CS 4701: Digit Recognizer

Hsiao-Tung Chen(hc847), Cheng-Han Lin (cl2287)

Professor Bart Selman, Cornell University

Abstract. In this project, we developed a system that could recognize hand-written digits with high accuracy. We used digit images and combined feature extraction and descriptor with learning algorithms. For the first step, we extracted features and descriptors from each image, and then we fed those information into our learning system to predict the digit number. For the learning algorithms, we used four different popular classifiers. Among them we found that the convolutional neural network achieved the highest accuracy.

Keywords: handwritten digits recognition, convolution neural network, support vector machine, k-nearest neighbor, feature extraction, global feature descriptor

1 Introduction

In this digital age, there are many cases where digit recognition are useful. For example, cameras that take photos of cars capture images of license plates that consist of letters and digits. Digit recognition can be applied to find out the car owner that the license plate matches to. When banks process checks, the often handwritten digits need to be recognized automatically by machines such as the ATM, when there is no bank employee to verify. When the post office processes envelope mail addresses, they can process faster if digital recognition of digits are faster. Digit recognition implemented for all these cases and more saves time and cost.

The goal of this project is to find methods that can accurately predict the digits in images of handwritten single digit. We used the dataset provided by kaggle, which includes 42,000 training examples and ?? testing data, and each of the data is a number images represented as a set of pixel intensity values. Since the useful information in one image is sparse, meaning that in a number image large proportion of it is background, so feature extraction and descriptor generation may be performed prior to running classification algorithms in order to improve the performance of our learning algorithm.

The learning algorithms used were k-nearest neighbor (kNN), Support vector machine (SVM), neural network, and convolutional neural network. We tested with different parameters and with different descriptors.

Fig. 1. Real world digit recognition

2 Issue to be address

2.1 Feature extraction and descriptor generation in dataset

The first issue would we given a image how could we find useful information inside it and describe the image with a new descriptor that could represent a image ? Take figure 2 below for example. One could notice that for a digit image the useful information are the white pixels and must of pixels are just background, so we need to find a detector that could locate the exact position of a digit, for example we could use edge detector to find the boundary of a digit.

Also, we as figure 2 shows when a digit has a rotation of translation, if we apply the raw image directly to some learning algorithm such as kNN which only compares the pixels located in corresponding positions that would definitely fail in those cases. In this way, we need to find a descriptor that could represent a whole image by global information, such as relative distribution of points or spatial structure of a object. In this project, we try three different kinds of descriptors includes: gist descriptor, shape context, and original raw image.

Fig. 2. Real world digit recognition

2.2 Learning algorithm

The mainly part of our project is to develop a learning algorithm that could recognize the handwritten digit with high accuracy and speed. Since a digit might be located in different positions on an image, we decide to use state-of-the-art classifier neural network as well as convolutional neural network. Also, we compared our result to some popular learning methods, including SVM and kNN.

2.3 Combination of descriptor and learning algorithm

Even though a descriptor could represent cone digit image properly, it might not always be effective when we apply different learning methods. In this way we tried each learning algorithm with different descriptor as input. In order to figure out the best combination of them.

3 Dataset

We use the dataset provided by kaggle called Mnist, which contains 42000 instances for training. Each instance is a handwritten single digit image, and they are translated to gray image with black background and white foreground. Those digit have different positions or rotations and even style, which make this task a challenge one. Below is some example for our dataset.

Fig. 3. Mnist example

Each of image is 28 pixels in both height and width for total 784 pixels, and they are reshape to 784-dimensional vector in the dataset. Each pixel is a single

value associated with the intensity of that pixel, which represents the lightness of that pixel. Range of intensity is between 0 to 255, and the larger the number is, the darker the pixel is.

Beside, on the training set, kaggle also provide "label" each dataset, that is the ground truth for each instance. In this way, we can train our model by some state-of-art learning algorithms.

