哈尔滨工业大学深圳研究生院

Machine Learning Report

Team member: Aditya Ardiya 15SF51800

Team member: 李淑敏 15S051006

Team member: 何 威 15S051024

Team member: 李朝竟 15S051026

Team member: 方汉生 15S051055

Catalog

1.Introduction

The ultimate goal of digital image processing is to use computer to replace human to recognize the image and find out the target which people are interested in. This is the main content of computer pattern recognition. Pattern recognition technology is to use the machine to simulate a variety of human identification, the current main simulation of human visual and auditory ability. Image pattern recognition is an important problem in the processing and identification of text, image, image and scene information by machine, which is used to solve the problem of direct communication between computer and the external environment. Image recognition consists of three parts, namely, data acquisition, data processing, discriminant classification. To solve the problem of image recognition, it can be divided into statistical pattern recognition, structural pattern recognition, fuzzy image recognition and intelligent pattern recognition. In the eighties of the 20th century new artificial neural network, as a kind of generalized intelligence pattern recognition method, with highly parallel, distributed storage, good fault-tolerant, adaptive and associative memory function, the height of the nonlinear processing ability, has many of the traditional methods are difficult to achieve success in the field of pattern recognition. And the image recognition technology of the neural network with the modern computer technology, image processing, artificial intelligence, pattern recognition theory etc. developed a new image recognition technology, is in the traditional method of image recognition based on fusion of neural network algorithm an image recognition method.

With the development of the Internet and mobile terminals, and now many matters through the Internet will be able to easily get. And with the development of information technology, many cumbersome procedures have been simplified. Improve the work efficiency of the relevant areas, but also changed the way people work in a certain way, the way of life.

So we consider allowing users to achieve camera input card number, no need to manually enter. The move to enhance the work efficiency of the industry has made tremendous contributions. From then on staff no longer need to manually input the customer number, as long as the alignment customers for the bank card to take photographs or scan, the system will automatically input card information. Both can reduce the errors caused by manual operation to the customer to cause unnecessary trouble, but also to improve work efficiency, enhance the user's satisfaction.

2.Related research

There are two interrelated motives and goals behind developing this approach.

The first is to fully utilize the power and capability of machine learning methods, and ANNs in particular, in solving the DP recognition problem. Despite the widely acknowledged success of ANNs in solving other pattern recognition problems [1], they have not yet been fully applied in the field of DP recognition. To the best of the authors' knowledge, all what any machine learning method has been used for is to filter out false positives as in [2] and [3], or to reduce the search space as in [4] and [5].

The second motive of developing this approach is to completely learn the recognition rules and features from DP instances implemented in real world applications. Almost all existing DP recognition techniques depend, to different extents, on rules/features derived from theoretical descriptions and are not validated on implemented instances [5]. Some of these techniques are based on the assumption that the theoretical DP descriptions are accurately reflected in the implemented instances although it is not usually the case [6]. To utilize the knowledge hidden in the implemented instances and to improve the recognition accuracy, our approach does not pre-set any rules or features based the DP descriptions. Alternatively, features will be selected by using feature selection methods and rules will be learnt by training ANNs.

Learning rules this way helps to avoid the problem of rules granularity which faces many of the existing recognition techniques [7], assuming that the ANNs are properly trained and not over or under fitted. Setting crisp logical rules is not actually a sensible approach because of the vague and abstract nature of DPs which allows them to be implemented in various ways [8] [9]. Also, since DPs are based on common and conventional object-oriented concepts (e.g. polymorphism), it is likely to have structures of classes that only accidentally have similar relationships as found in some DPs [2]. So, for any DP recognition technique to produce accurate results, it should have the capability to predict the intent behind constructing such structures, and whether or not they are intended to solve the problems for which the corresponding DPs are invented [10]. The failure to do so is likely to result in many false positives. The natural, and probably the best, answer to these problems and challenges lays in the capability and strength of machine learning methods, of which ANN is a powerful technique.

3.Our Solution and method

In our project, we firstly preprocess the bankcard images we obtained using our mobile phones, and then we use the functions from OpenCV foundation library to process these colorful images. After we split the images of the digits, we can use them as training samples to train a model. Finally we just use the application we developed to take a photo of a bankcard and transmit it to the sever. After testing sample come to the server which can call our model and executable file for preprocessing the image, we can just get the results of our experiment. And it will show on both the application and the web. The flow chat of the procedure is as follows:

