POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Master of Science in Computer Engineering

Master Degree Thesis

Exploiting Parallel Neural Networks for Automatic Recognition of Characters and Mathematical Symbols

Supervisor

Prof. Paolo PRINETTO

Tutor

Dr. Giuseppe AIRO FARULLA

Dr. Nadir MURRU

Candidate

Yu YE

student no. 222204

December 2017

Abstract

The development of a reliable Optical Character Recognition (OCR) software specifically tailored for scientific texts has been long required from the community of blind and visually impaired people, since only very few scientific texts are accessible. This thesis deals with a recognition core of OCR leveraging on the artificial neural networks (ANN) for recognizing both normal and scientific text (i.e. Latin characters, Greek characters and mathematical symbols). We present the procedures for training each neural network with the back propagation method, and then develop three voting strategies in order to make these neural networks work in parallel. The overall architecture of our OCR is based on the parallel neural networks (PNN). The outcome of experimental validation demonstrates that the voting strategy based on the mean Euclidean norm has the best and steadiest behaviors with accuracy 94.9%.

Keywords: Optical Character Recognition, Artificial Neural Network, Parallel Neural Network

Acknowledgements

I would like to express my deepest appreciation to Prof. Paolo Prinetto, Prof. Anna Capietto, Dr. Giuseppe Air*ò* Farulla, and Dr. Nadir Murru for their guidances and supports.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optical Character Recognition (OCR) software is an application of pattern recognition, which converts a scanned image of typewritten, handwritten, or printed text into a machine editable format. Assistive technology including the assistive devices for the blind and visually impaired people performs satisfactorily with respect to screen reader software and screen display software based on the machine-editable format, which is generated from the machine or converted from the uneditable format. There exists OCR software, which mainly focus on the characters (Latin). However, they still have a long way to go as far as more scientific text is concerned. Currently, InftyReader is the unique OCR application that recognizes and translate the scientific documents into several machine-editable formats (e.g. LaTeX). The development of a reliable OCR software specifically tailored for scientific texts has been long required from the community of blind and visually impaired people.

In this thesis, we develop the recognition core of OCR based on the artificial neural network. This thesis and relative research activities are carried in cooperation with Universit*à* degli Studi di Torino, Department of Mathematics and CINi National Lab on Assistive Technologies.

In our study, we aim at pattern recognition of the Latin characters, Greek characters, and mathematical symbols in different font sizes. However, it's very difficult to put all characters and symbols into one neural network. In fact, a single ANN can not discriminate properly among hundreds of different patterns. So we decide to train six feed-forward neural networks, each of which is in charge of different type of characters and font sizes, and then make these trained neural networks work in parallel with the voting algorithm.

We create the image files as the raw materials for training and testing the neural networks. As we only focus on the pattern recognition without segmentation, each image only includes one character. The image files are generated automatically by the Matplotlib, which is a Python library. Then we train each neural networks with the back-propagation method separately and adjust their structure (i.e. the number of neurons in the hidden layer) and parameters (i.e. learning rate and the initial range of weights) for the optimal performance evaluated by the elapsed time and training step. Next, we test the neural networks and readjust the structure and parameters to improve their study abilities (i.e. testing accuracy). After testing, we get neural networks, each of which can recognize a certain number of characters or symbols with high accuracy, and then make them working in parallel. The overall system uses a voting algorithm for selecting the most proper output among outputs from the paralleled-working neural networks as the final result. We develop the strategies for the voting algorithm, evaluate their accuracies against our data set and then analyze their behaviors.

This thesis is divided into four sections.

Section one is the introduction. The chapter is subdivided into three parts. Part one describes the OCR software development nowadays, part two describes our goal, and part three is the outline of the order of information in this thesis.

Section two is the overview of the artificial neuron network. The chapter consists of four parts. Part 1 focuses on how single neuron works and part 2 introduces how a neuron network works. Part 3 describes the learning paradigms, especially the supervised learning, for the artificial neuron network. Part 4 is about the back-propagation method used in this thesis for training the neuron networks.

Section three describes the procedures of experimental tests including the creation of images, image preprocessing, training and testing each neural network separately and developing the strategies for the parallel neural network.

Section four summarizes the general procedures and their corresponding results.

Chapter 2

Backgrounds

2.1 Neuron

2.1.1 The Biological Neuron

The artificial neuron network is inspired by the human brain. It's a simulation of the real nervous system. At the beginning of this chapter, let's have a general view of how a biological neuron works. The sketch is shown in Figure 2.1.1. [1].

There are several main components of a neuron which are dendrites, cell body, and axons. The dendrites are the terminal for receiving the signal, i.e. electronic stimulation, from the other neurons or from the outside of the world. The cell body contains the mechanisms, especially nucleus, which keeps the cell alive. The axon is the terminal for sending the electronic stimulation to other neurons.

Generally, all neurons have three basic functions, which is receiving signals, integrating incoming signals, and sending signals to target neurons.

Dendrites Most neurons receive the input signals from their dendrite trees. A neuron usually has a brunch of dendrites, each of which could receive many input

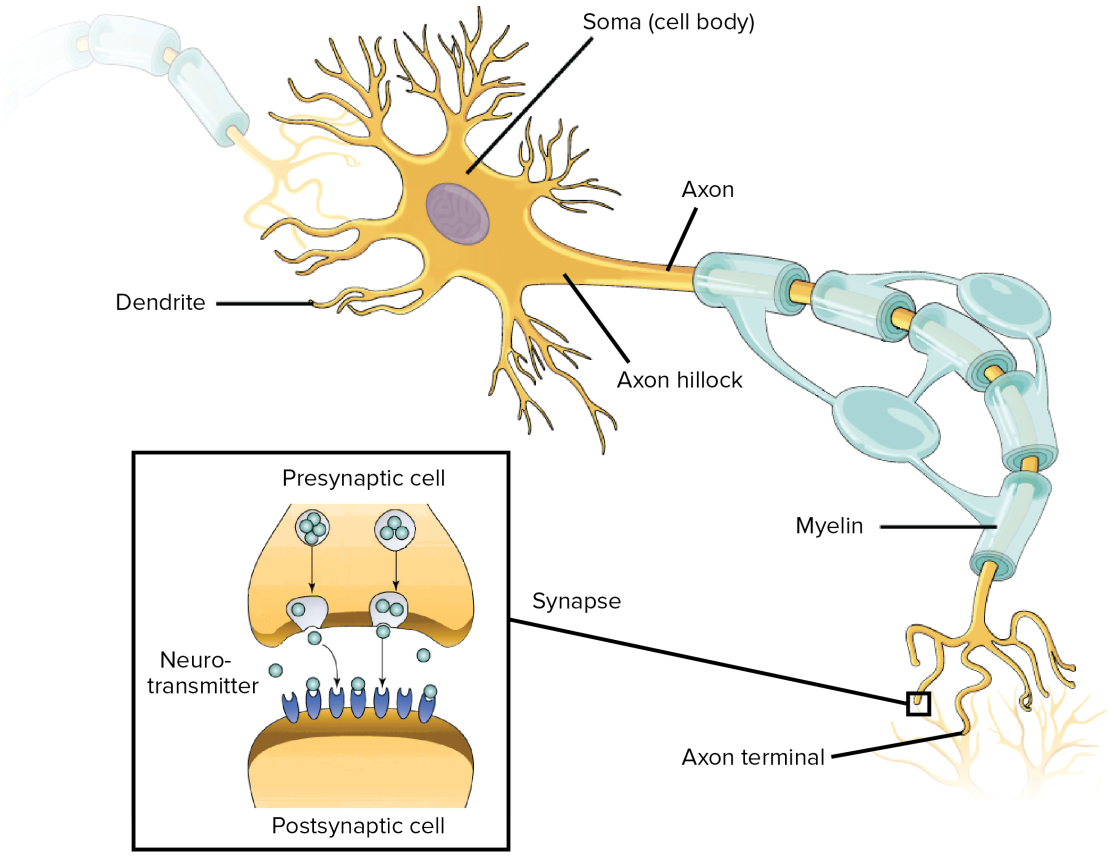


Figure 2.1.1: The structure of a neuron

signals. Whether a neuron is stimulated to firing an impulse depends on the sum of all the signals (positive and negative), which it receives. If the sum is over a certain threshold, the nerve impulse is conducted down the axon, which is in charge of sending the signals [2].

Axon Compared with dendrites, the axon is different. Many axons are covered with myelin, which is an insulating substance, that helps axons to convey nerve impulse rapidly. When an axon receives the signals conducted from the dendrites, it distributes the signals into its branches, which are also known as never terminals. These terminal make the connections to the target neurons.

Synapses The connections between neurons are made onto the dendrites and somas of other neurons, These connections known as synapses, carries information or signals from the pre-synaptic neuron (sender) to post-synaptic neuron (receiver or target). In most synapses, a signal is transmitted in neurotransmitters which are the forms of chemical messengers. When an action potential is conducted into an axon terminal, the release of neurotransmitter is triggered. It released from the pre-synaptic cell to post-synaptic cell for conveying a positive or negative signal.

Action potentials In the human body, there are many sensors with respect to the different functions. For example, in the eye, there are rods and cones, which are neurons that can be stimulated by the light, and then produce pulses called action potentials.

The action potential is a temporary shift in the neuron's potential, which is cause ions suddenly flowing in and out of the neuron. We only mention about the three states rather than too many details about how the ions flow. These three states are deactivated (i.e. at rest), activated and inactive (i.e. after the depolarizes).[2]

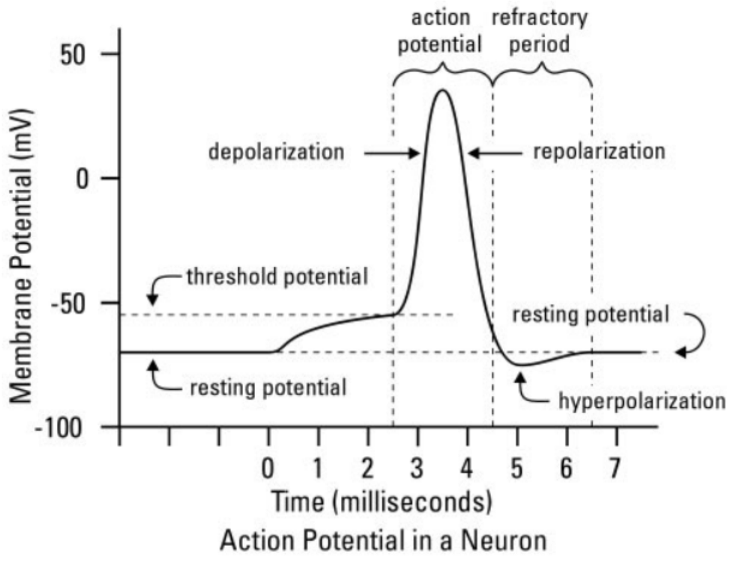


Figure 2.1.2: The action potential generated when a neuron is stimulated

The general procedure is shown in Figure 2.1.2[3], before stimulation, the neuron is at rest, which means the neuron is charged up and ready to produce a pulse. While the stimulation applied, the neuron initializes an action potential, which is the depolarized of a neuron. After reaching the peak value, the voltage drops down rapidly and then re-balance to the rest state (below the threshold voltage). Additionally, if the dendrites of a neuron receive many simulations from another, it produces more action potentials, and vice verse, which is also shown in Figure 2.1.3. The whole process shows how the signal transmits inside a single neural.

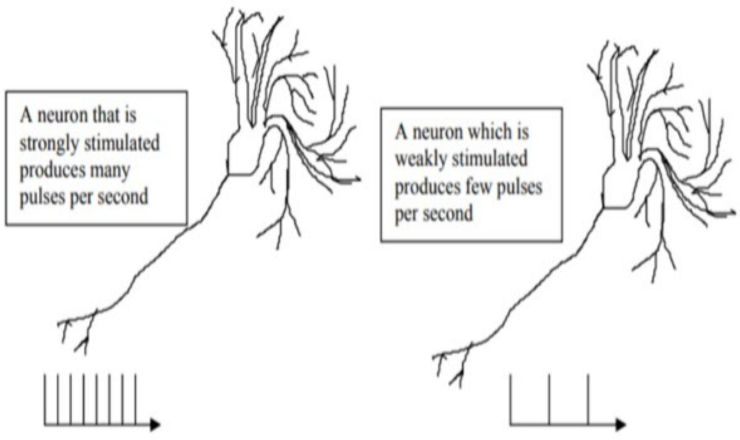

A neuron that is strongly stimulated produces many pulses per second
A neuron which is weakly stimulated produces few pulses per second

Figure 2.1.3: The value of action potential is related with stimulation level

2.1.2 The Artificial Neuron

After the introduction of the biological neuron. We can begin to study artificial neuron.

The concept of the artificial neuron is nearly started with Warren McCulloch and Walter Pitts who proposed a simple artificial model called Perceptron in 1943. Then in the 1950s, it's developed by Frank Rosenbaltt[4], who is inspired by the previous work.

The sketch is shown in the Figure 2.1.4. There are three inputs, and each of them is equipped with the corresponding weights, which represents the strength of the synaptic connection of its dendrite in the biological neuron. Generally, the perceptron could have more or fewer inputs.

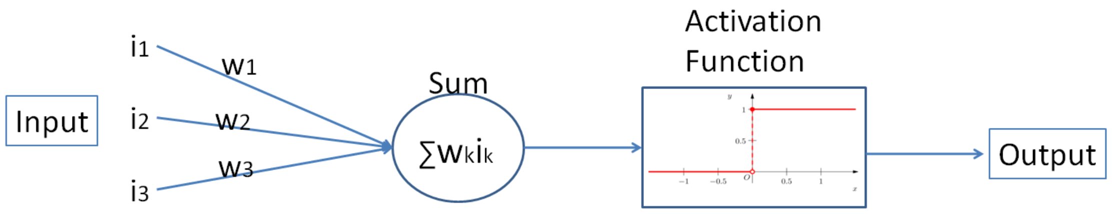


Figure 2.1.4: The basic structure of Perceptron

A perceptron takes these binary inputs and produces a binary output. The output is zero or one, which depends on whether the weighted sum of input is less or greater than a certain value of the threshold. The formulas are shown as following.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \qquad (1)$$

In this basic mathematical model, the perceptron is able to solve some simple decision problems. In a practical way, we could set some factors as the input and put the weights, then we sum up with weights to get the final yes or no binary output.

