

**HANDWRITTEN COURTESY AMOUNT
RECOGNITION**

By

CHONG YONG SHEAN

A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Science (Hons.)

Applied Mathematics With Computing

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2014

DECLARATION OF ORIGINALITY

I hereby declare that this project report entitled "**HANDWRITTEN COURTESY AMOUNT RECOGNITION**" is my own work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled "**HANDWRITTEN COURTESY AMOUNT RECOGNITION**" was prepared by **CHONG YONG SHEAN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Hons.) Applied Mathematics With Computing at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : _____

Date : _____

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of University Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

©2014, CHONG YONG SHEAN. All rights reserved.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my parents for supporting me throughout the research, my supervisor Dr Tay Yong Haur for guiding me on the road to completion of the project, and my friends for helping me on improving the project. They have provided support to me in terms of supplying ideas, providing feedbacks and guidance.

CHONG YONG SHEAN

HANDWRITTEN COURTESY AMOUNT RECOGNITION

CHONG YONG SHEAN

ABSTRACT

Cheque processing software is used in banks to help bank employees to clear cheques in a more efficient manner. In this project, A system that is capable of reading courtesy amount written on the cheque is implemented. This system will be useful to the bank in reducing cheque processing cost in terms of time and labour.

The programming language Python is used to implement the handwritten courtesy amount recognition system. Python is an open source programming language and together with numpy and scipy libraries, it creates a good environment for image processing and machine learning. Python is chosen because it is user-friendly and well documented, and the libraries contain many useful functions that could be applied easily.

First, various techniques from existing researches are reviewed and explained. Then the amount recognition system is implemented. The implementation consists of five components — field extraction, segmentation, feature extraction, recognition, and post-processing. The recognition module requires a classifier to be trained, which in this project Support Vector Machines (SVM) is used. It is designed to perform recognition effectively and accurately. Different techniques are studied and tested to evaluate its performance.

TABLE OF CONTENTS

TITLE	i
DECLARATION OF ORIGINALITY	ii
ACKNOWLEDGEMENTS	vii
ABSTRACT	viii
LIST OF FIGURES	xii
LIST OF TABLES	xiv
CHAPTER 1 Introduction	1
1-1 Motivation	1
1-2 Objective	2
1-3 Problem Statement	2
1-4 Scope and Planning	3
CHAPTER 2 Literature Review	4
2-1 Summary of Techniques	4
2-2 Details of Techniques	4
2-2-1 Pre-processing	4
2-2-2 Field Extraction	6
2-2-3 Segmentation	7
2-2-4 Feature Extraction	10
2-2-5 Classification	11
CHAPTER 3 Methodology	15
3-1 System Overview	16
3-2 Major Stages of the System	17
3-2-1 Pre-processing	17
3-2-2 Field Extraction	17
3-2-3 Segmentation	18
3-2-4 Feature Extraction	19
3-2-5 Classification	21
3-2-6 Post-processing	23

3-3 Software Implementation	24
3-3-1 Training and Testing of Single Digits	24
3-3-2 Testing with cheque images	28
CHAPTER 4 Experiment and Analysis	32
4-1 Experiment 1	32
4-1-1 Problem Statement	32
4-1-2 Methodology	32
4-1-3 Experimental Setup	33
4-1-4 Result and Analysis	33
4-1-5 Conclusion	34
4-2 Experiment 2	35
4-2-1 Problem Statement	35
4-2-2 Methodology	35
4-2-3 Result and Analysis	36
4-2-4 Conclusion	37
4-3 Experiment 3	38
4-3-1 Problem Statement	38
4-3-2 Methodology	38
4-3-3 Result and Analysis	39
4-3-4 Conclusion	40
4-4 Experiment 4	40
4-4-1 Problem Statement	40
4-4-2 Methodology	41
4-4-3 Result and Analysis	41
4-4-4 Conclusion	42
CHAPTER 5 Conclusion and Future Improvements	43
5-1 Summary	43
5-2 Contributions	43
5-3 System Limitation	43
5-3-1 Future Improvement	44

APPENDIX A Source Code	1
A-0-2 Experiment 1	1
A-0-3 Experiment 2	2
A-0-4 Experiment 3	3
A-0-5 Experiment 4	6
A-0-6 Handwritten Courtesy Amount Recognition System GUI	8
A-0-7 Training and Testing of MNIST Digits GUI	19
A-0-8 Training and Testing of Combined Dataset	53
A-0-9 Shared scripts - required for the above scripts	56

LIST OF FIGURES

1.1	Cheque layout specified by Bank Negara Malaysia, 2007	2
2.1	An example of slant correction. Picture adopted from (Cheung 1998).	6
2.2	Pixel-to-pixel path construction based on the current position. Picture adopted from (Khan 1998).	8
2.3	An example of touching numerals and its contours. Picture adopted from citepyunlei04.	9
2.4	Applying over-segmentation on a connected component and find the best segmentation path based on word likelihood. Picture adopted from (Y.H. et al. 2003).	10
2.5	A typical feed-forward neural network architecture used in backpropagation. Picture adopted from (<i>Neural Networks</i> 2011).	12
2.6	The optimal hyperplane that gives the maximum margin between classes. Picture adopted from (<i>Introduction to SVM</i> 2011).	14
3.1	The overall process and the components involved in training and testing module.	16
3.2	Example of splitting connected components. Picture adopted from (Chaabani 2007).	18
3.3	The 8-directions and its corresponding digit that indicates each direction.	19
3.4	The pixels of a zero digit image in binary form.	20
3.5	Transforming x to $\phi(x)$ to become linearly separable.	22
3.6	Screenshot of the <i>Training</i> tab.	25
3.7	Screenshot of the <i>Testing</i> tab.	26
3.8	Screenshot of the <i>Analysis</i> pop-up window.	27
3.9	Screenshot of the <i>Classify</i> tab.	27
3.10	Screenshot of the <i>Handwritten Courtesy Amount Recognition</i> demonstration GUI.	28
3.11	A successful example of recognition.	29
3.12	Recognition error: A decimal point is missed.	30

3.13 Recognition error: Classify ',' as '1'	31
4.1 Examples of successful dropfall segmentation.	36
4.2 Examples of unsuccessful dropfall segmentation.	37
4.3 Frequency of occurrence of error on the recognition of 100 cheques before and after training.	39

LIST OF TABLES

4.1	Accuracy of training and testing set for 3-fold cross validation on Normal SVM	41
4.2	Accuracy of training and testing set for 3-fold cross validation on Weighted SVM	41

CHAPTER 1: INTRODUCTION

According to *Financial Stability and Payment Systems Report 2012* (2013), cheques account for 11.1% of the non-cash transactions, and nearly 203.8 million cheques worth RM2033 billion were processed in 2012 in Malaysia. Despite the rapid emergence of credit and debit cards and other electronic means of payment, paper cheques are still widely used in bank transactions. Since most of the cheques need to be partially processed by hand, there is significant interest in the banking industry for new approaches that can be used to read paper cheques automatically. Such approaches can greatly reduce the workload of bank employees.

In the field of handwritten character recognition, there are two types of systems, which are known as on-line systems and off-line systems. On-line systems involve recognition of handwritten characters by touch input, in other words, on-line systems recognise the character written by the user according to the input strokes. On the other hand, off-line systems involve character recognition by scanned image, where no stroke information is known to the system. Generally, it is more challenging to implement off-line systems because of limited information available. In this case, the amount recognition system is an off-line system since scanned cheque image is used as the input. (Cheung 1998)

1-1 Motivation

According to Bank Negara deputy governor, it is estimated that cheque processing costs Malaysia RM768 million a year. This amount is huge and a sheer wastage. (*Malaysians still prefer cheques says Bank Negara 2013*) Reading paper cheques and typing into computers manually is very time-consuming and labour-intensive. Therefore, accurate and efficient cheque recognition systems are in high demand by banks.

There are already existing solutions to automatic bank cheque recognition, however, the use of such systems has not become widespread worldwide is that cheques do not conform to a single standard. Cheques have many different sizes, and the courtesy amount field is not located in the same place on all cheques. Therefore there is suf-

ficient reason to develop a tailor-made bank cheque recognition system for Malaysia cheques.

1-2 Objective

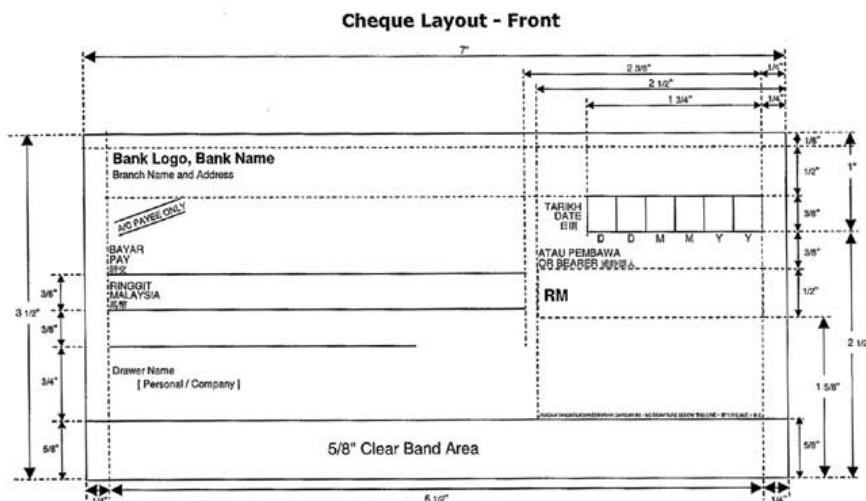
The aim of this project is to create a system that is able read handwritten digits on the amount field of scanned bank cheques and to reject a cheque for human recognition in case of doubt.

The objectives are:

- Study the methods of feature extraction and compare the performance of each method
- Implement field extraction and segmentation techniques on bank cheques
- Design the methods to identify decimal points, commas, and junk segments

1-3 Problem Statement

A standard bank cheque consists of the following fields: issue date box, payee's name field, courtesy amount field, legal amount field, and signature pane. This project focuses on handwritten character recognition on courtesy amount field.



BNM CDSS, 18/04/2007

Figure 1.1: Cheque layout specified by Bank Negara Malaysia, 2007

Recognising courtesy amount field is one of the most challenging tasks for the automatic bank cheque recognition process. The difficulty is due to great variability in handwriting styles, handwriting devices, and a lack of patterns and symbols used by writers to prevent fraud. (Shah et al. 2010) This makes it difficult to set the criteria to whether accept or reject a cheque.

The major challenges of this project are to train the machine to recognise a variety of writing styles and to differentiate between useful characters and junk characters.

1-4 Scope and Planning

This project focuses particularly on segmentation and recognition of the components in the courtesy amount field. A simple field extraction is implemented to demonstrate the whole process of a basic bank cheque recognition.

Discussion on the important stages or steps in the recognition process is included. Some techniques on post-processing were described briefly.

CHAPTER 2: LITERATURE REVIEW

2-1 Summary of Techniques

A general recognition system consists of the following steps in the order of how they are listed: input of image, field extraction, segmentation, feature extraction, recognition. In some cases, pre-processing and post-processing are required to improve the performance of the system.

The recognition step requires prior machine learning or training. To train the machine to perform recognition, a large dataset and features to be extracted from the dataset are needed. Then, an appropriate classifier is chosen to be trained with the extracted features. After which the classifier is trained, it is utilised in the recognition engine.

2-2 Details of Techniques

In this section, the techniques involved in the steps of training and recognition are discussed in-depth.

2-2-1 Pre-processing

A typical procedure of processing of a scanned document image is to binarise the image, background extraction, and noise removal. Threshold is required to define which part of the image will be white pixels (background) and which belongs to black pixels (foreground). For background extraction, Shah et al. (2010), Wheelock (2010) and Kurniawan et al. (2011) uses Otsu method to perform image thresholding to obtain the binary image. In Kurniawan et al. (2011), it explains how Otsu's method work and implemented in computer. Instead of defining a fixed threshold value for all kinds of images, Otsu's method searches for the threshold that minimises the variance within class and maximises the variance between classes. There are two classes in image binarisation: background and foreground. The variance is defined as a weighted sum of variances of the two classes as follows:

$$\sigma_{\omega}^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \quad (2.1)$$

Weights ω_i are the probabilities of the two classes separated by a threshold t and variances σ^2 of these classes. Otsu defined that minimising the variance within class is similar as maximising the variance between class:

$$\sigma_b^2(t) = \sigma^2 - \sigma_{\omega}^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2 \quad (2.2)$$

Equation 2.2 is expressed in terms of class probabilities ω_i and class means μ_i which can be updated iteratively. To put it in illustration, Y.H. et al. (n.d.) mentioned by the histogram of grayscale values of a scanned document image, the task of binarisation is to determine the optimal value in the valley between the two peaks: a larger peak corresponding to the white background and a smaller peak corresponding to the foreground.

The binarised image will still contain some noise which can reduce the performance of the recognition system. Some examples of noise in a scanned cheque are the background patterns, smudge or dirt on the paper, and paper crease. Even though it is not guaranteed that by noise extraction all noises will be removed, it is helpful to reduce the noise in the image. Smoothing operations are often used to eliminate the artifacts introduced during image scanning.

Sometimes the characters in the scanned image can be blur or appear disconnected due to poor image quality, as well as erratic hand movement. In Lei et al. (n.d.), Y.H. et al. (n.d.), Nath & Rastogi (2012), Wheelock (2010), Sofiene & Samia (n.d.), stroke filling is performed to improve stroke connectivity. In Lei et al. (n.d.), Nath & Rastogi (2012), morphological close is implemented to connect the broken strokes.

For cheque images, handwriting normalisation such as baseline construction, slant and skew correction are performed. To construct the baseline of a string of characters, the local minima and maxima of the handwriting signal is extracted by running a contour following algorithm on the internal and external contours of the string image (Y.H. et al. n.d.).

Due to the inaccuracies in writing style and scanning process, the writing might be slightly tilted within the image. This can affect the effectiveness of recognition and therefore should be detected and corrected. (Nath & Rastogi 2012) Slant correction is

the process which attempts to adjust the slant of the handwriting to the vertical line. (Wheelock 2010) This can be performed by adjusting the baseline to horizontal and aligning the local minima of the lower contour to its baseline. In Cheung (1998), a rather simple implementation of slant correction is to rotate the bitmapped string of characters both clockwise and counter-clockwise around the centre of the bitmap. The best rotation that will provide a slant-corrected presentation is a rotation that results in the smallest horizontal width of the string.

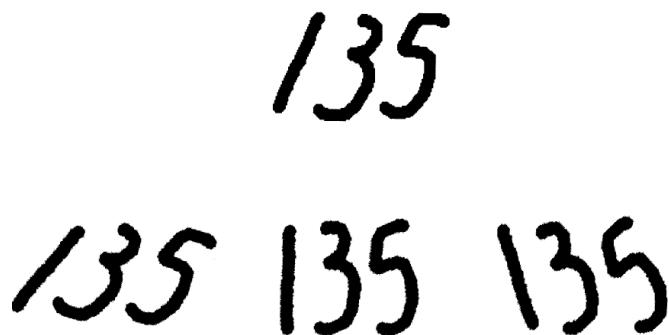


Figure 2.1: An example of slant correction. Picture adopted from (Cheung 1998).

In certain papers such as Cheung (1998) and Rafael et al. (2002), thinning and rethickening is performed to standardise the thickness of the stroke of characters.

2-2-2 Field Extraction

The common approaches on extracting the region of interest are smearing and smoothing (Agarwal et al. 1995) , or by observation on the position of the field in all cheques (Sofiene & Samia n.d.).

The idea behind smearing and smoothing is to determine the overall structure of a page, then breaks this down into regions and lines. First, the image is smeared horizontally. This results in the adjacent characters and words become smeared onto one another, and the smeared image become the mask of region extraction. The regions are obtained by subtracting the masked area from the cheque image. Smoothing is similar to smearing, whereby the resolution of the image is lowered, and the connected components in the low resolution image usually represent the large blocks or strings in the image.

Sofiene & Samia proposed that field extraction can be done with a direct extrac-

tion from the cheque image by defining the standard physical and logical structure of bank cheques. This is by the assumption that all the cheques processed have the same structure.

2-2-3 Segmentation

There are three categories of segmentation: segmentation of the extracted field/string , segmentation of the connected components, or segmentation of both at the same time. For the string segmentation, Khan (1998) and Agarwal et al. (1995) use depth first search of connected components. The basic idea behind connected component extraction is to find a black pixel and then find all the pixels which are connected to it. (Khan 1998) These connected pixels form a connected component.

How do we know if the pixels are connected? It is defined that if they are adjacent, in other words, they are neighbours to each other in four or eight directions. Khan (1998) wrote that it is unusual for two pixels which are only diagonal to each other to be part of the same character, therefore only four directions (above, below, left, right) from that pixel will be considered. For implementation, depth first search is used in Agarwal et al. (1995) to compute and obtain the set of connected components in the cheque image.

If the characters in the cheque image are well separated when they are written, then each component would represent a whole character. Unfortunately, this is seldom the case. If some characters are touching, overlapping, or disjoint, then each connected component would consists of multiple characters or parts of the character. To confront this issue, the second category of segmentation is introduced.

For the second category, dropfall algorithm is used in Khan (1998), Rafael et al. (2002), Clement et al. (2006), Rui et al. (2009) to segment touching numerals. Dropfall algorithm is a simulation of acid dropping from the top of the number, flowing along the edge of number and then corrodes (cut) the edge when it has nowhere to go.

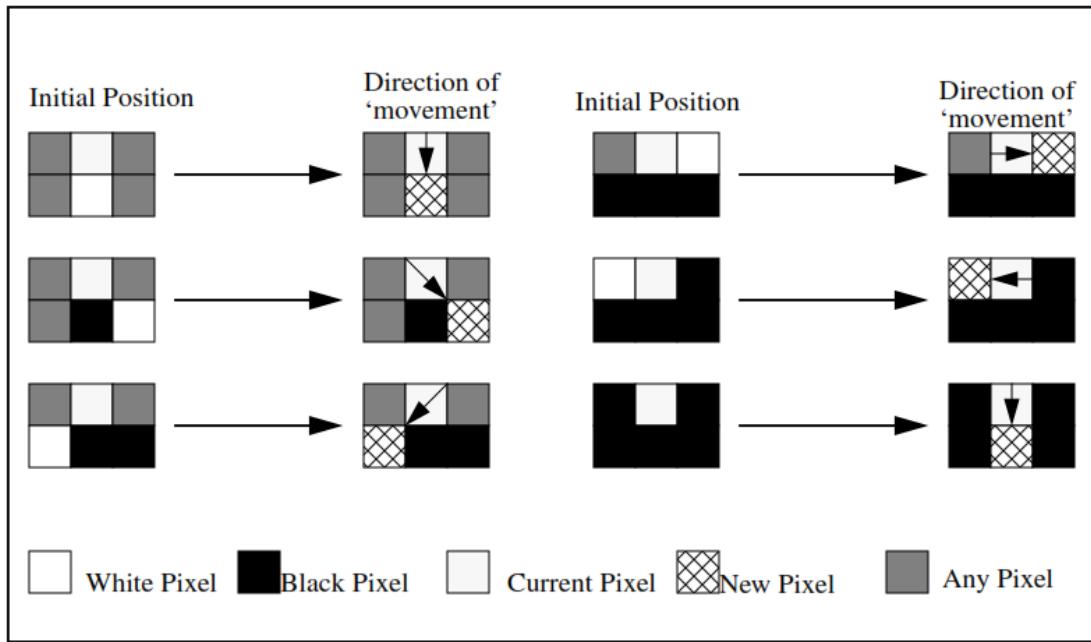


Figure 2.2: Pixel-to-pixel path construction based on the current position. Picture adopted from (Khan 1998).

The movement rules can be represented in pseudocode as follows (Rafael et al. 2002):

Rule 1: if down is white, then move down

The acid falls. This is the normal movement through the background.

Rule 2: if down-right is white, then move down-right

Move to the right empty diagonal. This is the movement around the contour of one character, leaving it to the left.

Rule 3: if down-left is white, then move down-left

If the diagonal movement to the right was not possible, then the acid tries a diagonal movement to the left.

Rule 4: if right is white, then move right

If the acid reaches a flat area, it tries to reach the contour by horizontal move before beginning to corrode the character.

Rule 5: if left is white and the drop does not come from the left, then move left

In a flat area the right move is tried first. If it is not possible, then horizontal left movement is performed.

Rule 6: else move down #cut

Another approach is to perform contour analysis on the extracted string of characters. Contour is defined as foreground pixels that do not have any other foreground pixels at top, right and bottom up to some vertical distance. (Lei et al. n.d.) From the prospective contours produced by contour analysis, prospective segmentation points are located. These are the possible points which contains the cutting point between connected characters. Since the points are numerous, some heuristic rule is defined to eliminate unnecessary points. These rules are written in detail in Kurniawan et al. (2011).

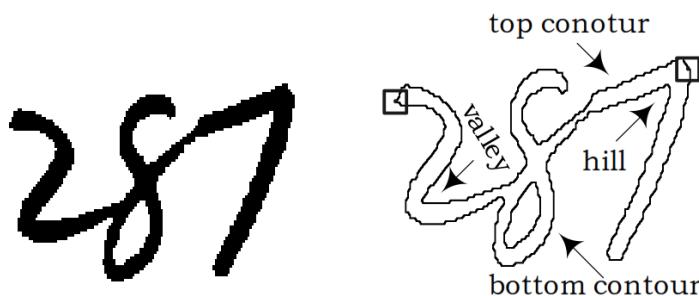


Figure 2.3: An example of touching numerals and its contours. Picture adopted from citepyunlei04.

However, it is difficult to determine how many characters or digits are there in the connected component. If it fails to determine, then the techniques in the second category cannot be used. Therefore, a third category of segmentation techniques are mentioned in (Y.H. et al. n.d.), (Y.H. et al. 2003), (Md Tanvir & Sabri A. 2013) and (Gattal & Chibani 2012). Their techniques are similar, which is based on sliding window segmentation. In Y.H. et al. (2003), the author uses hybrid Neural Network - Hidden Markov Model (NN-HMM) as a basis for segmentation of cursive handwritten words. In each iteration of sliding window, part of the word that is sliced (letter hypothesis) is fed into neural network to compute its observation probability. This continues until all possible segmentation paths, the best path is chosen based on the likelihood computed by HMM. It achieved a high accuracy of 96.1%.