4 Data Pre-processing

4.1 Raw Image

Original dataset, each image is represented by 784-dimensional vector, and each element is intensity of certain pixel. In some learning algorithm such as convolution neural network, we just take this vector to represent an image and each element in vector could be an attribute.

4.2 Descriptor

As one can image, using the intensity for each pixel in some learning algorithm such as kNN is not only inefficient but also not effective in recognition. One reason is that we need to use 784 dimensional vector to represent an image. Also, one could observe that the black background has larger proportion to white foreground on most of cases, meaning that the useful information actually only lies in a small area of one image. Beside, when using kNN and support vector machine, we need to compare the similarity between pixel to pixel with same position. That is to say, when a digit moves its position or has a small rotation, will make a significant effect in decision what digit it is.

In this way, we need to find a way to extract useful points in an image, and represent an image with global information which would invariant to reasonable scale and rotation changes. In next two subsections, we would introduce two popular image descriptor which would represent image with low dimensional vector, and they represent image using "global" information rather than local information only.

4.3 GIST Descriptor

The idea of GIST Descriptor is to develop a low dimensional representation of the scene, which does not require any form of segmentation. The GIST descriptor focuses on the shape of scene itself, on the relationship between the outlines of the surfaces and their properties, and ignores the local objects in the scene and their relationships. This criteria makes it a suitable approach for our work.

The detail of gist descriptor including following steps. Firstly, we split image into a grid on several scales, and the response of each cell is computed using a series of Gabor filters. Then all of the cell responses are concatenated to form the feature vector. We represented each image with a gist descriptor which has 512 dimension.

4.4 Shape Context

Shape context is a novel approach to measuring similarity between shapes and utilize those information for object recognition. The idea of this descriptor is that we observe relative positions of feature points which are points on the boundary of certain shape. For example, we pick n points on the contours of the shape then for certain point p_i, where $p_i \in n$, the coarse histogram of the relative coordinates of the remaining n - 1 points is defined as the shape context of p_i. Also, this descriptor is robust to scale and rotation changes. This descriptor is suitable for our work since same numbers might have different scale, angle and even style, but they share the same shape. We represented each image with a gist descriptor which has 2172 dimension.

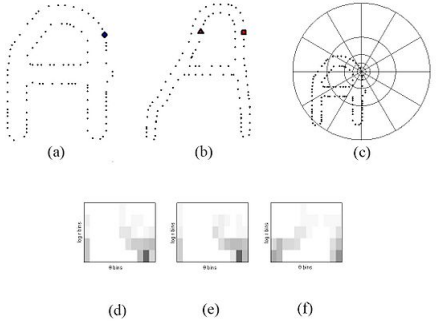

Fig. 4. (a) and (b) are the sampled edge points of the two shapes. (c) is the diagram of the log-polar bins used to compute the shape context. (d) is the shape context for the point marked with a circle in (a), (e) is that for the point marked as a diamond in (b), and (f) is that for the triangle. As can be seen, since (d) and (e) are the shape contexts for two closely related points, they are quite similar, while the shape context in (f) is very different.

5 Learning Algorithm

5.1 K-Nearest Neighbor

In pattern recognition, the kNN algorithm is a non-parametric method used for classification. In this project, firstly, we transfer every image in to a N- dimensional vector,which could be gist descriptor, shape context, or grayscale raw image. Given a query image, we compute the euclidean distance between it and every image in training dataset. Then we find the k most similar images (nearest neighbors) from training dataset which has most small euclidean distance, and we predict the label of query image from the majority label of its nearest neighbors. We tried k from 1 to 20 to find the most effective parameter.