After the introduction about perceptron, it's time to talk about the type of the neuron. In the previous formula (1), the threshold value is a step function, It's easy to calculate. However, as there are more and more complicated problems applied, it's found that the continuous one is more flexible for the real application, for example, the sigmoid and tanh function. The explanation is shown in the next section, when we discuss the artificial neural network.

The development of the perceptron is based on the single neuron and the linear system. However, it's very limited for analyzing the real-life problem. The most famous problem is Exclusive-Or problem. In this case, no matter how you change the weigh of each input, it's not possible to get the expected output. So In 1969, Marvins Minsky and Semour Papert published a book 'Perceptrons'.[5] In this book, they proposed a new model of the neural net where a collection of connected neurons rather than a single neuron, and it's the fundamental model of the neural network nowadays, which is able to recognize the complex patterns.

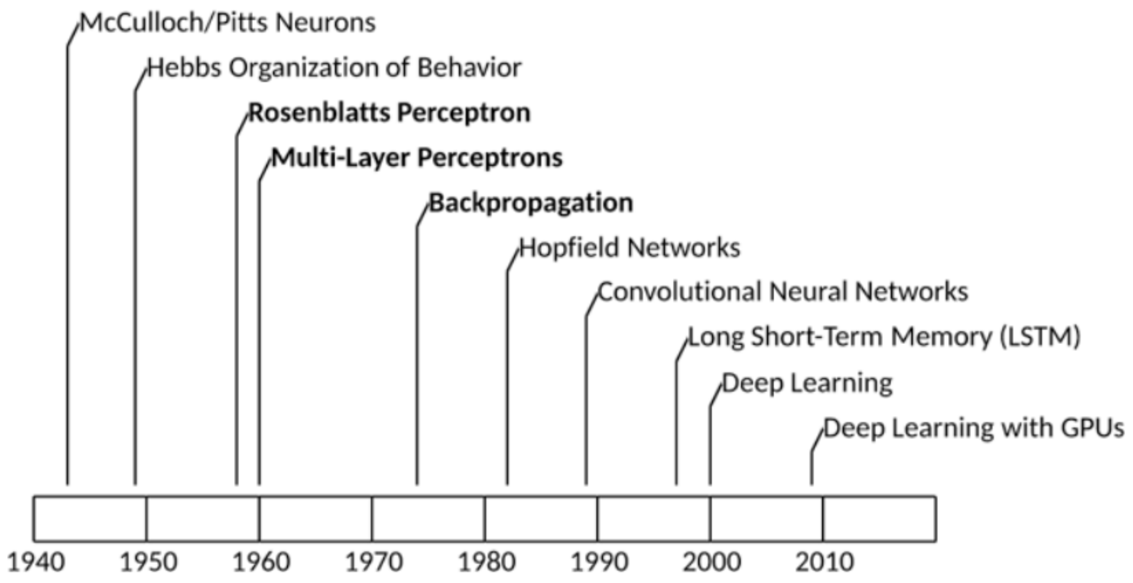


Figure 2.1.5: The history of neural network

Generally, as shown in Figure 2.1.5, the whole history of the neural network began with the computational model of neuron in the early 1940s. Later, Hebbian learning observed from the behavior of synapse was created. In 1958, perceptron, which is a simple neural model that could classify the data into two sets. However, as the study went further, it's found that this model can not deal with the complicated problems, one of which is the XOR. Then in 1969, this model was purposed with its limitation, and the researchers' attention turned back to the development of symbolic method. In 1975, the backpropagation method was created, which could train the multi-layer perceptrons successfully. This method is used in our study, and it is introduced in section 2.4.

2.2 The Artificial Neural Network

The artificial neural network is inspired by the nervous system introduced before, as the Figure 2.2.1[6] shows. In this part, we introduce the feed-forward artificial neural network used in our study.

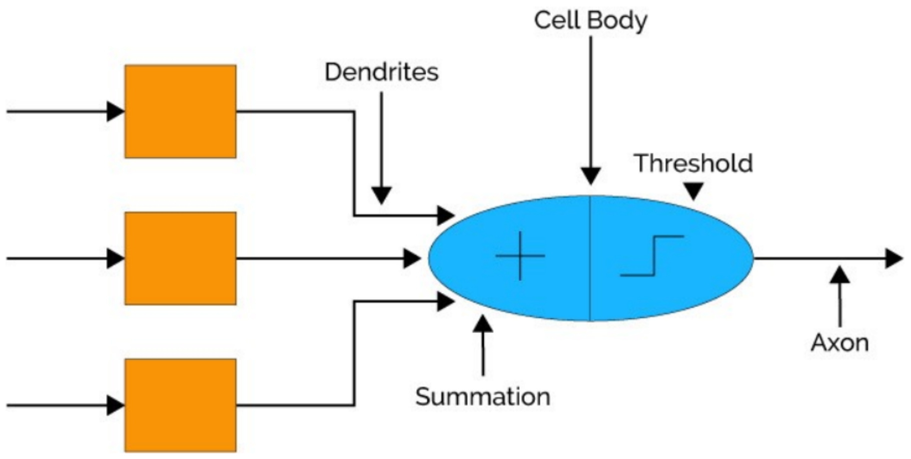


Figure 2.2.1: Analogy of Artificial Neural Network With Biological Neural Network

2.2.1 Structure

A neural network is composed of the neurons and their connections.

The neurons inside the network are divided into three main layers, which are the input layer, the hidden layer, and the output layer. The number of hidden layers could be more than one, so that kind of neural network is also called Multiple-Layer Neural Network.

Each layer includes a number of neurons, which depends on the practical case. The basic structure is shown in Figure 2.2.2.

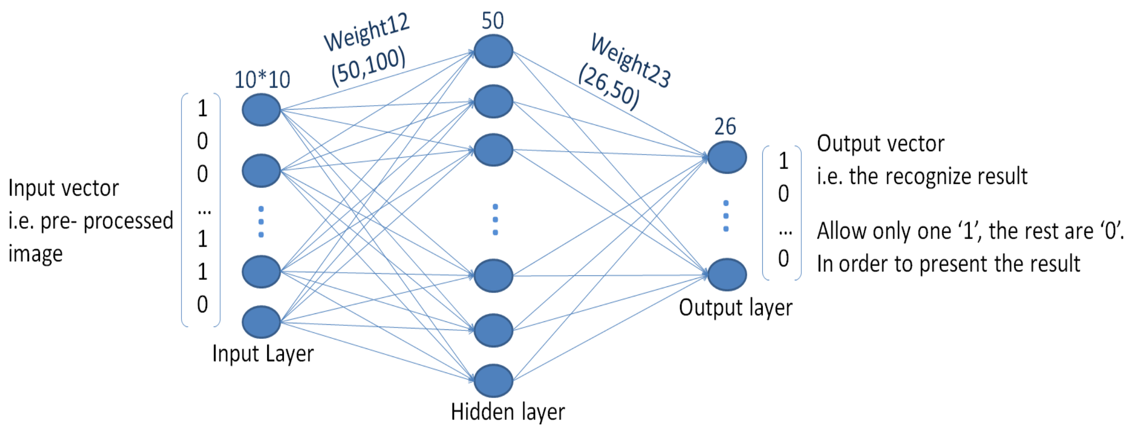


Figure 2.2.2: The simple structure of artificial neural network

Input Layer Each neuron in this layer represents a feature unit to be analyzed later. For example, we want to recognize a scanned image with feature pixels 10∗10. In this case, 100=10*10 neurons are assigned to the input layer of the neural network. Each neuron is scaled between zero and one (since we remove the irrelevant color information in this study). As there are 100 valid units to be analyzed, each neuron in the input layer is in charge of a pixel and processes the info of this pixel to the neurons in next layer.

Output Layer Each neuron in this layer represents a recognized element. There is a certain threshold set manually to evaluate whether each neuron is active or not. Usually, this threshold is 0.5 as default. For example, if an ANN focuses on recognizing the 26 Latin characters, then usually the number of neurons in the output layer is fixed in 26, each neuron is in charge of a corresponding result. If this ANN gets the output result which is a vector with the first element active and the rest of the elements inactive, it recognizes this image as character assigned in the first position.

Hidden Layer The neurons in this layer make the connections between neurons input layer and neurons in output layer. It's important for the artificial neural network to learn the complicated staff (e.g. XOR problem). Different with the input and output layer, the number of neurons in the hidden layer is not fixed with a certain value. So assigning the number of neurons in the hidden layer becomes an essential step in the construction of the artificial neural network. In Section 3, we do the repetition test with different values of the number of neurons in hidden layer, in order to find the optimal structure for each neural network. Referring to Jeff Heaton's idea, "the optimal number of neurons in the hidden layer is between the number of neurons in the input and output layer"[7]. If it's too many, overfitting might occur, as there is only a limited number of images (limited dataset) to be processed, which is not enough to train all the hidden neurons. If it's too few, underfitting may occur, as the limited number of the hidden neurons is not enough to process so much information. According to his idea, the neural networks in this study should have the number of neurons in the hidden layer between [24, 18200= 130*140]. 24 is the smallest number of neurons in the output layer, which is from the Greek neural networks, and 18200 is the largest number of the input layer which is from the mathematical neural networks. However, with the consideration of the elapsed time, we decide to limit this value within [100, 800]. As for the number of hidden layer in our study, we only use the single hidden layer to build the neural networks.

Connections The connections are used for the neurons in the different layers. Any two neurons in the adjoining layers have a connection. Each connection is equipped with a corresponding weight, which processes the value from the predecessor to the successor. For the initial weights, we generate them randomly within the range of [-kk/2, kk/2]. In Section 3, we do the repetition test with the different values of kk for exploiting the optimal performance of the neural network. The kk is also an important parameter of the back-propagation method mentioned in Section 2.4.

Neuron In the last subsection, we introduce the perceptron neuron with the step function, and also mention that as the problems become more and more complicated, this kind of the discontinuous function is no more suitable to be applied. Suppose we have a neuron network that learns to recognize a character from an image. This neuron network faces a sequence of the pixels that include the features of this character. We need to train it over and over again to reduce the difference between the real learning outcome and the expected one by adjusting the internal structure which is the weight. Usually, this adjustment is very small, and the result is composed of even thousands of these small changes. In this case, if we use the discontinuous function in each neuron, like perceptron, sometimes even a small change in the weight may then cause the output of this neuron to completely flip (e.g. from zero to one), and this filp might cause the result of the rest of the network to completely change into more complicated condition. So we introduce the continuous activation function used in our study. Figure 2.2.3 shows the plot and formula of the Hyperbolic Tangent. It's obvious that the output of tanh is not just zero and one, but is a smooth result range from -1 to 1. It's so useful in the study, and we could explain it in a simple example. A neuron wants to recognize the character "A". If it's a perceptron, then there are only two results, which are yes or no. However, a tanh neuron could give a certain result, which is that we think it is character A with 80% of possibility but it might be not A with 20% possibility. So we could set a threshold that if the possibility is over 75%, then the tanh neuron would make a positive decision. [8]

Feed-forward network "A feedforward neural network is an ANN, in which the connections between the neurons do not form a loop", i.e. acylic[9]. In this neural network, all the transmission is one direction. In our study, we use three-layer neural networks which means the output neurons only receive signals from the neurons in hidden layer, and the hidden layer neurons only receive signals from the neurons in input layer. In Section 2.4, we talk about the back propagation method which

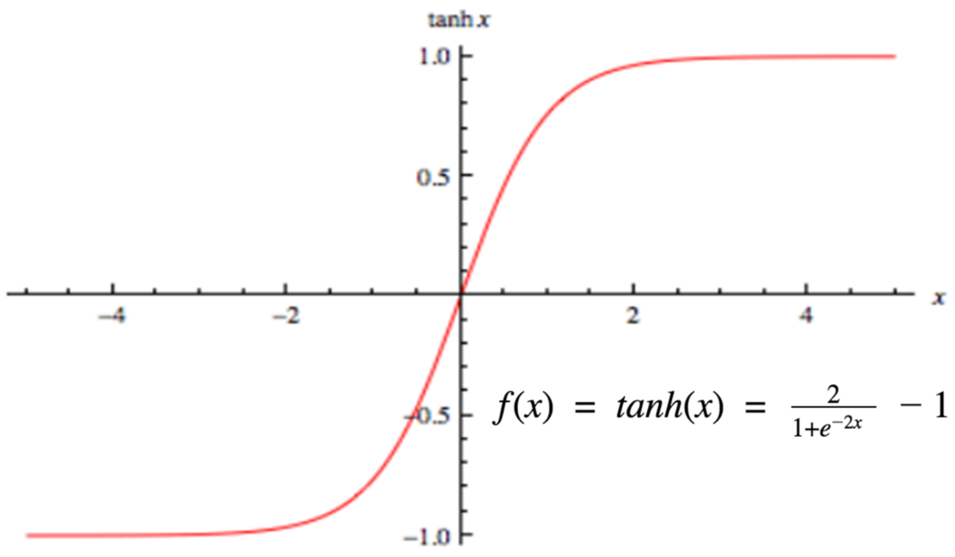


Figure 2.2.3: The activation function (Hyperbolic Tangent)

calculates the error contribution for each neuron after the data is processed and then adjusts the corresponding weights in order to reduce the error contribution between the real output and the expected one. It's a method for training the artificial neural network rather than the direction of information propagation.

2.2.2 Overfitting

In this part, we use an example to explain the typical problem of the neural network. The first time when we trained the neural network of the Greek symbols we met the problem that it can not stop until the maximum time is reached. we tried to use many approaches to test it over and over again, until we found that we met the overfitting. Overfitting is a main problem for predictive analytics. Especially in the artificial neural network, there are complicated models. Within a number of images, the neural network needs to analyze all the features and get the ability to predict based on their previous training. If the number of the image is not enough or their features are completely different, the neural network would be confused,

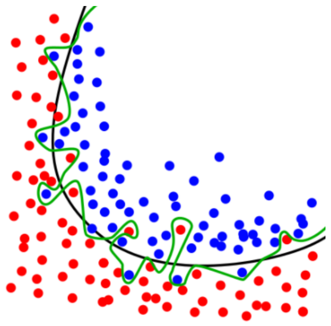


Figure 2.2.4: An example of overfitting

and keep training for getting a better result. That's why in neural network field, the overfitting is also known as the overtraining. The result for the overfitting, in most cases, leads to the worse result.