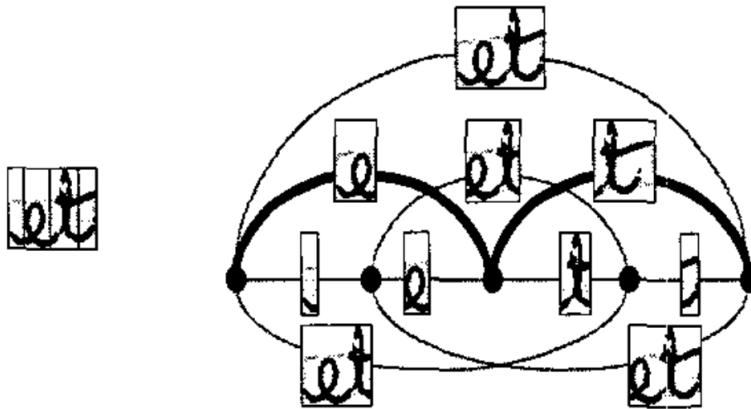


Figure 2.4: Applying over-segmentation on a connected component and find the best segmentation path based on word likelihood. Picture adopted from (Y.H. et al. 2003).

2-2-4 Feature Extraction

Many efforts have been done in finding a good feature. A good feature should describe the image precisely yet being able to describe other similar images in the same way. It also should be scale invariant and rotation invariant, in other words, it should not be affected by the size or resolution of the input image (if size is not a criterion) nor the angle of rotation of the image. Generally there are two categories of features: topological features and statistical features.

An example of topological features is Fourier descriptors. In Nath & Rastogi (2012) and Sofiene & Samia (n.d.), Fourier Descriptors are used to represent the shape of boundary of a character. Fourier descriptors are a way of encoding the shape of a two-dimensional object by taking the Fourier transform of the Centroid Distance Function $r(t)$, which is expressed by the distance of the boundary points from the centroid (g_x, g_y) of the character. (*Fourier Descriptors* 2012)

$$r(t) = \sqrt{[(x(t) - g_x)^2 + (y(t) - g_y)^2]} \quad (2.3)$$

The discrete Fourier Transform of $r(t)$ is given by:

$$a_n = \frac{\sum_{t=0}^{N-1} \exp\left(\frac{-j^2\pi nt}{N}\right)}{N}, n = 0, 1, \dots, N - 1 \quad (2.4)$$

The larger the number of Fourier Descriptor, the more precise it is to represent the shape of boundary of a particular image. Normalised Fourier Descriptors are scale-,

translation-, and rotation-invariant.

For statistical features, a variety of algorithms and measures are used, such as Chaincode algorithm (Cheung 1998), PCA-based features (Das et al. 2012), histogram analysis (Wheelock 2010) and pixel average (Stefano et al. 2009). Chaincoding is a statistical contour-based feature extractor, which represents an image by storing the image boundaries in terms of directions. (Cheung 1998) The construction of chaincode is mentioned in detail in Danescu (n.d.). Chain codes may be made position-independent by normalising the start point. Other information that characterise an image such as pixel density, centroid, height, aspect ratio are also used in Y.H. et al. (n.d.), Md Tanvir & Sabri A. (2013), Pal et al. (2001).

An interesting approach - Directional Distance Distribution (DDD) in Oh & Suen (1997) is a measure of representation of overall pixel distribution. DDD computes the distance information for black and white pixels in eight directions. Since the input image is binary, DDD is a reliable feature which considers both the black and white pixels and their distributions. The detail of implementation is provided in Section 3.5.

2-2-5 Classification

One of the most widely used classifier in digit recognition is artificial neural network (ANN). ANN is an abstract simulation of a real nervous system, which is adaptive, distributed and mostly nonlinear. (Eduardo Gasca 2007) It consists of a network of artificial neurons or nodes that are interconnected. Because a lot of real world problems are nonlinear, ANN becomes widely used in solving nonlinear problems such as pattern recognition, prediction, and optimisation of functions. (Eduardo Gasca 2007)

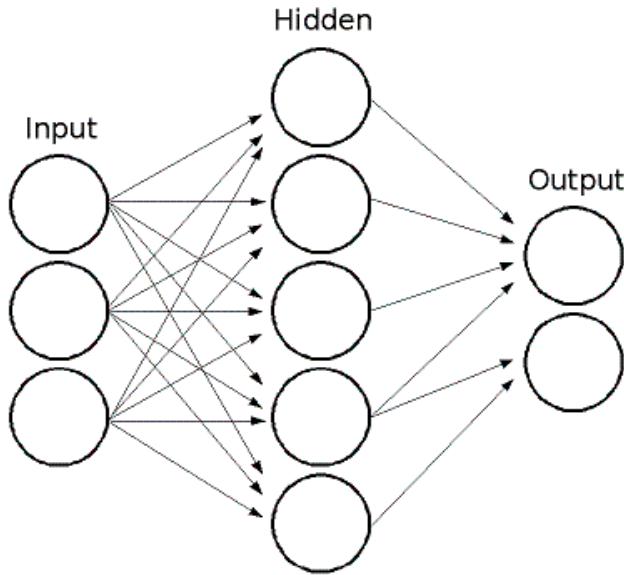


Figure 2.5: A typical feed-forward neural network architecture used in backpropagation. Picture adopted from (*Neural Networks* 2011).

One of the most important models in the artificial neural networks is called the Multilayer Perceptron (MLP). This is of the supervised learning and is feedforward. It is constituted by one or several layers of hidden neurons, between the input and the output nodes.

There are various algorithms in achieving the learning objective. In Cheung (1998) and Wheelock (2010), the backpropagation algorithm is defined over a multilayer feed-forward neural network.

The Backpropagation Algorithm

(Sergios & Konstantinos 2006)

- *Initialisation:* Initialise all the weights with small random values from a pseudo-random sequence generator.
- *Forward computations:* For each of the training feature vectors $x(i)$, $i = 1, 2, \dots, N$, compute all the predictions and compare with the actual class $y(i)$ it belongs. The error is $(\text{prediction} - \text{actual})$.
- *Backward computations:* compute the weight correction for all weights from hidden layer to output layer; and then the weight correction for all weights from

input layer to hidden layer.

- *Update the weights:* until all training feature vectors are classified correctly or a stopping criterion is satisfied. Let \mathbf{w}_j^r be the weight vector of the j th neuron in the r th layer. For $r = 1, 2, \dots, L$ (layers of neurons) and $j = 1, 2, \dots, k_r$ (number of neurons in the r th layer), $\mathbf{w}_j^r(\text{new}) = \mathbf{w}_j^r(\text{old}) + \Delta \mathbf{w}_j^r$

In the case of pattern recognition, the number of elements in the feature vector represents the input nodes, and the number of classes which the pattern can be classified is the number of output nodes. But how many hidden layers and hidden nodes we should assign to the MLP?

Kanellopoulos et al [Kanellopoulos, 1997] suggest that the number of nodes in the first hidden layer should be exactly the same as the maximum value that results when estimating between two and four times the amount of nodes in the input later, or two or three times the number of nodes of the output layer.

The second type of neural network used in Fabien et al. (2007), Ciresan (2008) and Ian J. et al. (2013) is convolutional neural network. Convolution is a mathematical term, defined as applying a function repeatedly across the output of another function. It can extract topological properties from an image. (Fabien et al. 2007) It extracts features from the raw image in its first layers and classify the pattern with its last layers. The advantage of convolutional neural network over MLP is the ratio of the number of trainable parameters to the number of connections is very small, hence the effectiveness of training is improved.

Besides ANN, Support Vector Machines (SVM) is also widely used in many classification problems. The goal of SVM is to find the optimal separating hyperplane in a feature space. (Fabien et al. 2007) The idea of finding such hyperplane is based on the maximisation of the margin. Originally, SVMs were designed to solve binary classification problems, but were later generalised to solve multi-class problems.

Let $x_i = 1, 2, \dots, N$, be the feature vectors of the training set, X . The goal is to find a hyperplane, that is

$$g(x) = \mathbf{w}^T \mathbf{x} + w_0 = 0 \quad (2.5)$$

where $w = [w_1, w_2, \dots, w_l]^T$ is known as the weight vector and w_0 as the threshold. Such hyperplane is not unique. There can be many hyperplanes that can classify all

points of training set correctly, but not all of the hyperplanes can classify testing points accurately. This is related to the generalisation capability of the function $g(x)$. (Sergios & Konstantinos 2006) More details on SVM will be discussed in Section 3.6.

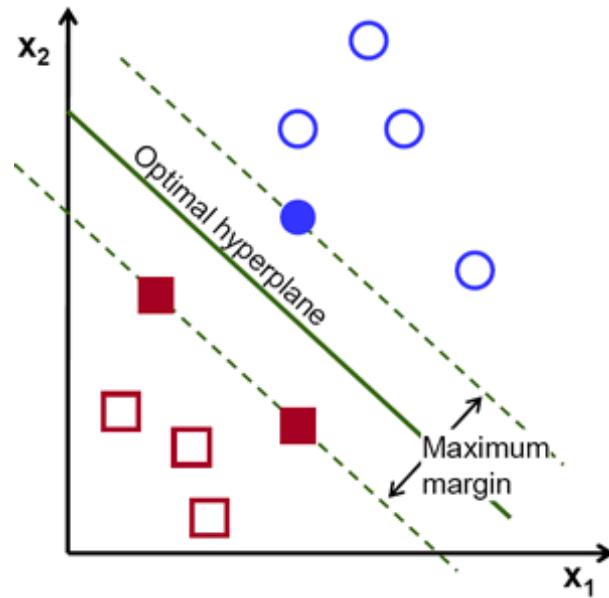


Figure 2.6: The optimal hyperplane that gives the maximum margin between classes.
Picture adopted from (*Introduction to SVM* 2011).

In most applications, the confidence score is how much the output is activated. The level of activation function produced at the output is a widely used confidence measure.

Other classifiers such as K-Means algorithm (Wheelock 2010), Hidden Markov Model (Wheelock 2010), and Bayesian Networks (Cheung 1998, Stefano et al. 2009) are also methods of supervised learning. However, these methods are not as widely used as ANN and SVM in handwriting recognition so they will not be discussed here.

According to Clement et al. (2006), a good digit classifier should perform the two tasks as follows:

- *Discrimination*: The classifier should output the correct digit class with a high confidence value.
- *Detection*: The classifier should be able to reject outliers which are not part of digits.

CHAPTER 3: METHODOLOGY

3-1 System Overview

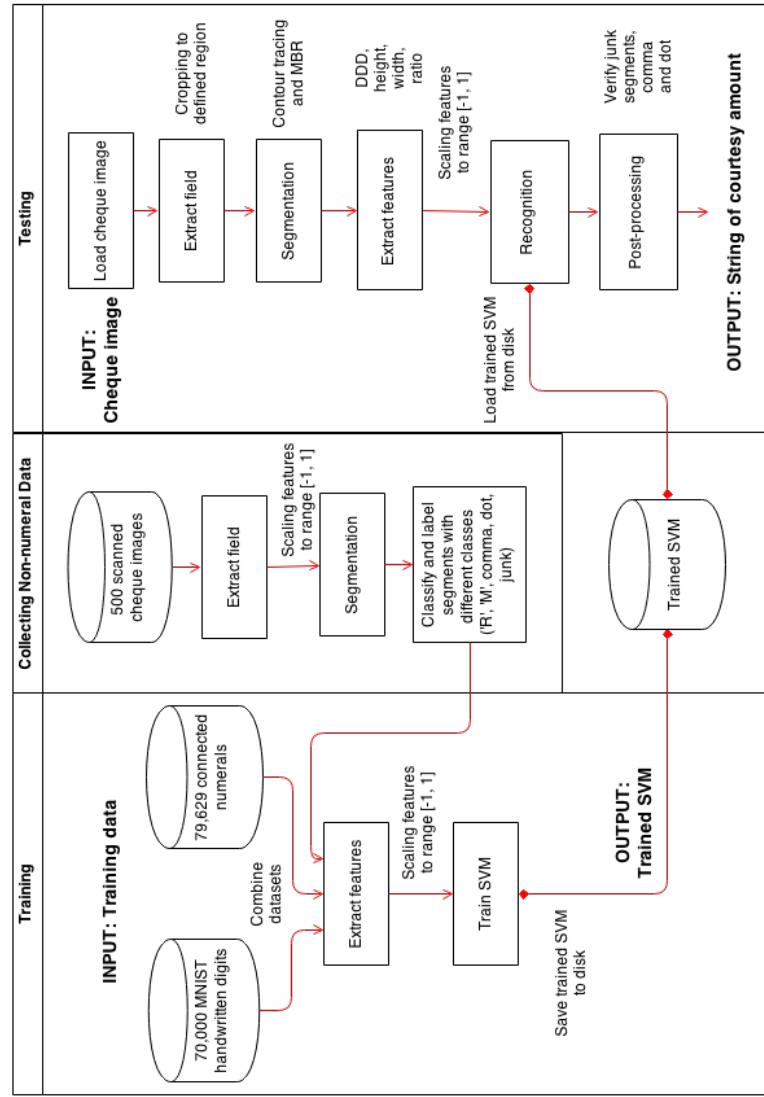


Figure 3.1: The overall process and the components involved in training and testing module.

3-2 Major Stages of the System

This section explains the major steps involved in the training and recognition process of the system.

3-2-1 Pre-processing

In this paper, no pre-processing technique is applied for simplicity and assumption that the cheque image is scanned in bitonal form is hold. Scanning software can be easily configured to perform the binarisation on its own.

3-2-2 Field Extraction

Field extraction refers to the extraction of region of interest, which in this case, is the courtesy amount field. Since the location of courtesy amount field is about the same in all cheques, the relative position of the bounding box to the whole cheque is used to locate courtesy amount field in other cheques. The bounding box is defined by 4 values: top, left, bottom, right. 40 cheques are observed and the coordinates of the bounding box is recorded to calculate the relative position of the box.

Despite the fact that the courtesy amount field are located at the same place of all cheques, the size of the bounding box varies from bank to bank. To ensure that the region cropped contain the courtesy amount fully, the bounding box coordinates are taken to be the values which define the maximum area of all possible bounding boxes. Since the aspect ratio of all cheques are the same, the possible bounding box area is calculated as follows:

$$\text{Top-left coordinate}(x, y) = (0.6 \times \text{image width}, 0.4 \times \text{image height}) \quad (3.1)$$

$$\text{Top-right coordinate} = (x + \text{width of bounding box}, y) \quad (3.2)$$

where $\text{length of bounding box} = 0.5x$

$$\text{Bottom-left coordinate} = (x, y + \text{height of bounding box}) \quad (3.3)$$

where $\text{height of bounding box} = 0.4y$

The bounding box is cropped according to the coordinates provided and the resulted image will have a lot of white space surrounding the courtesy amount field. To remove this space, sliding window is used to scan the cropped area for the first black pixel from the left and define the left border of courtesy amount field from this point. The horizontal scanning continues until encounters the last black pixel on the right side of the cropped area. This point marks the right border of the courtesy amount field. The border is then updated and the new region is cropped. The new cropped region is then sent to the next step for further processing.

3-2-3 Segmentation

This step utilises OpenCV library to find the contour points of the region of black pixels, which are characters in this case. Using OpenCV function *findContours*, the points which are the edge of each character are stored in vector form. This way can be used to split each connected component from the extracted field as the boundaries of the components are traced. It effectively split disjoint numbers from the image, but could not split if two or more numbers are connected or touching each other.



Figure 3.2: Example of splitting connected components. Picture adopted from (Chaabani 2007).

For the case of connected components, no splitting procedure is used to segment the touching numerals. Instead, all two-digit connected numerals from '00' to '99' are trained so that the system will treat the connected numerals as a whole without the need of further segmentation. The database of touching digits is obtained from *Touching Digits* (2010)

3-2-4 Feature Extraction

Feeding the entire image of the digit into the classifier is not a good idea. Merely using pixel information cannot produce a classifier that is accurate enough when it is tested with other images. Prior to feature extraction, the segments are resized to 16x16 to ensure the computation time is short enough.

Directional Distance Distribution is a type of distance information between the black pixels and white pixels in the image. It calculates the distance to the closest black pixel for each pixel of the source image. It also calculate similarly for white pixels. This feature is based on the distance information computed for both black pixels and white pixels in 8 directions. The output of calculation for each pixel is a length-16 vector:

W0	W1	W2	W3	W4	W5	W6	W7	B0	B1	B2	B3	B4	B5	B6	B7
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

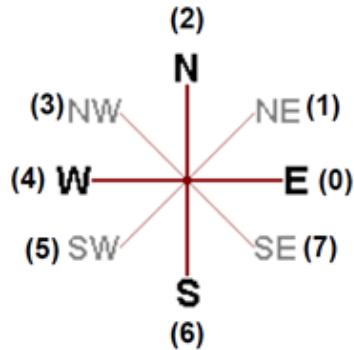


Figure 3.3: The 8-directions and its corresponding digit that indicates each direction.

The rule of calculation is simple: if the pixel is white, then set W (W0, W1, ..., W7) will be filled with 0; else, set B (B0, B1, ..., B7) will be filled with 0. Scanning and counting pixels in 8 directions will fill up the entire length-16 vector with some values. We call this process 'WB encoding'.

To illustrate how it is calculated, the following image is used: '-' indicates a white pixel, '*' indicates a black pixel.

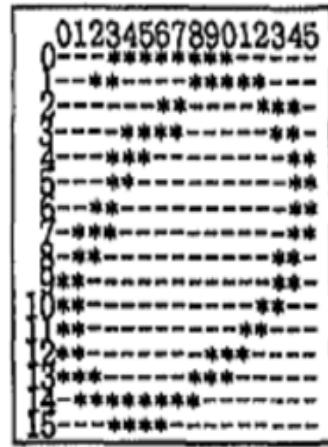


Figure 3.4: The pixels of a zero digit image in binary form.

For example, the WB encoding at pixel (column, row) = (8,2) is

0	0	0	0	0	0	0	0	4	1	1	2	1	1	11	6
---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	---

And the WB encoding at pixel (8,1) is

5	2	2	4	1	9	1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

By this calculation, the resulting array would have $16 \times 16 \times 16 = 4096$ values for just one image. This will consume a huge amount of space and further processing steps will be very slow and require a lot of memory. To reduce the dimension, the 16x16 image is divided into 4x4 grids, and for each 4x4 grid the average of each value in the length-16 vector is calculated and stored. The final dimension of the feature vector would be just 16 (each 4x4 grid) x 16 (16-bit WB encoding).

Advantage of DDD over distance transform is it treats the array as being circular when computing the distance, and also it consider both distances from black pixel to white pixel and from white pixel to black pixel. Hence the feature contains both the black/white distribution and the directional distance distribution. Computation time is short since the dominant operations are integer comparison and addition.

In addition to DDD, information on the height and width of the original segment is also used. The aspect ratio of the original segment is also used. Aspect ratio is the ratio of height to width. These features can effectively differentiate between single digits and connected numerals due to the fact that connected numerals generally have a larger width and larger aspect ratio.

3-2-5 Classification

Standard Scaling. Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data.

For instance, the objective function of a learning algorithm assumes that all features are centered around 0 and have variance in the same order. If a feature has a variance that has order of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. (Khan 1998)

This operation standardises features by scaling the values in the feature vector so that the values would fall within a specified range, say (-1,1). Let x be an element in the feature vector X and the range specified by (\min, \max) . Then the scaled x_{scaled} is computed as follows.

$$x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.4)$$

$$x_{scaled} = \frac{x_{std}}{(max - min) + min} \quad (3.5)$$

SVM. One of the most attractive properties of SVM, is that different kernel functions can be specified for the decision function. A linear hyperplane may be sufficient for linearly separable problems, however it is impossible to find a linear hyperplane for the case of nonlinear separable classes such as the Exclusive-OR (XOR) problem. For such case, some kernel function $\phi(x)$ can be used to map the feature vectors x so that they become linearly separable.

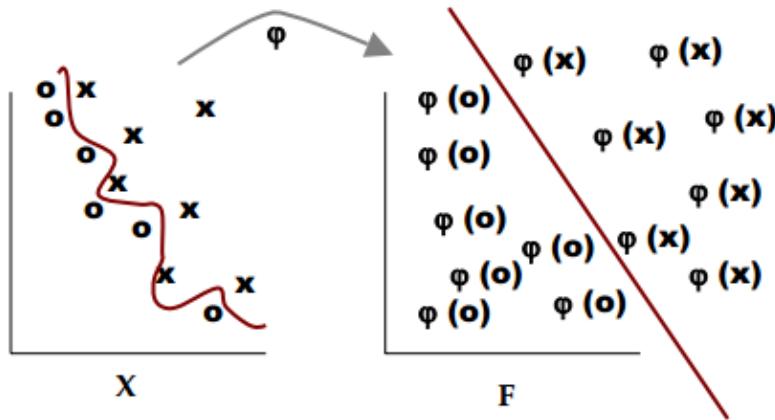


Figure 3.5: Transforming x to $\phi(x)$ to become linearly separable.

In this paper, a SVM with the Radial Basis Function (RBF) kernel is used. RBF kernel is defined as Chih-Chung & Chih-Jen (2013):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} \quad (3.6)$$

When training with RBF kernel, two parameters have to be specified: cost C and γ . A low C makes the decision surface smooth (more generalised), while a high C aims at classifying all training examples correctly (more specific). γ defines how much influence a single training example has. The larger the value of γ is, the closer other examples are to be affected. (*Support Vector Machine* 2010)

Even though SVM is a binary classifier on its own, there are developed approaches in dealing with multi-class case using one-against-one or one-against-all approach. In this paper, a function of SVM in the scikit-learn library, SVC, implement the "one-against-one" approach (Knerr et al., 1990) for multi-class classification.

For one-against-one approach, if k is the number of classes, $\frac{k(k-1)}{2}$ binary classifiers are trained and each classifier separates a pair of classes. The decision strategy is made on the basis of majority vote. Taking training of digit '1' as an example, the images of '1' are feed into the SVM as the positive samples, images of one digit at a time are fed as the negative samples, let's say, '9'. Then in the next iteration the images of '1' are trained against another digit, let's say '8'. This process continues until all $\frac{k(k-1)}{2}$ binary classifiers are trained. The disadvantage is that a relatively large number of binary classifiers has to be trained.

The reason that one-against-all approach is not selected is that this technique will results in asymmetric binary classifiers because the training is carried out with many more negative than positive samples. This becomes more serious when the number of classes is relatively large. (Sergios & Konstantinos 2006)

SVM itself does not have a measure of confidence. It is computed using algorithms and formulae proposed by Ting-Fan et al..