5.2 Support Vector Machine

SVM is an efficient technique for data classification. Given a training set of instance-label pairs $(\mathbf{x}_i, y_i)$, $i = 1, ..., l$ where $\mathbf{x}$ is the input instance features and y is label for each instance. SVM solves the following optimization problem:

$$\min_{\mathbf{w},\zeta} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i$$

subject to

$$y_i(\mathbf{w^T}\phi(\mathbf{x_i}) + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0$$

Here the training instances are mapped to a higher dimensional space by function ϕ. SVM aims at finding a hyperplane that maximizes the margin. $C > 0$ is the penalty parameter of the error term. In our task, x represents input descriptor that are bag-of-word and gist descriptor, and y is label for each instance that is ground truth for every instance. We used LIBSVM for this project, and it supports SVM with four basic kernels:

linear:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$$

polynomial:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma\mathbf{x}_i^T\mathbf{x}_j + r)^d, \gamma > 0$$

radial basis function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma|\mathbf{x}_i - \mathbf{x}_j|^2), \gamma > 0$$

sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma\mathbf{x}_i^T\mathbf{x} + r)$$

However, the LIBSVM has some limitation that is it can not deal with input instances with large dimension, when we used shape context as input it would fail, so we also used another method called LIBLINEAR which is a linear classifier for data with millions of instances and features. It supports L2-regularized classifiers and L2-loss linear SVM, L1-loss linear SVM.

5.3 Neural Network

Neural Network is a learning algorithm inspired by the nervous system of animals, in particular the network of neurons in the brain. Neural network serves the purpose to estimate functions $f : X \rightarrow Y$ to determine an output Y while taking in a large number of inputs X that are generally unknown. The layers of neural network consist of the input layer, hidden layer, and the output layer. The input layer's number of nodes correspond to the number of features that an example has, since the neural network takes in one input value for each feature. The output layer's number of nodes correspond to the number of classes that

the examples can fall under, since the neural network output a predicted classification of an example. The number of total hidden layer nodes required grows exponentially with the number of input nodes. For example, a function with n boolean inputs will need $2^n/n$ hidden nodes to encode all possible hypotheses that predict the example classes. In general for optimal prediction accuracy, the number of hidden layers is between the number of features (i.e. number of inputs) and the number of possible classes (i.e. number of outputs).

Multilayer feed-forward neural networks are used to solve functionality that are not linearly separable, meaning that the examples when plotted in a space of n dimensions (n= number of features), the examples could not be separated by a hyperplane. This can be seen from understanding the inner workings of a neural network. Each node except the input node has input that is the weighted sum of all incoming inputs. Thus the output node's output is a function of the inputs and weights of previous nodes. The output expression can have its derivative taken with respect to the weights in order to optimize the weights of the inputs to each node, by the method of gradient-descent loss minimization. Because the network can represent and solve a highly nonlinear function (composed of nested nonlinear soft threshold functions), neural network can be used as a tool to do nonlinear regression.

Learning the Neural Network: Thus, neural network can be applied to digit recognition by analysis of a set of pixel intensities, since the pixel intensities can be extracted into features, which are typically the strokes for the digits. The features are modeled by the network nodes, and the nodes fire higher values to the next layer's nodes if the weighted sum of pixel intensity inputs hit the node's threshold value. The weight of each network connection (i.e. edge) is randomly initialized, and we propagate the weighted sum outputs and inputs forward through the network until the output layer. Let $g(x)$= activation function, in = weighted sum input, a= output activation, k= k^{th} node of this layer, and j= j^{th} node of the previous layer. At the output layer, the error of the k^{th} node is $\Delta_k = Err_k \times g'(in_k)$, where g'(x) is derivative of the activation function. The weight update rule is

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$$

where α is the learning rate chosen by the user. The error of the previous hidden layer node j is as such: $\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k$. The updated weight values as a function of errors are back-propagated from the output layer to the previous hidden layers, and then another round of forward propagation by the same inputs is performed. Enough iterations of forward propagation and back-propagation are done to achieve the gradient-descent loss minimization that optimizes the weight values for the neural network.

5.4 Convolutional Neural Network

A regular neural network could be further optimized to reduce training time and improve accuracy by performing the preprocessing step of convolution and pooling, before feeding the inputs into the neural network.