In the following, there is a typical solution to solve the overfitting by using the training, testing and validation sets for building a neural network. The training set is for training the neural network with the backpropagation method for decreasing the error distributions for a certain number of the inputs. The testing set is for testing the previous trained neural network based on its structure and the weights, in order to test the recognition ability. As for the validation set, which could be regarded as a part of the training set, it's used for early stop the training procedure even without reaching the expected result. It could avoid the overfitting by detecting whether the accuracy of recognizing its dataset keeps improving. Once the accuracy gets worse, the validation set will force the training to stop.

2.3 Supervised Learning

In the machine learning algorithm, there are three main learning paradigms based on their different functions. These are supervised learning based on the labeled dataset, unsupervised learning based on the descriptive model, and reinforcement learning. The sketch for each paradigm is shown in Figure 2.3.1.

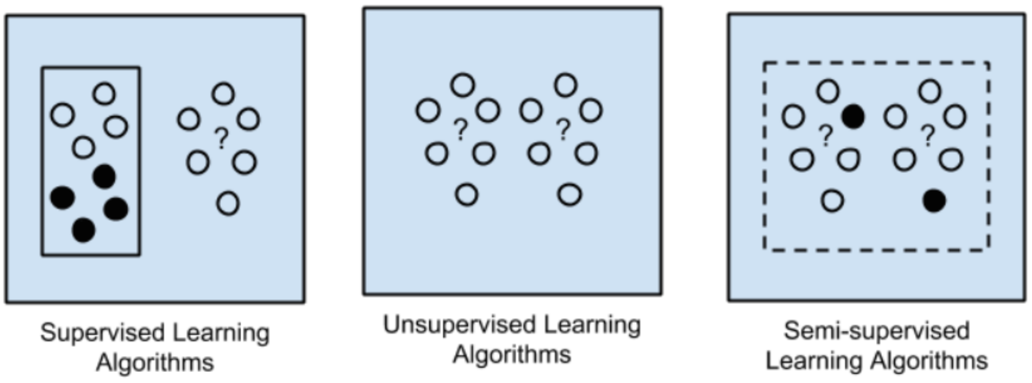

Supervised Learning Algorithms
Unsupervised Learning Algorithms
Semi-supervised Learning Algorithms

Figure 2.3.1: Machine Learning Algorithm

In our study, all the training procedures of the artificial neural network are based on the supervised learning, "which is the machine learning task that inferring functions from labeled training data" [10]. The labeled data comes from the solution-equipped data set. For example, there is an image file including the printed format of Latin character a, and this file is named with "9pt_Arial_a.png". In this case, we could digitize all pixels of the image to a vector as the input object, and then classify the filename which includes the name of this Latin character as the expected output object. Overall, we get a pair of the data for training or testing our neural networks.

The supervised learning is also the reason why we finally decided to build our dataset. We train the neural networks to recognize Latin characters, Greek characters, and mathematical symbols, which requires the huge and various datasets to support the learning procedure. In this case, it's also required to generate the dataset automatically by image preprocessing and filename recognition. If the datasets come from different resources, it's so hard to manage all of them with a piece of code.

For supervised learning, the training process keeps the neural network adjusting its internal representation (weight) in order to get the expected result. The difference between the real result and the expected result is calculated as the error of training, and then the neural network uses the back-propagation method in order to reduce this error. Finally, when the neuron network can get the expected result based on its internal structure, the training successes. The back-propagation method mentioned above is explained in Section 2.5.

After training step, we still need the testing dataset to check the learning ability of the neural network. In the training step, the label of the dataset is used for conducting the neural network to study. However, in the test step, the labeled dataset is used for testing whether the trained neural network could get the corresponding expected output from any arbitrary input.

2.4 Backpropagation Method

"Backpropagation, also called backward propagation of errors, is an approach commonly used in ANN to calculate the error contribution of each neuron after a number of data is processed"[11]. There is a famous paper written by David Rumelhart, Geoffrey Hinton, and Ronald Williams[12]. This paper describes that compared with other approaches the backpropagation is much faster, which was able to solve some previously insoluble problems at that time. Because of this paper, the backpropagation method began to be appreciated.

Let's consider a feedforward neural network with M layers[13].

The input of i th neuron in layer $k{+}1$ is as Formula 2.1. It shows that the input of the $k{+}1$ layer is depend only from the outputs of the neurons in k layer, and the weight between the *i* neuron from *k+1* layer and the neuron *j* neuron from *k* layer

$$n^{k+1}(i) = \sum_{j=1}^{Sk} w^{k+1}(i,j)a^{k}(j) \tag{2.1}$$

The output of this neuron is shown in Formula 2.2, which is applied the input of neuron with an activation function *f*. Usually this function could be sigmoidal or hyperbolic tan. In this thesis, we use the hyperbolic tan as it range from [-1, 1], without the bias.

$$a^{k+1}(i) = f^{k+1}\left(n^{k+1}(i)\right). \tag{2.2}$$

For an M layer network, the system equations in matrix form are

$$\begin{gathered} \underline{a}^{0} = \underline{p} \\ \underline{a}^{k+1} = \underline{f}^{k+1}\left(W^{k+1}\underline{a}^{k} + \underline{b}^{k+1}\right), \\ k = 0, 1, \cdots, M-1. \end{gathered} \tag{2.3}$$

The task of the neural network is trained to learn the mapping between inputs and their corresponding output by back propagation method which adjusts the internal weights. How the backpropagation success by each time adjusting the weights slightly? Let's us consider in the following way[8]. There is a small change $\triangle w^{l}_{jk}$ to the weight w^{l}_{jk} (Figure 2.4.1 (a)). This change will result in a change in the output of the corresponding neuron $\triangle a^{l}_{j}$ (Figure 2.4.1 (b)). It leads to a change in the inputs of neurons in the next layer, and later the outputs of these neurons will also change (Figure 2.4.1 (c)). Finally the cost function changes $\triangle C$ (Figure 2.4.1 (d)).

The change of the cost is related to the change of that weight, which is shown as follows.

$$\Delta C \approx \frac{\partial C}{\partial w^{l}_{jk}} \Delta w^{l}_{jk}. \tag{2.4}$$

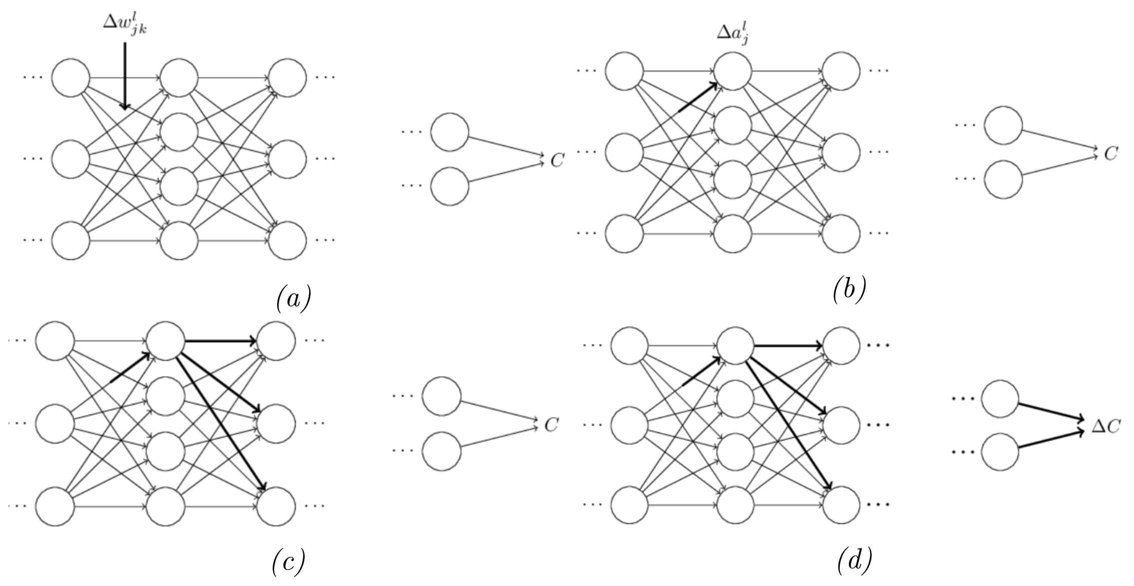
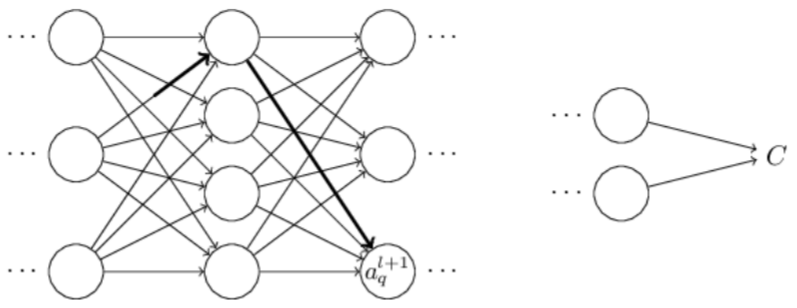


Figure 2.4.1: Backpropagation Method (part.1)

The change of the weight Δw^l_{jk} also leads to the change of corresponding activation function, which is from the j th neuron in the i th layer. This change will lead to the change of activation functions in all the neurons of *i+1* th layer. As the Figure 2.4.2 shows.



$$\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}. \tag{2.5}$$

Figure 2.4.2: Backpropagation Method (part.2)

$$\Delta a^{l+1}_q \approx \frac{\partial a^{l+1}_q}{\partial a^l_j} \Delta a^l_j. \tag{2.6}$$

Substitute the Formula 2.6 with Formula 2.5, we get:

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l. \tag{2.7}$$

Moreover, this change will lead to the change of activation functions in the next layer. We get the general sketch that the change of the very initial weight will cause the change of the final cost.

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l, \tag{2.8}$$

In order to compute the final change of the cost, we need to consider all the path available in this network, which is shown in formula 2.9.

$$\Delta C \approx \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l, \tag{2.9}$$

Substituted with Formula 2.4, we get:

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}. \tag{2.10}$$

Above we compute the derivative of c with respect to the weight. As for the every single the change is shown in Figure 2.4.3.

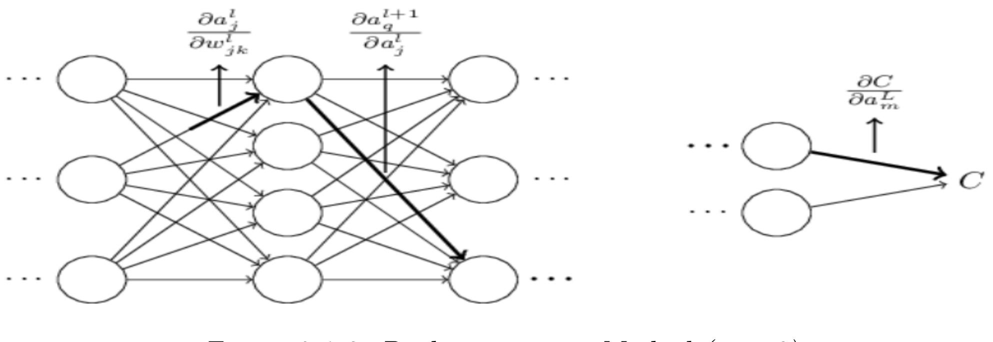

$\frac{\partial a_j^l}{\partial w_{jk}^l}$
$\frac{\partial a_q^{l+1}}{\partial a_j^l}$
$\frac{\partial C}{\partial a_m^L}$
C

Figure 2.4.3: Backpropagation Method (part.3)

2.4.1 Optimize the Back propagation

There are some methods of optimizing the backpropagation method for accelerating the learning. In the following, there are two methods used in our thesis for improving the performance of backpropagation algorithm.

Initialization of weights Initialization of the weights plays an important role in the performance of the feed-forward neural network[14]. There are many methods of the weights initialization which aims at reducing the influence to the performance of the neural network. In this study, we use the random initialization due to its simplicity, and set the different range of initial weights. By the repetition test, we aim at finding the optimal range of the initial value for improving the performance of the neural network.

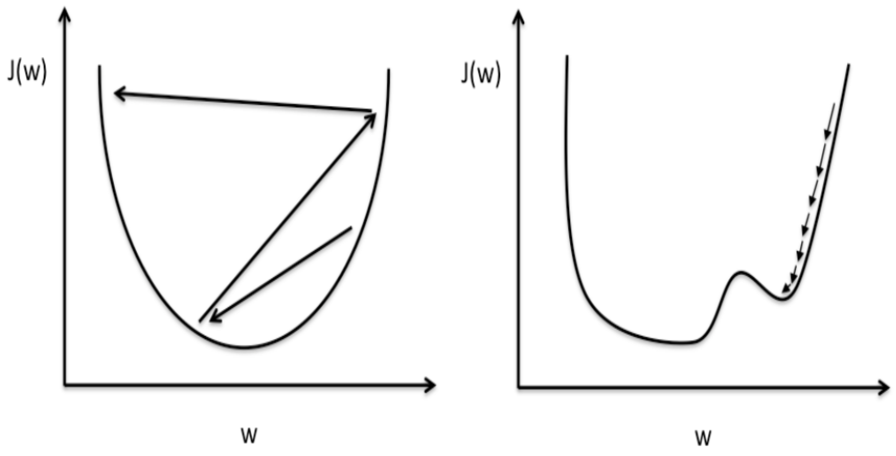


Figure 2.4.4: The learning rate of backpropagation

Learning rate Using the adaptive learning rate by modifying learning rate (eta). With the standard gradient descent, the learning rate is a fixed constant in the whole training process. It's an essential parameter for the performance of the neural network (i.e. convergent rate). If it's too small, the algorithm will take a long time to converge. However, if it's too large, the algorithm is very unstable[15]. The Figure 2.4.4 shows these relationships in general.