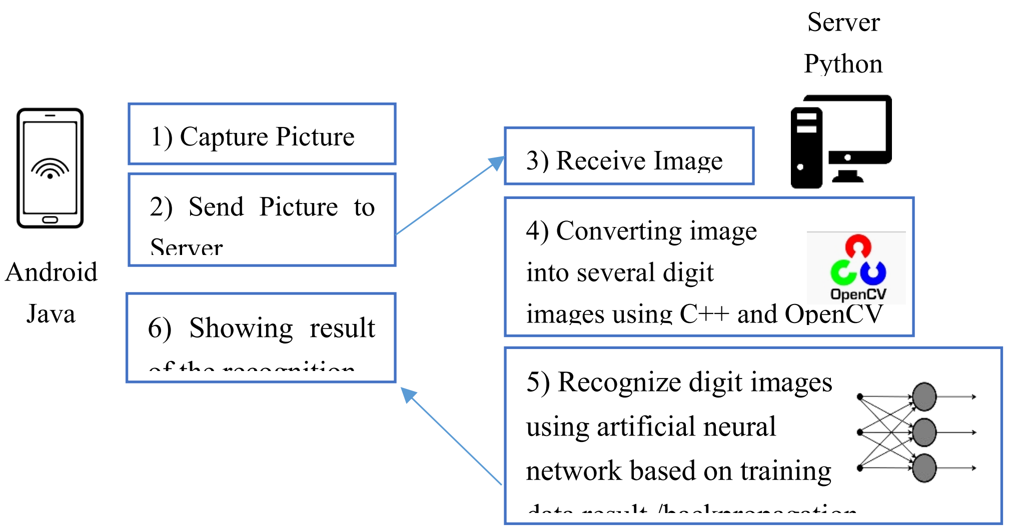

Server
Python
Android
Java
1) Capture Picture
2) Send Picture to Server
6) Showing result
3) Receive Image
4) Converting image into several digit images using C++ and OpenCV
OpenCV
5) Recognize digit images using artificial neural network based on training

Fig 3.1 Flow chat of our system

3.1 Data Preprocessing

In this step, we implement the segmentation of the bankcard images. We use our phones to get the bankcard images we need firstly. This step request the number of bankcard must be large, or we can not get accurate results. This step involves using C++ programming language, OpenCV foundation library and Visual Studio 2013 development environment. We use some methods to implement the segmentation of the bankcard pictures, and we use Image2gray.cpp files to binarize the colorful pictures:

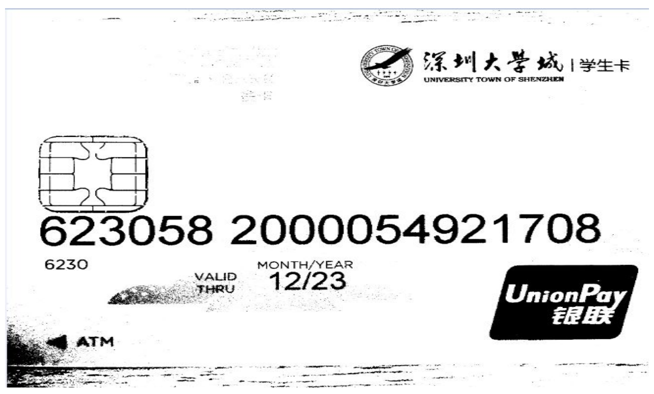


Fig 3.2 Binary bankcard image

Then we use the bankCard.cpp to segment the binary bankcard image and save them in a file fold, the results are given in Fig 3.3:

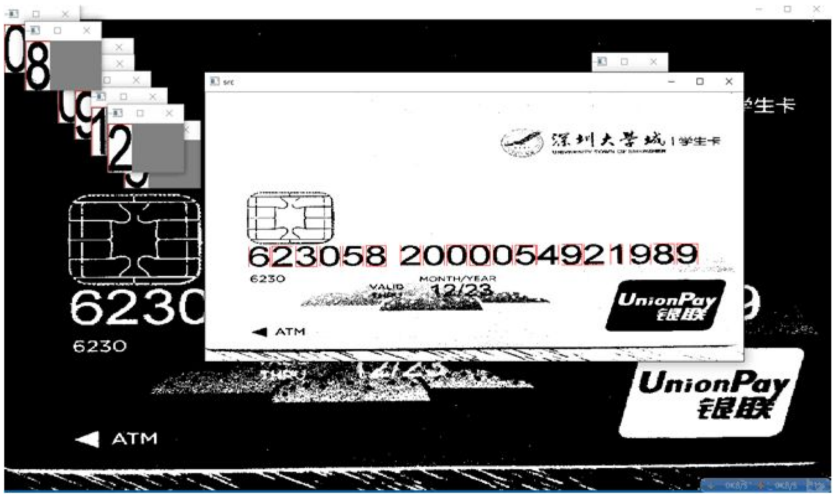


Fig 3.3 Segmentation for the bankcard

The mainly procedure of this part is in Fig 3.4. We firstly use the histogram equalization to make the pictures we obtained more clearly. Then we transform it into grey scale image through using fixed threshold to the binaryzation. Finally we can extract the components and get the training datasets.