Classes. There are 114 classes setup for training: 10 digits (0 9), 99 connected numerals (00 99 except '11'), 'R', 'M', commas, decimal dots, junk segments. Because the number of images (samples) of each class is different, class weight is computed for each class such that the weight is inversely proportional to the number of samples in that class.

3-2-6 Post-processing

After feature vectors is being fed into the SVM and predicted output is produced, some post-processing procedure is required to check the validity of the result of recognition.

Before the output is displayed, the following criteria and rules must be met in order for the output to be valid:

- Dot cannot be located at the first position.
- The character precede dot must be a number.
- No character can precede RM.
- If the character is recognised as junk but the confidence of it being a junk is less than 0.7, then we check if it is a dot or comma or simply a junk. If probability of it being a dot is higher than of comma, we check if the next two characters are numbers. If yes, then it is a dot. Else it remains as a junk.
- No comma or dot can precede the first number in the string.
- If there is no dot and only one comma, and next three characters are not all numbers, then the comma will be replaced as dot.
- To differentiate between '1' and comma, we calculate the absolute distance between the regression line and the maximum. If the value is larger than 0.4 and

the maximum is below the regression line, then it is possibly a comma. Else it should be '1'.

However, it is difficult and sometimes impossible to differentiate between the digit '1' and comma by the image segment alone without knowing the context. To introduce context information into this recognition, the position of the maximum of each segment is checked and these maxima are used to construct a regression line. Generally a comma would locate at a position lower than other characters, hence, a possible comma fulfils the criterion that its actual maximum would be lower than the predicted maximum by the constructed regression line.

3-3 Software Implementation

Two GUI programs are created for easy visualisation and demonstration of the process described in the previous section.

3-3-1 Training and Testing of Single Digits

This program is created for the purpose of testing and analysis of SVM training for MNIST database. With a visual display of datasets, results and charts, it is easier to compare the performance of each parameter setting. There are 3 tabs in this Graphical User Interface (GUI), namely '*Training*', '*Testing*', and '*Classify*' tab.

Training Tab

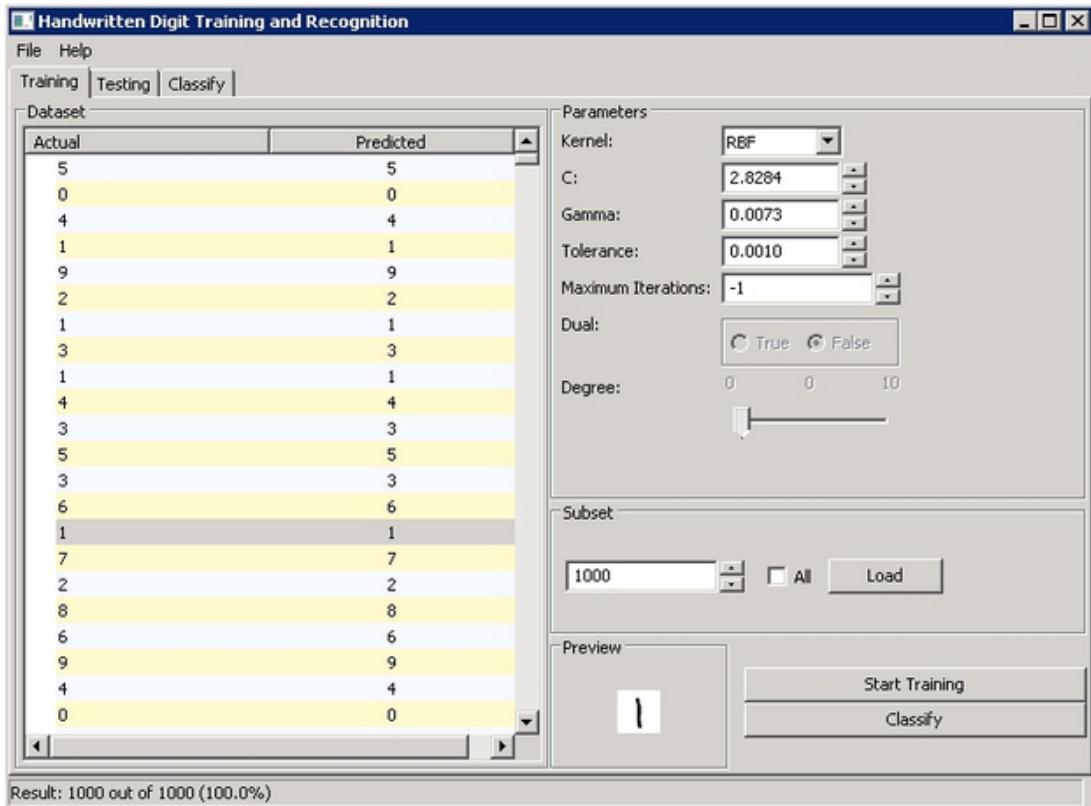


Figure 3.6: Screenshot of the *Training* tab.

The dataset field consists of a list with two columns: the actual class (digit) and the predicted class by the trained SVM. Initially, the list is empty. When the 'Load' button is clicked, the images and labels from the MNIST training set will load, according to the number of 'subset' specified. After loading, users can click on each entry in the list, and the corresponding image will show in the preview pane. To adjust parameters, choose the desired kernel, and the options appropriate to the kernel will be enabled for user control. Then, click on 'Start Training' to start extracting features of the listed digits, and training of SVM will follow. After it is done, users can click 'Classify' to test the accuracy for the training set. The result will show in the 'Predicted' column and also in the statusbar. Any misclassified digit will be marked as red.

Note: It can also load a trained SVM, by choosing File -> Load -> A trained SVM and select the appropriate pickled file. Users will still be required to click on 'Start Extracting' button before they can click 'Classify'. This is because a trained SVM will still require extracted features to work.

After performing the training, if the result is satisfying, users can save the trained SVM from the menubar as well, by clicking File -> Save -> The trained SVMâ€ and the file will be saved in the selected directory.

Testing Tab

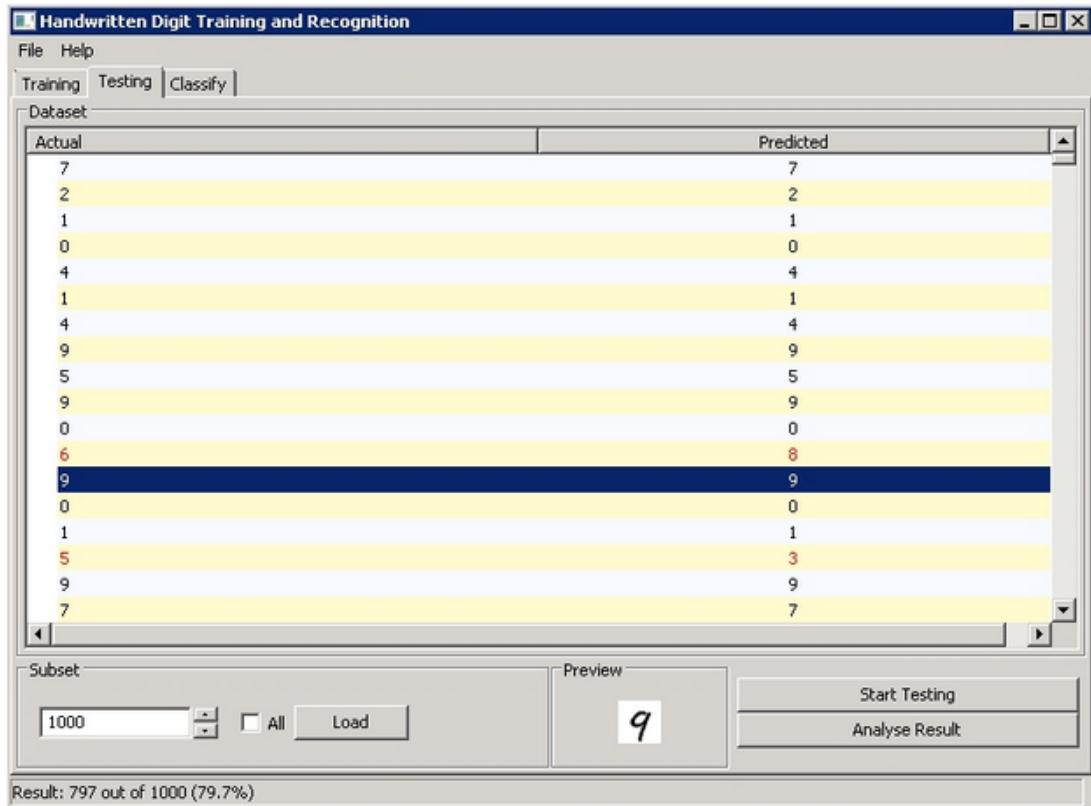


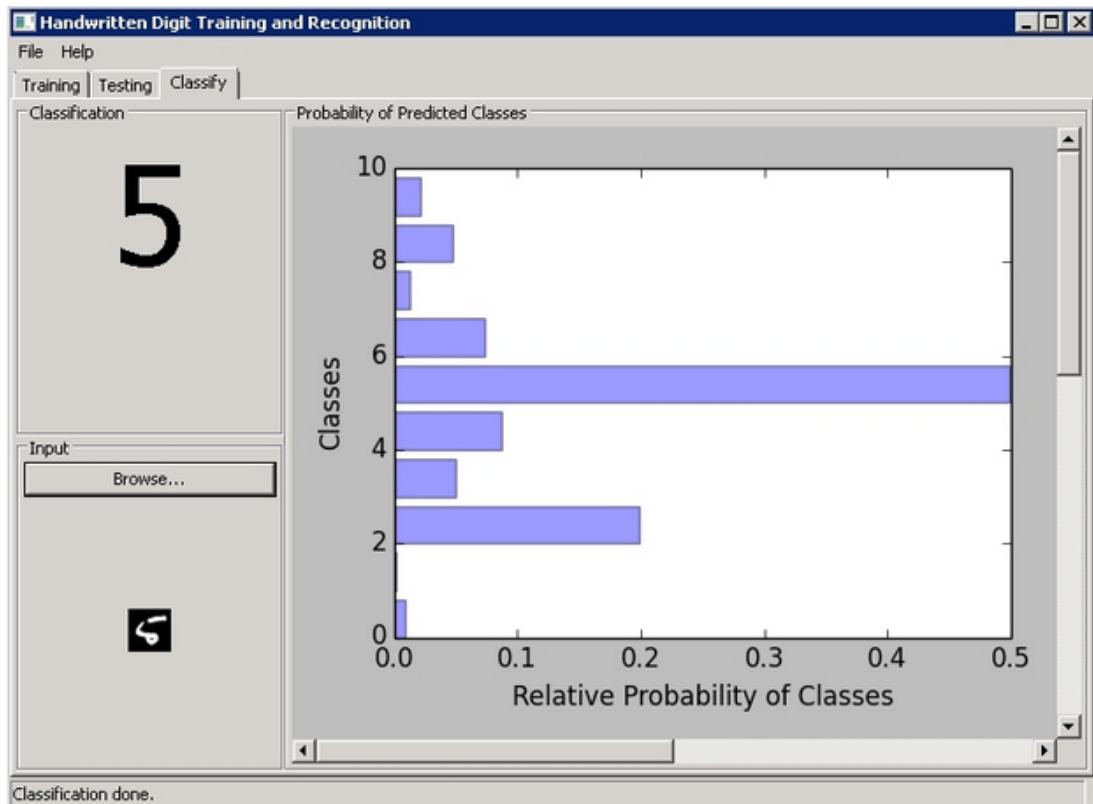
Figure 3.7: Screenshot of the *Testing* tab.

After training the SVM (or after loading a trained SVM), users can then load the MNIST testing set by clicking the 'Load' button. Similar to the training tab, clicking on each entry in the list will show the corresponding image in the preview pane. Clicking on 'Start Testing' button will extract the features from the subset, and the predicted digits will be shown in the 'Predicted' column. Any misclassified digits will be marked as red. After the result is computed and displayed in the statusbar, users can click on 'Analyse Result' to see the confusion matrix, as shown below.

		0	1	2	3	4	5	6	7	8	9
0	20	1	1	0	0	0	3	1	0	0	0
1	0	118	2	0	1	2	1	0	0	0	1
2	0	1	96	6	1	0	3	7	4	2	
3	1	0	5	81	0	5	0	1	6	2	
4	0	4	1	2	86	2	6	5	4	7	
5	0	1	0	9	1	63	7	1	7	1	
6	2	1	1	0	2	4	64	1	3	1	
7	0	0	2	0	3	2	1	77	1	6	
8	1	0	8	6	2	9	2	0	59	1	
9	1	0	0	3	14	0	0	6	5	73	

Figure 3.8: Screenshot of the *Analysis* pop-up window.

Classify Tab

Figure 3.9: Screenshot of the *Classify* tab.

This is where users can input their own digit images and see what will the SVM classify. To choose the input image, click on 'Browse...' and select a JPEG image of a single digit. Then the result will show in the Classification pane and a corresponding bar chart of probability will be plotted on the right. The longer the bar is, the more

likely the digit is to be predicted as.

3-3-2 Testing with cheque images

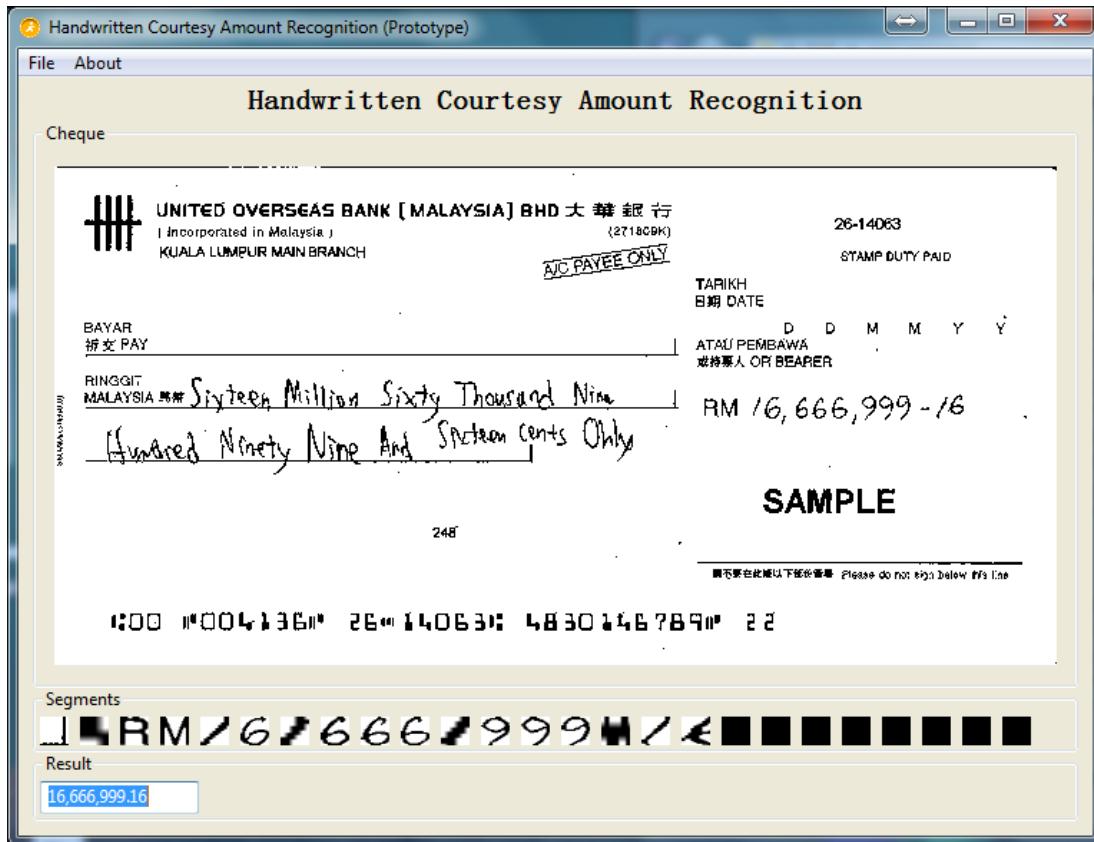


Figure 3.10: Screenshot of the *Handwritten Courtesy Amount Recognition* demonstration GUI.

This program can load a JPEG cheque image from disk and recognise the courtesy amount on it. It still cannot process grayscale images by now, so the image loaded must be in binary.

When does the recognition most likely to succeed?

- When numbers are clearly written.
- When there is no background pattern or noise.

Any noise in the image will become a segment too. It could be misclassified as other number or character.

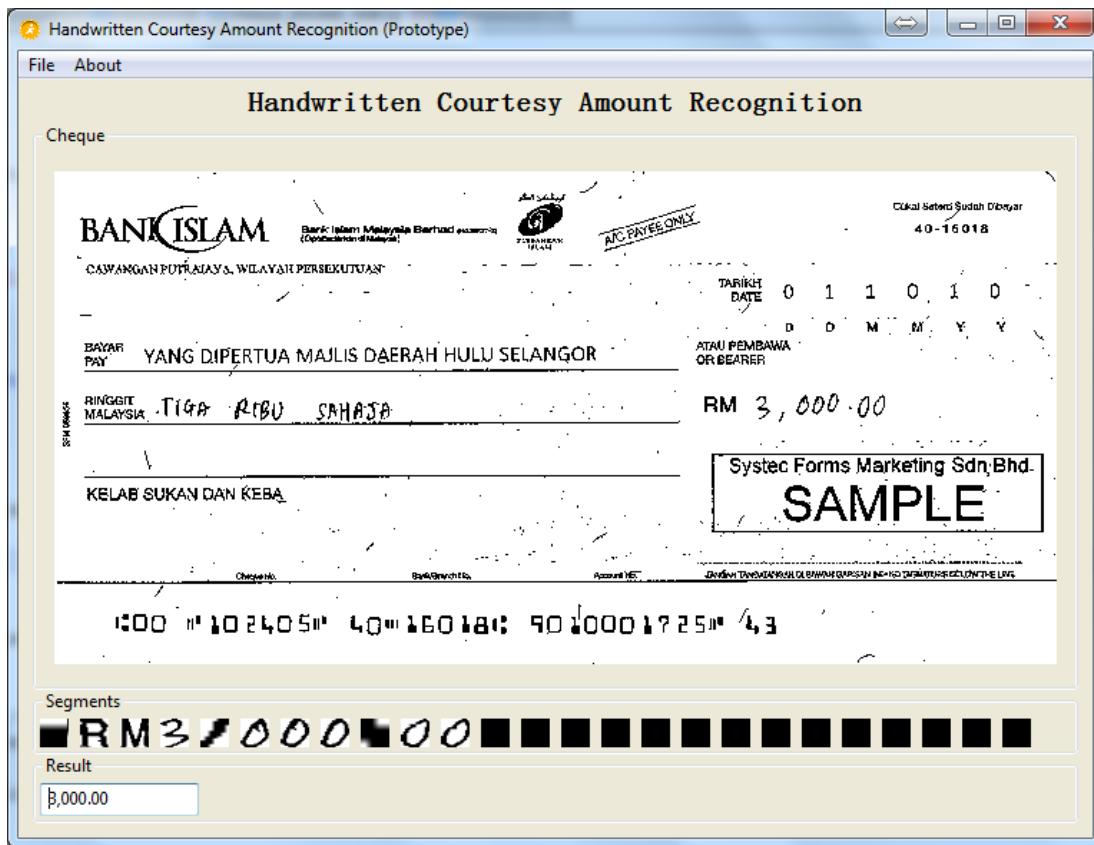


Figure 3.11: A successful example of recognition.

When does the recognition most likely to fail?

- When the decimal point is too small.

Any segment smaller than a certain size will be eliminated before feature extraction. Hence the segment will not be recognised and be displayed in the output.

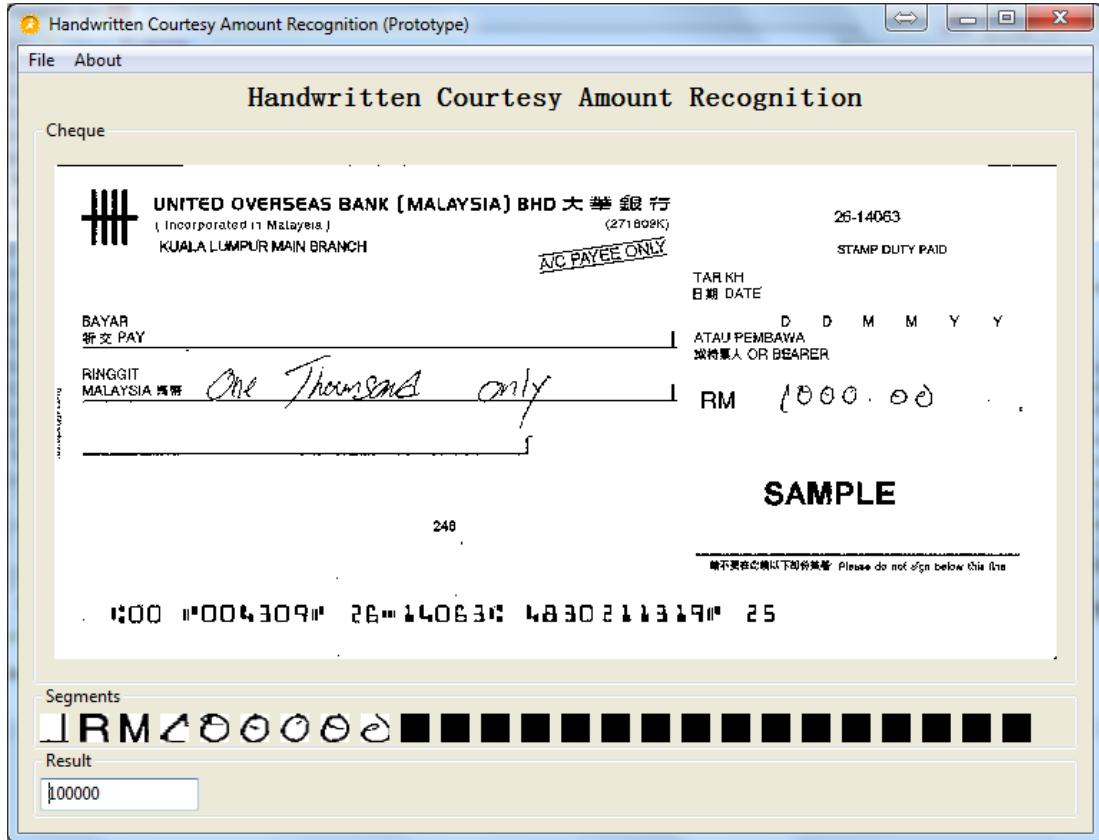


Figure 3.12: Recognition error: A decimal point is missed.