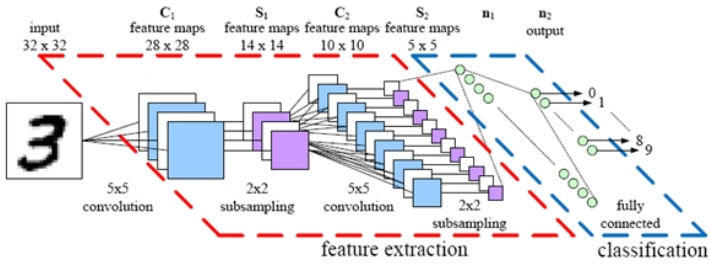

Fig. 5. Convolutional Neural Network Model

First, the input image is convoluted and then pooled (i.e. subsampled) twice before feeding into neural network. Convolution means applying a small-size filter over an image at all possible offsets to produce a smaller image. The pixel value inside the filter multiplied by their weights are translated into 1 single pixel. For example, a 32 x 32 image is translated into 28 x 28 images (called feature maps) with a filter of size 5 x 5, by shifting the filter one pixel at a time in 1 direction (horizontally then vertically). Then, the 28 x 28 feature maps can be pooled into 14 x 14 images by having just the maximum-value pixel represent each 2x2 grid of pixels.

After first applying convolution, the resulting feature maps represent a series of overlapping receptive fields with sparse connectivity. Each feature map now is more responsive to a larger area of pixel space. Pooling (i.e. subsampling) has the added benefit of enhancing translational invariance, meaning that a similar or same feature can still be detected even if translated across to different positions on the image. Combining convolution and pooling produces maps of high level features that can be used to more efficiently discern image features. Also, less inputs (features maps instead of all pixels) mean less weights to optimize so neural networks can potentially go deeper (have more layers) to be a better classifier. In general, convolution increases classification accuracy and reduces training time.

6 Experimental Evaluation

6.1 kNN

The kNN testing we split our data set into two parts: training and testing. We used 40000 instances among 42000 dataset for training and others for testing. Firstly, we tried different descriptors including raw image, gist descriptor and shape context for kNN, the results are shown below. Since in this data set digit in an image do not have large translation and rotation, so merely the raw image could reach high accuracy.

Raw	Gist	SC
96.6225%	96.4875%	9.955%

Table 1. Different descriptors for kNN

Also, we tried different k for kNN, to figure out the must suitable parameter, the result is shown as below, and we could find that when k equals to 3 or 4 the learning algorithm could reach highest accuracy.

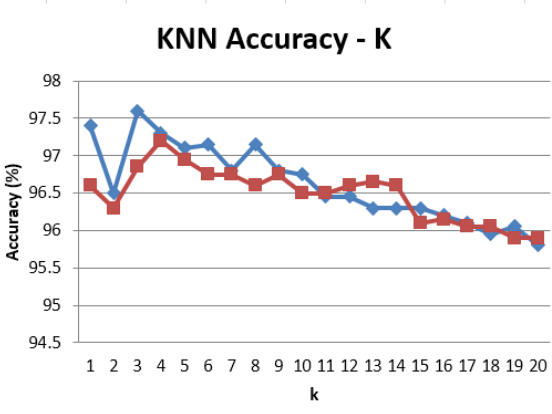

Fig. 6. kNN Accuracy with different k

6.2 SVM

In SVM the data setting is same to kNN, we split our data set into two parts: training and testing. We used 40000 instances among 42000 dataset for training and others for testing. Firstly, we tried different descriptors including raw image, gist descriptor and shape context as well as different kernels for SVM, the results are shown below, since the LIBSVM could not handle large dimensional data so the table 2 for LIBSVM only shows a descriptors. We could see that for raw descriptor the polynomial kernel has high accuracy and for gist descriptor linear kernel have highest accuracy.

We also tried LIBLINEAR with different classifier as well as descriptors. The result are shown below. We could observe that most of cases have good performance under linear SVM especially when we apply shape context as descriptor. For those classifiers below, s1 is L2-regularized L2-loss support vector classification (dual), s2 is L2-regularized L2-loss support vector classification (primal) and s3 is L2-regularized L1-loss support vector classification (dual), s4 is L1-regularized L2-loss support vector classification.