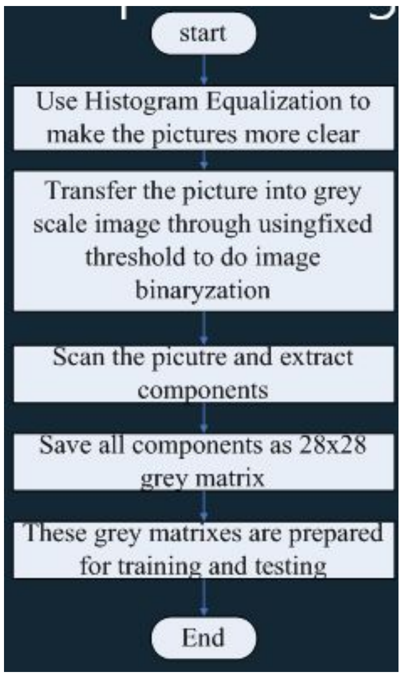

start
Use Histogram Equalization to make the pictures more clear
Transfer the picture into grey scale image through usingfixed threshold to do image binaryzation
Scan the picutre and extract components
Save all components as 28x28 grey matrix
These grey matrixes are prepared for training and testing
End

Fig 3.4 Flow chat of the image processing

3.2 Training models

Actually we have achieve three models for training our models. They are neural network algorithm , kNN algorithm, and BP algorithm. But after doing some tests, we find the neural network works best, so choose that algorithm. Since we implement these models buy different members, therefore we choose python and C++ language respectively.

Then we use the open dataset mnist.pkl.gz for training and testing our model since we can not obtained enough training dataset at the beginning. This dataset has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. The following pictures in Fig 3.5 shown are some of them:

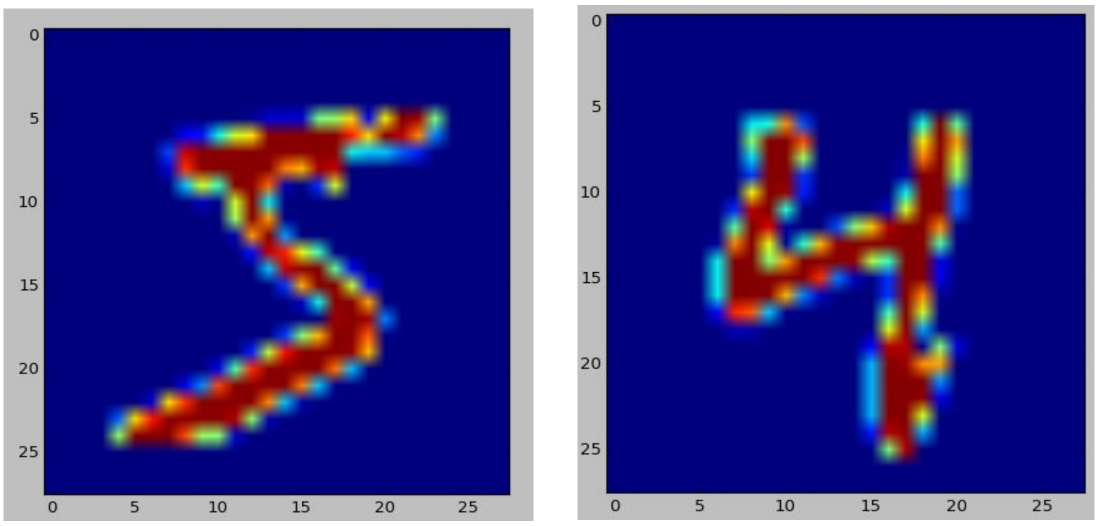
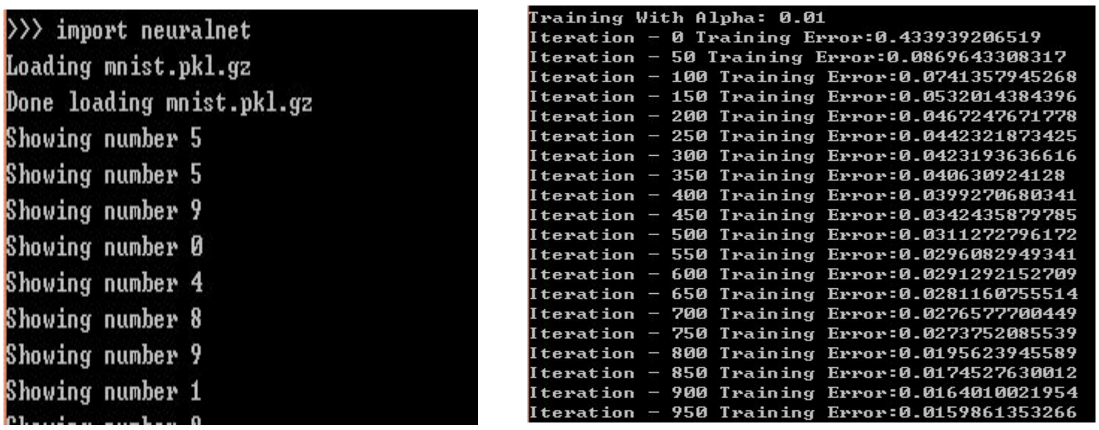