- When there are connected numerals consists of more than two digits. Only connected numerals of two-digits are trained. Any other types of connected numerals will be recognised wrongly.
- When there are many junk segments produced during the segmentation stage. In the post-processing stage, the local maximum of each segment is used to construct the linear regression line. To differentiate between comma and '1', the ratio of its distance (between the maximum and regression line) to the maximum distance (computed from all segments) is calculated. When there is junk segments, the value contributed by the junk segment will affect the result.

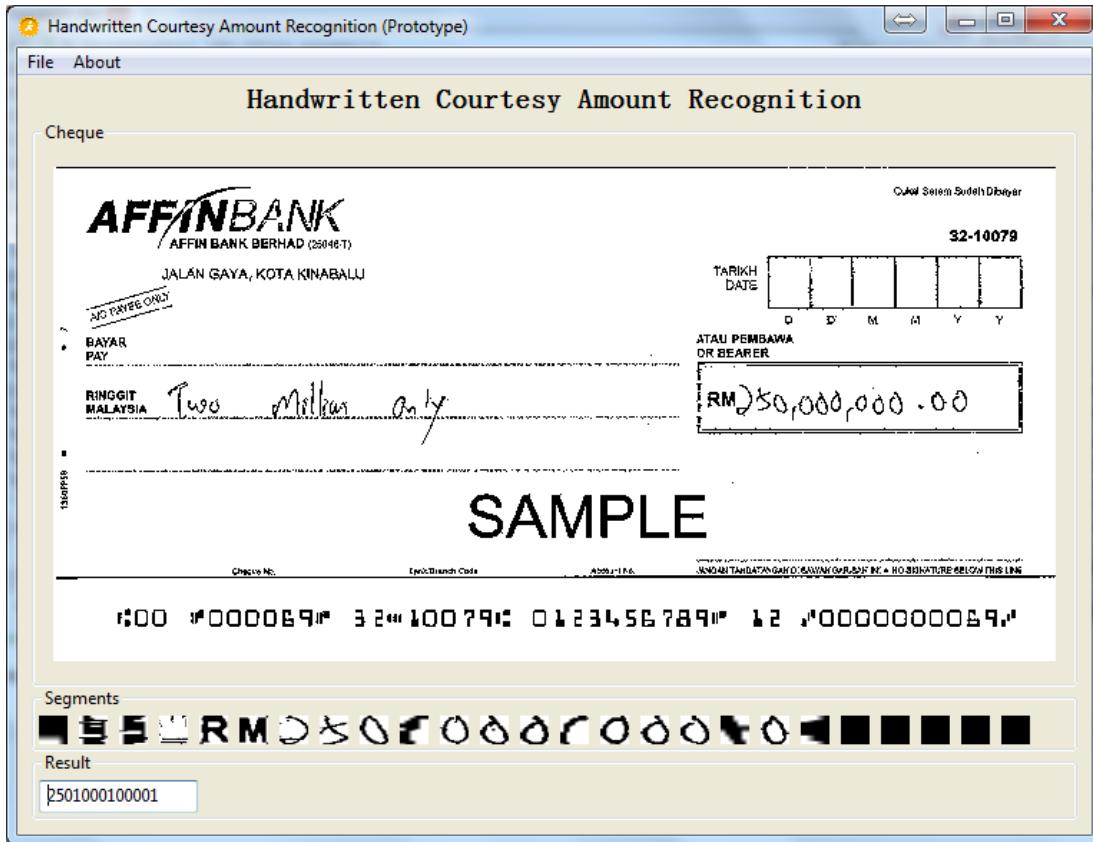


Figure 3.13: Recognition error: Classify ',' as '1'.

An error analysis on the system will be discussed in Section 3.3.3.

CHAPTER 4: EXPERIMENT AND ANALYSIS

4-1 Experiment 1

Question: What kernel and parameters of SVM is suitable?

Objective: To find the most suitable kernel and parameters over a specified range of choices.

4-1-1 Problem Statement

There is no set of parameters that always perform better than another because the choice of parameters is problem-specific. Proper choice of kernel, C and γ is critical to the SVM's performance. However, no rule or formula has been defined for what parameters would be the most suitable.

In the process of finding a good set of parameters, the usual way is to split the dataset into training and testing set and compute the classification scores on the testing set while continue to adjust the parameters until we get the best possible scores.

However, there is a risk of overfitting on the testing set because the parameters are tweaked until the classifier performs optimally. The result of evaluation then would not be reflect the generalisation performance. To solve this problem, the original dataset is split into three different sets, which the extra set is the validation set. Instead of adjusting the parameters based on the result of testing set, validation set is used to adjust the parameters. Then the final performance is tested with the testing set.

4-1-2 Methodology

This results in another issue, which is the reduction in the amount of training data, hence affecting the overall performance of the learning model. To solve this issue, k -fold cross-validation is carried out on the dataset. To do this, the dataset is split into k smaller parts, where $k - 1$ part will be used to train the classifier, and the remaining part to validate the performance. The process is repeated for different choices of parameters.

Stratified K -Fold Cross Validation

In this case, stratified k -fold cross validation is used to evaluate the performance of the classifier. Stratified k -fold produces stratified folds whereby each set contains the same percentage of samples of each class as the complete set.

Grid search with cross validation

Grid search is a method of searching over a range of parameters of the classifier systematically. During the search it is accompanied by k -fold cross validation, which is used to evaluate the performance of the choice of parameters.

In this experiment, the grid search is performed twice: the first time on a wider range of values, the second time on a more narrowed down range.

4-1-3 Experimental Setup

Dataset

The dataset used is the feature vectors of all available images of 114 classes. Features are extracted using the method as mentioned in Section 3.5. For the first attempt, the number of data used is all extracted feature vectors (151699); whereas for the second attempt, due to time constraint so to reduce the training and testing time, the number of data used is 5000 out of 151699 feature vectors.

Grid of search

The range of choices of parameters specified are as follows.

<u>Attempt 1</u>	<u>Attempt 2</u>
Kernel: Linear, RBF	Kernel: Linear, RBF
C: 1.0, 2.0, 3.0	C: 2.2, 2.5, 2.8, 3.0
γ : 0.006 0.007, 0.008, 0.009	γ : 0.00675, 0.007, 0.00725, 0.0075

4-1-4 Result and Analysis

Table below shows the top 5 set of parameters that produce the highest mean validation score and lowest standard deviation during the first attempt.

Rank	Mean Validation Score	Standard Deviation	Kernel	C	γ
1	1.000	0.000	Linear	1.0	0.006
2	1.000	0.000	RBF	1.0	0.006
3	1.000	0.000	Linear	1.0	0.007
4	1.000	0.000	RBF	1.0	0.007
5	1.000	0.000	Linear	1.0	0.008

From the result obtained the first attempt, not much information can be extracted since the scores are all the same for top 5 set of parameters. So by a rough guess the second grid of search is specified and evaluated.

Table below shows the top 5 set of parameters that produce the highest mean validation score and lowest standard deviation during the second attempt.

Rank	Mean Validation Score	Standard Deviation	Kernel	C	γ
1	0.841	0.001	RBF	2.8	0.0075
2	0.841	0.002	RBF	3.0	0.00725
3	0.840	0.001	RBF	2.5	0.00725
4	0.840	0.002	RBF	2.8	0.007
5	0.840	0.002	Linear	2.8	0.00725

Even though there is a tradeoff of validation score due to reduced dataset, it is now clearer to see the advantage of certain set of parameters over the rest. The best choice (rank 1) of parameters is to choose RBF kernel, C to be 2.8, and γ to be 0.0075. Note that in the following experiments, if it involves the use of SVM, the above mentioned parameters would be applied.

4-1-5 Conclusion

Grid search with cross validation is a rather 'fair' way to determine a good set of parameters because the risk of overfitting a testing set is greatly reduced compared to a fixed training and testing set. The final choice of parameters is RBF kernel, with C to be 2.8 and γ to be 0.0075.

4-2 Experiment 2

Question: How good is the accuracy of dropfall segmentation?

Objective: To evaluate the performance of dropfall segmentation.

4-2-1 Problem Statement

The contour tracing algorithm mentioned in Section 3.4 does the job of segmenting the extracted field into single characters, but only in the case where all characters are spaced by at least one pixel in any direction. In many cases, some digits are written in a way that they touch or overlap one another. We call this touching or connected numerals.

To deal with this case, some special method or algorithm would be needed to split the connected numerals into single digits. One of the many approaches reviewed in Section 2.2.3 is dropfall algorithm. Though dropfall algorithm is not used in the recognition system, its effectiveness is evaluated in this section.

4-2-2 Methodology

Dataset

To test the effectiveness a number of connected numerals are required. The connected numeral database contains 79,466 samples distributed into the 100 classes of touching pairs, which correspond to the possible combinations of two digits. The database is generated by Oliveira et al. (2005) and provided by The Laboratory of Vision, Robotics and Imaging (VRI). However, the class correspond to the touching pair '11' is empty due to the fact that it is impossible to write the number 11 in such manner.

Two samples were selected from each non-empty class — 198 samples were selected at random to be segmented using dropfall algorithm, as reviewed in Section 2.2.3.

Method

There are four variations of dropfall algorithm: top-left, top-right, bottom-left, bottom-right. All these variations will produce different result of segmentation.

In my implementation, all four variations are tested for each pair of connected numerals. Then from all variations, the segments that result in the highest sum of confidence value are chosen to be the final segments.

4-2-3 Result and Analysis

Among 198 connected numerals, there are 65 successful pairs of segments produced by the dropfall algorithm. The criteria of a successful segmentation is that the numbers are somehow splitted from another and can be recognised by the single-digit classifier. Figures below show some examples of successful and unsuccessful cases.

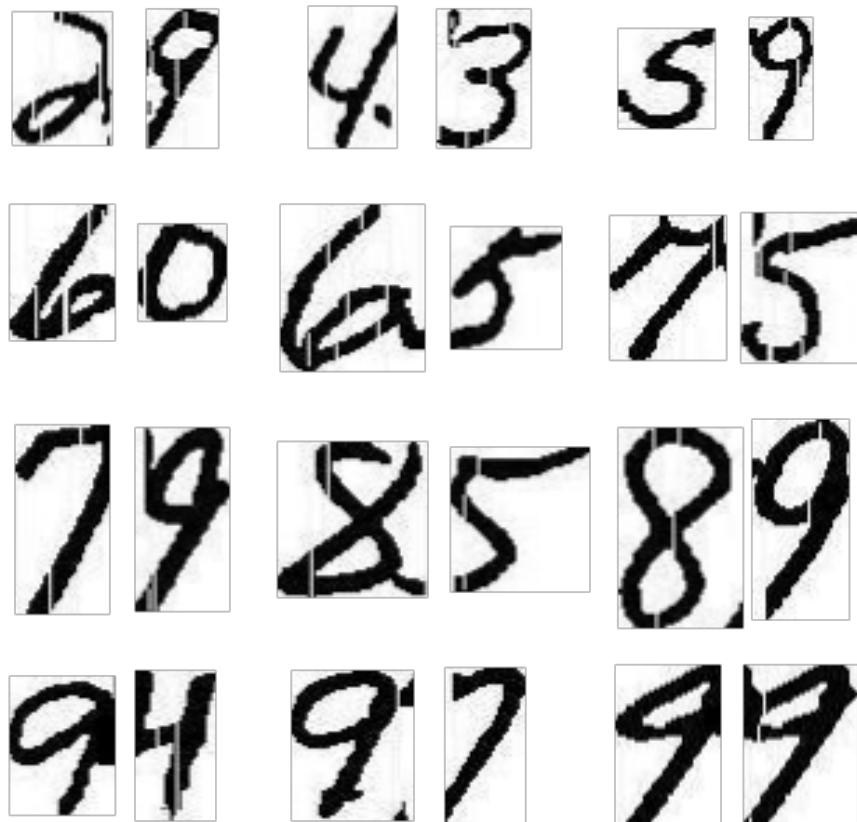


Figure 4.1: Examples of successful dropfall segmentation.

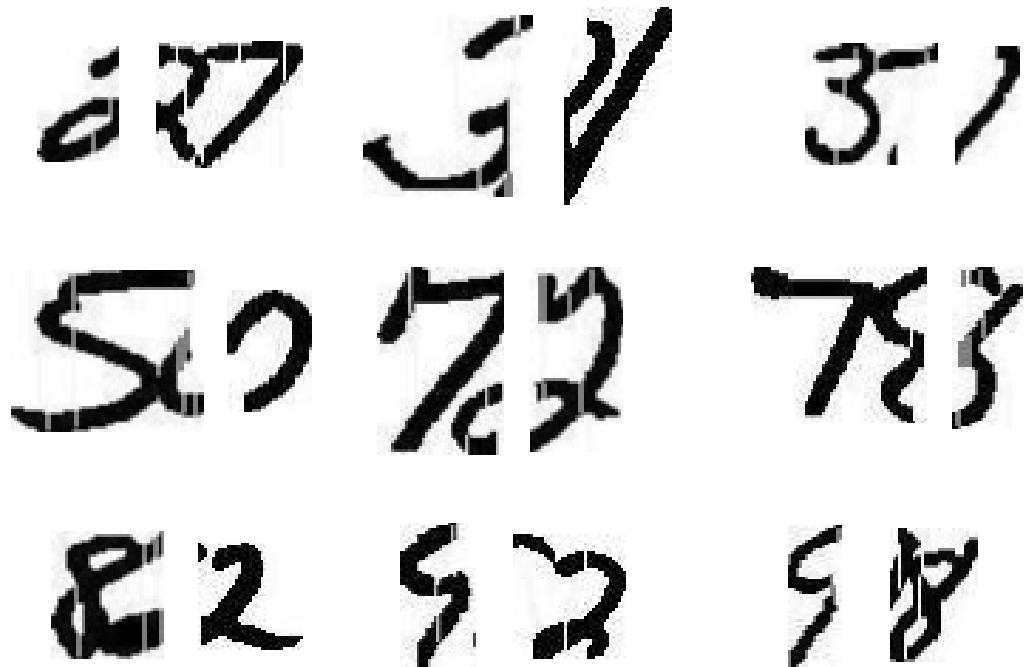


Figure 4.2: Examples of unsuccessful dropfall segmentation.

Dropfall algorithm seems to work well with connected numerals starting by '9', and works badly with connected numerals which are started by '1'. Also the segmentation is bad with numerals that are slanted. Most of the connected numerals tested in this experiment are slanted. In slanted numerals, dropfall algorithm tends to cut at a wrong point.

Besides, due to own implementation issue, the segmentation produces many broken or corroded lines and this issue is yet to be fixed.

4-2-4 Conclusion

The result of segmentation (success rate of 32.83%) is below satisfactory, and even if the dropfall segmentation perform well enough, it is challenging to determine if a segment is a connected numeral. Though dropfall algorithm can be used to segment connected numerals of more than two digits, it would take a long time to segment. Due to difficulties in implementation, this algorithm is not used in the software implementation.

4-3 Experiment 3

Question: How would training connected numerals, junk segments, currency sign, decimal points, and commas make a difference?

Objective: To compare the error analysis with and without training connected numerals, junk segments, currency sign, decimal points, and commas.

4-3-1 Problem Statement

Initially, only patterns of all possible single digits (0-9) from the MNIST dataset were trained. It performs well on single digit recognition with 98.41%, however, this is insufficient for a courtesy amount recognition system to work well because there are other characters involved such as currency sign and commas. Though in an ideal case the currency sign can be ignored by ignoring the first two segments returned by the segmentation function, but in many cases there will be junk segments which could lie in any position in the extracted field.

To deal with this issue, a possible solution is to train the SVM with samples of these junk segments, currency sign, commas and even decimal points. Once the SVM has learned to recognise these segments, then it would be easier to eliminate unwanted output in the result of recognition. Also, instead of using dropfall algorithm to segment potential connected numerals, all possible two-digit connected numerals are trained in the consideration that two-digit connected numerals occurs much more frequently than three- or four-digit connected numerals.

4-3-2 Methodology

Dataset

Connected numerals database is the same as mentioned in Section 4.2.2. The junk segments, currency sign, decimal points and commas are obtained by performing contour tracing segmentation on 500 cheque samples.

Training

As usual, the classifier is SVM. The parameters set are according to Section 4.1.4.

Testing

Each output is checked against the original cheque sample manually to determine and count the possible causes of error. The observation is recorded and plotted in the charts in the next sub-section.

4-3-3 Result and Analysis

Before the connected numerals, junk segments, currency sign, decimal points, and commas are trained, the performance of the system is evaluated by testing with 100 cheques.

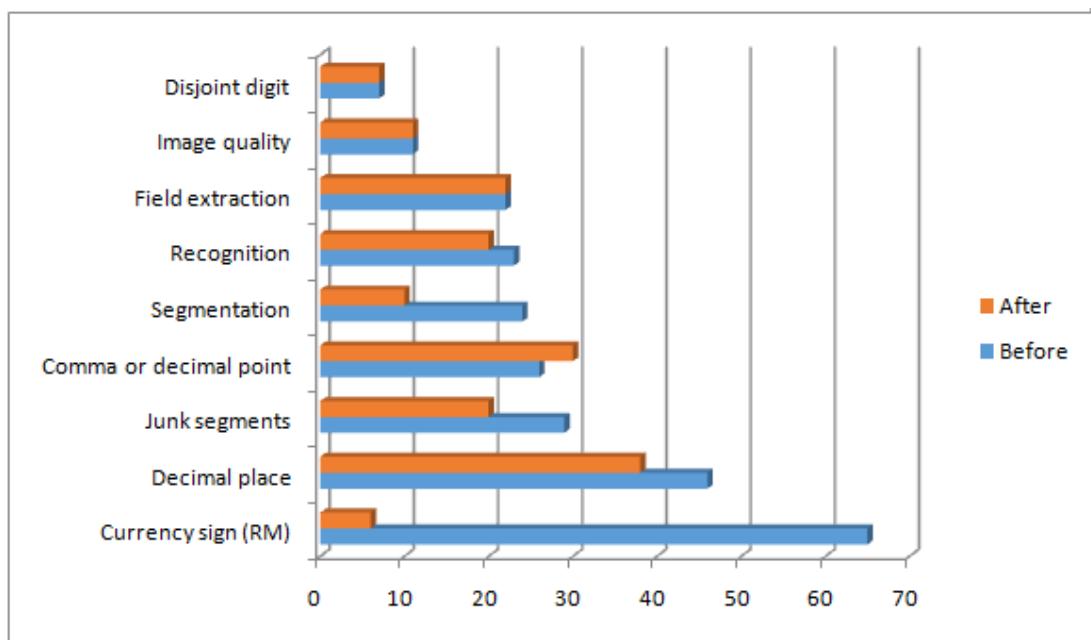


Figure 4.3: Frequency of occurrence of error on the recognition of 100 cheques before and after training.

From the bar chart, the performance of recognising currency sign is quite promising, which reduces the error caused by wrongly recognised currency sign as other digits by 90.77%. The reduction in decimal place could possibly be the result of both training of decimal dots and post-processing. Error caused by segmentation also has been reduced drastically because segmentation of two-digit connected numerals is no longer required. There is also some reduction in the error caused by junk segments, but not as much due to the huge variation in the junk segments.

However, comma or decimal point could still be recognised as '1' in many cases because comma, decimal point and '1' sometimes look very alike and difficult to differentiate without proper context information.

Also, among 100 cheques tested, there are 13 correctly recognised amount after the training. This is a huge improvement compared to the performance before the training, which only 1 amount is correctly recognised.

4-3-4 Conclusion

Allowing the classifier to learn the patterns of non-digit characters in the courtesy amount improves the overall performance of the system by reducing the error caused by the unrecognised segments.

4-4 Experiment 4

Question: How would setting proper class weights affect the performance of SVM in recognition?

Objective: To compare the performance of properly-weighted SVM with the normal even-weighted SVM.

4-4-1 Problem Statement

Leaving a dataset to have different amount of data in each class would result in an undesirable classifier that tends to assign values to the majority. To handle data imbalance, a direct way is to obtain more data for the smaller class. If it is difficult to obtain more data, some distortion or elastic transformation can be performed on the existing dataset in order to obtain more data with variations. A simpler way is to sample less data from the bigger class, but this would also reduce the learning samples for the classifier hence reduces the generalisation ability of SVM.

In problems where it is desired to give more importance to certain classes or certain individual samples, class weights are computed and defined as a parameter of SVM so that the issue of unbalanced data would be solved.

4-4-2 Methodology

Dataset

The dataset used is the set of feature vectors built from all available character images.

Stratified k -fold cross validation

Again this method is used to evaluate the performance of SVM on both training and testing set.

4-4-3 Result and Analysis

Fold	Training	Testing
1	90.49%	90.11%
2	90.65%	89.85%
3	90.56%	90.15%
Average	90.57%	90.04%

Table 4.1: Accuracy of training and testing set for 3-fold cross validation on Normal SVM

Fold	Training	Testing
1	99.96%	98.06%
2	99.97%	97.85%
3	99.97%	97.97%
Average	99.97%	97.96%

Table 4.2: Accuracy of training and testing set for 3-fold cross validation on Weighted SVM

From the result above, it is proven that training SVM with a properly defined class weights have significant improvement on its performance than the normal SVM with evenly weighted classes.

By having uneven weights for unbalanced data, the SVM assigns a greater penalty to misclassification errors related with the less likely instance (classes with less data).

In this way, though the number of instances for the minority classes is not increased, the performance of SVM is enhanced, in this case, by approximately 8%.

4-4-4 Conclusion

Having class weights defined for unbalanced data is a good idea because it would avoid the tendency of assigning to some instance from the bigger class and hence improving the performance of the classifier.

CHAPTER 5: CONCLUSION AND FUTURE IMPROVEMENTS

5-1 Summary

In this research, I utilised a combination of existing techniques that are easy to apply and flexible. For field extraction, I used the sliding window technique to detect the border black pixels in the defined region. For feature extraction, I used DDD together with height, width and aspect ratio of the segment. Then the extracted features is fed into the trained SVM for recognition. Some post-processing conditioning are applied to the result of recognition and the final output is produced.

5-2 Contributions

Images of single digits, connected numerals, other characters such as comma, dot and the currency sign, and also junk segments are used to train SVM. Training the non-number characters does help to identify the courtesy amount more accurately.

Unlike other literatures which tried to segment connected numerals, I made the classifier learn to recognise connected numerals as a whole to avoid segmentation errors. By this technique the computation time is also greatly reduced.

