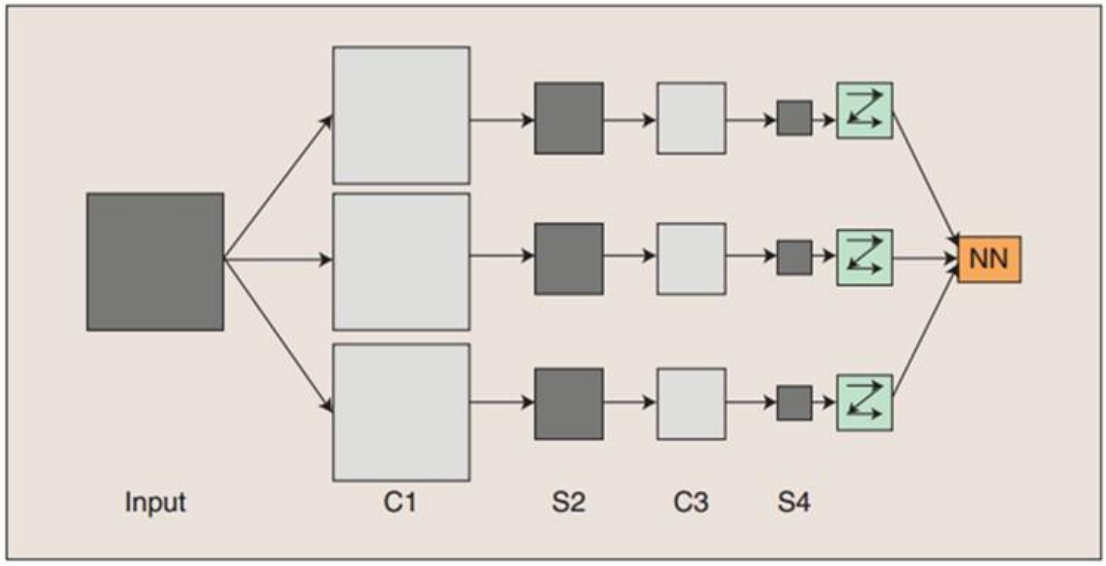


***Figure 6： The Architecture of CNN***

The CNNs is comprised of a sequence of convolution process and subsampling process[12]. As figure 6 shows, the convolution process convolves an input with a trainable filter $f_x$ and adds a trainable bias $\mathrm{b}_x$ to produce the convolution layer $C_x$, and the subsampling process sums a neighborhood(four pixels), weights by scalar $w_{x+1}$, adds the trainable bias $\mathrm{b}_{x+1}$, and passes through a sigmoid function to produce a smaller feature map $\mathrm{s}_{x+1}$. Then, $\mathrm{C}_x$ is treated as the input data, converted to a set of smaller-dimension feature maps $S_{x+1}$ via the subsampling process.

CNNs adopts the Back-Propogation to update the weights between every two adjacent layers. Therefore, one entire CNNs procedure helps to calculate the weight update. We should run this procedure several times until convergence is reached. Other improvements like using Fast Fourier Transform(FFT) algorithms to filter the input data or using max-pooling to subsampling are related to either the convolution process or subsampling process. Most of them are based on the classical methods in signal processing.

## Test 1

In test 1, we use the CNN of deeplearningtoolbox to recognize MNIST data. There are some parameters we should set. First, the input layer, convolution layer and sub

sampling layer, here we use the 6c-2s-12c-2s Convolutional neural network. Someone use different network, such as [13], we can see from the previous part, network is important to the result of CNN, but we just use the default setting 6c-2s-12c-2s here. And the learning rate here is 1, it is important to the process of back propagation， the bigger the learning rate，the faster the training speed, but if the learning rate is out of bounds, we will meet some problem[14].In fact, it is better that we set different learning rate between different layer [13]. The next is the batchsize, it decides how many training cases we choose to train at one time, here the batchsize is 50. The last one is numepochs, it decides our epochs, and the default value is 1, but if you choose a larger numepochs, the error rate will become smaller. As we can see from figure 7.