Fig 3.5 Original dataset used for selecting models

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. After we setting the parameters, we can find that neural network work best, the following picture shows the experimental results:



```
>>> import neuralnet
Loading mnist.pkl.gz
Done loading mnist.pkl.gz
Showing number 5
Showing number 5
Showing number 9
Showing number 0
Showing number 4
Showing number 8
Showing number 9
Showing number 1
```

```
Training With Alpha: 0.01
Iteration - 0 Training Error:0.433939206519
Iteration - 50 Training Error:0.0869643308317
Iteration - 100 Training Error:0.0741357945268
Iteration - 150 Training Error:0.0532014384396
Iteration - 200 Training Error:0.0467247671778
Iteration - 250 Training Error:0.0442321873425
Iteration - 300 Training Error:0.0423193636616
Iteration - 350 Training Error:0.040630924128
Iteration - 400 Training Error:0.0399270680341
Iteration - 450 Training Error:0.0342435879785
Iteration - 500 Training Error:0.0311272796172
Iteration - 550 Training Error:0.0296082949341
Iteration - 600 Training Error:0.0291292152709
Iteration - 650 Training Error:0.0281160755514
Iteration - 700 Training Error:0.0276577700449
Iteration - 750 Training Error:0.0273752085539
Iteration - 800 Training Error:0.0195623945589
Iteration - 850 Training Error:0.0174527630012
Iteration - 900 Training Error:0.0164010021954
Iteration - 950 Training Error:0.0159861353266
```

Fig 3.6 Results from the original dataset

Then we start to train the model use our dataset obtaining from the bankcard shown in the following pictures:

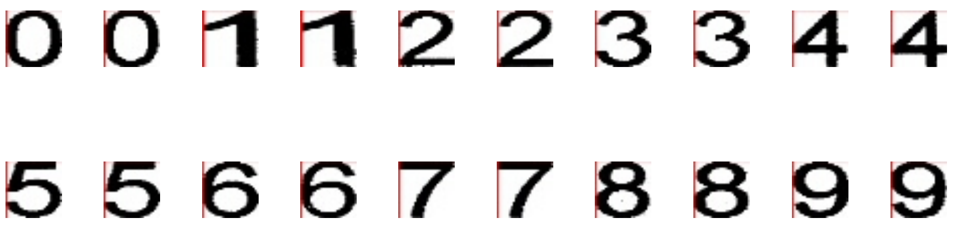


Fig 3.7 Dataset we segment

And correspondingly we train the model use the training dataset, before we use the dataset, we also transform the dataset into the 28x28 pixels. The size of these images are similar to the minist dataset, here we just put some of them in Fig3.8:

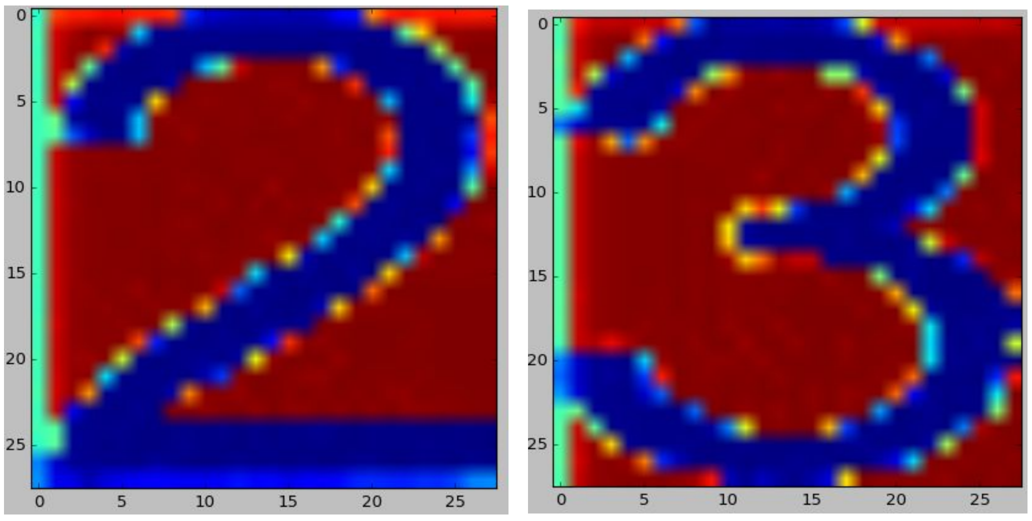
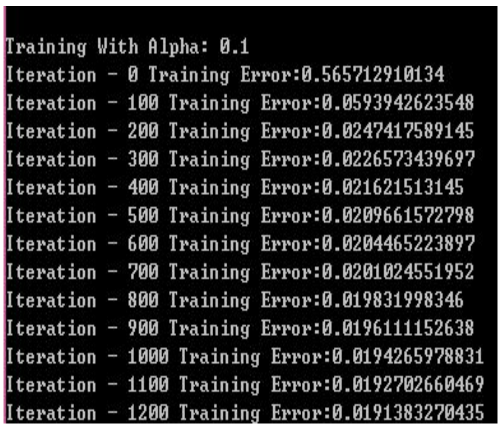
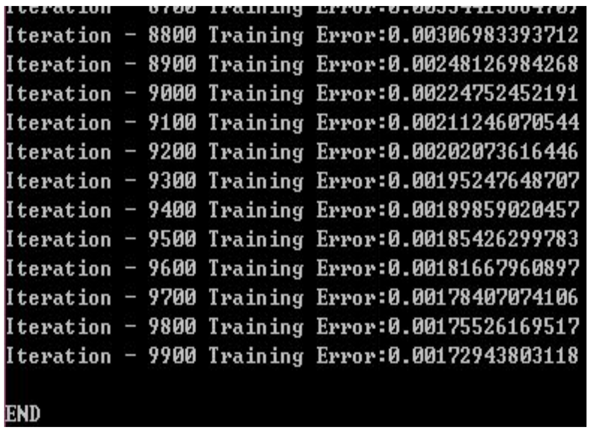