5-3 System Limitation

- The system can only accept input of cheque image that is binary and clear of background pattern. The system does not binarise the input image nor remove noise from the image.
- The defined region sometimes does not cover the whole courtesy amount field. This issue is yet to be solved.
- There are many junk segments produced by the segmentation algorithm. This sometimes affect the result of recognition. A more effective junk elimination

process needs to be implemented, or a better segmentation algorithm should be used.

- The existing post-processing rules is not sufficient to classify all segments correctly. Sometimes the system would misclassify '1' as ',' or vice versa.

The disadvantages of support vector machines include:

- Time of training is very long, up to 3 hours per run. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. (*Support Vector Machine* 2010)
- Cannot 'update' the classifier with more samples in future, can only retrain the classifier, which is very time consuming.

5-3-1 Future Improvement

The future research could be focused on the implementation of the following techniques.

Feature Selection

More statistical and topological features such as centroid, ink-crossing, and Fourier Descriptors as described in Section 2.2.4 can be included in the feature vector.

Artificial Neural Network (ANN)

The popularity of ANN in classification tasks is greatly due to its effectiveness and performance. Hence it is an excellent alternative of SVM in terms of time and accuracy.

Segmentation

A better segmentation technique that can cater for single and multiple digits is sliding window segmentation with Hidden Markov Model and Viterbi algorithm. By determining the best segmentation path through the search on all possible paths, and if this technique is combined with the previously defined rules of output validity, it is believed that the cheque processing process would become simpler and faster.

REFERENCES

- Agarwal, A., Hussein, K., Gupta, A. & Wang, P. S. P. (1995), ‘Detection of courtesy amount block on bank checks’, *Journal of Electronic Imaging* **5**, 214–224.
- Chaabani, I. (2007), ‘Applying domain knowledge to the recognition of handwritten zip codes’.
- Cheung, K. L. (1998), ‘A study of improvements for the winbank courtesy amount recognition system’.
- Chih-Chung, C. & Chih-Jen, L. (2013), *LIBSVM: A Library for Support Vector Machines*, National Taiwan University.
- Ciresan, D. (2008), ‘Avoiding segmentation in multi-digit numeral string recognition by combining single and two-digit classifiers trained without negative examples’, *Symbolic and Numeric Algorithms for Scientific Computing* pp. 225–230.
- Clement, C., Laurent, H. & Thierry, P. (2006), *Document Analysis Systems VII*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, chapter Segmentation-Driven Recognition Applied to Numerical Field Extraction from Handwritten Incoming Mail Documents, pp. 564–575.
- Danescu, R. (n.d.), Border tracing algorithm. Image Processing Lecture Notes, Available at: users.utcluj.ro/~rdanescu/PI-L6e.pdf.
- Das, N., Reddy, J. M., Sarkar, R., Basu, S., Kundu, M., Nasipuri, M. & Basu, D. K. (2012), ‘A statistical-topological feature combination for recognition of handwritten numerals’, *Applied Soft Computing* **12**(2), 2486–2495.
- Eduardo Gasca, A. (2007), ‘Artificial neural networks’, Available at: <http://edugi.uni-muenster.de/eduGI.LA2/downloads/02/ArtificialNeuralNetworks240506.pdf>. [Accessed 28th November 2013].
- Fabien, L., Ching Y., S. & Gerard, B. (2007), ‘A trainable feature extractor for handwritten digit recognition’, *Pattern Recognition* **40**(6), 1816–1824.

- Financial Stability and Payment Systems Report 2012* (2013), Available at: http://www.bnm.gov.my/files/publication/fsp/2012/zcp07_table_A.29.pdf. [Accessed 3rd March 2014].
- Fourier Descriptors* (2012), Available at: <http://demonstrations.wolfram.com/FourierDescriptors/>. [Accessed 14th March 2014].
- Gattal, A. & Chibani, Y. (2012), ‘Segmentation and recognition strategy of handwritten connected digits based on the oriented sliding window’, *2012 International Conference on Frontiers in Handwriting Recognition* . [pg297-301].
- Ian J., G., Yaroslav, B., Julian, I., Sacha, A. & Vinay, S. (2013), ‘Multi-digit number recognition from street view imagery using deep convolutional neural networks’, *Computer Vision and Pattern Recognition* .
- Introduction to SVM* (2011), Available at: [http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html]. [Accessed 28th November 2013].
- Khan, S. A. (1998), ‘Character segmentation heuristics for check amount verification’.
- Kurniawan, F., Rahim, M. S. M., Sholihah, N., Rakhmadi, A. & Md, D. (2011), ‘Characters segmentation of cursive handwritten words based on contour analysis and neural network validation’, *ITB Journal of Information and Communication Technology* **5**(1), 1–16.
- Lei, Y., Liu, C., Ding, X. & Fu, Q. (n.d.), ‘A recognition-based system for segmentation of touching handwritten numeral strings’, *Proceedings of the 9th Int'l Workshop on Frontiers in Handwriting Recognition* .
- Malaysians still prefer cheques says Bank Negara* (2013), Available at: <http://www.themalaymailonline.com/malaysia/article/malaysians-still-prefer-cheques-says-bank-negara>. [Accessed 3rd March 2014].
- Md Tanvir, P. & Sabri A., M. (2013), ‘Offline arabic handwritten text recognition: A survey’, *ACM Computing Surveys* **45**(2), 23:1–23:36.

- Nath, R. K. & Rastogi, M. (2012), ‘Improving various off-line techniques used for handwritten character recognition’, *International Journal of Computer Applications* **49**(18), 11–18.
- Neural Networks* (2011), Available at: http://docs.opencv.org/modules/ml/doc/neural_networks.html. [Accessed 28th November 2013].
- Oh, I.-S. & Suen, C. Y. (1997), ‘A feature for character recognition based on directional distance distributions’, *Document Analysis and Recognition* **1**, 288–292.
- Oliveira, L., Jr, A. B. & Sabourin, R. (2005), ‘Probability estimates for multi-class classification by pairwise coupling’, *8th International Conference on Document Analysis and Recognition* pp. 207–211.
- Pal, U., Belaid, A. & Choisy, C. (2001), ‘Water reservoir based approach for touching numeral segmentation’, *Document Analysis and Recognition* pp. 892–896.
- Rafael, P., Amar, G. & Patrick S.P., W. (2002), Reading courtesy amounts on handwritten paper checks, Technical report, MIT Sloan School of Management.
- Rui, M., Jie, D., Yunhua, G. & Yunyang, Y. (2009), ‘An improved drop-fall algorithm based on background analysis for handwritten digits segmentation’, *WRI Global Congress* **4**, 374–378.
- Sergios, T. & Konstantinos, K. (2006), *Pattern Recognition*, 4 edn, Elsevier Inc.
- Shah, M. S., Haque, S. A., Islam, M. R., Ali, M. A. & Hassan, M. S. (2010), ‘Automatic recognition of handwritten bangla courtesy amount on bank checks’, *International Journal of Computer Science and Network Security* **10**(12), 154–163.
- Sofiene, H. & Samia, M. (n.d.), ‘Extraction of handwritten areas from colored image of bank checks by an hybrid method’, *International Conference on Machine Intelligence*.
- Stefano, C. D., D’elia, C. & Freca, A. S. D. (2009), ‘Classifier combination by bayesian networks for handwriting recognition’, *International Journal of Pattern Recognition and Artificial Intelligence* **23**(5), 887–905.

- Support Vector Machine* (2010), Available at: <http://scikit-learn.org/stable/modules/svm.html>. [Accessed 29th November 2013].
- Ting-Fan, W., Chih-Jen, L. & Ruby C., W. (2004), ‘Probability estimates for multi-class classification by pairwise coupling’, *Journal of Machine Learning Research* **5**, 975–1005.
- Touching Digits* (2010), Available at: <http://web.inf.ufpr.br/vri/image-and-videos-databases/touching-digits>. [Accessed 29th November 2013].
- Wheelock, X. (2010), ‘Whole word handwritten numerals recognition using word-pattern statistics’.
- Y.H., T., Khalid, M., Yusof, R. & Viard-Gaudin, C. (2003), ‘Offline cursive handwriting recognition system based on hybrid markov model and neural networks’, *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation* pp. 1190–1195.
- Y.H., T., Pierre-Michel, L., Marzuki, K., Christian, V.-G. & Stefan, K. (n.d.), ‘An offline cursive handwritten word recognition system’, *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology*.

APPENDIX A: SOURCE CODE

A-0-2 Experiment 1

```
1 # grid_search.py
2
3 print(__doc__)
4
5 import numpy as np
6
7 from time import time
8 from operator import itemgetter
9 from scipy.stats import randint as sp_randint
10
11 from sklearn.preprocessing import MinMaxScaler
12 from sklearn.grid_search import GridSearchCV
13 from sklearn.pipeline import Pipeline
14 from sklearn import svm
15 from sklearn.externals import joblib
16
17 _kernel = 'rbf'
18 _c = 2.8
19 _gamma = 0.0073
20 _tol = 0.001
21 _max_iter = -1
22 _dual = False
23 _deg = 5
24
25 # get some data
26 features = np.load('features.npy')
27 labels = np.load('labels.npy')
28 X, y = features, labels.flatten()
29
30 # build a classifier
31 min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
32
33 X = min_max_scaler.fit_transform(X)
34 samp = np.random.choice(len(X), 5000)
35 X = X[samp]
36 y = y[samp]
37 class_weights = np.bincount(y)
38 d = dict()
39 for i,w in zip(range(len(class_weights)+1),class_weights):
40     d[i] = w
41 # Set weights here #
42 clf = svm.SVC(tol=_tol, max_iter=_max_iter, probability=True,
43                 class_weight=d)
43 #scaling_svm = Pipeline([('scaler', min_max_scaler), ('svm',
44 #clf)])
```

```

44
45 # Utility function to report best scores
46 def report(grid_scores, n_top=10):
47     top_scores = sorted(grid_scores, key=itemgetter(1),
48                         reverse=True) [:n_top]
49     for i, score in enumerate(top_scores):
50         print("Model_with_rank:{0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}{16}{17}{18}{19}{20}{21}{22}{23}{24}{25}{26}{27}{28}{29}{30}{31}{32}{33}{34}{35}{36}{37}{38}{39}{40}{41}{42}{43}{44}{45}{46}{47}{48}{49}{50}{51}{52}{53}{54}{55}{56}{57}{58}{59}{60}{61}{62}{63}{64}{65}{66}{67}{68}{69}{70}
51             score.mean_validation_score,
52             np.std(score.cv_validation_scores)))
53         print("Parameters:{0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}{16}{17}{18}{19}{20}{21}{22}{23}{24}{25}{26}{27}{28}{29}{30}{31}{32}{33}{34}{35}{36}{37}{38}{39}{40}{41}{42}{43}{44}{45}{46}{47}{48}{49}{50}{51}{52}{53}{54}{55}{56}{57}{58}{59}{60}{61}{62}{63}{64}{65}{66}{67}{68}{69}{70}
54         print("")
55
56 # use a full grid over all parameters
57 param_grid = {'kernel':('linear', 'rbf'),
58                 'C':[2.2, 2.5, 2.8, 3.0],
59                 'gamma':[0.00675, 0.007, 0.00725, 0.0075]}
60
61 # run grid search
62 if __name__ == '__main__':
63     print "Running_grid_search..."
64     grid_search = GridSearchCV(clf, param_grid=param_grid,
65                                n_jobs=-1, verbose=1)
65     start = time()
66     grid_search.fit(X, y)
67
68     print("GridSearchCV_took_{0} seconds for {1} candidate_"
69           "parameter_settings."
70     % (time() - start, len(grid_search.grid_scores_)))
70     report(grid_search.grid_scores_)

```

A-0-3 Experiment 2

```

1 # Experiment: Dropfall segmentation on connected numerals
2
3 from __future__ import print_function
4 log = open("c:\\dropfall_log2.txt", "w")
5
6 from functions import * # file must be in the directory
7
8 #img = Image.open('cn('+str(i)+').jpg')
9 #area = ExtractField(img) #only for cheque inputs
10
11 direct = ['top-left','top-right','bottom-left','bottom-right']
12 clf = joblib.load("clf_min_max_scaled.pkl")
13
14 for i in range(100,199):
15     print ("Processing_image_number..."+str(i+1), file=log)
16     img = cv2.imread('test_('+str(i+1)+').jpg', cv2.
17                       CV_LOAD_IMAGE_GRAYSCALE)
17     arr_conf = []

```

```

18     for k in range(4):
19         tile1, tile2 = dropfall(img, direction=direct[k])
20         try:
21             if tile1.shape[0]<10 or tile1.shape[1]<10 or tile2.
22                 shape[1]<10 or tile2.shape[0]<10:
23                 print ("Dropfall_error!", file=log)
24                 continue
25         except:
26             print ("Dropfall_error!", file=log)
27             continue
28         img_lst = []
29         img_lst.append(tile1)
30         img_lst.append(tile2)
31         #img_lst = np.array(img_lst)
32         input = FeatureExtraction(img_lst)
33         pred = clf.predict(input)
34         confidence = clf.predict_proba(input)
35         arr_conf.append([confidence[0][pred[0]]+confidence[1][
36             pred[1]]])
37         arr_conf = np.array(arr_conf)
38         index = arr_conf.argmax()
39         tile1, tile2 = dropfall(img, direction=direct[index])
40         try:
41             if tile1.shape[0]<10 or tile1.shape[1]<10 or tile2.shape
42                 [1]<10 or tile2.shape[0]<10:
43                 print ("Dropfall_error!", file=log)
44                 continue
45         except:
46             print ("Dropfall_error!", file=log)
47             continue
48         img_list = []
49         img_list.append(tile1)
50         img_list.append(tile2)
51         input = FeatureExtraction(img_list)
52         pred = clf.predict(input)
53         print ("Predicted: "+str(pred[0])+str(pred[1])+" with "
54             probability "+str(clf.predict_proba(input[0])[0][pred
55                 [0]])+" and "+str(clf.predict_proba(input[1])[0][pred
56                 [1]]), file=log)
57         if i<=10:
58             io.imsave('cn_'+str(i)+'_tile1.jpg',tile1)
59             io.imsave('cn_'+str(i)+'_tile2.jpg',tile2)
60         else:
61             io.imsave('cn_'+str(i+1)+'_tile1.jpg',tile1)
62             io.imsave('cn_'+str(i+1)+'_tile2.jpg',tile2)

```

A-0-4 Experiment 3

```

1 # Experiment: Test on 100 cheques
2
3 from functions import *
4 from skimage.io import imread, imsave

```

```

5  from PIL import Image
6
7 #from __future__ import print_function
8 #log = open("c:\\cheque_exp_log.txt", "w")
9
10 total_output = []
11 for i in range(1,101):
12     print "Processing_\"cheque_number\"", str(i), "..."
13     chq = Image.open('chq_('+str(i)+').jpeg')
14     img = ExtractField(chq)
15     img = pil_to_cv(img)
16
17     height, width = img.shape
18
19     img_lst, mean_height, mean_width, y_arr = Segmentation(img)
20     datalr_y = [[y] for y in y_arr]
21     datalr_X = [[x] for x in range(1,len(img_lst)+1)]
22
23     datalr_X = np.array(datalr_X)
24     datalr_y = np.array(datalr_y)
25     from sklearn.linear_model import LinearRegression
26
27     regr = LinearRegression()
28     regr.fit(datalr_X, datalr_y)
29
30     clf = joblib.load("scaling_svm_overall.pkl")
31     input = FeatureExtraction(img_lst)
32     pred = clf.predict(input)
33     confidence = clf.predict_proba(input)
34
35     dists = regr.predict(datalr_y)-datalr_y
36     dists = abs(dists)
37     max_dist = dists.max()
38
39     output = []
40     for item in pred:
41         if item<=9:
42             output.append(item)
43         if item>=10 and item<=19:
44             output.append('0'+str(item-10))
45         if item>=20 and item<=109:
46             if item==21:
47                 output.append(-1)
48                 #print "Junk"
49             else:
50                 output.append(str(item-10))
51         if item>=110 and item<=113:
52             if item==110:
53                 output.append(",")
54             if item==111:
55                 output.append(".")

```

```

56     if item==112:
57         output.append(-1)
58         #print "R"
59     if item==113:
60         output.append(-1)
61         #print "M"
62
63     i=0
64
65     for dist in dists:
66         height, width = img_lst[i].shape
67         if i==0 and pred[i]==111:
68             #print "Pos", i, ": probably a junk"
69             output[i] = -1
70         if i>0 and pred[i]==111 and pred[i-1]>=110:
71             #print "Pos", i, ": probably a junk"
72             output[i] = -1
73         if i>0 and i<pred.shape[0]-2 and pred[i]<110 and pred[i+1]==112 and pred[i+2]==113:
74             output[i] = -1
75         if pred[i]==21:
76             if confidence[i][21]<0.7:
77                 # check if it's a dot or comma
78                 if confidence[i][111] > confidence[i][110]:
79                     # if prob of it being a dot is higher than of
80                     # comma
81                     if i<pred.shape[0]-2 and pred[i+1]!=21 and pred[i+1]<110 and pred[i+2]!=21 and pred[i+2]<110:
82                         #print "Pos", i, ": probably a dot"
83                         output[i] = '.'
84                     if i>0 and pred[i-1]<110: # extra condition
85                         #print "Pos", i, ": probably a junk due to
86                         # non-number before dot"
87                         output[i] = -1
88         else:
89             if dist>0.2*max_dist:
90                 #print "Pos", i, ": probably being a comma"
91                 output[i] = ','
92             else:
93                 #print "Pos", i, ": probably being a 1"
94                 output[i] = 1
95         if pred[i]==1 or pred[i]==110:
96             if dist>0.2*max_dist:
97                 #print "Pos", i, ": probably being a comma"
98                 output[i] = ','
99             else:
100                 #print "Pos", i, ": probably being a 1"
101                 output[i] = 1
102
103     i+=1

```

```

102     output = filter(lambda a: a != -1, output)
103     index_of_first_number = -1
104     for k in range(len(output)):
105         if output[k]<100 and output[k]!=-1:
106             index_of_first_number = k
107             break
108     for n in range(index_of_first_number):
109         if output[n]==',' or output[n]=='.':
110             output[n] = -1
111
112     output = filter(lambda a: a != -1, output)
113     cnt_comma = 0
114     cnt_dot = 0
115     m = -1
116     for k in range(len(output)):
117         if output[k]==',':
118             cnt_comma += 1
119             m = k
120         if output[k]=='.':
121             cnt_dot += 1
122         if cnt_comma==1 and cnt_dot==0:
123             output[m] = '.'
124     total_output.append(output)
125 print "Completed."

```

A-0-5 Experiment 4

```

1 # train_dataset.py
2
3 from sklearn.preprocessing import MinMaxScaler
4 from sklearn.cross_validation import StratifiedKFold
5 import numpy as np
6 import time
7 from sklearn.pipeline import Pipeline
8 from sklearn import svm
9 from sklearn.externals import joblib
10
11 _kernel = 'rbf'
12 _c = 2.8
13 _gamma = 0.0073
14 _tol = 0.001
15 _max_iter = -1
16 _dual = False
17 _deg = 5
18
19 print "Loading_training_set..."
20 features = np.load('features.npy')
21 labels = np.load('labels.npy')
22
23 start_run = time.time()
24 min_max_scaler = MinMaxScaler()
25

```

```

26 class_weights = np.bincount(labels.flatten())
27 d = dict()
28 for i,w in zip(range(len(class_weights)+1),class_weights):
29     d[i] = w
30 # Set weights here #
31 if _kernel=='linear':
32     clf = svm.LinearSVC(C=_c, dual=_dual, tol=_tol)
33 elif _kernel=='poly':
34     clf = svm.SVC(kernel='poly',C=_c, gamma=_gamma, tol=_tol,
35                     max_iter=_max_iter, degree=_deg, probability=True)
36 elif _kernel=='rbf':
37     clf = svm.SVC(kernel='rbf', C=_c, gamma=_gamma, tol=_tol,
38                     max_iter=_max_iter, probability=True)
39 elif _kernel=='sigmoid':
40     clf = svm.SVC(kernel='sigmoid', C=_c, gamma=_gamma, tol=_tol,
41                     max_iter=_max_iter, probability=True)
42
43 print "Splitting_training_and_testing_set_using_Stratified_K-
44 fold..."
45 skf = StratifiedKFold(labels.flatten(), n_folds=3)
46 k=1
47 for train_index, test_index in skf:
48     features_train, features_test = features[train_index],
49         features[test_index]
50     labels_train, labels_test = labels[train_index], labels[
51         test_index]
52
53 print "Training_dataset..."
54 scaling_svm = Pipeline([('scaler', min_max_scaler), ("svm",
55         clf)])
56 scaling_svm.fit(features_train, labels_train)
57 joblib.dump(scaling_svm, 'scaling_svm_fold'+str(k)+'.pkl')
58 end_run = time.time()
59 print "Training_completed."
60 print "Fold_no."+str(k)+": Time elapsed", round(end_run-
61 start_run, 4), "seconds"
62
63 print "Testing_dataset..."
64 labels_train_predicted = scaling_svm.predict(features_train
65 )
66 correct = 0
67 for i in range(len(labels_train)):
68     if labels_train_predicted[i]==labels_train[i]:
69         correct+=1
70 print "Accuracy:", str(correct/float(len(labels_train)))
71 np.save('labels_train_predicted_fold'+str(k)+'.npy',
72         labels_train_predicted)
73 labels_test_predicted = scaling_svm.predict(features_test)
74 correct = 0
75 for i in range(len(labels_test)):
76     if labels_test_predicted[i]==labels_test[i]:

```

```

67         correct+=1
68     print "Accuracy:", str(correct/float(len(labels_test)))
69     np.save('labels_train_predicted_fold'+str(k)+'.npy',
70             labels_train_predicted)
70     k+=1