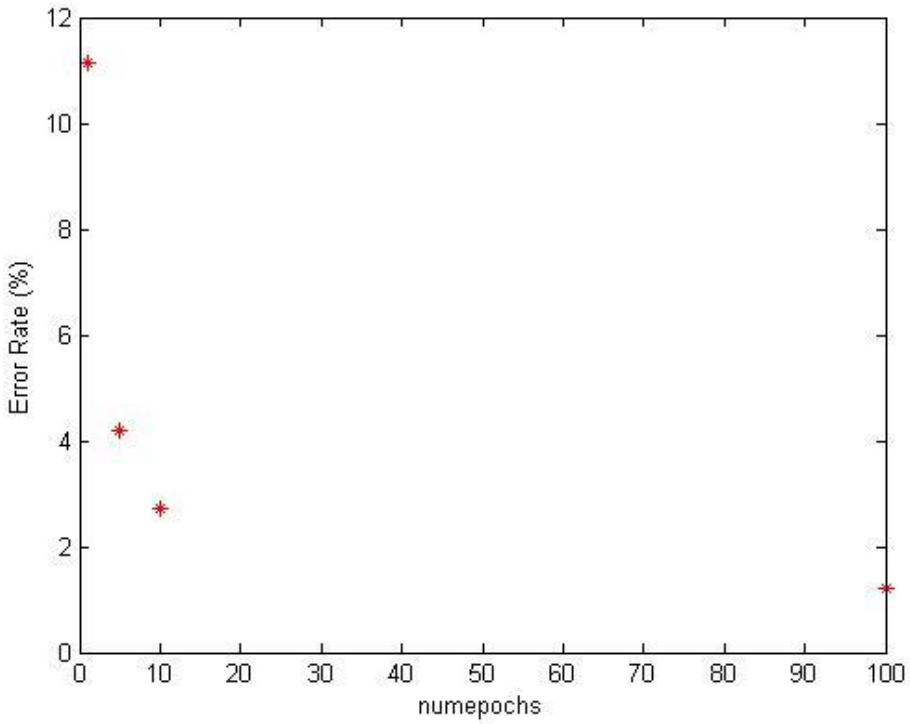


***Figure 7:Error rate at different numepochs***

The error rate at different numepochs shows that the bigger the numepochs, the smaller the error rate, but I will not use bigger numepochs to test, the reason is that it takes me several hours to run the program when numepochs is 100. But we can retrieve several information here, the normal Mnist data unlike other dataset, it is more similar to images with same digit and it contributes the result.

## Test2

In test 2, data sets have been changed to Mnist variations[15]. First, we use the previous method in test 1, but the result is really terrible. See table 1. The other parameters are the same as test 1, but we change the numepochs like test1, and we can see the error rate changes smaller after we set a larger numepochs.

| Types / Error rate(%) | basic | background images | random background | rotated | Rotated + background |
|---|---|---|---|---|---|
| numepoches=10 | 9.166 | 28.85 | 18.432 | 43.014 | 81.406 |
| Numepoches=20 | 6.252 | 20.532 | 12.66 | 27.06 | 65.108 |

***Table 1: Error Rate at different numepoches and variable images***

## Model Improvements:

From the table 1, we can see that the pure CNN can't solve the variable digits, supposing that every single digit has different images if we rotate the digits or add a background, which results the over fittings during training. To solve the problem, we can use a method called dropout[16]. After using dropout, we'll use Denoising Autoencoder, and we choose the SAE in deeplearningtoolbox.

In a large feedforward neural network, overfitting can be greatly reduced by randomly omitting half of the hidden units on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random "dropout" gives big improvements on many benchmark tasks and sets new records for object recognition and molecular activity prediction.

And we use the NN of deeplearningtoolbox to test the dropout method, and see how it can improve our recognitions. We choose the $784 \times 100 \times 10$ network architecture, and the dropfunction is 0.5, batchsize is 50. The table 2 shows that NN with dropout is worse than NN without dropout. The reason is that the MNIST images are almost normal digits, the dropout method will lower the precision.

| Types / Error rate(%) | NN without dropout | NN with dropout |
|---|---|---|
| numepoches=5 | 5.29 | 6.35 |
| numepoches=20 | 4.42 | 5.3 |

***Table 2: NN+dropout in MNIST***

| Types / Error rate(%) | basic | background images | random background | rotated | Rotated + background |
|---|---|---|---|---|---|
| numepoches=5 with dropout | 28.448 | 86.032 | 90.302 | 77.168 | 90.668 |
| Numepoches=10 With dropout | 9.502 | 87.726 | 89.332 | 61.97 | 89.128 |

| | | | | | |
|---|---|---|---|---|---|
| Numepoches=20 With dropout | 16.346 | 87.626 | 90.374 | 54.712 | 89.568 |