Chapter 3

Implementation

3.1 Parallel neural networks

Based on the previous introduction, we have a general idea of how to train single neural network with the backpropagation method for improving its performance.

In our study, we aim at pattern recognition for the Latin characters, Greek characters, and mathematical symbols. However, it's not practical to put all of these objects into one neural network. Since for a neural network, a large number of different patterns is hard to achieve the fast convergence[16]. In this case, we decide to separate all these characters into several groups and each of them is recognized by a neural network. With the trained neural networks, we propose the idea of making them working in parallel to improve the performance of the overall system. The parallel neural networks are based on an amount of separately-trained neural networks. Each neural network is required to recognize the same image, and produce its own output based on its structure and parameters. The system of the parallel neural network will select one of the outputs as the most proper result of the voting algorithm then produce. This idea proposed aims at improving the average performance of the neural networks.

3.2 Ensemble Methods

Ensemble method is a meta algorithm which uses other machine learning algorithms as a component to learn a collection of predictors. It's based on the attitude that many predictors always preform better than one. One view of ensemble learning is related to a psychological phenomenon which is called the wisdom of the crowds. That is a bunch of algorithms may work better than s single one.

Predictors of the same or different types are combined usually by the un-weighted average or weight average. These predictors could be neural networks, support vector machines, or decision trees. But in this part, we only focus on the same type of learners, and introduce some typical examples like boosting, which uses only single kind of the learner but many different instances to perform a better job.

Before introducing the ensemble methods, there are two concepts, bias and variance. Bias is the error from the model choice. "If the value of bias is high, which means the model that keeps missing valuable trends." [17]. Variance is the randomness from the dataset. "If in a model, the variance is high, this model will overfit and perform badly on the observation beyond training". As shown in Figure 3.2.1, "the increase of the complexity of the model could reduce the error due to the low bias. However, from a certain point, the high variance would have an influence on the high-complexity of the model"[17].

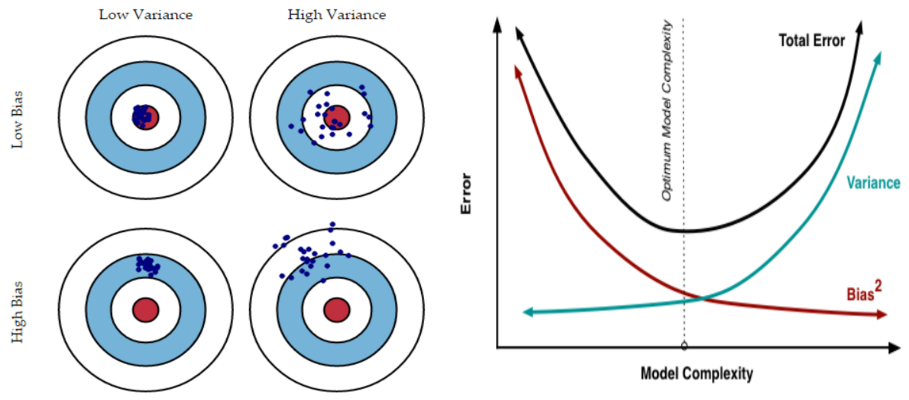


Figure 3.2.1: The bias and variance

In the following paragraphs, there are two major ensemble methods.

Bagging It's the Bootstrap AGGregation, which learn from many classifiers, each of which only with a portion of the data, and then combine these classifiers with model averaging. The general pipeline is shown in Figure 3.2.2. The basic idea behind this method is to avoid the overfitting by using the cross-validation which makes many splits of the dataset for training and immediately testing. The cross-validation could detect the overfitting efficient[18]. Bootstrap is used as the subroutine of its learner. It creates K learners independently, with the previous training set. Each learner generates its own training data set from the original full training set by replacement, then train a classifier on those data set. Train these learners in parallel and test on all K (the number of the learners) of them. Finally, do the combination of all the predictors. For the classifiers, there is the majority vote, and for the regression, the weight or un-weighted average is calculated. This method aims at reducing the complexity and avoid overfitting which dues to the un-memorized of dataset.

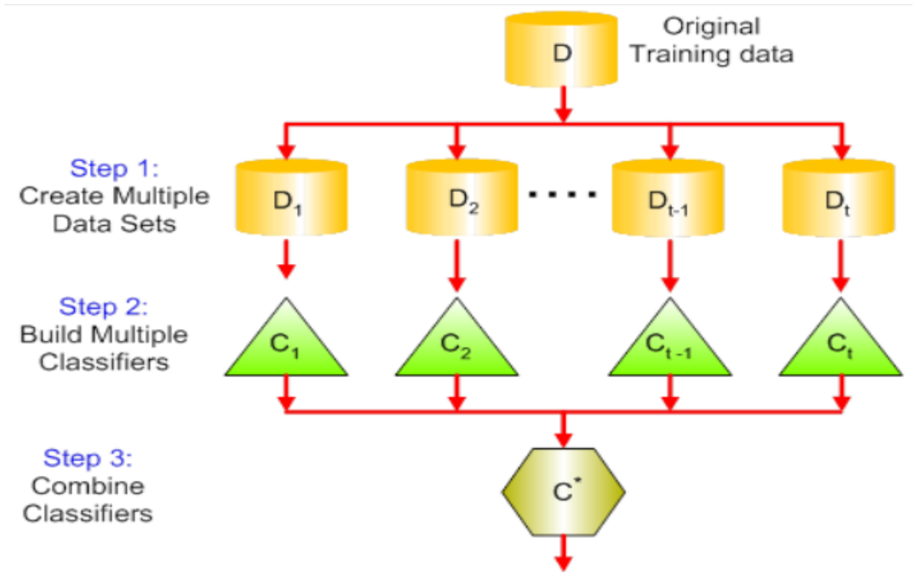

D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Figure 3.2.2: The Bootstrap AGGregation (Bagging)

Boosting Boosting a technique for producing regression models consisting of a collection of regressors. Learner learns sequentially with early learners. Errors from the earlier predictors are marked in "difficult" examples draw the later predictions' attention. This method successfully converts many "weaker learners" into one "strong learner".

Gradient Boosting is one of the methods of boosting. It starts with a simple regression model, and each learner predicts the error subsequently based on the previous prediction. Overall the final prediction is given by the weighted sum from each model. The sketch is shown in Figure 3.2.3. Each of the prediction is based on the previous prediction and its error residual and tends to predicts in a better way.

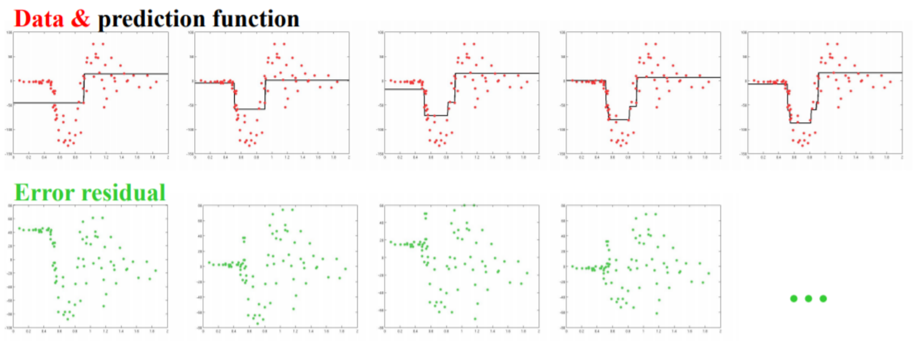


Figure 3.2.3: The Gradient Boosting

Let's consider as an example the simple classifier shown in Figure 3.2.4. The original dataset D1 is split into 2 parts with most of the decisions correct. But there are still three symbols (two negative and one positive) which are predicted wrongly. So in the next step, the weights for wrong objects, which are got from the previous step, are emphasized. In this step, the new classifier classifies most of the symbols correctly, especially the high-weighted ones. However, it misses the low-weighted symbols and classifies them wrongly. So in the following step, the weights are calculated again based on the correctness in the current situation. After training each classifier, the combination with weights is obtained. These weights for each classifier are based on the training precision.

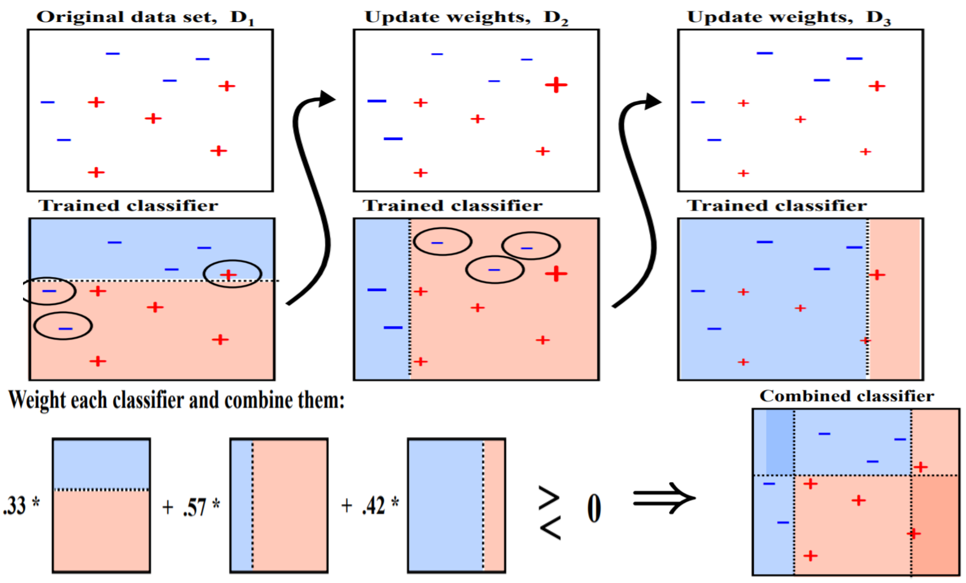


Figure 3.2.4: The Gradient Boosting 2

AdaBoost is Adaptive Boosting, which aims at minimizing the surrogate loss (i.e exponential loss) as shown in Formula 3.1.

$$C_{ada} = \sum_i \exp[-y^{(i)} f(x^i)] \tag{3.1}$$

A typical application of AdaBoost is the face detection algorithm, which would combine many weak classifiers and define many features.

Chapter 4

Results and Performance Evaluation

4.1 Generate Image Files

The first step for the experimental validation is the preparation of the raw image files. Our study focuses on recognizing the scientific text which are the Greek characters and mathematical symbols, and also recognizing the normal characters, i.e 26 Latin characters. In this case, the database including the images of these characters and symbols is required. On the one hand, even if there are already many public databases of mathematical symbols, which are freely usable for research activities, but it's impossible to find one which includes all the symbols we need. On the other hand, the images from the different database might not be in the same format.So we decided to create our own image data.

We aim at training six neural networks encompassing mathematical symbols, Greek characters and Latin characters with the font size from 9 pt to 12 pt. In general, we choose 26 Latin lower-case characters, 24 Greek characters and 44 frequently-used characters as the training goal. For the Latin characters, there are

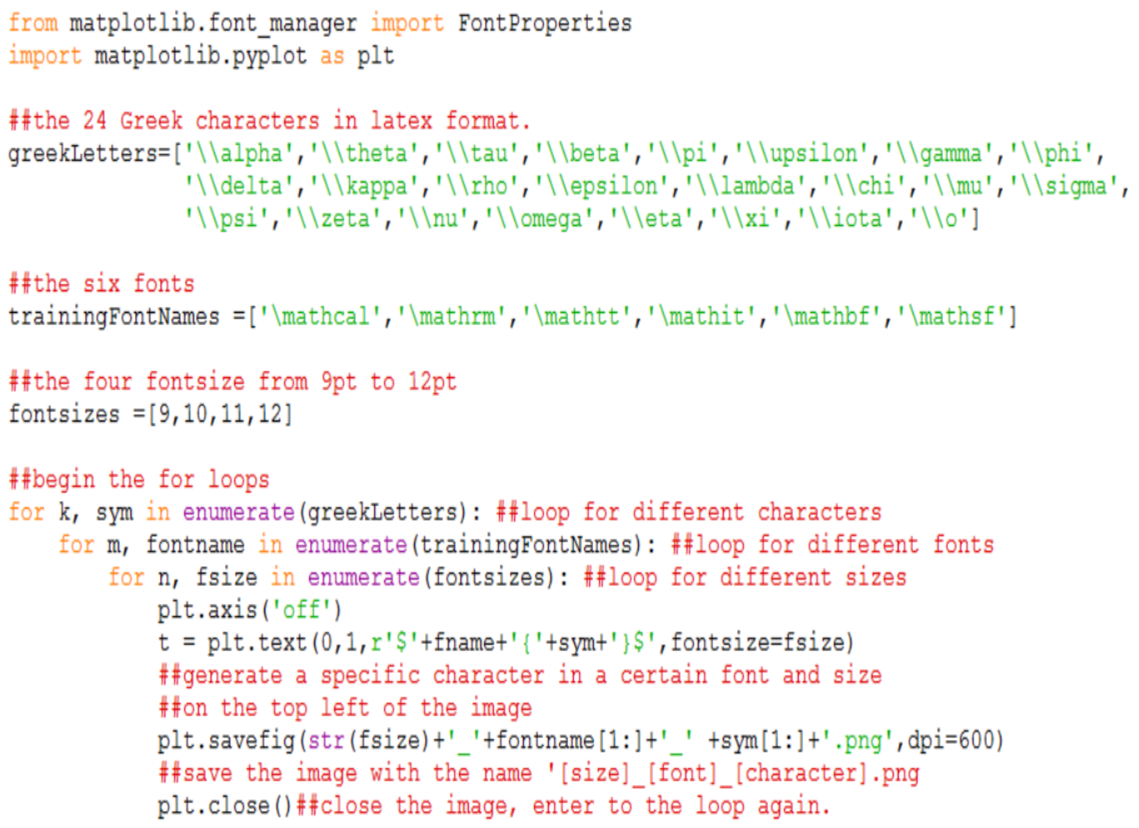
115 different fonts and for the Greek characters and mathematical symbols there are 6 fonts The details for each type of characters is in the Table 4.1.1.