```

A-0-6 Handwritten Courtesy Amount Recognition System GUI

```

1  #!/usr/bin/env python
2  # -*- coding: CP936 -*-
3  #
4  # generated by wxGlade 0.6.8 (standalone edition) on Fri Nov
5  # 22 10:50:18 2013
6  #
7  import wx
8  import os
9  from functions import *
10 from cheque_process_neat import *
11
12 # begin wxGlade: dependencies
13 import gettext
14 # end wxGlade
15
16 # begin wxGlade: extracode
17 # end wxGlade
18
19
20 class MyFrame(wx.Frame):
21     def __init__(self, *args, **kwds):
22         # begin wxGlade: MyFrame.__init__
23         kwds["style"] = wx.DEFAULT_FRAME_STYLE
24         wx.Frame.__init__(self, *args, **kwds)
25
26         self.output = _("")
27
28         # Menu Bar
29         self.frame_1_menubar = wx.MenuBar()
30         wxglade_tmp_menu = wx.Menu()
31         self.open = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _
32                               ("Open_Cheque.."), "", wx.ITEM_NORMAL)
33         wxglade_tmp_menu.AppendItem(self.open)
34         wxglade_tmp_menu.AppendSeparator()
35         self.Exit = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _
36                               ("Exit"), "", wx.ITEM_NORMAL)
37         wxglade_tmp_menu.AppendItem(self.Exit)
38         self.frame_1_menubar.Append(wxglade_tmp_menu, _("File"))
39
40         wxglade_tmp_menu = wx.Menu()
41         self.author = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _
42                               ("Author_Information"), "", wx.ITEM_NORMAL)
43         wxglade_tmp_menu.AppendItem(self.author)

```

```

40         self.frame_1_menuBar.Append(wxglade_tmp_menu, _("About
41             "))
42             self.SetMenuBar(self.frame_1_menuBar)
43             # Menu Bar end
44             self.label_title = wx.StaticText(self, wx.ID_ANY, _(""
45                 "Handwritten_Courtesy_Amount_Recognition"), style=wx
46                 .ALIGN_CENTRE)
47             self.MaxImageSize = 700
48             Img = wx.Image("C:\Users\YongShean\Desktop\FYP\cars_
49                 corrected\chq_(1).jpeg", wx.BITMAP_TYPE_ANY)
50             #Img = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap
51                 (700, 700))
52             # scale the image, preserving the aspect ratio
53             W = Img.GetWidth()
54             H = Img.GetHeight()
55             if W > H:
56                 NewW = self.MaxImageSize
57                 NewH = self.MaxImageSize * H / W
58             else:
59                 NewH = self.MaxImageSize
60                 NewW = self.MaxImageSize * W / H
61             Img = Img.Scale(NewW, NewH)
62             self.bitmap_1 = wx.StaticBitmap(self, wx.ID_ANY, wx.
63                 BitmapFromImage(Img))
64             self.sizer_4_staticbox = wx.StaticBox(self, wx.ID_ANY,
65                _("Cheque"))
66             self.bitmap_2 = wx.StaticBitmap(self, wx.ID_ANY, wx.
67                 EmptyBitmap(20, 20))
68             #self.bitmap_3 = wx.StaticBitmap(self, wx.ID_ANY, wx.
69                 Bitmap("C:\\Documents and Settings\\YongShean\\My
70                 Documents\\img 3.jpg", wx.BITMAP_TYPE_ANY))
71             self.bitmap_3 = wx.StaticBitmap(self, wx.ID_ANY, wx.
72                 EmptyBitmap(20, 20))
73             self.bitmap_4 = wx.StaticBitmap(self, wx.ID_ANY, wx.
74                 EmptyBitmap(20, 20))
75             self.bitmap_5 = wx.StaticBitmap(self, wx.ID_ANY, wx.
76                 EmptyBitmap(20, 20))
77             self.bitmap_6 = wx.StaticBitmap(self, wx.ID_ANY, wx.
78                 EmptyBitmap(20, 20))
79             self.bitmap_7 = wx.StaticBitmap(self, wx.ID_ANY, wx.
80                 EmptyBitmap(20, 20))
81             self.bitmap_8 = wx.StaticBitmap(self, wx.ID_ANY, wx.
82                 EmptyBitmap(20, 20))
83             self.bitmap_9 = wx.StaticBitmap(self, wx.ID_ANY, wx.
84                 EmptyBitmap(20, 20))
85             self.bitmap_10 = wx.StaticBitmap(self, wx.ID_ANY, wx.
86                 EmptyBitmap(20, 20))
87             self.bitmap_11 = wx.StaticBitmap(self, wx.ID_ANY, wx.
88                 EmptyBitmap(20, 20))
89             self.bitmap_12 = wx.StaticBitmap(self, wx.ID_ANY, wx.
90                 EmptyBitmap(20, 20))

```

```
71     self.bitmap_13 = wx.StaticBitmap(self, wx.ID_ANY, wx.
72         EmptyBitmap(20, 20))
73     self.bitmap_14 = wx.StaticBitmap(self, wx.ID_ANY, wx.
74         EmptyBitmap(20, 20))
75     self.bitmap_15 = wx.StaticBitmap(self, wx.ID_ANY, wx.
76         EmptyBitmap(20, 20))
77     self.bitmap_16 = wx.StaticBitmap(self, wx.ID_ANY, wx.
78         EmptyBitmap(20, 20))
79     self.bitmap_17 = wx.StaticBitmap(self, wx.ID_ANY, wx.
80         EmptyBitmap(20, 20))
81     self.bitmap_18 = wx.StaticBitmap(self, wx.ID_ANY, wx.
82         EmptyBitmap(20, 20))
83     self.bitmap_19 = wx.StaticBitmap(self, wx.ID_ANY, wx.
84         EmptyBitmap(20, 20))
85     self.bitmap_20 = wx.StaticBitmap(self, wx.ID_ANY, wx.
86         EmptyBitmap(20, 20))
87     self.bitmap_21 = wx.StaticBitmap(self, wx.ID_ANY, wx.
88         EmptyBitmap(20, 20))
89     self.bitmap_22 = wx.StaticBitmap(self, wx.ID_ANY, wx.
90         EmptyBitmap(20, 20))
91     self.bitmap_23 = wx.StaticBitmap(self, wx.ID_ANY, wx.
92         EmptyBitmap(20, 20))
93     self.bitmap_24 = wx.StaticBitmap(self, wx.ID_ANY, wx.
94         EmptyBitmap(20, 20))
95     self.bitmap_25 = wx.StaticBitmap(self, wx.ID_ANY, wx.
96         EmptyBitmap(20, 20))
97     self.bitmap_26 = wx.StaticBitmap(self, wx.ID_ANY, wx.
98         EmptyBitmap(20, 20))
99     self.sizer_3_staticbox = wx.StaticBox(self, wx.ID_ANY,
100         _("Segments"))
101     self.text_ctrl_1 = wx.TextCtrl(self, wx.ID_ANY, self.
102         output)
103     self.sizer_2_staticbox = wx.StaticBox(self, wx.ID_ANY,
104        _("Result"))

105     self.__set_properties()
106     self.__do_layout()
107     # end wxGlade

108     self.Bind(wx.EVT_MENU, self.OnOpen, self.open)
109     self.Bind(wx.EVT_MENU, self.OnExit, self.Exit)
110     self.Bind(wx.EVT_MENU, self.OnAbout, self.author)

111     def __set_properties(self):
112         # begin wxGlade: MyFrame.__set_properties
113         self.SetTitle(_("Handwritten_Courtesy_Amount_"
114             "Recognition_(Prototype)"))
115         _icon = wx.EmptyIcon()
116         _icon.CopyFromBitmap(wx.Bitmap("icon.ico", wx.
117             BITMAP_TYPE_ANY))
118         self.SetIcon(_icon)
```

```
103     self.SetBackgroundColour(wx.Colour(236, 233, 216))
104     self.label_title.SetFont(wx.Font(15, wx.DEFAULT, wx.
105         NORMAL, wx.BOLD, 0, ""))
106     self.bitmap_1.SetMinSize((700, 350))
107     # end wxGlade
108
109     def __do_layout(self):
110         # begin wxGlade: MyFrame.__do_layout
111         sizer_1 = wx.BoxSizer(wx.VERTICAL)
112         self.sizer_2_staticbox.Lower()
113         sizer_2 = wx.StaticBoxSizer(self.sizer_2_staticbox, wx
114             .HORIZONTAL)
115         self.sizer_3_staticbox.Lower()
116         sizer_3 = wx.StaticBoxSizer(self.sizer_3_staticbox, wx
117             .HORIZONTAL)
118         grid_sizer_1 = wx.GridSizer(1, 25, 5, 0)
119         self.sizer_4_staticbox.Lower()
120         sizer_4 = wx.StaticBoxSizer(self.sizer_4_staticbox, wx
121             .VERTICAL)
122         sizer_1.Add(self.label_title, 0, wx.ALL | wx.
123             ALIGN_CENTER_HORIZONTAL, 5)
124         sizer_4.Add(self.bitmap_1, 0, wx.ALL | wx.EXPAND | wx.
125             ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
126             10)
127         sizer_1.Add(sizer_4, 0, wx.LEFT | wx.RIGHT | wx.EXPAND
128             , 10)
129         grid_sizer_1.Add(self.bitmap_2, 0, wx.
130             ALIGN_CENTER_VERTICAL, 0)
131         grid_sizer_1.Add(self.bitmap_3, 0, wx.
132             ALIGN_CENTER_VERTICAL, 0)
133         grid_sizer_1.Add(self.bitmap_4, 0, wx.
134             ALIGN_CENTER_VERTICAL, 0)
135         grid_sizer_1.Add(self.bitmap_5, 0, wx.
136             ALIGN_CENTER_VERTICAL, 0)
137         grid_sizer_1.Add(self.bitmap_6, 0, wx.
138             ALIGN_CENTER_VERTICAL, 0)
139         grid_sizer_1.Add(self.bitmap_7, 0, wx.
140             ALIGN_CENTER_VERTICAL, 0)
141         grid_sizer_1.Add(self.bitmap_8, 0, wx.
142             ALIGN_CENTER_VERTICAL, 0)
143         grid_sizer_1.Add(self.bitmap_9, 0, wx.
144             ALIGN_CENTER_VERTICAL, 0)
145         grid_sizer_1.Add(self.bitmap_10, 0, wx.
146             ALIGN_CENTER_VERTICAL, 0)
147         grid_sizer_1.Add(self.bitmap_11, 0, wx.
148             ALIGN_CENTER_VERTICAL, 0)
149         grid_sizer_1.Add(self.bitmap_12, 0, wx.
150             ALIGN_CENTER_VERTICAL, 0)
151         grid_sizer_1.Add(self.bitmap_13, 0, wx.
152             ALIGN_CENTER_VERTICAL, 0)
```

```
133     grid_sizer_1.Add(self.bitmap_14, 0, wx.
134                     ALIGN_CENTER_VERTICAL, 0)
135     grid_sizer_1.Add(self.bitmap_15, 0, wx.
136                     ALIGN_CENTER_VERTICAL, 0)
137     grid_sizer_1.Add(self.bitmap_16, 0, wx.
138                     ALIGN_CENTER_VERTICAL, 0)
139     grid_sizer_1.Add(self.bitmap_17, 0, wx.
140                     ALIGN_CENTER_VERTICAL, 0)
141     grid_sizer_1.Add(self.bitmap_18, 0, wx.
142                     ALIGN_CENTER_VERTICAL, 0)
143     grid_sizer_1.Add(self.bitmap_19, 0, wx.
144                     ALIGN_CENTER_VERTICAL, 0)
145     grid_sizer_1.Add(self.bitmap_20, 0, wx.
146                     ALIGN_CENTER_VERTICAL, 0)
147     grid_sizer_1.Add(self.bitmap_21, 0, wx.
148                     ALIGN_CENTER_VERTICAL, 0)
149     grid_sizer_1.Add(self.bitmap_22, 0, wx.
150                     ALIGN_CENTER_VERTICAL, 0)
151     grid_sizer_1.Add(self.bitmap_23, 0, wx.
152                     ALIGN_CENTER_VERTICAL, 0)
153     grid_sizer_1.Add(self.bitmap_24, 0, wx.
154                     ALIGN_CENTER_VERTICAL, 0)
155     grid_sizer_1.Add(self.bitmap_25, 0, wx.
156                     ALIGN_CENTER_VERTICAL, 0)
157     grid_sizer_1.Add(self.bitmap_26, 0, wx.
158                     ALIGN_CENTER_VERTICAL, 0)
159     sizer_3.Add(grid_sizer_1, 1, wx.EXPAND | wx.
160                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
161                 0)
162     sizer_1.Add(sizer_3, 0, wx.LEFT | wx.RIGHT | wx.EXPAND
163                 , 10)
164     sizer_2.Add(self.text_ctrl_1, 0, 0, 0)
165     sizer_1.Add(sizer_2, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM
166                 | wx.EXPAND, 10)
167     self.SetSizer(sizer_1)
168     sizer_1.Fit(self)
169     self.Layout()
170     # end wxGlade
171
172     def GetBitmap(self, npimage):
173         from scipy.misc import imresize
174         npimage = imresize(npimage, (20,20))
175         height, width = npimage.shape
176         image = wx.EmptyImage(width,height)
177         pilimage = Image.fromarray(npimage)
178         rgb = pilimage.convert('RGB')
179         image.SetData(rgb.tostring())
180         wxBitmap = wx.BitmapFromImage(image)
181         return wxBitmap
```

```

166 def OnOpen(self, event): # wxGlade: MainFrame.<
167     event_handler>
168     self.bitmap_2.SetBitmap(wx.EmptyBitmap(20, 20))
169     self.bitmap_3.SetBitmap(wx.EmptyBitmap(20, 20))
170     self.bitmap_4.SetBitmap(wx.EmptyBitmap(20, 20))
171     self.bitmap_5.SetBitmap(wx.EmptyBitmap(20, 20))
172     self.bitmap_6.SetBitmap(wx.EmptyBitmap(20, 20))
173     self.bitmap_7.SetBitmap(wx.EmptyBitmap(20, 20))
174     self.bitmap_8.SetBitmap(wx.EmptyBitmap(20, 20))
175     self.bitmap_9.SetBitmap(wx.EmptyBitmap(20, 20))
176     self.bitmap_10.SetBitmap(wx.EmptyBitmap(20, 20))
177     self.bitmap_11.SetBitmap(wx.EmptyBitmap(20, 20))
178     self.bitmap_12.SetBitmap(wx.EmptyBitmap(20, 20))
179     self.bitmap_13.SetBitmap(wx.EmptyBitmap(20, 20))
180     self.bitmap_14.SetBitmap(wx.EmptyBitmap(20, 20))
181     self.bitmap_15.SetBitmap(wx.EmptyBitmap(20, 20))
182     self.bitmap_16.SetBitmap(wx.EmptyBitmap(20, 20))
183     self.bitmap_17.SetBitmap(wx.EmptyBitmap(20, 20))
184     self.bitmap_18.SetBitmap(wx.EmptyBitmap(20, 20))
185     self.bitmap_19.SetBitmap(wx.EmptyBitmap(20, 20))
186     self.bitmap_20.SetBitmap(wx.EmptyBitmap(20, 20))
187     self.bitmap_21.SetBitmap(wx.EmptyBitmap(20, 20))
188     self.bitmap_22.SetBitmap(wx.EmptyBitmap(20, 20))
189     self.bitmap_23.SetBitmap(wx.EmptyBitmap(20, 20))
190     self.bitmap_24.SetBitmap(wx.EmptyBitmap(20, 20))
191     self.bitmap_25.SetBitmap(wx.EmptyBitmap(20, 20))
192     self.bitmap_26.SetBitmap(wx.EmptyBitmap(20, 20))
193     wildcard = "JPQ_file_(*.jpg)|*.jpg|\JPEG_file_(*.jpeg"
194     wildcard += "|*.jpeg"
195     dialog = wx.FileDialog(None, "Choose_a_file", os.
196                           getcwd(),
197                           "", wildcard, wx.OPEN)
198     if dialog.ShowModal() == wx.ID_OK:
199         cheque = dialog.GetPath()
200
201         Img = wx.Image(cheque, wx.BITMAP_TYPE_ANY)
202         # scale the image, preserving the aspect ratio
203         W = Img.GetWidth()
204         H = Img.GetHeight()
205         if W > H:
206             NewW = self.MaxImageSize
207             NewH = self.MaxImageSize * H / W
208         else:
209             NewH = self.MaxImageSize
210             NewW = self.MaxImageSize * W / H
211         Img = Img.Scale(NewW, NewH)
212         self.bitmap_1.SetBitmap(wx.BitmapFromImage(Img))
213         self.output, segments = ChequeOutput(cheque)

214         for i in range(len(segments)):
215             Img = self.GetBitmap(segments[i])

```

```
214         #Img = Img.SetHeight(20)
215         #Img = Img.SetWidth(20)
216         if i==0:
217             self.bitmap_2.SetBitmap(Img)
218         if i==1:
219             self.bitmap_3.SetBitmap(Img)
220         if i==2:
221             self.bitmap_4.SetBitmap(Img)
222         if i==3:
223             self.bitmap_5.SetBitmap(Img)
224         if i==4:
225             self.bitmap_6.SetBitmap(Img)
226         if i==5:
227             self.bitmap_7.SetBitmap(Img)
228         if i==6:
229             self.bitmap_8.SetBitmap(Img)
230         if i==7:
231             self.bitmap_9.SetBitmap(Img)
232         if i==8:
233             self.bitmap_10.SetBitmap(Img)
234         if i==9:
235             self.bitmap_11.SetBitmap(Img)
236         if i==10:
237             self.bitmap_12.SetBitmap(Img)
238         if i==11:
239             self.bitmap_13.SetBitmap(Img)
240         if i==12:
241             self.bitmap_14.SetBitmap(Img)
242         if i==13:
243             self.bitmap_15.SetBitmap(Img)
244         if i==14:
245             self.bitmap_16.SetBitmap(Img)
246         if i==15:
247             self.bitmap_17.SetBitmap(Img)
248         if i==16:
249             self.bitmap_18.SetBitmap(Img)
250         if i==17:
251             self.bitmap_19.SetBitmap(Img)
252         if i==18:
253             self.bitmap_20.SetBitmap(Img)
254         if i==19:
255             self.bitmap_21.SetBitmap(Img)
256         if i==20:
257             self.bitmap_22.SetBitmap(Img)
258         if i==21:
259             self.bitmap_23.SetBitmap(Img)
260         if i==22:
261             self.bitmap_24.SetBitmap(Img)
262         if i==23:
263             self.bitmap_25.SetBitmap(Img)
264         if i==24:
```

```
265                     self.bitmap_26.SetBitmap(Img)
266
267             self.text_ctrl_1.ChangeValue(self.output)
268             dialog.Destroy()
269             self.Layout()
270
271     def OnExit(self, event): # wxGlade: MainFrame.<
272         event_handler>
273         event.Skip()
274         self.Destroy()
275
276     def OnAbout(self, event): # wxGlade: MainFrame.<
277         event_handler>
278         self.dialog = AboutDialog(None)
279         #self.SetTopWindow(self.frame)
280         self.dialog.Show()
281         event.Skip()
282
283 # end of class MyFrame
284
285 class AboutDialog(wx.Dialog):
286     def __init__(self, *args, **kwds):
287         # begin wxGlade: ResultDialog.__init__
288         kwds["style"] = wx.DEFAULT_DIALOG_STYLE
289         wx.Dialog.__init__(self, *args, **kwds)
290         self.label_1 = wx.StaticText(self, wx.ID_ANY, _("This
291             _program_is_written_by_Chong_Yong_Shean_\n            _Compiled
292             _using_Python"))
293         self.button_1 = wx.Button(self, wx.ID_OK, "")
294
295         self.__set_properties()
296         self.__do_layout()
297
298         self.Bind(wx.EVT_BUTTON, self.OnClick, self.button_1)
299
300     # end wxGlade
301
302     def __set_properties(self):
303         # begin wxGlade: ResultDialog.__set_properties
304         self.SetTitle(_("About_the_Author"))
305         _icon = wx.EmptyIcon()
306         _icon.CopyFromBitmap(wx.Bitmap("icon.ico", wx.
307             BITMAP_TYPE_ANY))
308         self.SetIcon(_icon)
309         self.label_1SetFont(wx.Font(14, wx.MODERN, wx.NORMAL,
310             wx.BOLD, 0, ""))
311     # end wxGlade
312
313     def __do_layout(self):
314         # begin wxGlade: ResultDialog.__do_layout
315         sizer_3 = wx.BoxSizer(wx.VERTICAL)
```

```

310         sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
311         sizer_4.Add(self.label_1, 0, wx.
312                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
313                     0)
312         sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
313         sizer_3.Add(self.button_1, 0, wx.
314                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
315                     0)
314         self.SetSizer(sizer_3)
315         sizer_3.Fit(self)
316         self.Layout()
317         # end wxGlade
318
319     def OnClick(self, event): # wxGlade: ResultDialog.<
320         event_handler>
320         self.Destroy()
321         event.Skip()
322
323 # end of class ResultDialog
324 class MyApp(wx.App):
325     def OnInit(self):
326         wx.InitAllImageHandlers()
327         frame_1 = MyFrame(None, wx.ID_ANY, "")
328         self.SetTopWindow(frame_1)
329         frame_1.Show()
330         return 1
331
332 # end of class MyApp
333
334 if __name__ == "__main__":
335     gettext.install("app") # replace with the appropriate
336                             catalog name
336
337     app = MyApp(0)
338     app.MainLoop()

1 # cheque_process_neat.py
2 from functions import *
3 """
4 #_Log_file
5 from __future__ import print_function
6 log_=open("c:\\cheque_exp_log.txt", "w")
7
8 #_Save_all_output_in_an_array_(for_many_cheques)
9 total_output_=[]
10
11 #_Start_loop_to_process_cheque
12 for i in range(1,101):
13     print("Processing cheque number ", str(i), "... ")
14     chq_path_='chq_(' + str(i) + ').jpeg'
15     # Insert/create_a_function_that_processes_cheque
16     output_=ChequeOutput(chq_path)[0]

```

```

17 """
18 def ChequeOutput(chq_path):
19     # Load cheque image
20     # chq = Image.open('chq ('+str(i)+').jpeg')
21     chq = Image.open(chq_path)
22
23     # Extract field
24     img = ExtractField(chq)
25     img = pil_to_cv(img)
26
27     height, width = img.shape
28
29     # Segmentation
30     img_lst, mean_height, mean_width, y_arr = Segmentation(img)
31
32     # Construct reference line
33     datalr_y = [[y] for y in y_arr]
34     datalr_X = [[x] for x in range(1, len(img_lst)+1)]
35
36     datalr_X = np.array(datalr_X)
37     datalr_y = np.array(datalr_y)
38     from sklearn.linear_model import LinearRegression
39
40     regr = LinearRegression()
41     regr.fit(datalr_X, datalr_y)
42
43     dists = regr.predict(datalr_y)-datalr_y
44     #dists = abs(dists)
45     max_dist = abs(dists).max()
46
47     # Extract features
48     input = FeatureExtraction(img_lst)
49
50     # Recognition
51     clf = joblib.load("scaling_svm_overall.pkl")
52     input = FeatureExtraction(img_lst)
53     pred = clf.predict(input)
54     confidence = clf.predict_proba(input)
55
56     output = []
57     for item in pred:
58         if item<=9:
59             output.append(item)
60         if item>=10 and item<=19:
61             output.append('0'+str(item-10))
62         if item>=20 and item<=109:
63             if item==21:
64                 output.append(-1)
65                 #print "Junk"
66             else:
67                 output.append(str(item-10))