***Table 3: NN+dropout in Variations of Mnist***

The Denoising Autoencoder (DA) is an extension of a classical autoencoder and it was introduced as a building block for deep networks in [18]. We use the SAE in deeplearningtoolbox to test the normal MNIST data and variable MNIST. And we got the table 4 and table 5.

| Error rate(%) | SAE without dropout |
|---|---|
| numepoches=1 | $8.26 \pm 0.1$ |
| numepoches=5 | $4.7 \pm 0.1$ |
| numepoches=20 | $2.69 \pm 0.2$ |

***Table 4: SAE in MNIST***

| Types / Error rate(%) | basic | background images | random background | rotated | Rotated + background |
|---|---|---|---|---|---|
| numepoches=5 | 7.8 | 46.96 | 62.36 | 32.74 | 76.24 |
| numepoches=10 | 6.5 | 35.48 | 44.71 | 24.95 | 69.03 |
| Numpoches=20 | 5.32 | 30.49 | 23.28 | 19.72 | 65.64 |
| Numepoches=50 | 4.40 | 27.34 | 17.07 | 18.50 | 61.69 |

***Table 5: SAE in Variations of MNIST***

As we can see from the table 2 to table 5, we see that the NN with dropout is not good to detect the noisy images, but compare with NN, SAE is better with lower error rate and faster training speed. The parameters in these models are all choose according to other people's work, we choose sigmoid function to be our activation_function, the learning rate is 1, inputZeroMaskedFraction is 0.5, the batchsize is 50. You may wonder why I only choose to change numepoches but no other parameters. Here I'll explain that choose the value of parameters are a continuous work, I just choose the numepoches to show this process, the other parameters are also set through this method.

We can see that the different model shows good performance to different types of digits, such as SAE is better at rotated digits, and CNN is better at digits without rotation, and we can combine these two models to test these data. And there are two kinds of CAE model, first is Contractive Auto-Encoders and the second is Convolutional Auto-Encoders, the result of others shows that these two kinds of method performs well to variable MNIST [19].

# References：


[1] The MNIST database of handwritten digits, Retrieved June 27, 2014 from http://yann.lecun.com/exdb/mnist/

[2] k-nearest neighbors algorithm, Retrieved June 27, 2014 from Wikipedia, http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[3] Gaurav Jain,Jason Ko. ”Handwritten Digits Recognition”, 2008.

[4] Calin Enachescu, Cristian-Dumitru Miron. “Handwritten Digits Recognition Using Neural Computing”, Inter-eng 2009.

[5] Euclidean distance, Retrieved June 27, 2014 from http://en.wikipedia.org/wiki/ Euclidean_distance.

[6] Cityblock, Retrieved June 27, 2014, from http://en.wikipedia.org/wiki/ City_block.

[7] Pafnuty Chebyshev, Retrieved June 27, 2014 from http://en.wikipedia.org/wiki / Pafnuty_Chebyshev.

[8] Feature extraction using convolution. Retrieved June 27,2014 from http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial.

[9] Pooling, Retrieved June 27, 2014 from http://deeplearning.standford.edu/wiki / index.php/Pooling

[10] LeNet, Retrieved June 27, 2014 from http://deeplearning.net/tutorial/lenet.html.

[11] D.C.Ciresan, U.Meier, J.Masci, L.M. Gambardella, and J.Schmidhuber, “Flexible, high performance convolutional neural networks for image classification,” in International Joint Conference on Artificial Intelligence, 2011.

[12] Dandan Mo, “A survey on deep learning: one small step toward AI”, 2012.

[13] Patrice Y. Simard, Dave Steinkraus, John C. Platt, ”Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis”, ICDAR 2003.

[14] Gradient Descent, from http://en.wikipedia.org/wiki/Gradient_descent.

[15] Variations on the MNIST digits, Retrieved from http://www.iro.umontreal.ca/ ~lisa/twiki/bin/view.cgi/Public/MnistVariations

[16] Hinton, G. E., et al. (2012). "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580.

[17] Principal component analysis, retrieved from http://en.wikipedia.org/wiki/Principal_component_analysis.

[18] Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML‘08), pages 1096 - 1103, ACM, 2008.

[19] S Rifai, “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction”, ICML2011.