Fig 3.8 Training dataset actually use

the following pictures show the running results of our model:



```
Training With Alpha: 0.1
Iteration - 0 Training Error:0.565712910134
Iteration - 100 Training Error:0.0593942623548
Iteration - 200 Training Error:0.0247417589145
Iteration - 300 Training Error:0.0226573439697
Iteration - 400 Training Error:0.021621513145
Iteration - 500 Training Error:0.0209661572798
Iteration - 600 Training Error:0.0204465223897
Iteration - 700 Training Error:0.0201024551952
Iteration - 800 Training Error:0.019831998346
Iteration - 900 Training Error:0.0196111152638
Iteration - 1000 Training Error:0.0194265978831
Iteration - 1100 Training Error:0.0192702660469
Iteration - 1200 Training Error:0.0191383270435
```



```
Iteration - 8800 Training Error:0.00306983393712
Iteration - 8900 Training Error:0.00248126984268
Iteration - 9000 Training Error:0.00224752452191
Iteration - 9100 Training Error:0.00211246070544
Iteration - 9200 Training Error:0.00202073616446
Iteration - 9300 Training Error:0.00195247648707
Iteration - 9400 Training Error:0.00189859020457
Iteration - 9500 Training Error:0.00185426299783
Iteration - 9600 Training Error:0.00181667960897
Iteration - 9700 Training Error:0.00178407074106
Iteration - 9800 Training Error:0.00175526169517
Iteration - 9900 Training Error:0.00172943803118

END
```

3.3 Our model and system

In our neural model, We use 85 hidden nodes and 10 output layers, training iteration is 10000 times. After we testing, we choose 0.1 as the value of the learning rate. The input image size is 28x28 pixels to adapt our system, and we use two sigmoid function layers. The main algorithm we use is as follows:

Each training examples is a pair of the form($\vec{x},\vec{t}$), where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of the target network output values.

η is the learning rate. n_{in} is the number of network output inputs, n_{hidden} is the number of units in the hidden layer, and n_{out} is the number of output units.

The input from unit i into unit j is denoted x_{ji} , and the weight from unit i to unit j is denoted ω_{ji} .

Create a feed-back network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.

Initialize all network weights to small random numbers.
Until the termination condition is met, do:

For each ($\vec{x},\vec{t}$) in training examples, do

Propagate the input forward through the network:

1.Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2.For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. For each hidden unit h, calculate its error δ_{h}

$$\delta_{\mathrm{h}} \leftarrow o_h(1-o_h) \sum_{k\in outputs} \omega_{kh}\delta_k$$

4. Update each network weight ω_{ji}

$$\omega_{ji} \leftarrow \omega_{ji} + \Delta\omega_{ji}$$

Where

$$\Delta\omega_{\mathrm{ji}} = \eta\delta_j x_{ji}$$

The most common type of artificial neural network consists of three groups, or layers, of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units. Just like the Fig3.9 shows below:

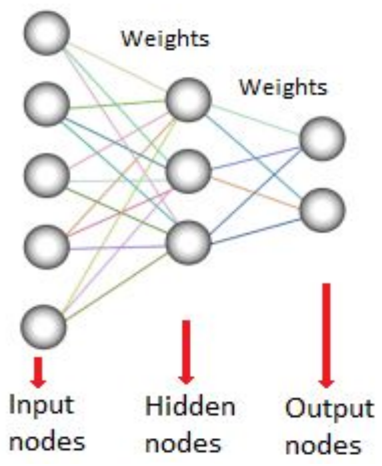


Fig 3.9 An example of a simple feedforward network

The activity of the input units represents the raw information that is fed into the network.

The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organization. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.

For sigmoid units, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurones than do linear or threshold units, but all three must be considered rough approximations. Fig 3.10 shows an example of sigmoid threshold unit.