Kernel	Linear	Polynomial	Radial basis	Sigmoid
Raw	88.15%	97.55%	10.85%	10.85%
Gist	96.55%	10.85%	55.05%	23.6

Table 2. Different kernels and descriptors for LIBSVM

Classifier	s1	s2	s3	s4
Raw	85%	90.5%	84.05%	60.7
Gist	96.05%	96.1%	95.15%	96.25
SC	98.75%	98.7%	98.5%	98.75

Table 3. Different classifiers and descriptors for LIBLINEAR

6.3 Neural Network

# Iterations	# Nodes in Hidden Layer	Training time(s)	Training Set Accuracy	Test Set Accuracy
57	20	18.6	95.3%	92.3%
139	50	60.6	99.4%	95.4%
177	100	117	100%	96.6%
161	350	301	100%	97.2%
200	500	525	100%	97.5%
200	700	732	100%	97.5%
200	1200	1250	100%	97.4%

Table 4. Training and Testing Results for Neural Network

All non-convolutional neural network experiments were trained on 10,000 examples and tested on 10,000 examples. The results showed that by varying from 20 to 1200 hidden layer nodes, the accuracy on the test set increased from 92.3% to 97.4%, but the increase from 350 nodes to 1200 nodes saw the test set accuracy plateaued at low 97%. The training time increased from 18.6s to 1250s, with the time of 301s with 350 nodes hitting a plateauing accuracy of 97.2%. Trading off between training efficiency and classification accuracy, the result showed that around 400 hidden layer nodes would be optimal for training and testing.

6.4 Convolutional Neural Network

Neural Network is a nonlinear regression classification tool that can be applied to perform digit recognition. The neural network algorithm parses a set of training

data that includes the classes (0 to 9, one class for each digit) and pixel 0 to 738's intensities of many examples. The neural network takes in 739 inputs, one for each pixel, and has 10 outputs. An example with digit class of 2 would have the output vector close to [0010000000]. The learning rate is taken to be 1. The number of nodes in the hidden layer are varied from 20 to 1200. Since the number of hidden layer nodes should be between the number of features and the number of outputs, we varied the nodes from 20 nodes to 1200 nodes to assess neural network's classification accuracy. All weights are initialized to 1. The weighted sum signals are propagated from the input layer, to the hidden layer, and to the output layer. The errors at the output layer are calculated and the weight updates as a function of the errors are back-propagated from the output layer, as such: $w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$.

The neural network algorithm also included a preprocessing option to perform convolution and pooling before feeding the feature map pixels as inputs into the neural network. All convolution filters have size 5 x 5 pixels and all pooling filters have 2 x 2 pixels. The algorithm performed 2 times the convolution and pooling. The number of feature maps for each stage of preprocessing would be varied to assess its impact on the training time and classification accuracy.

# feature maps from 1st convolution	# feature maps from 2nd convolution	Training time(s)	Training Set Accuracy	Test Set Accuracy
2	13	14.5	97.7%	96.7%
5	25	32.5	99.0%	98.1%
10	50	83.4	99.6%	98.1%
20	100	273	99.8%	98.3%
40	200	1030	99.9%	98.6%
80	400	4870	99.9%	98.6%

Table 5. Training and Testing Results for Convolutional Neural Network

For convolutional neural network, all experiments trained on 10,000 examples and tested on 10,000 examples, for 10 iterations. All convolution filter sizes are 5x5 pixels and pool sizes are 2x2 pixels. The parameters were tuned such that the number of feature maps from the first convolution increased from 2 to 80, and from 13 to 400 from the second convolution, respectively. The results indicated that the training time increased from 14.5s to 4870s, while the test set accuracy slightly increased from 96.7% to 98.6%, and hitting a plateau of low 98% when the feature maps were 5 and 25 for each stage of convolution. Trading off between training and testing efficiency and classification accuracy, the 2 preprocessing stages should produce 20 and 100 feature images, respectively. In all experiments, given the same training time, convolutional neural network model yielded better class prediction accuracy than regular neural network model.