Type	Elements	Training Fonts	Tot. images	Testing Fonts	Tot. images
Latin	26	115	11960	6	624
Greek	24	6	576	3	288
Math.	44	6	1056	3	528

Table 4.1.1: Image assignment for training and testing each type of text

Since the image processing is a long, tiring and error-prone procedure. It's unfeasible to generate images manually. So we use Matplotlib which is a Python 2D plotting library. It produces publicatable-quality of the figures in different formats and interactive environments across platform[19]. We generate both the image files to be used for the neural networks and the previews of the characters in all available fonts for better introducing them

Figure 4.1.1 illustrates the code developed to generate the 24 Greek characters with 4 sizes and 6 fonts automatically. The generation of each character is based on its corresponding latex code. And the result of this part of the code is the image files used for training the neural networks. As the above code shows that all the images are generated one by one automatically. Also every time an image is generated, it is named after the symbol's name, font, and size (E.g., 9pt_mathit_subset.png), and these specific file names help us in the next step to produce the labeled data set. In each image, there is a single character located in the top left, size is among 9, 10, 11 and 12 pt. The character is black in the white background to present all the symbols clearly, the previews of the characters for training the artificial neural networks are shown in the Figure 4.1.2- 4.1.4. In these figures, there are many different fonts for the Latin characters, with various shapes and real sizes. For the Latin characters, there are the Arial, Calibri and etc, which are the most common fonts. And for the Greek and mathematical characters, there are 6 fonts which are specific from the math font family. That's the main feature of the artificial neural network or even



```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

##the 24 Greek characters in latex format.
greekLetters=['\\alpha','\\theta','\\tau','\\beta','\\pi','\\upsilon','\\gamma','\\phi',
              '\\delta','\\kappa','\\rho','\\epsilon','\\lambda','\\chi','\\mu','\\sigma',
              '\\psi','\\zeta','\\nu','\\omega','\\eta','\\xi','\\iota','\\o']

##the six fonts
trainingFontNames =['\mathcal','\mathrm','\mathtt','\mathit','\mathbf','\mathsf']

##the four fontsize from 9pt to 12pt
fontsizes =[9,10,11,12]

##begin the for loops
for k, sym in enumerate(greekLetters): ##loop for different characters
    for m, fontname in enumerate(trainingFontNames): ##loop for different fonts
        for n, fsize in enumerate(fontsizes): ##loop for different sizes
            plt.axis('off')
            t = plt.text(0,1,r'$'+fname+'{'+sym+'}$',fontsize=fsize)
            ##generate a specific character in a certain font and size
            ##on the top left of the image
            plt.savefig(str(fsize)+'_'+fontname[1:]+'_' +sym[1:]+'.png',dpi=600)
            ##save the image with the name '[size]_[font]_[character].png
            plt.close()##close the image, enter to the loop again.
```

Figure 4.1.1: The code for the generation of images with Greek characters

the machine learning, which is required a large amount of data set for training and then get the ability to recognize the character even with some noises.

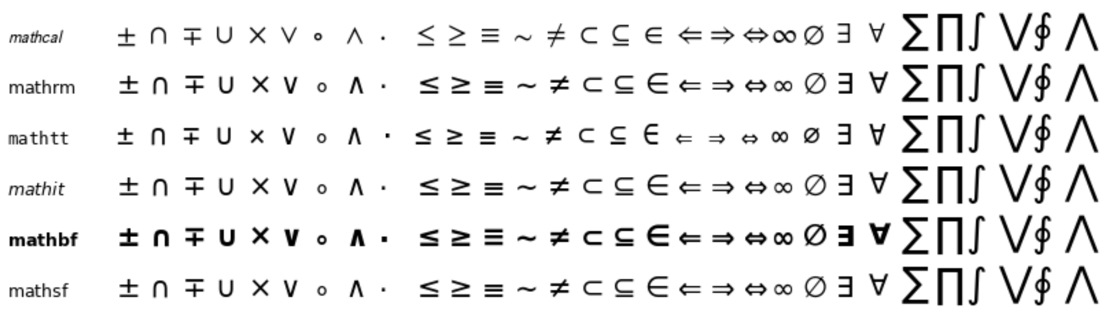


Figure 4.1.2: The preview of training set for mathematical symbol

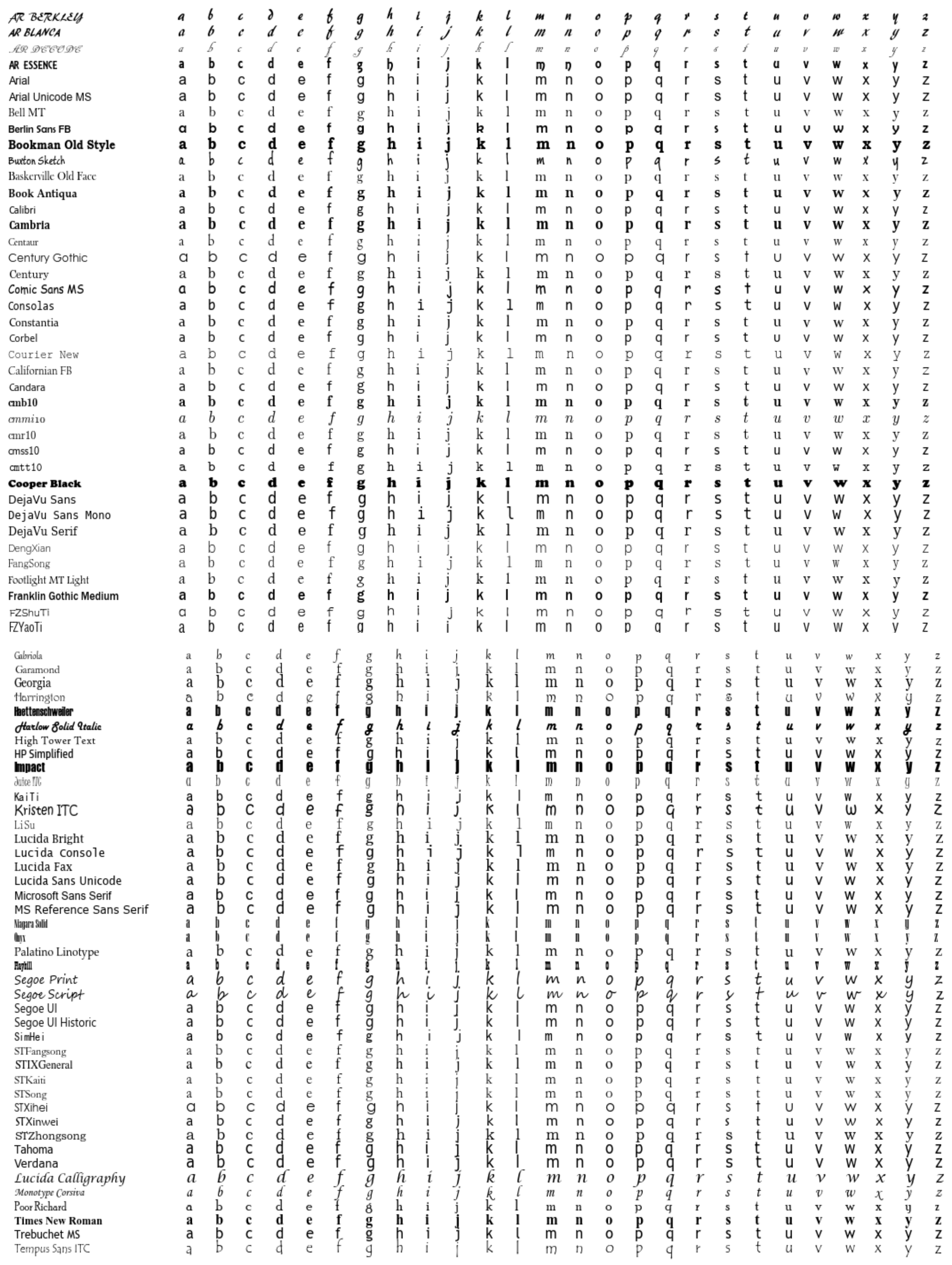


Figure 4.1.3: The preview of training set for Latin characters

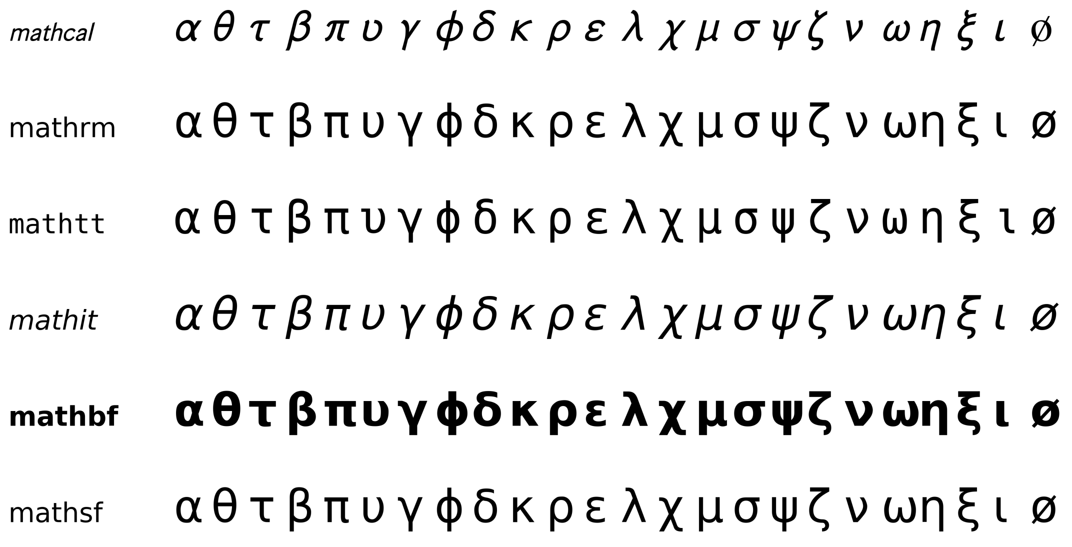

mathcal α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø
mathrm α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø
mathtt α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø
mathit α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø
mathbf α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø
mathsf α θ τ β π υ γ ϕ δ κ ρ ε λ χ μ σ ψ ζ ν ω η ξ ι ø

Figure 4.1.4: The preview of training set for Greek symbol

After the generation of image files, we assign them into different neural networks. The detailed assignment is in the following Table 4.1.2. For example, there are 5980 images for training and 312 images for testing in nn1, which is in charge of the Latin characters with size 9pt and 10pt. As shown in the column Output Neurons, for nn1, there are 26, which means this neural network can recognize 26 patterns, i.e., 26 Latin characters.

Neural Network	Type	Font Size (pt)	Training Set	Testing Set	Output Neurons
nn1	Latin	9-10	5980	312	26
nn2	Latin	11-12	5980	312	26
nn3	Greek	9-10	288	144	24
nn4	Greek	11-12	288	144	24
nn5	Math.	9-10	528	264	44
nn6	Math.	11-12	582	264	44

Table 4.1.2: Data set assignment for each neural network

4.2 Image Preprocessing

All the images need to undergo the preprocessing and features extraction before being analyzed by the ANN. In this part, all the image contents are converted into vectors[20], and the corresponding file names are decoded also into vectors.

4.2.1 Image content analysis

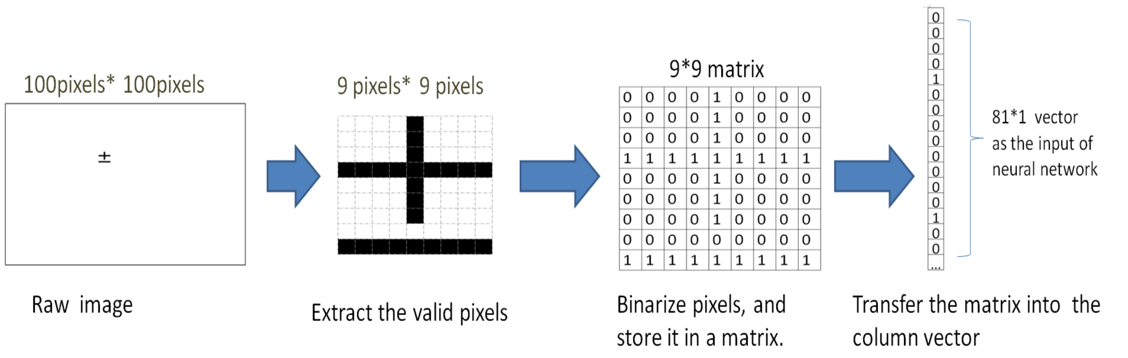

100pixels* 100pixels
±
Raw image
9 pixels* 9 pixels
Extract the valid pixels
9*9 matrix
Binarize pixels, and store it in a matrix.
81*1 vector as the input of neural network
Transfer the matrix into the column vector

Figure 4.2.1: A simple example of image preprocessing

The draft of the image preprocessing is shown in the Figure 4.2.1 . We implement them with Matlab program. In the following, we discuss each of them in details.

Binarization Remove the irrelevant color elements based on a threshold in order. It's implemented in Matlab by the Function im2bw() which converts the grayscale image to binary image. As a result, all the color pixels are converted into the black (one) or white (zero) pixel, which reduces the subsequent analysis complexity, and for each image file there is a corresponding matrix, so that one element in the matrix represents one pixel in the image.

Normalization In the binarization, from each image, we obtain a corresponding binary matrix which presents all the pixel information. However, this matrix might include some invalid element (e.g. padding of the image). So we want to minimize the size of the matrix without losing any valid information loss. In this step, for each image, its matrix is to be re-sized by nr and nc, which are the number of pixels in the row and column, respectively, and fixed for a neural network (the reason will be explained in the next step about linearization.). Starting from the right and the bottom, the padding is removed until the first row or column where non-empty is found. The re-size of matrices can reduce the computational complexity later. Finally, we find the minimum values of nr and nc for each neural network, which can cover all the valid information of the character. Also, the product of nr and nc is equal to the number of the neurons in the input layer for each neural network, as shown in Table 4.2.1.