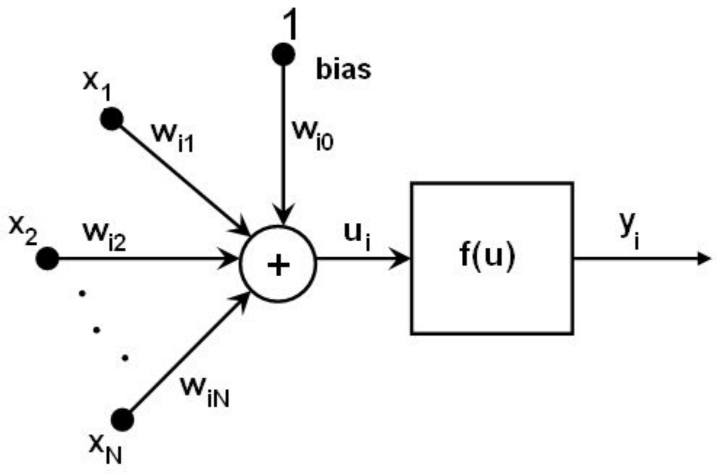


Fig 3.10 Sigmoid threshold unit

After training the neural network model using our training dataset and obtained a model, we focus on the application development. First we create the server in python to run the preprocessing using executable .exe generated from OpenCV modules and to feedforward the ANN using the weight resulted from the training model.

The web server use Flask framework as the foundation of the HTTP. It handles the GET request of URL “/” by showing some web form to upload an image. It also provides a service to handle POST request of URL “/upload” by taking the request file image, saving it into disk, do the preprocessing using executables from OpenCV, feed-forward the Artificial Neural Network, and finally show the output using JSON formats which includes the image name(for it to be shown in the android application), the prediction(the index the highest value of nodes in the output layer), and confidence(the value of the highest node in the output layer).

Then, we create the android application on the Android Studio IDE. We use the android as the basis for the smartphone to easily use the camera to take a picture of the bank card. We also provides a way for user to choose an image from the gallery as an alternative to the camera. We also provide button to send the image to the server and get the result of the prediction. The bank card number will be shown in a large, bold, TextView. The details of each prediction digits will be shown on the ListView. Each of the ListView items contains 1 ImageView to show the segmented image from the webserver, 3 TextView to show the filename, prediction, and confidence, and 1 ProgressBar to show the level of the confidence. We also provide a setting to change the server url, and save it using android preferences. The server url acts as the connector with the webserver in the local network.

When we finish the development of the android APP, we use the APP to obtain the test picture and send it to our server. After recognizing the card, it will show the results (The account number of the bankcard) on the interface. On the other hand, we can also show upload the pictures on the phone or web, and its results can also be shown in the web.

4.Experimental results representation

In this part, we are going to show the details of our project results. We have implemented this function on both PC and Android application. Firstly, we use the application to obtain an image of the bankcard, Fig 4.1 shows this:

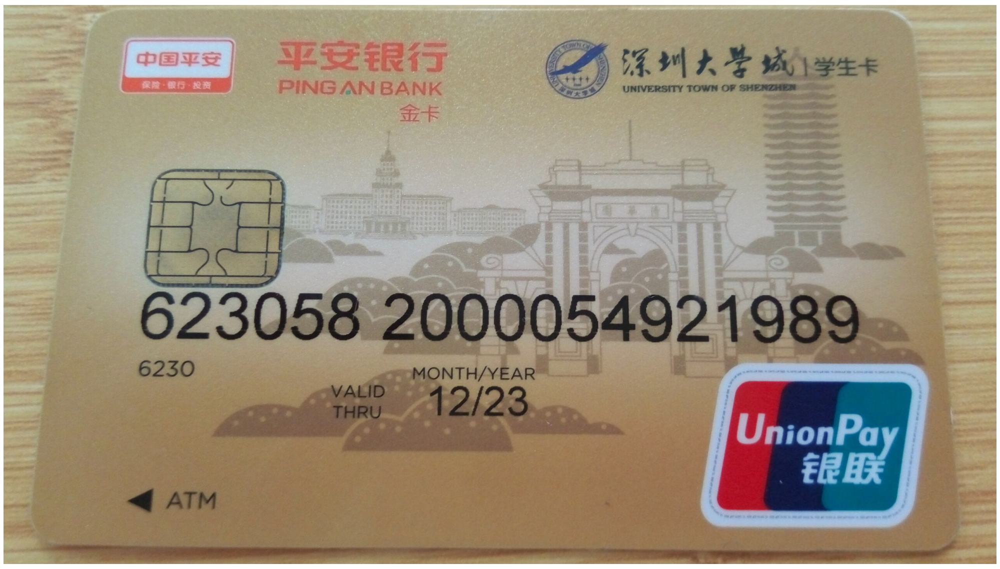
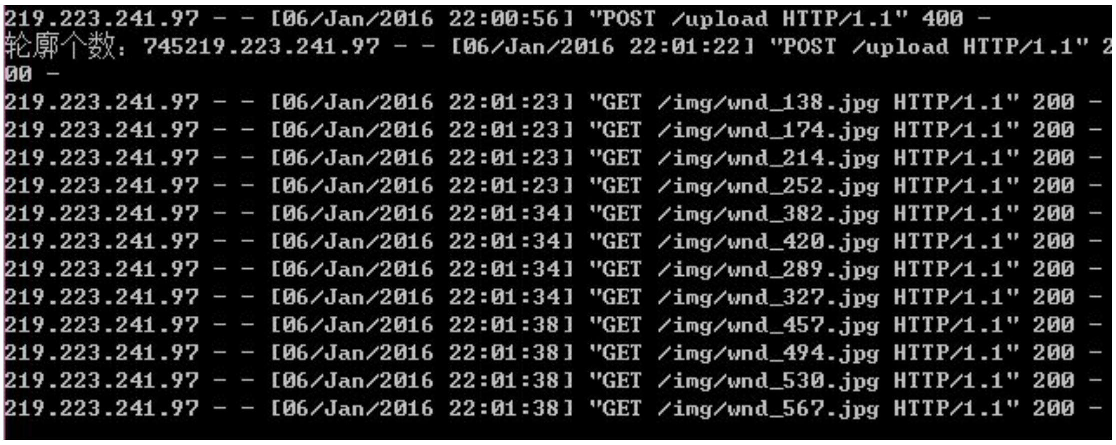