Fig. 7. Convolutional vs Non-Convolutional Neural Network Class Prediction Accuracy

7 Discussion

The results showed that given the same training time, convolutional neural network model predicted digit image class more accurately than regular neural network model. For example, after 301s of training, the regular neural network yielded 97.2% accuracy, but after 273s of training, convolutional neural network yielded 98.3% accuracy. Although convolutional neural network was consistently better by yielding higher accuracy across all training times, the accuracy improvement was small, only less than 1.5% given the same training time. However, convolutional neural network's preprocessing stages of convolution and pooling would not cause significant overhead to negatively impact performance, so it is still more preferable to use convolutional neural network for classification prediction modeling. The better class prediction performance can also be supported theoretically. Application of convolution produced feature maps that represent a series of overlapping receptive fields with sparse connectivity. Each feature map is now more responsive to a larger area of pixel space. Pooling (i.e. subsampling) enhances translation invariance- the ability to detect similar or same features translated to different positions on the image. Convolution and pooling together produce maps of high level features that when used in neural network can better distinguish between digits. Less inputs (feature maps instead of all pixels) means less weights to optimize so convolutional neural network can potentially go deeper (have more layers) to be a more accurate classifier. Both the experimental results and theoretical underpinnings support the assertion that convolutional neural network makes more accurate classification of digit images more efficiently with less training time.

kNN and SVM, in general predicted with better accuracy when first applied with feature extraction descriptors. kNN and SVM, although may yield high classification accuracy- 96.62% and 98.75% respectively, did so because of good feature extraction. If kNN and SVM only dealt with corresponding pixel-to-pixel comparison, the classification accuracy decreases (raw descriptor accuracy is in general lower than Gist and Shape Context descriptor accuracy). Neural network dealt better with translational invariance, meaning that a similar or same feature can still be detected even if translated across to different positions on the image. That's because in neural network, all pixel inputs are fed into all inner nodes, so all pixels are tested for the presence of certain features. Convolution enhanced the ability to deal with translational invariance even better since convolution produced high level feature maps.

8 Conclusions

From training our learning algorithms kNN, SVM, neural network, and convolutional neural network on the digit dataset, we can see what the best classifiers are. We wanted to classify a handwritten digit image as digit 0, 1, 2,..., or 9, given the pixel intensity values. Linear SVM is good when we applied shape context as the descriptor, reaching 98.75% accuracy. Convolutional neural network is also a good classifier with classification accuracy of 98.6%. Given that there were 784 pixel inputs, the neural network yielded the highest classification accuracy with around 400 hidden layer nodes, and the convolutional neural network did so with around 20 and 100 feature maps for the 2 stages of preprocessing. Convolutional neural network (CNN) should be a better classifier than kNN and SVM because CNN can better account for translational invariance, meaning that a similar or same feature can still be detected even if translated across to different positions on the image.

9 Reference

[1] Belongie, Serge, Jitendra Malik, and Jan Puzicha. "Shape matching and object recognition using shape contexts." Pattern Analysis and Machine Intelligence, IEEE Transactions on 24.4 (2002): 509-522.

[2] "An Introduction to Convolutional Neural Networks." Stanford University, n.d. Web. 15 Dec. 2014.

[3] "Convolutional Neural Networks (LeNet)." Convolutional Neural Networks (LeNet) DeepLearning 0.1 Documentation. LISA Lab, n.d. Web. 15 Dec. 2014.

[4] Russell, Stuart J., Peter Norvig, and Ernest Davis. Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ: Prentice Hall, 2010. Print.

[5] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011.

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. JMLR, 9, 2008.
[7] Lazebnik, Svetlana, Cordelia Schmid, and Jean Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories." Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. Vol. 2. IEEE, 2006.