Neural Network	Type	Font Size (pt)	nr * nc
nn1	Latin	9-10	90 * 100
nn2	Latin	11-12	110 * 120
nn3	Greek	9-10	80 * 90
nn4	Greek	11-12	90 * 100
nn5	Math.	9-10	110 * 115
nn6	Math.	11-12	135*140

Table 4.2.1: The the number of neurons in the input layer of each neural networks after image pre-processing

Linearization In the normalization, a re-sized matrix is obtained for each image file. The size nr*nc is also the size of input layer in a neural network, as one neuron in the input layer is in charge of one pixel of the image to manage a group of the images for training in a neural network, it's necessary to transfer a matrix for an image into a column vector, which is the procedure of linearization. Then we collect all the column vectors and present them as a collection of all the images by a matrix which is the input matrix used for the neural network.

4.2.2 Image naming convention

During the generation of image file, we make the image file name with the format of "[fontsize]_[fontname]_[character].png". While analyzing the content of the image to generate input vector of a neural network, we could also get the expected output vector from the file name. For example, there is an image file with name 9*pt_mathit_a.png*. Character "a" is the first element among 26 elements which could be analyzed by the Latin neural networks NN1 and NN2 as we assigned, and size 9pt belongs to the NN1. So the expected output is generated by a 26-length column vector with the first element in one and the rest of 25 elements in zero. Finally, the collection of these expected output vector is managed together as an output matrix used for the neural network. The combination of output matrix with its corresponding input matrix becomes the labeled data set used by supervised learning in training the neural network.

4.3 Training the Neural Network

After the preparation of labeled datasets (input matrices and expected output matrices), we begin to train the six neural networks separately by means of BP algorithm with the corresponding training dataset.

In this part, the neural networks are trained separately with different structures and parameters to find their best performances, which are assessed by the elapsed time and the steps needed for training. After training, each neural network should be able to recognize all the images in its training set with 100% accuracy.

The implementation in Matlab is shown as follows:

$$\begin{aligned}&[weights\_12, weights\_23, nStep, elapsedTime] =\\ &\quad train\_neural\_network(starting\_weights\_12, starting\_weights\_23,\\ &\qquad trainingSet, trainingOutput, eta, maxStep);\end{aligned}$$

starting_weights_12, starting_weights_23 are initial weights between the first and second layer and between the second and third layer for a neural network. The size of weight matrix depends on the number of neurons in each layer. The number of neurons in the input and output layer are fixed by the valid dimension of each image and the number of symbols to be recognized by this neural network, respectively. However, the number of neurons (i.e., *N_2*) in the hidden layer need to be explored. For the initialization, we use the random generator (i.e., Matlab function *rand()*) to generate the weight for each connection with the range of [-kk/2,kk/2]. *kk* is predefined and usually below 1. *trainingSet* is the collection of the input vectors generated from images and *trainingOutput* is the collection of expected output vectors generated from the image file names. *eta* is the learning rate of the back propagation method. *maxStep* is the maximum steps allowed. If the neural network after maxStep still can not reach the expected value, an error message is prompted.

In the training step, our aim is to find the optimal solution by adjusting the values of kk, eta, and N_2, which is related to the performance of back propagation method and structure of the neural network. We firstly focus on the neural networks of the Greek characters and mathematical symbols.

We assess the behavior of training process by the elapsed time and the training steps and the percentage of the errors from testing the training dataset by the trained neural network, which should be zero.

Table 4.3.1 to Table 4.3.8 report the experimental results on NN3, NN4, NN5 and NN6 to achieve the 100% accuracy. From Table 4.3.1 to Table 4.3.8, the average elapsed time and steps for training NN3, NN4, NN5 and NN6 are shown respectively with different values of N_2, kk and eta in 30 repetition tests. The shortest elapsed time and the fewest steps in each sub-table are marked in red.

N_2=100						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	2.41	3.76	7.58	25.38	28.24
	0.40	2.00	1.93	2.04	2.68	4.19
	0.50	3.19	2.13	1.79	1.99	2.51
	0.60	5.24	3.00	2.39	2.10	2.28
	0.70	7.00	4.35	3.04	2.48	2.45
N_2=150						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	2.86	4.57	6.81	12.73	20.35
	0.40	3.40	2.89	3.06	3.50	5.79
	0.50	5.37	3.74	3.28	3.55	4.10
	0.60	10.64	6.43	5.35	5.07	5.09
	0.70	24.89	11.08	6.27	5.39	5.09
N_2=200						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	2.80	4.37	6.96	13.93	20.87
	0.40	4.14	4.16	4.76	6.00	9.84
	0.50	6.63	5.29	5.56	5.74	5.89
	0.60	23.85	9.28	8.03	7.93	7.62
	0.70	56.41	17.85	10.70	10.83	10.65

Table 4.3.1: Average elapsed time by training NN3 with different values of N_2, kk and eta

N_2=100						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	36.27	52.47	110.87	367.47	440.47
	0.40	37.20	34.40	38.93	51.93	74.27
	0.50	58.33	40.53	35.13	38.13	45.73
	0.60	93.00	54.53	43.73	38.33	40.13
	0.70	130.00	86.73	59.87	49.13	46.27
N_2=150						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	34.60	49.73	72.20	133.07	234.93
	0.40	38.87	37.27	45.87	55.93	85.60
	0.50	75.00	51.20	46.80	49.13	55.67
	0.60	141.87	83.67	64.13	60.33	61.13
	0.70	304.00	140.80	93.40	80.53	80.27
N_2=200						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	41.35	55.75	80.25	140.65	209.40
	0.40	52.05	50.20	56.00	66.40	108.55
	0.50	88.60	68.40	65.35	68.15	74.15
	0.60	252.70	112.80	92.75	92.15	92.65
	0.70	672.20	215.80	133.30	129.15	127.50

Table 4.3.2: Average training steps of NN3 with different values of N_2, kk and eta

As we see in Table 4.3.1, with different values of each group of parameters, the elapsed times vary from [1.93, 28.24]. So to choose the optimal result among the red marked ones, we consider more about the number of neurons of the hidden layer, but relatively shorter elapsed time and fewer training steps.

N_2=100						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	2.63	4.06	7.68	18.12	43.63
	0.40	2.14	2.11	2.46	3.47	5.44
	0.50	3.48	2.36	2.00	2.22	2.96
	0.60	5.46	3.52	2.60	2.41	2.64
	0.70	7.19	4.65	3.35	3.03	2.99
N_2=150						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	3.56	5.42	9.38	15.52	46.98
	0.40	3.87	3.41	3.65	4.37	7.10
	0.50	6.03	4.29	3.87	4.31	5.08
	0.60	11.10	6.81	6.36	6.45	6.61
	0.70	25.63	11.46	7.49	6.89	6.42
N_2=200						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	3.10	5.02	8.45	18.68	33.29
	0.40	4.50	4.79	5.60	7.73	12.04
	0.50	8.32	6.49	6.73	7.02	7.81
	0.60	21.24	10.62	10.18	9.80	10.00
	0.70	46.28	18.68	13.72	13.41	13.97

Table 4.3.3: Average elapsed time by training NN4 with different values of N_2, kk and eta

N_2=100						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	35.53	51.47	100.07	239.47	631.87
	0.40	38.00	33.27	40.87	58.13	83.33
	0.50	56.67	39.60	33.80	38.20	48.87
	0.60	87.67	60.40	43.80	41.13	42.33
	0.70	123.20	83.47	59.27	53.73	50.60
N_2=150						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	37.93	53.07	89.27	149.67	491.40
	0.40	40.53	40.40	48.40	62.47	94.80
	0.50	73.33	51.87	51.93	51.60	62.60
	0.60	133.67	79.73	68.27	68.93	71.60
	0.70	281.73	133.27	99.13	93.20	92.87
N_2=200						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	42.05	58.55	92.15	177.05	314.70
	0.40	52.20	53.75	60.80	80.35	124.95
	0.50	99.90	75.70	73.80	77.65	89.05
	0.60	212.00	119.05	109.45	108.85	113.70
	0.70	509.85	213.70	158.15	149.95	156.00

Table 4.3.4: Average training steps of NN4 with different values of N_2, kk and eta

N_2=100						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.40	3.04	2.62	2.58	2.92	5.53
	0.50	5.52	3.41	2.63	2.31	2.18
	0.60	9.77	5.15	3.69	2.80	2.05
	0.70	12.88	9.67	5.44	3.51	3.08
N_2=150						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	2.24	2.15	5.13	5.47	10.48
	0.40	4.43	3.10	3.65	3.45	4.04
	0.50	12.56	7.43	4.52	3.68	3.37
	0.60	26.57	14.55	7.49	4.65	3.88
	0.70	42.20	19.96	10.89	7.72	5.28
N_2=200						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	3.18	5.79	6.83	12.81	18.78
	0.40	14.43	8.37	5.85	7.40	10.03
	0.50	36.96	19.14	11.54	7.67	8.58
	0.60	102.97	50.41	25.27	15.60	11.85
	0.70	193.59	77.47	23.90	15.65	11.87

Table 4.3.5: Average elapsed time by training NN5 with different values of N_2, kk and eta

N_2=100						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.40	75.87	51.13	47.00	52.47	94.93
	0.50	122.13	74.87	49.27	41.93	39.07
	0.60	194.20	128.80	73.80	54.20	45.40
	0.70	243.93	180.47	101.87	78.00	60.00
N_2=150						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	52.33	48.47	95.67	80.53	131.60
	0.40	77.60	52.40	49.07	44.80	52.00
	0.50	167.47	97.47	59.00	47.80	43.60
	0.60	366.33	218.67	112.20	69.27	57.67
	0.70	601.07	305.40	159.53	110.20	77.13
N_2=200						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	52.33	48.47	95.67	80.53	131.60
	0.40	77.60	52.40	49.07	44.80	52.00
	0.50	167.47	97.47	59.00	47.80	43.60
	0.60	366.33	218.67	112.20	69.27	57.67
	0.70	601.07	305.40	159.53	110.20	77.13

Table 4.3.6: Average training steps of NN5 with different values of N_2, kk and eta

N_2=100						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.40	3.37	2.87	2.90	3.26	5.19
	0.50	5.69	4.27	3.25	2.57	2.78
	0.60	11.41	6.07	4.41	3.32	2.35
	0.70	14.27	9.66	6.81	3.59	3.41
N_2=150						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	3.01	3.23	5.10	8.94	13.02
	0.40	6.48	4.08	3.87	4.33	5.64
	0.50	19.94	10.09	5.40	4.54	4.41
	0.60	36.66	12.60	9.26	5.97	5.21
	0.70	58.93	24.55	14.13	8.72	6.99
N_2=200						
Elapsed Time		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	3.64	6.97	9.40	16.71	25.28
	0.40	17.59	10.35	8.06	9.83	13.82
	0.50	106.78	19.20	14.28	10.77	13.46
	0.60	165.22	50.97	24.89	20.55	18.28
	0.70	321.15	69.72	25.59	14.58	16.82

Table 4.3.7: Average elapsed time by training NN6 with different values of N_2, kk and eta

N_2=100						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.40	65.67	46.60	44.20	49.60	76.33
	0.50	102.80	76.13	51.00	38.93	42.00
	0.60	186.87	121.87	73.53	53.60	42.33
	0.70	229.07	152.40	106.20	64.07	55.20
N_2=150						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	49.00	51.00	70.13	100.33	133.80
	0.40	87.07	52.80	41.53	45.27	58.80
	0.50	212.93	106.73	56.80	47.47	46.07
	0.60	398.73	149.33	109.60	70.33	61.20
	0.70	663.13	292.93	163.53	98.93	81.87
N_2=200						
Steps		eta				
		0.30	0.40	0.50	0.60	0.70
kk	0.30	45.05	44.80	60.05	85.25	128.35
	0.40	101.05	59.05	49.80	55.70	77.95
	0.50	613.95	116.70	76.75	68.35	78.10
	0.60	898.45	286.15	138.10	106.00	103.50
	0.70	1895.95	670.35	239.55	142.15	145.60

Table 4.3.8: Average training steps of NN6 with different values of N_2, kk and eta

With the evaluation of elapsed time and the number of training steps, we get the optimal structure and parameters for each neural networks with the requirement of 100% accuracy of recognition on its training data set. The details are shown in the Table 4.3.9.

neural network	N_2	KK	ETA
nn1	120	0.4	0.8
nn2	120	0.4	0.8
nn3	200	0.3	0.3
nn4	200	0.3	0.3
nn5	150	0.3	0.4
nn6	150	0.3	0.3

Table 4.3.9: The optimal parameters for accurate the neural network

4.4 Testing the Neural Network

This section deals with training the neural networks with the testing set, which are generated from the images the neural networks haven't study before. The images for testing are similar to the images for training before, but with some slight differences. For example, the Latin characters in Arial are used for training, and the Latin characters in Cambria could be used for testing. During the training procedure, a neural network learns from many enough images and has the ability to recognize characters according to their main features. In the testing procedure, the learning ability is to be assessed by images with the same main features but with some different features which are not trained before. After the testing and re-adjusting, the performance of each neural networks, which is assessed by the ability to recognize the strange but similar images, is improved. This step is implemented by the Matlab function shown as follows:

$$[testOutput, elapsedTime, pErr, nErr] = test\_neural\_network$$
$$(weights\_12, weights\_23, testSet, expectedOutput)$$

weights_12, weights_23 are the trained weights from the last step. *testSet* is similar with the one mentioned previously, which is the collection of input vectors generated from each image. *expectedOutput* is the collection of expected output vectors, which is generated from the image file name. *pErr* and *nErr* are the values present the accuracy of the trained neural network with the image which is not learned before. The test result for each trained table is show in Table 4.4.1, which might be optimized in the following.