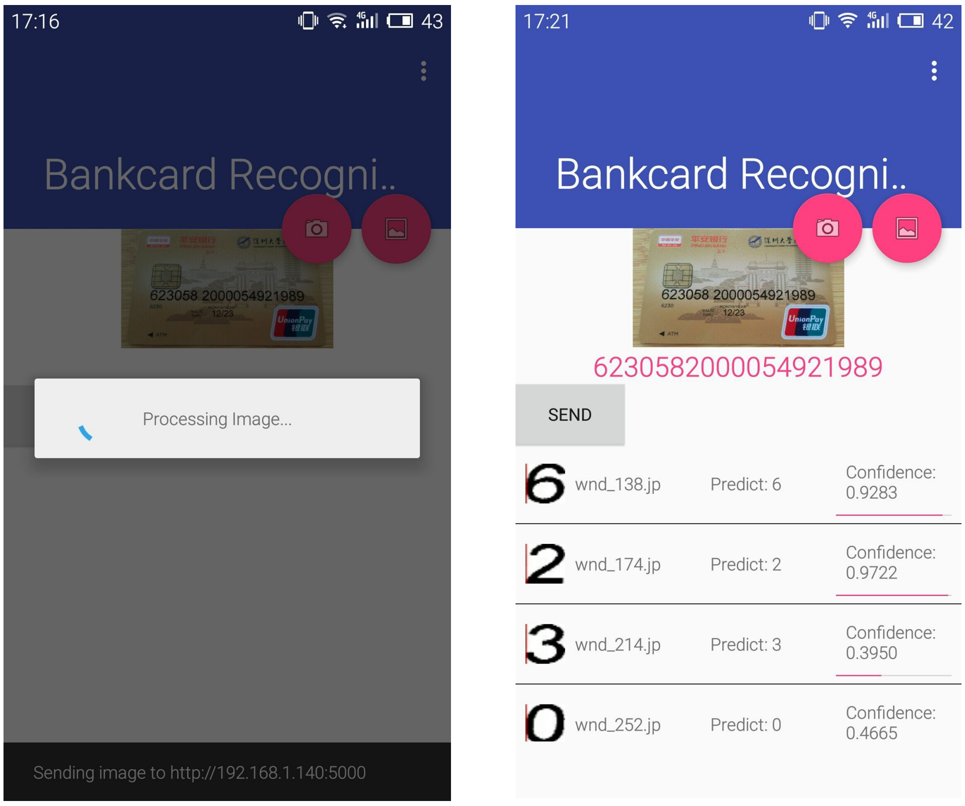
Fig 4.1 bankcard image we obtain and use

Then we put the upload button to transmit the image to the server. After the server get the image, it will call the executable file to do the segmentation. The dos interface shows the procedure in Fig 4.2 as follows:



```
219.223.241.97 - - [06/Jan/2016 22:00:56] "POST /upload HTTP/1.1" 400 -
轮廓个数: 745219.223.241.97 - - [06/Jan/2016 22:01:22] "POST /upload HTTP/1.1" 2
00 -
219.223.241.97 - - [06/Jan/2016 22:01:23] "GET /img/wnd_138.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:23] "GET /img/wnd_174.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:23] "GET /img/wnd_214.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:23] "GET /img/wnd_252.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:34] "GET /img/wnd_382.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:34] "GET /img/wnd_420.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:34] "GET /img/wnd_289.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:34] "GET /img/wnd_327.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:38] "GET /img/wnd_457.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:38] "GET /img/wnd_494.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:38] "GET /img/wnd_530.jpg HTTP/1.1" 200 -
219.223.241.97 - - [06/Jan/2016 22:01:38] "GET /img/wnd_567.jpg HTTP/1.1" 200 -
```

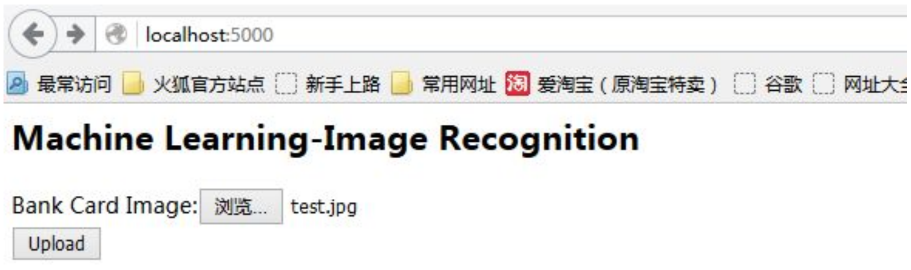
At the same time, we can get the results from the server and show them in our application in Fig 4.3:



Sending image to http://192.168.1.140:5000

Fig 4.3 results shown in our application

After we check each number of the bankcard account, we can find it make no mistakes in recognition. We also show the results presented by the web in Fig 4.4 and Fig 4.5 respectively:

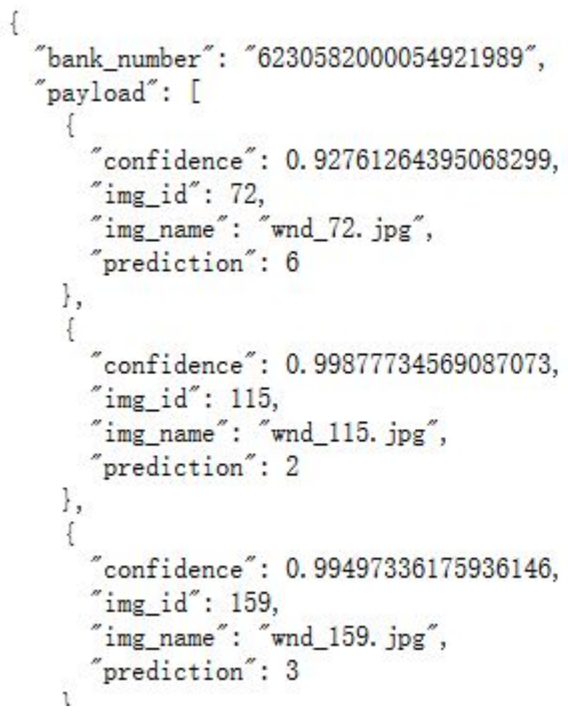


Machine Learning-Image Recognition

Bank Card Image: 浏览... test.jpg

Upload

Fig 4.4 Upload image on the web



```
{
  "bank_number": "6230582000054921989",
  "payload": [
    {
      "confidence": 0.92761264395068299,
      "img_id": 72,
      "img_name": "wnd_72.jpg",
      "prediction": 6
    },
    {
      "confidence": 0.99877734569087073,
      "img_id": 115,
      "img_name": "wnd_115.jpg",
      "prediction": 2
    },
    {
      "confidence": 0.99497336175936146,
      "img_id": 159,
      "img_name": "wnd_159.jpg",
      "prediction": 3
    }
```

Fig 4.5 Results shown in the web

5.Conclusion

In this project, we have learn a lot through the whole procedure, including the image preprocessing, dataset obtaining, models selecting and training, web server buliding and Android application developing. Most importantly, we can use what we learn in this course to work for our project. It is useful for us to understand the machine learning algorithms.

At the same time, we learn how to cooperate with each other, how to make a team work efficiently. Thanks to all the team members, we wouldn't finish this project without everyone's efforts. And also, we need to say thanks to the teacher and TAs for contributions to this course.

6.Reference

[1] C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995.
[2] R. Ferenc, A. Beszedes, L. Fulop, and J. Lele, "Design pattern mining enhanced by machine learning," in Proceedings of The 21st IEEE International Conference on Software Maintenance, 2005, pp. 295–304. [3] M. Zanoni, "Data mining techniques for design pattern detection," Ph.D. dissertation, Universita degli Studi di Milano-Bicocca, 2012.
[4] S. Uchiyama, H. Washizaki, Y. Fukazawa, and A. Kubo, "Design pattern detection using software metrics and machine learning," in Proceedings of The First International Workshop on Model-Driven Software Migration, 2011, pp. 38–47.
[5] Y.-G. Gueh´ eneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design ´ patterns," in Proceedings of The 11th Working Conference on Reverse Engineering, 2004, pp. 172–181.
[6] Y.-G. Gueh´ eneuc, J.-Y. Guyomarch, and H. Sahraoui, "Improving ´ design-pattern identification: a new approach and an exploratory study," Software Quality Journal, vol. 18, no. 1, pp. 145–174, 2010.
[7] J. Niere, W. Schafer, J. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery," in Proceedings of The 24rd International Conference on Software Engineering, 2002, pp. 338–348.
[8] Z. Balanyi and R. Ferenc, "Mining design patterns from c++ source code," in Proceedings of The International Conference on Software Maintenance, 2003, pp. 305–314.
[9] J. Smith and D. Stotts, "Spqr: flexible automated design pattern extraction from source code," in Proceedings of The 18th IEEE International Conference on Automated Software Engineering, 2003, pp. 215–224.
[10] G. Antoniol, G. Casazza, M. Di Penta, and R. Fiutem, "Object-oriented design patterns recovery," Journal of Systems and Software, vol. 59, no. 2, pp. 181–196, 2001.