```

```

68     if item>=110 and item<=113:
69         if item==110:
70             output.append(",")
71         if item==111:
72             output.append(".")
73         if item==112:
74             output.append(-1)
75             #print "R"
76         if item==113:
77             output.append(-1)
78             #print "M"
79
80     # Post-processing
81     i=0
82     for dist in dists:
83         height, width = img_lst[i].shape
84         if i==0 and pred[i]==111: #Dot is located at the first
85             position
86             #print "Pos", i, ": probably a junk"
87             output[i] = -1
88         if i>0 and pred[i]==111 and pred[i-1]>=110: #The
89             character precede dot must be a number
90             #print "Pos", i, ": probably a junk"
91             output[i] = -1
92         if i>0 and i<pred.shape[0]-2 and pred[i]<112 and pred[i
93             +1]==112 and pred[i+2]==113: #There are some
94             characters precede RM
95             output[i] = -1
96         if pred[i]==21: #It's a junk
97             if confidence[i][21]<0.7: #Confidence of it being a
98                 junk is less than 0.7
99                 # check if it's a dot or comma
100                if confidence[i][111] > confidence[i][110]:
101                    # if prob of it being a dot is higher than of
102                    comma
103                    if i<pred.shape[0]-2 and pred[i+1]!=21 and pred
104                        [i+1]<110 and pred[i+2]!=21 and pred[i
105                            +2]<110: #Next two characters are numbers
106                            #print "Pos", i, ": probably a dot"
107                            output[i] = '.'
108                            if i>0 and pred[i-1]>110: # The character
109                                precede this segment is not a number
110                                #print "Pos", i, ": probably a junk due to
111                                    non-number before dot"
112                                output[i] = -1
113                            else:
114                                if dist<0 :
115                                    #print "Pos", i, ": probably being a comma"
116                                    #output[i] = ','
117                                else:
118                                    #print "Pos", i, ": probably being a 1"

```

```

109                      #output[i] = 1
110      if pred[i]==1 or pred[i]==110: # Differentiate between
111          '1' and ','
112          if dist<-0.4*max_dist: # Absolute distance between
113              the regression line and the maximum is larger than
114              0.4 and the maximum is below the regression line
115              #print "Pos", i, ": probably being a comma"
116              output[i] = ','
117      else:
118          #print "Pos", i, ": probably being a 1"
119          output[i] = 1
120
121      i+=1
122
123      output = filter(lambda a: a != -1, output)
124      index_of_first_number = -1
125      for k in range(len(output)):
126          if output[k]<100 and output[k]!=-1:
127              index_of_first_number = k
128              break
129      for n in range(index_of_first_number): # Eliminate any
130          comma or dot before the first number
131          if output[n]==',' or output[n]=='.':
132              output[n] = -1
133
134      output = filter(lambda a: a != -1, output)
135      cnt_comma = 0
136      cnt_dot = 0
137      m = -1
138      for k in range(len(output)):
139          if output[k]==',':
140              cnt_comma += 1
141              m = k
142          if output[k]=='.':
143              cnt_dot += 1
144          if cnt_comma==1 and cnt_dot==0 and m<=len(output)-3: #If
145              there is no dot and only one comma and next three
146              characters are not all numbers
147          if output[m+1]<100 and output[m+1]!=-1 and output[m
148              +2]<100 and output[m+2]!=-1:
149              output[m] = '.'
150
151      output = ''.join(map(str,output))
152
153      return output, img_lst

```

A-0-7 Training and Testing of MNIST Digits GUI

```

1 import wx
2 from gui import *
3 from popups import *
4 #import databases
5 #import functions
6

```

```
7 # begin wxGlade: dependencies
8 import gettext
9 # end wxGlade
10
11 class MyApp(wx.App):
12     def OnInit(self):
13         wx.InitAllImageHandlers()
14         frame_main = MyFrame(None, wx.ID_ANY, "")
15         self.SetTopWindow(frame_main)
16         frame_main.Show()
17         return 1
18
19 # end of class MyApp
20
21 if __name__ == "__main__":
22     gettext.install("app") # replace with the appropriate
23                             catalog name
24
25     app = MyApp(0)
26     app.MainLoop()
27
28 #!/usr/bin/env python
29 # -*- coding: CP936 -*-
30 #
31 # generated by wxGlade 0.6.8 (standalone edition) on Thu Dec
32 #   12 14:35:02 2013
33 #
34
35 #
36
37 import wx
38 import wx.grid
39 from wx.lib.agw import floatspin as FS
40 import thread
41 from functions import *
42 from ObjectListView import FastObjectListView, ColumnDefn
43 from entry import *
44 from popups import *
45 from ddd import *
46
47 import numpy as np
48 from PIL import Image, ImageChops
49 from skimage import io
50 from skimage.filter import threshold_otsu
51 import cv2
52 from skimage import img_as_float, img_as_ubyte
53 import itertools
54 from sklearn import preprocessing
55 from sklearn.decomposition import RandomizedPCA
56 from sklearn.preprocessing import StandardScaler, MinMaxScaler
57 import gc
58 from sklearn.externals import joblib
59 from sklearn import svm
60 from sklearn.pipeline import Pipeline
```

```
31
32 import matplotlib
33 import matplotlib.pyplot as plt; plt.rcParams()
34 from matplotlib.figure import Figure
35 from matplotlib.backends.backend_wxagg import
36     FigureCanvasWxAgg as FigureCanvas
37
38 from skimage import img_as_ubyte
39
40 class MyFrame(wx.Frame):
41     def __init__(self, *args, **kwds):
42         # begin wxGlade: MyFrame.__init__
43         kwds["style"] = wx.DEFAULT_FRAME_STYLE
44         wx.Frame.__init__(self, *args, **kwds)
45
46         self.notebook_2 = wx.Notebook(self, wx.ID_ANY, style
47             =0)
48         self.notebook_2_pane_1 = wx.Panel(self.notebook_2, wx.
49             ID_ANY)
50         self.list_ctrl_train = FastObjectListView(self.
51             notebook_2_pane_1, wx.ID_ANY, style=wx.LC_REPORT | wx.SUNKEN_BORDER)
52
53         #self.list_ctrl_train.rowFormatter = self.
54             rowFormatterTrain
55         self.tested = False
56         self.classed = False
57
58         self.sizer_4_staticbox = wx.StaticBox(self.
59             notebook_2_pane_1, wx.ID_ANY, _("Dataset"))
60         self.label_1 = wx.StaticText(self.notebook_2_pane_1,
61             wx.ID_ANY, _("Kernel:"))
62         self.choice_kernel = wx.Choice(self.notebook_2_pane_1,
63             wx.ID_ANY, choices=[_("Linear"), _("Polynomial"),
64             _("RBF"), _("Sigmoid")])
65         self.label_2 = wx.StaticText(self.notebook_2_pane_1,
66             wx.ID_ANY, _("C:"))
67         self.spin_ctrl_c = FS.FloatSpin(self.notebook_2_pane_1
68             , -1, min_val=0, max_val=20, increment=0.01)
69
70         self.label_3 = wx.StaticText(self.notebook_2_pane_1,
71             wx.ID_ANY, _("Gamma:"))
72         self.spin_ctrl_gamma = FS.FloatSpin(self.
73             notebook_2_pane_1, -1, min_val=0, max_val=10,
74             increment=0.0001)
75
76         self.label_4 = wx.StaticText(self.notebook_2_pane_1,
77             wx.ID_ANY, _("Tolerance:"))
78         self.spin_ctrl_tol = FS.FloatSpin(self.
79             notebook_2_pane_1, -1, min_val=0, max_val=1,
80             increment=0.0001)
```

```
64
65      self.label_5 = wx.StaticText(self.notebook_2_pane_1,
66          wx.ID_ANY, _("Maximum_Iterations:"))
67      self.spin_ctrl_max = wx.SpinCtrl(self.
68          notebook_2_pane_1, wx.ID_ANY, "-1", min=-1, max
69          =1000)
70      self.label_6 = wx.StaticText(self.notebook_2_pane_1,
71          wx.ID_ANY, _("Dual:"))
72      self.radio_box_dual = wx.RadioBox(self.
73          notebook_2_pane_1, wx.ID_ANY, "", choices=[_("True")
74          , _("False")], majorDimension=2, style=wx.
75          RA_SPECIFY_COLS)
76      self.label_7 = wx.StaticText(self.notebook_2_pane_1,
77          wx.ID_ANY, _("Degree:"))
78      self.slider_deg = wx.Slider(self.notebook_2_pane_1, wx
79          .ID_ANY, 0, 0, 10, style=wx.SL_HORIZONTAL | wx.
80          SL_LABELS)
81      self.sizer_6_staticbox = wx.StaticBox(self.
82          notebook_2_pane_1, wx.ID_ANY, _("Parameters"))
83      self.spin_ctrl_set = wx.SpinCtrl(self.
84          notebook_2_pane_1, wx.ID_ANY, "60000", min=0, max
85          =60000)
86      self.checkbox_all = wx.CheckBox(self.notebook_2_pane_1
87          , wx.ID_ANY, _("All"))
88      self.button_load = wx.Button(self.notebook_2_pane_1,
89          wx.ID_ANY, _("Load"))
90      self.sizer_7_staticbox = wx.StaticBox(self.
91          notebook_2_pane_1, wx.ID_ANY, _("Subset"))
92      self.bitmap_train_prev = wx.StaticBitmap(self.
93          notebook_2_pane_1, wx.ID_ANY, wx.EmptyBitmap(28, 28)
94          )
95      self.sizer_10_staticbox = wx.StaticBox(self.
96          notebook_2_pane_1, wx.ID_ANY, _("Preview"))
97      self.button_train = wx.Button(self.notebook_2_pane_1,
98          wx.ID_ANY, _("Start_Training"))
99      self.button_class = wx.Button(self.notebook_2_pane_1,
100         wx.ID_ANY, _("Classify"))
101     self.notebook_2_pane_2 = wx.Panel(self.notebook_2, wx.
102         ID_ANY)
103     self.list_ctrl_test = FastObjectListView(self.
104         notebook_2_pane_2, wx.ID_ANY, style=wx.LC_REPORT | 
105         wx.SUNKEN_BORDER)
106
107     #self.list_ctrl_test.rowFormatter = self.
108     #rowFormatterTest
109
110
111     self.sizer_13_staticbox = wx.StaticBox(self.
112         notebook_2_pane_2, wx.ID_ANY, _("Dataset"))
```

```

89         self.spin_ctrl_set_test = wx.SpinCtrl(self.
90             notebook_2_pane_2, wx.ID_ANY, "10000", min=1, max
91             =10000)
92         self.checkbox_all_test = wx.CheckBox(self.
93             notebook_2_pane_2, wx.ID_ANY, _("All"))
94         self.button_load_test = wx.Button(self.
95             notebook_2_pane_2, wx.ID_ANY, _("Load"))
96         self.sizer_2_staticbox = wx.StaticBox(self.
97             notebook_2_pane_2, wx.ID_ANY, _("Subset"))
98         self.bitmap_test_prev = wx.StaticBitmap(self.
99             notebook_2_pane_2, wx.ID_ANY, wx.EmptyBitmap(28, 28)
100            )
101        self.sizer_15_staticbox = wx.StaticBox(self.
102            notebook_2_pane_2, wx.ID_ANY, _("Preview"))
103        self.button_test = wx.Button(self.notebook_2_pane_2,
104            wx.ID_ANY, _("Start Testing"))
105        self.button_analyse = wx.Button(self.notebook_2_pane_2
106            , wx.ID_ANY, _("Analyse Result"))
107        #self.bitmap_test_prev = wx.StaticBitmap(self.
108            notebook_2_pane_2, wx.ID_ANY, wx.Bitmap("C:\\\\
109             Documents and Settings\\\\YongShean\\\\My Documents\\\\
110             img3.jpg", wx.BITMAP_TYPE_ANY))
111        #self.sizer_15_staticbox = wx.StaticBox(self.
112            notebook_2_pane_2, wx.ID_ANY, _("Preview"))
113        #self.button_test = wx.Button(self.notebook_2_pane_2,
114            wx.ID_ANY, _("Start Testing"))
115        #self.button_analyse = wx.Button(self.
116            notebook_2_pane_2, wx.ID_ANY, _("Analyse Result"))
117        self.notebook_2_pane_3 = wx.Panel(self.notebook_2, wx.
118            ID_ANY)
119        self.label_num = wx.StaticText(self.notebook_2_pane_3,
120            wx.ID_ANY, _("7"), style=wx.ALIGN_CENTRE)
121        self.sizer_19_staticbox = wx.StaticBox(self.
122            notebook_2_pane_3, wx.ID_ANY, _("Classification"))
123        self.button_browse = wx.Button(self.notebook_2_pane_3,
124            wx.ID_ANY, _("Browse..."))
125        self.bitmap_input = wx.StaticBitmap(self.
126            notebook_2_pane_3, wx.ID_ANY, wx.EmptyBitmap(28, 28)
127            , style=wx.ALIGN_CENTRE)
128        self.sizer_20_staticbox = wx.StaticBox(self.
129            notebook_2_pane_3, wx.ID_ANY, _("Input"))
130        self.window_plot = WindowPlot(self.notebook_2_pane_3)
131        self.sizer_21_staticbox = wx.StaticBox(self.
132            notebook_2_pane_3, wx.ID_ANY, _("Probability_of_
133             Predicted_Classes"))

134
135
136    # Menu Bar
137    self.frame_main_menubar = wx.MenuBar()
138    self.file = wx.Menu()

```

```

114     self.selectdir = wx.MenuItem(self.file, wx.ID_ANY, _(""
115         Select_database_directory..."), _("Select_the_"
116         directory_of_MNIST_files"), wx.ITEM_NORMAL)
117     self.loadsvm = wx.MenuItem(self.file, wx.ID_ANY, _("A_
118         trained_SVM..."), _("Select_a_trained_SVM_pipelined_
119         _with_standard_scaler"), wx.ITEM_NORMAL)
120     self.loadpca = wx.MenuItem(self.file, wx.ID_ANY, _("A_
121         fitted_PCA..."), _("Unimplemented"), wx.ITEM_NORMAL
122     )
123     #self.loadstd = wx.MenuItem(self.file, wx.ID_ANY, _("A
124         fitted Standard Scaler..."), "", wx.ITEM_NORMAL)
125     self.savesvm = wx.MenuItem(self.file, wx.ID_ANY, _(""
126         The_trained_SVM..."),_("Save_the_trained_SVM_
127         pipelined_with_standard_scaler_into_disk"), wx.
128         ITEM_NORMAL)
129     self.savepca = wx.MenuItem(self.file, wx.ID_ANY, _(""
130         The_fitted_PCA..."), _("Unimplemented"), wx.
131         ITEM_NORMAL)
132     #self.savestd = wx.MenuItem(self.file, wx.ID_ANY, _(""
133         The_fitted_Standard_Scaler..."), "", wx.ITEM_NORMAL
134     )
135     self.file.AppendItem(self.selectdir)
136     self.submenu_load = wx.Menu()
137     self.submenu_load.AppendItem(self.loadsvm)
138     self.submenu_load.AppendItem(self.loadpca)
139     #self.submenu_load.AppendItem(self.loadstd)
140     self.file.AppendMenu(wx.ID_ANY, _("Load"), self.
141         submenu_load)
142     self.file.AppendMenu(wx.ID_ANY, _("Save"), self.
143         submenu_save)
144     self.file.AppendSeparator()
145     self.exit = wx.MenuItem(self.file, wx.ID_ANY, _("Exit"
146         ), "", wx.ITEM_NORMAL)
147     self.file.AppendItem(self.exit)
148     self.frame_main_menubar.Append(self.file, _("File"))
149     self.help = wx.Menu()
150     self.view_m = wx.MenuItem(self.help, wx.ID_ANY, _(""
151         View_manual"), "", wx.ITEM_NORMAL)
152     self.help.AppendItem(self.view_m)
153     self.help.AppendSeparator()
154     self.about = wx.MenuItem(self.help, wx.ID_ANY, _(""
155         About_the_author"), "", wx.ITEM_NORMAL)
156     self.help.AppendItem(self.about)
157     self.frame_main_menubar.Append(self.help, _("Help"))
158     self.SetMenuBar(self.frame_main_menubar)
159     # Menu Bar end
160