As the results show that in the nn3, nn4, nn5 and nn6 the accuracy is 100%. However, in nn1 and nn2, which are in charge of the Latin characters, the percentage of error is much higher. So we adjust the structure and the parameters of these

Neural Network	Content	pErr(%)
nn1	Latin 9-10pt	26.5749
nn2	Latin 11-12pt	24.5963
nn3	Greek 9-10pt	0
nn4	Greek 11-12pt	0
nn5	math 9-10pt	0
nn6	math 11-12pt	0

Table 4.4.1: percentage of error (pErr) for each trained neural networks with the testing set

two neural networks, but this doesn't improve the performance. Another experiment has been carried with respect to the increase of the number of neurons in the hidden layer, which obviously improves their performances. The experimental result is shown in Table 4.4.2 and Figure 4.4.1, in which as the increase of neurons in the hidden layer, the percentage of the error in recognizing images from testing dataset rapidly decreases.

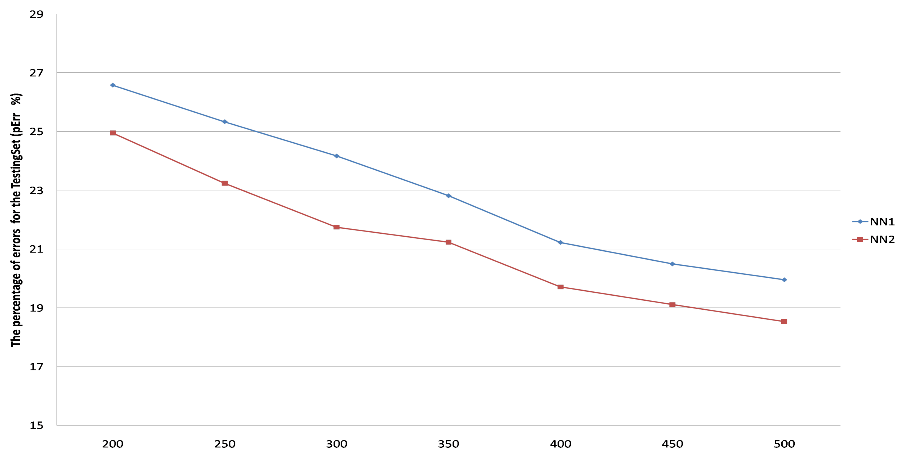


Figure 4.4.1: The plot of pErr of NN1 and NN2 trained by different N_2 values with kk=0.4 and eta=0.8

Of course, as the number of neurons in the hidden layer increases, the elapsed time is also longer. However, compared with the training elapsed time, we concentrate more on the accuracy, as the accuracy for the single neuron has the influence on the

Number of neurons in the hidden layer	pErr (%)	
	NN1	NN2
200	26.5769	24.9487
250	25.3333	23.2392
300	24.1667	21.7436
350	22.8141	21.2372
400	21.2244	19.7115
450	20.4936	19.109
500	19.955	18.5321

Table 4.4.2: pErr of NN1 and NN2 trained by different N_2 values with kk=0.4 and eta=0.8

accuracy of the parallel neural networks.

In a word, the optimal structure and parameters for Latin neural networks NN1 and NN2 are 500 neurons in the hidden layer, the range of random initial weight for training is [-0.1,0.1] and the learning rate is 0.8. Then the optimal percentage of error are 15% and 8% for the nn1 and nn2, respectively.

We also check the images which are wrongly recognized by NN2 and present them in Figure 4.4.2. with the pixel grid.

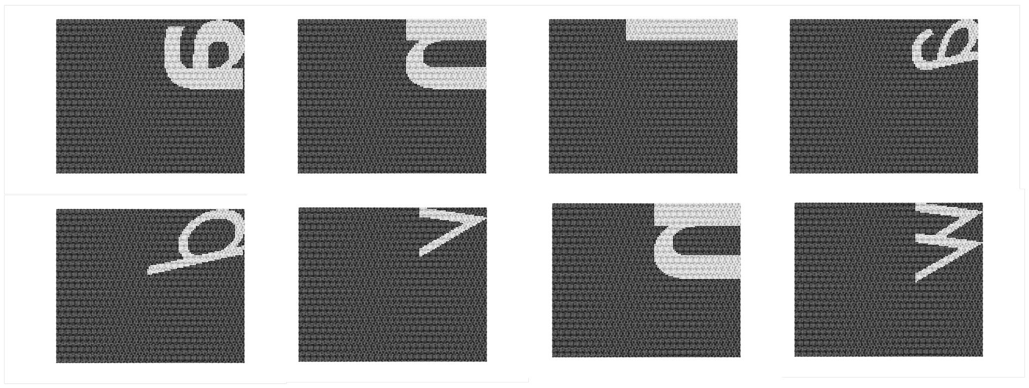


Figure 4.4.2: The testing images which are wrongly recognized by NN2.

Overall, after the testing trained neural networks with testing data set. The optimal accuracy and its relative structure and parameters are shown on the Table 4.4.3.

neural network	content	N_2	KK	ETA	pErr(%)
nn1	Latin 9-10pt	500	0.2	0.8	15.3467
nn2	Latin 11-12pt	500	0.2	0.8	8.5597
nn3	Greek 9-10pt	200	0.30	0.3	0
nn4	Greek 11-12pt	200	0.3	0.3	0
nn5	math 9-10pt	150	0.3	0.4	0
nn6	math 11-12pt	150	0.3	0.3	0

Table 4.4.3: percentage of error (pErr) for each trained neural networks with the testing set

Until now, we find the optimal parameters and structure for each neural network based on the performance of the testing data set.

4.5 Parallel Neural Networks: Voting Algorithm

In previous parts, we get the six independently-trained neural networks, and optimize their performances. Our aim is combining them together and build our final OCR. With the parallel neural networks, each neural network still works separately.

We focus on making the neural networks work in parallel and each of them produces its own output based on its structure and weights. There is the voting algorithm which selects the most proper output as the final outcome. That's the general idea of the parallel neural networks (PNN).

These results could be invalid (i.e., more than one *1* in the final output vector), valid but wrong (e.g., recognize Latin character 'o' as number zero), or valid and correct.

In the single neural network, the errors are only from wrong recognition of the patterns. However, In parallel neural networks, errors might derive from the misclassifications of the voting algorithm.

So in this part, the main difficulty is how to choose the only correct result as the final outcome of OCR among the valid ones.

4.5.1 Voting strategies

In Figure 4.5.1 and Figure 4.5.2, there are the brief pipelines about our idea of parallel neural networks.

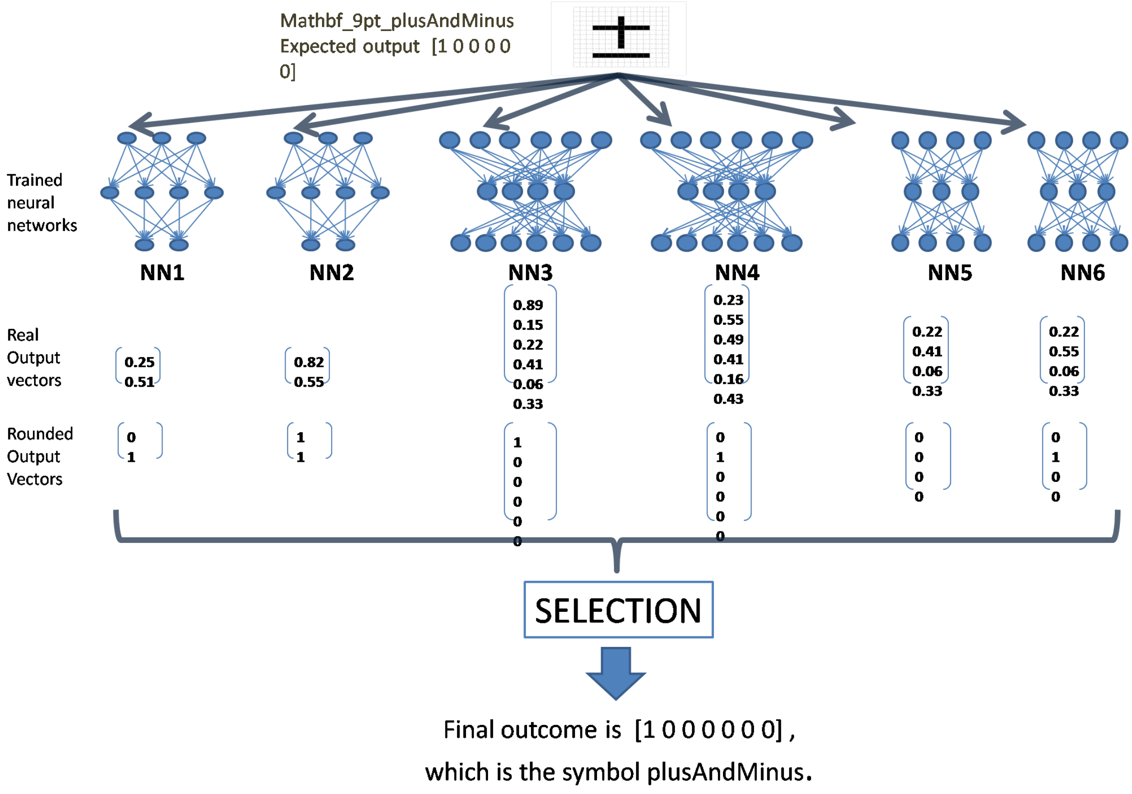


Figure 4.5.1: The parallel neural networks

Firstly, we set a group of image files as our testing dataset, which includes all the testing sets for six neural networks. It could check easily whether the accuracy of the parallel neural network is as good as the average accuracy of the six single neural networks. Secondly, we apply the testing images to the six trained neural

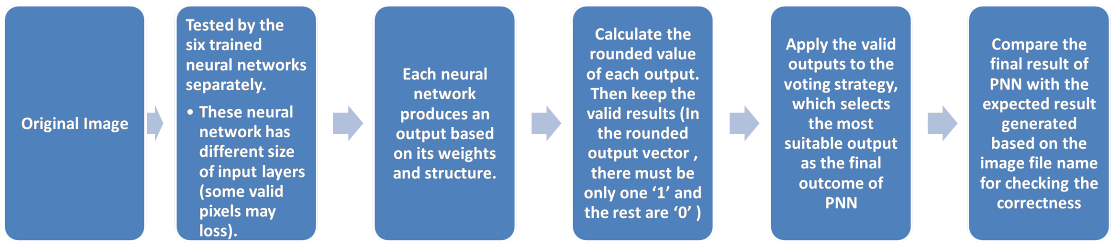

Original Image
Tested by the six trained neural networks separately.
• These neural network has different size of input layers (some valid pixels may loss).
Each neural network produces an output based on its weights and structure.
Calculate the rounded value of each output. Then keep the valid results (In the rounded output vector , there must be only one '1' and the rest are '0')
Apply the valid outputs to the voting strategy, which selects the most suitable output as the final outcome of PNN
Compare the final result of PNN with the expected result generated based on the image file name for checking the correctness

Figure 4.5.2: The pipeline of the parallel neural networks

networks to create six testing sets and one expected testing output. This means that if we apply a character in 12pt into the neural network which is specific for size 9pt. There is a large possibility that the pixels including the character feature will be lost, which means that the wrong neural network couldn't recognize it as one of its members as usually. The only expected testing output is used for the final step to check the correctness of the output of parallel neural networks, which is generated according to the name of the image file.

Select a number of the image files generated before as the recognition objects for the parallel neural networks. These image files could generate six different input matrices by applied with six neural networks and one expected output matrix by analyzing their file names.

The reason for the six different inputs matrices is that for each neural network, the number of neurons in the input layer depends on nr and nc, which are the minimum sizes to cover all feature pixels without any information loss. Each neural network analyses a range of the characters with specific sizes, so that the values of nc*nr are different. In this case, there is the possibility that a character in size 12 pt is applied to a neural network which is specific for the characters in 9 pt, and some of the feature pixels are lost as the normalization size for this neural network is not large enough to cover all the feature pixels of this image. In this case, there is no error message prompted.

As for the only expected output matrix, rather than a real matrix, it's better to describe it as a collection of column vectors with various lengths. For example, a

Latin character produces its expected output vector with length 26, but a Greek character produces one with length 24, as we analyze 26 Latin characters and 24 Greek characters. This expected output collection is used in the last step for checking the correctness of outcome of parallel neural networks.

After the generations of six input data sets, we apply them to the corresponding neural networks. Based on their structure and weights, there are six real output matrices generated, and then also the rounded output matrix is generated.

From the rounded outputs, we could do the first selection, which works by removing the invalid outputs. The invalid output is defined by the composition of the rounded output which should consist of only one "high" element which points to one of the patterns this neural network could recognize, and the rest are " low" elements which mean the other patterns. In a word, in this step, the neural network who doesn't generate a N-length vector with only one '1' element and N-1 '0' element is not considered anymore.

After removing all the invalid results, it's needed to select among the valid results based our strategies. We provide three methods available for the selection.

Method A Calculate the square root of the sum of the difference between each element in the rounded and real output vector, which is called Euclidean norm. Each valid output will produce a Euclidean norm, which is divided by the number of output neurons and then the smallest one is selected, which means that the neural network, who produces this smallest mean norm, is selected to be the most suitable neural network for recognizing this image. In order to optimize its procedure, the mean Euclidean distance is used for avoiding the effect of the numbers of output neurons in different neural networks. The detail is shown in Formula 3.2. For example, the neural network with 24 output neurons and 44 output neurons, even the former performs better than the latter, but its Euclidean norm may still larger, which may cause the error.

$$MeanEuclideanNorm = \frac{\|\mathbf{O}_{Round} - \mathbf{O}_{Real}\|}{n} \tag{4.1}$$

Method B Previously all the invalid outputs have been removed, and the rest of the outputs are all valid, which means in all output vectors there is only an element in the range of [0.51, 0.99] and the rest in the range [0, 0.49]. So we decide to choose the largest element in each valid output, which, in some ways, emphasis the most important value and remove other noise. Then compare all the largest values select the maximum among them. The output from the neural network who own this selected value is processed as the outcome of the parallel neural networks.

Method C Based on the Method A, add the previous precision on mean Euclidean distance. This method is inspired by the Boosting, which is an ensemble method used for constructing the weak classifiers. The minimum value of the weighted mean Euclidean distance is selected, and the output from the neural network who produce this min value is selected as the final outcome of parallel neural networks.

Finally, after the final outcome of the parallel neural networks is selected by three methods mentioned above, it is compared with the expected outcome generated before to check its correctness.

4.5.2 Experimental test

In this part, we focus on testing the parallel neural networks (PNN) by two main parts of data sets: one is training data set, in which previously we get the 100% accuracy in corresponding neural networks working separately as a result, and the other is testing data set, in which previously we get on average of 95% accuracy.

Testing PNN with the training set could check errors from the wrong decision of voting algorithm. And testing PNN with the testing set could evaluate the general performance of PNN (i.e., OCR)

Based on the Table 4.4.3. We could regard the NN3, NN4, NN5, and NN6 as the strong learners and NN1 and NN2 as the weaker learners. The strong learner has the high ability to recognize its corresponding objects and the weak learner has the relatively low ability to recognize its objects, which is classified based on their previous performance.

We test the PNN based on the previous trained neural networks (case 1), and then improve the strong learners and weaker learner respectively (case 2), Finally, combine the improvement of both strong and weak learners (case 3). In order to analyze the behaviors of three methods of voting strategies.

Each test works on three methods of voting strategy. For each strategy, the specific data sets (eg. Greek characters only) and the mixed data sets (i.e. a collection of Greek characters, mathematical symbols and Latin characters) are both applied in order to analyze the behaviors more clearly.

Case 1 Based on previous optimized-trained neural networks. The result is shown in the Table 4.5.1 and 4.5.2.

Training Data Set	pErr (%)		
	Method A	Method B	Method C
Greek (576)	17.01	11.11	12.67
math (1056)	8.24	13.92	3.41
Latin (1200)	10.17	14.08	17.83
Mixed (2832)	10.84	13.42	11.40

Table 4.5.1: Testing the parallel neural networks with previous training sets(v.1)

This case is based on the neural networks with 85% and 92% accuracies for NN1 and NN2, and 100% for NN3 to NN6. In the training set table, the accuracy should be 100%. Especially for the Greek data set, the percentage of the error is much higher than the mathematical data set. As for the testing set table, the accuracy

Testing Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (276)	5.43	5.43	4.35
math (528)	9.66	13.07	3.98
Latin (600)	13.17	18.5	20.83
Mixed (1404)	10.33	13.89	11.25

Table 4.5.2: Testing the parallel neural networks with testing sets(v.1)

should be about 95% on average, but actually, it's only in 10.33 with the best performance. However, compared with the result about training set table, it's not out of expectation. For the three strategies in this, method A performs well in recognizing the Latin characters and on the contrary method C performs better in the Greek and mathematical. But in general method A performs better.

Case 2 Based on the first case, we improve NN1 and NN2 by increasing the number of neurons in the hidden layer from 500 to 700. As a result, we get the new accuracy of NN1 and NN2 which are 91.1% and 90.7% respectively. The following Table 4.5.3 and 4.5.4 summarize experimental results.

Training Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (576)	21.88	11.81	15.63
math (1056)	10.23	13.92	4.36
Latin (1200)	5.5	11.83	7.58
Total	10.60	12.61	8.02

Table 4.5.3: Testing the parallel neural networks with previous training sets(v.2)

Testing Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (276)	17.39	8.7	5.43
math (528)	10.8	14.2	4.55
Latin (600)	10.17	15.67	16
Total(1404)	11.82	13.75	9.62

Table 4.5.4: Testing the parallel neural networks with testing sets(v.2)

We improve the performance of the weak learner (i.e. Latin ones) in this case.

For method A, the accuracy of recognizing the Greek and the mathematical is much worse than before but for the Latin character, it's much better. Generally, its performance is the same with case 1.

For method B, the general performance is not improved.

For method C, the accuracy of Greek and the mathematical dataset is only slightly worse than before, but the recognition of the Latin characters is much better. In general, method B performs better than the last case.

In this case, we change the average accuracy by improving the weak learners in the parallel neural network, the method C evaluated by the different set of the images performs better than the other two methods.

Case 3 Based on Case 1, we improve NN3, NN4, NN5 and NN6 by raising the requirements of training(i.e., with higher threshold), which also reduces the influence of random weight generation. The improvement of strong learners is based on adjusting the threshold for training, as in the previous testing, these neural networks have already get the accuracy of 100%. The results are shown on the Table 4.5.5 and 4.5.6.

Training Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (576)	1.04	8.68	0.17
math (1056)	0	5.96	0
Latin (1200)	11.67	13.67	21
Total	5.16	9.78	8.93

Table 4.5.5: Testing the parallel neural networks with previous training sets(v.3)

We could get from the result that all of the three strategies perform much better than before. Especially for the method A and C, the accuracy for recognizing both Greek characters and the mathematical characters are nearly 99%. Compared with

Testing Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (276)	1.09	7.61	0
math (528)	0	3.41	0
Latin (600)	16.33	20.17	24.33
Total(1404)	7.19	11.4	10.4

Table 4.5.6: Testing the parallel neural networks with testing sets(v.3)

method C, method A perform better in general. However, for the method B, the general performance is nearly the same as the last two cases.

Case 4 Combine the improvement from the second and third case.

Training Data Set	pErr(%)		
	Method A	Method B	Method C
Greek	1.91	7.64	0.52
math	0	4.92	0
Latin	7.08	10.67	10.17
Total	3.39	7.91	4.42

Table 4.5.7: Testing the parallel neural networks with previous training sets(v.4)

We improve the performance of both strong learners and weak learners. As Table 4.5.7 and Table 4.5.8 shows, we get the best accuracy of PNN which is 5.06% in method A. For the method C, it's a little bit worse than method A which is 7.62%. As for method B, it is the worst.

Testing Data Set	pErr(%)		
	Method A	Method B	Method C
Greek (276)	2.17	11.96	2.17
math (528)	0	2.28	0
Latin (600)	10.83	17.33	16.83
Total(1404)	5.06	10.61	7.62

Table 4.5.8: Testing the parallel neural networks with testing sets(v.4)

Since it's the best performance of PNN could get among the four cases. In the following, we would analyze an image (Latin character "a") which is recognized by six neural networks separately. In Figure 4.5.3, these subfigures are plotted by the input vectors generated from each neural network. All the three strategies of PNN select the output from the Greek neural network (nn4), so that this image is recognized as Greek character α.

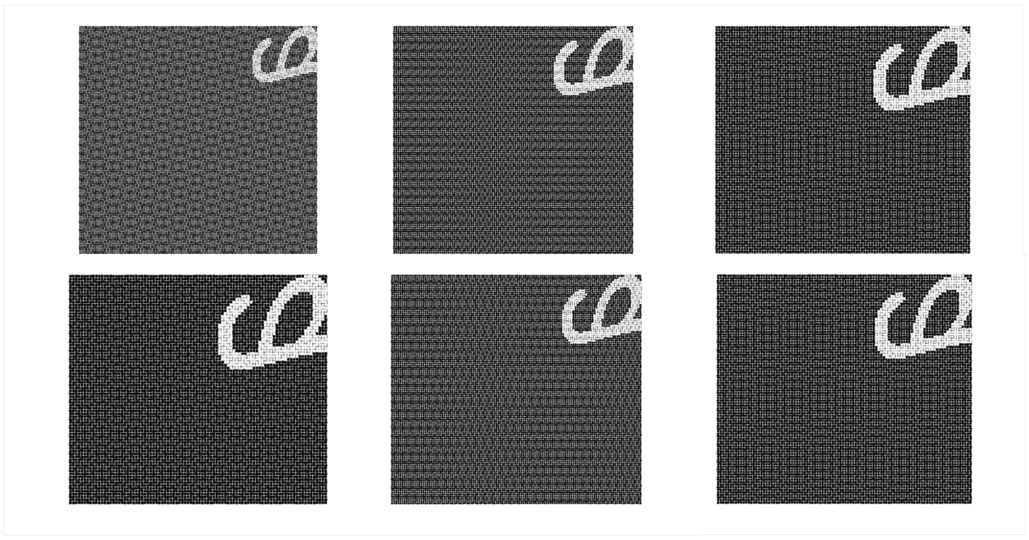


Figure 4.5.3: The testing images which are wrongly recognized by PNN.

Summary from all cases The method A performs steadily in general, which could face the challenges from different neural networks. In the first, third and fourth case, its performance is the best among three methods, and with the higher possibility to be improved For method B, the selection is very limited, which only based the highest element of an output vector. Compared with other two methods, it's the worst one. For method C, compared with the method A, it's a little bit unsteady, although it's developed based on it. However, in the second case, when we improve the study ability of weak learners, its performance is improved obviously with respect to the method A.

There is still a long way to exploit the voting strategies of the parallel neural networks. In our study, the best performance of the strategy is 94.94% accuracy.

Chapter 5

Conclusion

The goal of this thesis is to study some tools and strategies for developing an automatic Optical Character Recognition (OCR) system specifically tailored for scientific text in order to assist blind and visually impaired persons during their studies. In this thesis, the pattern recognition algorithm of OCR is based on Artificial Neural Networks (ANN).

As the main outcome of this thesis, we have developed a recognition core based on six ANNs, each of which focuses on different sets of patterns (like Latin characters, Greek characters and mathematical symbols) and font sizes.

In order to train the neural network, the labeled data sets are required. In our study, we selected the 26 Latin characters, 24 Greek characters and 44 the most frequently-used mathematical symbols as the training objects.

The database consists of image files which are generated using Matplotlib. This database is the second outcome of this thesis. These images vary in their fonts, sizes and symbols, and each one has specific file name. We worked on 13,592 files for training, and 1440 files for testing.

To get the labeled data set, we used Matlab to convert each image into a vector, the collection of which is the input matrix of a neural network. And the expected output of a neural network is generated by recognizing the name of each image file.

After preparing the labeled data sets, we trained the six neural networks separately by means of BP algorithm. In this step, each neural network is trained with different structures and parameters, i.e., the number of neurons in the hidden layer (N_2), the range of initial weights from random generators (kk) and the learning rate (eta) to get their best performances evaluated by the elapsed time and training steps. After training, each neural network is able to recognize all the images in its training dataset with 100% accuracy. However, the learning abilities for the neural networks are still to be assessed by the testing dataset. In this step, we re-adjusted the structures and parameters for each neural network with the aim of improving its performance which is evaluated by the accuracy of recognizing the images in testing dataset.

In the experimental test, both of the percentages of errors for testing dataset in NN1 and NN2, which are in charge of the Latin characters, were about 35%. With the previous experience, we decided to increase the training dataset from 30 fonts to 115 fonts and increase the number of neurons in the hidden layer from 100 to 500 which reduces the percentage of errors approximately to 12%. For the neural networks which are in charge of the Greek character and the mathematical symbols, the accuracy of the testing data set is 100%. We only focused on finding the optimal parameters to obtain the shorter elapsed time and less training steps by repetition tests. In a word, we trained and tested all six neural networks separately and then assessed their performances in the consideration of elapsed time, training steps, and learning ability.

Based on six trained neural networks, we exploited the parallel neural networks (PNN) by developing three strategies of voting algorithm which is used for selecting the most proper output from all the outputs of six parallel-running neural networks without considering the invalid outputs, in which there is not only one high element. In this part, we developed three methods for the voting algorithm. *Method A* is based on the mean Euclidean norm between the real output and the rounded output of a neural network. *Method B* focuses only on the maximum element in each output vector. *Method C* is based on the method A, but additionally applies the previous accuracy on the selection.

We assessed three strategies by the previous training set and testing set with different PNNs, and observed their performances. Generally, the behavior of method A is the most steadiest and the best, which could recognize the training set with the accuracy of 96.6% and the testing set with accuracy of 94.9%. The performance of method C, whose accuracies are 95.6% and 92.4% respectively, is slightly worse than method A. Additionally, the behavior of method C, much related to the accuracy of each single neural network, is not so predictable. As for the method B, its performance is the worst with the accuracies 92.1% and 89.4% for the training set and testing set.

Overall, in this thesis, we generated our own data set, and trained the six neural networks separately with the average accuracy 96.95%.

Also we developed three strategies of the voting algorithm for the parallel neural networks. Among them, the method, which has the best and steadiest behaviors, is based on the mean Euclidean norm with accuracy 94.9%.

In the future, there is still the large possibility to improve the performance of PNN by developing new voting strategies based on our current study. For the better behavior of voting strategies, for example, with the support of large-scale database, we could apply the occurrence frequency of each character to the voting strategies.

Bibliography

[1] Overview of neuron structure and function.

[2] Neuron action potentials: The creation of a brain signal.

[3] Understanding the transmission of nerve impulses.

[4] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[5] Seymour Papert Marvin Minsky. Perceptrons: an introduction to computational geometry. 1969.

[6] Overview of artificial neural networks and its applications.

[7] *Introduction to Neural Networks for Java.* Heaton Research. Inc, 2008.

[8] *Neural Networks and Deep Learning.* Determination Press, 2015.

[9] *Simulation Neuronaler Netze" [Simulation of Neural Networks] (in German) (1st ed.).* Addison-Wesley, 1994.

[10] Ameet Talwalkar Mehryar Mohri, Afshin Rostamizadeh. Foundations of machine learning. 2012.

[11] Backpropagation.

[12] Ronald J. Williams David E. Rumelhart, Geoffrey E. Hinton. Learning representations by back-propagating errors. *Neurocomputing: foundations of research*, 1988.

[13] M. B. Menhaj M. T. Hagan. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks.*

[14] E. Fiesler G. Thimm. High order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, 1997.

[15] N. Kandil, K. Khorasani, R. V. Patel, and V. K. Sood. Optimum learning rate for backpropagation neural networks. In *Proceedings of Canadian Conference on Electrical and Computer Engineering*, pages 465–468 vol.1, Sep 1993.

[16] Anna Capietto Nadir Murru Rosaria Rossini Giuseppe Air'o Farulla, Tiziana Armano. Artificial neural networks and fuzzy logic for recognizing alphabet characters and mathematical symbols. *arXiv:1607.02028v1 [cs.NE]*, 6 Jul 2016.

[17] TAVISH SRIVASTAVA. Basics of ensemble learning.

[18] Andrew Y. Ng. Preventing "overfitting" of cross-validation data. Technical report.

[19] The introduction of matplotlib.

[20] Sunil Karamchandani Parul Shah. Ocr-based chassis-number recognition using artificial neural networks.