```

```
146     self.frame_main_statusbar = self.CreateStatusBar(5, 0)
147
148     self.__set_properties()
149     self.__do_layout()
150
151     self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.
152         OnSelectTrainItem, self.list_ctrl_train)
153     self.Bind(wx.EVT_CHOICE, self.OnChoiceKernel, self.
154         choice_kernel)
155     self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinC, self.
156         spin_ctrl_c)
157     self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinGamma, self.
158         spin_ctrl_gamma)
159     self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinTol, self.
160         spin_ctrl_tol)
161     self.Bind(wx.EVT_SPINCTRL, self.OnSpinMax, self.
162         spin_ctrl_max)
163     self.Bind(wx.EVT_RADIOBOX, self.OnRadioDual, self.
164         radio_box_dual)
165     self.Bind(wx.EVT_COMMAND_SCROLL, self.OnSliderDeg,
166         self.slider_deg)
167     self.Bind(wx.EVT_SPINCTRL, self.OnSpinSet, self.
168         spin_ctrl_set)
169     self.Bind(wx.EVT_CHECKBOX, self.OnCheckAll, self.
170         checkbox_all)
171     self.Bind(wx.EVT_BUTTON, self.OnButtonLoad, self.
172         button_load)
173     self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.
174         OnSelectTestItem, self.list_ctrl_test)
175     self.Bind(wx.EVT_BUTTON, self.OnButtonTrain, self.
176         button_train)
177     self.Bind(wx.EVT_BUTTON, self.OnButtonClass, self.
178         button_class)
179     self.Bind(wx.EVT_BUTTON, self.OnButtonTest, self.
180         button_test)
181     self.Bind(wx.EVT_BUTTON, self.OnButtonAnalyse, self.
182         button_analyse)
183     self.Bind(wx.EVT_BUTTON, self.OnButtonBrowse, self.
184         button_browse)
185     self.Bind(wx.EVT_NOTEBOOK_PAGE_CHANGED, self.
186         OnPageTest, self.notebook_2)
187     self.Bind(wx.EVT_SPINCTRL, self.OnSpinTest, self.
188         spin_ctrl_set_test)
189     self.Bind(wx.EVT_CHECKBOX, self.OnCheckAllTest, self.
190         checkbox_all_test)
191     self.Bind(wx.EVT_BUTTON, self.OnButtonLoadTest, self.
192         button_load_test)
193 ##### Menu items #####
194     self.Bind(wx.EVT_MENU, self.OnAbout, self.about)
195     self.Bind(wx.EVT_MENU, self.OnSelectDir, self.
196         selectdir)
```

```
175     self.Bind(wx.EVT_MENU, self.OnLoadSVM, self.loadsvm)
176     #self.Bind(wx.EVT_MENU, self.OnLoadPCA, self.loadpca)
177     #self.Bind(wx.EVT_MENU, self.OnLoadSTD, self.loadstd)
178     self.Bind(wx.EVT_MENU, self.OnSaveSVM, self.savesvm)
179     #self.Bind(wx.EVT_MENU, self.OnSavePCA, self.savepca)
180     #self.Bind(wx.EVT_MENU, self.OnSaveSTD, self.savestd)
181     self.Bind(wx.EVT_MENU, self.OnExit, self.exit)
182     self.Bind(wx.EVT_MENU, self.OnViewManual, self.view_m)
183
184     def __set_properties(self):
185         # begin wxGlade: MyFrame.__set_properties
186         self.SetTitle(_("Handwritten_Digit_Training_and_
187             Recognition"))
188
189         # ObjectListView
190         actualColumn = ColumnDefn("Actual", "center", 50,
191             valueGetter="actual", minimumWidth=30)
192         predictedColumn = ColumnDefn("Predicted", "center", 70,
193             valueGetter="predicted", minimumWidth=50)
194         actualColumn.isSpaceFilling = True
195         predictedColumn.isSpaceFilling = True
196
197         # Floatspin options
198         self.spin_ctrl_c.SetFormat("%f")
199         self.spin_ctrl_c.SetDigits(4)
200         self.spin_ctrl_c.SetValue(2.8)
201         self.spin_ctrl_c.SetDefaultValue()
202         self.spin_ctrl_gamma.SetFormat("%f")
203         self.spin_ctrl_gamma.SetDigits(4)
204         self.spin_ctrl_gamma.SetValue(0.0073)
205         self.spin_ctrl_gamma.SetDefaultValue()
206         self.spin_ctrl_tol.SetFormat("%f")
207         self.spin_ctrl_tol.SetDigits(4)
208         self.spin_ctrl_tol.SetValue(0.001)
209         self.spin_ctrl_tol.SetDefaultValue()
210
211         # Default selection
212         self.radio_box_dual.setSelection(1)
213         self.choice_kernel.setSelection(2)
214         self.radio_box_dual.Enable(False)
215         self.slider_deg.Enable(False)
216
217         # Default values
218         self._kernel = 'rbf'
219         self._c = 2.8
220         self._gamma = 0.0073
```

```
221     self._tol = 0.001
222     self._max_iter = -1
223     self._dual = False
224     self._deg = 3
225     self._subset = 60000
226     self.button_train.Enable(False)
227     self.button_class.Enable(False)
228     self.button_test.Enable(False)
229     self.button_analyse.Enable(False)
230     self.data =
231         [0.02, 0.3, 0.45, 0.56, 0.67, 0.78, 0.89, 0.95, 0.21, 0.1]
232     self.mnist_path = '.'
233
234     self._subset_test = 10000
235
236     self.label_num.SetFont(wx.Font(68, wx.DEFAULT, wx.
237         NORMAL, wx.NORMAL, 0, ""))
238     self.frame_main_statusbar.SetStatusWidths([-1, 0, 0,
239         0, 0])
240     # statusbar fields
241     frame_main_statusbar_fields = [_("Ready")]
242     for i in range(len(frame_main_statusbar_fields)):
243         self.frame_main_statusbar.SetStatusText(
244             frame_main_statusbar_fields[i], i)
245     # end wxGlade
246
247     self.list_ctrl_train.rowFormatter = self.
248         rowFormatterTrain
249     self.list_ctrl_test.rowFormatter = self.
250         rowFormatterTest
251
252     def __do_layout(self):
253         # begin wxGlade: MyFrame.__do_layout
254         sizer_1 = wx.BoxSizer(wx.VERTICAL)
255         sizer_17 = wx.BoxSizer(wx.HORIZONTAL)
256         self.sizer_21_staticbox.Lower()
257         sizer_21 = wx.StaticBoxSizer(self.sizer_21_staticbox,
258             wx.HORIZONTAL)
259         sizer_18 = wx.BoxSizer(wx.VERTICAL)
260         self.sizer_20_staticbox.Lower()
261         sizer_20 = wx.StaticBoxSizer(self.sizer_20_staticbox,
262             wx.HORIZONTAL)
263         sizer_22 = wx.BoxSizer(wx.VERTICAL)
264         self.sizer_19_staticbox.Lower()
265         sizer_19 = wx.StaticBoxSizer(self.sizer_19_staticbox,
266             wx.HORIZONTAL)
267         """
268             sizer_12_=wx.BoxSizer(wx.VERTICAL)
269             sizer_14_=wx.BoxSizer(wx.HORIZONTAL)
270             sizer_16_=wx.BoxSizer(wx.VERTICAL)
271             self.sizer_15_staticbox.Lower()
```

```
263     sizer_15=wx.StaticBoxSizer(self.sizer_15_staticbox,wx.HORIZONTAL)
264     self.sizer_13_staticbox.Lower()
265     """
266
267     sizer_12 = wx.BoxSizer(wx.VERTICAL)
268     sizer_14 = wx.BoxSizer(wx.HORIZONTAL)
269     sizer_16 = wx.BoxSizer(wx.VERTICAL)
270     self.sizer_15_staticbox.Lower()
271     sizer_15 = wx.StaticBoxSizer(self.sizer_15_staticbox,
272                               wx.HORIZONTAL)
272     self.sizer_2_staticbox.Lower()
273     sizer_2 = wx.StaticBoxSizer(self.sizer_2_staticbox, wx.HORIZONTAL)
274
275     sizer_13 = wx.StaticBoxSizer(self.sizer_13_staticbox,
276                               wx.HORIZONTAL)
276     sizer_3 = wx.BoxSizer(wx.HORIZONTAL)
277     sizer_5 = wx.BoxSizer(wx.VERTICAL)
278     sizer_9 = wx.BoxSizer(wx.HORIZONTAL)
279     sizer_11 = wx.BoxSizer(wx.VERTICAL)
280     self.sizer_10_staticbox.Lower()
281     sizer_10 = wx.StaticBoxSizer(self.sizer_10_staticbox,
282                               wx.HORIZONTAL)
282     self.sizer_7_staticbox.Lower()
283     sizer_7 = wx.StaticBoxSizer(self.sizer_7_staticbox, wx.HORIZONTAL)
284     sizer_8 = wx.BoxSizer(wx.HORIZONTAL)
285     self.sizer_6_staticbox.Lower()
286     sizer_6 = wx.StaticBoxSizer(self.sizer_6_staticbox, wx.HORIZONTAL)
287     grid_sizer_1 = wx.FlexGridSizer(7, 2, 3, 3)
288     self.sizer_4_staticbox.Lower()
289     sizer_4 = wx.StaticBoxSizer(self.sizer_4_staticbox, wx.HORIZONTAL)
290     sizer_4.Add(self.list_ctrl_train, 1, wx.EXPAND, 0)
291     sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
292     grid_sizer_1.Add(self.label_1, 0, wx.ALL | wx.EXPAND |
293                      wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
294     grid_sizer_1.Add(self.choice_kernel, 0, 0, 0)
295     grid_sizer_1.Add(self.label_2, 0, wx.ALL | wx.EXPAND |
296                      wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
297     grid_sizer_1.Add(self.spin_ctrl_c, 0, 0, 0)
298     grid_sizer_1.Add(self.label_3, 0, wx.ALL | wx.EXPAND |
299                      wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
```

```
299         ALIGN_CENTER_VERTICAL, 3)
300     grid_sizer_1.Add(self.spin_ctrl_tol, 0, 0, 0)
301     grid_sizer_1.Add(self.label_5, 0, wx.ALL | wx.EXPAND |
302                       wx.ALIGN_CENTER_HORIZONTAL | wx.
303                           ALIGN_CENTER_VERTICAL, 3)
304     grid_sizer_1.Add(self.spin_ctrl_max, 0, 0, 0)
305     grid_sizer_1.Add(self.label_6, 0, wx.ALL | wx.EXPAND |
306                       wx.ALIGN_CENTER_HORIZONTAL | wx.
307                           ALIGN_CENTER_VERTICAL, 3)
308     grid_sizer_1.Add(self.radio_box_dual, 0, wx.EXPAND, 0)
309     grid_sizer_1.Add(self.label_7, 0, wx.ALL | wx.EXPAND |
310                       wx.ALIGN_CENTER_HORIZONTAL | wx.
311                           ALIGN_CENTER_VERTICAL, 3)
312     grid_sizer_1.Add(self.slider_deg, 0, wx.EXPAND, 0)
313     sizer_6.Add(grid_sizer_1, 1, wx.EXPAND, 0)
314     sizer_5.Add(sizer_6, 3, wx.EXPAND, 0)
315     sizer_8.Add(self.spin_ctrl_set, 0, wx.ALL | wx.
316                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
317                     5)
318     sizer_8.Add(self.checkbox_all, 0, wx.LEFT | wx.
319                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
320                     10)
321     sizer_8.Add(self.button_load, 0, wx.LEFT | wx.
322                     ALIGN_RIGHT | wx.ALIGN_CENTER_HORIZONTAL | wx.
323                         ALIGN_CENTER_VERTICAL, 10)
324     sizer_8.Add(sizer_7, 1, wx.EXPAND | wx.
325                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
326                     0)
327     sizer_5.Add(sizer_7, 1, wx.EXPAND | wx.
328                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
329                     0)
330     sizer_10.Add(self.bitmap_train_prev, 1, wx.EXPAND | wx
331                     .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL
332                     , 0)
333     sizer_9.Add(sizer_10, 1, wx.EXPAND | wx.
334                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
335                     0)
336     sizer_11.Add(self.button_train, 0, wx.EXPAND | wx.
337                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
338                     0)
339     sizer_11.Add(self.button_class, 0, wx.EXPAND | wx.
340                     ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
341                     0)
342     sizer_9.Add(sizer_11, 2, wx.LEFT | wx.ALIGN_RIGHT | wx
343                     .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL
344                     , 10)
345     sizer_5.Add(sizer_9, 1, wx.EXPAND, 0)
346     sizer_3.Add(sizer_5, 1, wx.EXPAND, 0)
347     self.notebook_2_pane_1.SetSizer(sizer_3)
348     sizer_13.Add(self.list_ctrl_test, 1, wx.EXPAND, 0)
349     sizer_12.Add(sizer_13, 5, wx.EXPAND, 0)
```

```
323     """
324     sizer_15.Add(self.bitmap_test_prev, 1, wx.
325                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
325     sizer_14.Add(sizer_15, 1, wx.EXPAND | wx.
326                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
326     sizer_16.Add(self.button_test, 0, wx.EXPAND | wx.
327                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
327     sizer_16.Add(self.button_analyse, 0, wx.EXPAND | wx.
328                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
328     sizer_14.Add(sizer_16, 1, wx.ALL | wx.ALIGN_RIGHT | wx
329                 .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 5)
329     sizer_12.Add(sizer_14, 1, wx.EXPAND, 0)
330     self.notebook_2_pane_2.SetSizer(sizer_12)
331 """
332     sizer_2.Add(self.spin_ctrl_set_test, 0, wx.ALL | wx.
333                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
333                 10)
333     sizer_2.Add(self.checkbox_all_test, 0, wx.ALL | wx.
334                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
334                 5)
334     sizer_2.Add(self.button_load_test, 0, wx.
335                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
335                 5)
335     sizer_14.Add(sizer_2, 3, wx.EXPAND | wx.
336                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
336                 0)
336     sizer_15.Add(self.bitmap_test_prev, 1, wx.
337                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
337                 0)
337     sizer_14.Add(sizer_15, 1, wx.EXPAND | wx.
338                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
338                 0)
338     sizer_16.Add(self.button_test, 0, wx.EXPAND | wx.
339                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
339                 0)
340     sizer_14.Add(sizer_16, 2, wx.ALL | wx.ALIGN_RIGHT | wx
340                 .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL
340                 , 5)
341     sizer_12.Add(sizer_14, 1, wx.EXPAND, 0)
342     self.notebook_2_pane_2.SetSizer(sizer_12)
343
344     sizer_19.Add(self.label_num, 1, wx.EXPAND | wx.
344                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
344                 5)
345     sizer_18.Add(sizer_19, 1, wx.EXPAND | wx.
345                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
345                 0)
346     sizer_22.Add(self.button_browse, 0, wx.EXPAND, 0)
```

```

347     sizer_22.Add(self.bitmap_input, 1, wx.EXPAND | wx.
348                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
349                 0)
350     sizer_20.Add(sizer_22, 1, wx.EXPAND, 0)
351     sizer_18.Add(sizer_20, 1, wx.EXPAND, 0)
352     sizer_17.Add(sizer_18, 1, wx.EXPAND, 0)
353     sizer_21.Add(self.window_plot, 1, wx.EXPAND | wx.
354                 ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
355                 0)
356     sizer_21.SetSizeHints(self)
357     sizer_17.Add(sizer_21, 3, wx.EXPAND, 0)
358     self.notebook_2_pane_3.SetSizer(sizer_17)
359     self.notebook_2.AddPage(self.notebook_2_pane_1, _(""
360                             "Training"))
361     self.notebook_2.AddPage(self.notebook_2_pane_2, _(""
362                             "Testing"))
363     self.notebook_2.AddPage(self.notebook_2_pane_3, _(""
364                             "Classify"))
365     sizer_1.Add(self.notebook_2, 1, wx.EXPAND, 0)
366     self.SetSizer(sizer_1)
367     sizer_1.Fit(self)
368
369     #self.window_plot.GetParent().GetSizer().Hide(self.
370           window_plot)
371
372     self.Layout()
373     # end wxGlade
374
375     # Events definition
376
377     def OnSelectTrainItem(self, event):
378         self.bitmap_train_prev.SetBitmap(wx.EmptyBitmap(28,28)
379             )
380         item = self.list_ctrl_train.GetSelectedObject()
381         self.bitmap_train_prev.SetBitmap(item.image)
382         self.bitmap_train_prev.GetParent().GetSizer().Layout()
383         self.Refresh()
384         event.Skip()
385
386     def OnChoiceKernel(self, event):
387         # Depending on the choice of kernel, certain parameter
388         # options would be disabled or changed.
389         sel = self.choice_kernel.GetSelection()
390         if sel==0: #Linear
391             self._kernel = 'linear'
392             #change tolerance to 0.0001
393             self.spin_ctrl_tol.SetValue(0.0001)
394             #enable dual option
395             self.radio_box_dual.Enable(True)
396             #disable gamma option
397             self.spin_ctrl_gamma.Enable(False)
398

```

```

388             #disable max_iter option
389             self.spin_ctrl_max.Enable(False)
390             #disable degree option
391             self.slider_deg.Enable(False)
392         if sel==1: #Polynomial
393             self._kernel = 'poly'
394             #enable degree option
395             self.slider_deg.Enable(True)
396             self.slider_deg.SetValue(3)
397             #disable dual option
398             self.radio_box_dual.Enable(False)
399             #change tolerance to 0.001
400             self.spin_ctrl_tol.SetValue(0.0010)
401             #enable gamma option
402             self.spin_ctrl_gamma.Enable(True)
403             #enable max_iter option
404             self.spin_ctrl_max.Enable(True)
405         if sel==2: #RBF or Sigmoid
406             self._kernel = 'rbf'
407             #change tolerance to 0.001
408             self.spin_ctrl_tol.SetValue(0.0010)
409             #disable dual option
410             self.radio_box_dual.Enable(False)
411             #disable degree option
412             self.slider_deg.Enable(False)
413             #enable gamma option
414             self.spin_ctrl_gamma.Enable(True)
415             #enable max_iter option
416             self.spin_ctrl_max.Enable(True)
417         if sel==3: #RBF or Sigmoid
418             self._kernel = 'sigmoid'
419             #change tolerance to 0.001
420             self.spin_ctrl_tol.SetValue(0.0010)
421             #disable dual option
422             self.radio_box_dual.Enable(False)
423             #disable degree option
424             self.slider_deg.Enable(False)
425             #enable gamma option
426             self.spin_ctrl_gamma.Enable(True)
427             #enable max_iter option
428             self.spin_ctrl_max.Enable(True)
429             event.Skip()
430
431     def OnSelectDir(self, event):
432         dialog = wx.DirDialog(None, "Choose_the_directory_of_"
433                               "MNIST_database", '.', style=wx.DD_DIR_MUST_EXIST)
434         if dialog.ShowModal() == wx.ID_OK:
435             self.mnist_path = dialog.GetPath()
436             event.Skip()
437     def OnLoadSVM(self, event):

```

```
438     wildcard = "PKL_file_(*.pkl)|*.pkl"
439     dialog = wx.FileDialog(None, "Choose_a_saved_SVM_file"
440                           , os.getcwd(),
441                           "", wildcard, wx.OPEN)
442
443     result = dialog.ShowModal()
444     if result==wx.ID_OK:
445         path = dialog.GetPath()
446         self.clf = joblib.load(path)
447         dialog.Destroy()
448         self.button_train.SetLabel('Start_Extracting')
449         return
450     elif result==wx.ID_CANCEL:
451         dialog.Destroy()
452         return
453
454     event.Skip()
455
456 def OnLoadPCA(self, event):
457     wildcard = "PKL_file_(*.pkl)|*.pkl"
458     dialog = wx.FileDialog(None, "Choose_a_saved_PCA_file"
459                           , os.getcwd(),
460                           "", wildcard, wx.OPEN)
461
462     result = dialog.ShowModal()
463     if result==wx.ID_OK:
464         path = dialog.GetPath()
465         self.pca = joblib.load(path)
466         dialog.Destroy()
467         return
468     elif result==wx.ID_CANCEL:
469         dialog.Destroy()
470     event.Skip()
471
472 def OnLoadSTD(self, event):
473     wildcard = "NPY_file_(*.npy)|*.npy"
474     dialog = wx.FileDialog(None, "Choose_a_saved_scaler_
475                           file", os.getcwd(),
476                           "", wildcard, wx.OPEN)
477
478     result = dialog.ShowModal()
479     if result==wx.ID_OK:
480         path = dialog.GetPath()
481         self.min_max_scaler = np.load(path)
482         dialog.Destroy()
483         return
484     elif result==wx.ID_CANCEL:
485         dialog.Destroy()
486     event.Skip()
487
488 def OnSaveSVM(self, event):
```

```
486     try:
487         self.clf
488     except AttributeError:
489         dial = wx.MessageDialog(None, "The_SVM_must_be_
        trained_before_saving!", 'Error', wx.OK | wx.
        ICON_ERROR)
490         dial.ShowModal()
491     return
492
493     dialog = wx.DirDialog(None, "Choose_the_directory_you_
        want_to_save_SVM_to", '.', style=wx.
        DD_DEFAULT_STYLE)
494     if dialog.ShowModal() == wx.ID_OK:
495         path = dialog.GetPath()
496         joblib.dump(self.clf, str(path)+"trained_svm.pkl")
497         dial2 = wx.MessageDialog(None, "The_file_is_saved_
            as 'trained_svm.pkl' in "+str(path)+"\nYou_may_
            load_this_file_on_future_startup.", 'Saved_
            successfully', wx.OK | wx.ICON_INFORMATION)
498         event.Skip()
499
500     def OnSavePCA(self, event):
501         try:
502             self.clf
503         except AttributeError:
504             dial = wx.MessageDialog(None, "The_SVM_must_be_
                trained_before_saving!", 'Error', wx.OK | wx.
                ICON_ERROR)
505             dial.ShowModal()
506         return
507
508     dialog = wx.DirDialog(None, "Choose_the_directory_you_
        want_to_save_PCA_to", '.', style=wx.
        DD_DEFAULT_STYLE)
509     if dialog.ShowModal() == wx.ID_OK:
510         path = dialog.GetPath()
511         joblib.dump(self.pca, str(path)+"fitted_pca.pkl")
512         dial2 = wx.MessageDialog(None, "The_file_is_saved_
            as 'fitted_pca.pkl' in "+str(path)+"\nYou_may_
            load_this_file_on_future_startup.", 'Saved_
            successfully', wx.OK | wx.ICON_INFORMATION)
513         event.Skip()
514
515     def OnSaveSTD(self, event):
516         try:
517             self.clf
518         except AttributeError:
519             dial = wx.MessageDialog(None, "The_SVM_must_be_
                trained_before_saving!", 'Error', wx.OK | wx.
                ICON_ERROR)
520             dial.ShowModal()
```

```

521         return
522
523     dialog = wx.DirDialog(None, "Choose_the_directory_you_
      want_to_save_standard_scaler_to", '.', style=wx.
      DD_DEFAULT_STYLE)
524     if dialog.ShowModal() == wx.ID_OK:
525         path = dialog.GetPath()
526         joblib.dump(self.min_max_scaler, str(path)+""
527                     "fitted_std.pkl")
528         dial2 = wx.MessageDialog(None, "The_file_is_saved_
      as'_fitted_std.pkl'_in"+str(path)+"\nYou_may_
      load_this_file_on_future_startup.", 'Saved_
      successfully', wx.OK | wx.ICON_INFORMATION)
529         event.Skip()
530
531     def OnViewManual(self, event):
532         link = "https://docs.google.com/document/d/1
      zOjEKk2vD_UMxxCtaSbat0bcW-_i9-pdyttQRP6kQLg/edit?
      usp=sharing"
533         dial = wx.MessageDialog(None, "You_are_about_to_open_
      the_manual_in_your_default_browser.\nClick_OK_to_
      proceed_or_click_CANCEL_to_close.", 'Warning', wx.
      OK | wx.CANCEL | wx.ICON_EXCLAMATION)
534         if dial.ShowModal() == wx.ID_OK:
535             wx.LaunchDefaultBrowser(link)
536         dial = None
537         event.Skip()
538
539     def OnExit(self, event):
540         event.Skip()
541         self.Destroy()
542
543     def OnAbout(self, event):
544         #self.dialog = AboutDialog(None)
545         #self.dialog.Show()
546         dial = wx.MessageDialog(None, "This_program_is_created
      _by_Chong_Yong_Shean.\nCompiled_using_Python.\nGUI_
      is_created_using_wxPython_with_the_aid_of_wxGlade."
      , 'About', wx.OK | wx.ICON_INFORMATION)
547         dial.ShowModal()
548         event.Skip()
549
550     def OnFloatSpinC(self, event):
551         floatspin = event.GetEventObject()
552         fmt = floatspin.GetFormat()
553         dgt = floatspin.GetDigits()
554         strs = ("%100." + str(dgt) + fmt[1])%floatspin.
555             GetValue()
556         self._c = float(strs.strip())
557         event.Skip()

```

```
557     def OnFloatSpinGamma(self, event):
558         floatspin = event.GetEventObject()
559         fmt = floatspin.GetFormat()
560         dgt = floatspin.GetDigits()
561         strs = ("%100." + str(dgt) + fmt[1])%floatspin.
562             GetValue()
563         self._gamma = float(strs.strip())
564         event.Skip()
565
566     def OnFloatSpinTol(self, event):
567         floatspin = event.GetEventObject()
568         fmt = floatspin.GetFormat()
569         dgt = floatspin.GetDigits()
570         strs = ("%100." + str(dgt) + fmt[1])%floatspin.
571             GetValue()
572         self._tol = float(strs.strip())
573         event.Skip()
574
575     def OnSpinMax(self, event):
576         self._max_iter = int(self.spin_ctrl_max.GetValue())
577         event.Skip()
578
579     def OnRadioDual(self, event):
580         if self.radio_box_dual.GetSelection() == 0:
581             self._dual = True
582         else:
583             self._dual = False
584         event.Skip()
585
586     def OnSliderDeg(self, event):
587         self._deg = int(self.slider_deg.GetValue())
588         event.Skip()
589
590     def OnSpinSet(self, event):
591         isChecked = self.checkbox_all.GetValue()
592         if isChecked:
593             self._subset = 60000
594         else:
595             self._subset = int(self.spin_ctrl_set.GetValue())
596         event.Skip()
597
598     def OnCheckAll(self, event):
599         isChecked = self.checkbox_all.GetValue()
600         if isChecked:
601             self._subset = 60000
602             self.spin_ctrl_set.SetValue(60000)
603             self.spin_ctrl_set.Enable(False)
604         else:
605             self.spin_ctrl_set.Enable(True)
606             self._subset = int(self.spin_ctrl_set.GetValue())
607         event.Skip()
```

```
606
607     def OnSpinTest(self, event):
608         isChecked = self.checkbox_all_test.GetValue()
609         if isChecked:
610             self._subset_test = 10000
611         else:
612             self._subset_test = int(self.spin_ctrl_set_test.
613                                     GetValue())
613         event.Skip()
614
615     def OnCheckAllTest(self, event):
616         isChecked = self.checkbox_all_test.GetValue()
617         if isChecked:
618             self._subset_test = 10000
619             self.spin_ctrl_set_test.SetValue(10000)
620             self.spin_ctrl_set_test.Enable(False)
621         else:
622             self.spin_ctrl_set_test.Enable(True)
623             self._subset_test = int(self.spin_ctrl_set_test.
624                                     GetValue())
624         event.Skip()
625
626     def OnButtonLoad(self, event):
627         #self.Hide()
628         self.SetStatusText("Loading_dataset...")
629         self.button_load.Enable(False)
630         msg = "Please_wait_while_the_dataset_is_being_loaded
631         ..."
631         self.busyDlg = wx.BusyInfo(msg)
632         thread.start_new_thread(self.OnStartLoad, ())
633         event.Skip()
634
635     def OnButtonLoadTest(self, event):
636         self.SetStatusText("Loading_dataset...")
637         self.button_load_test.Enable(False)
638         msg = "Please_wait_while_the_dataset_is_being_loaded
639         ..."
639         self.busyDlg = wx.BusyInfo(msg)
640         thread.start_new_thread(self.OnStartLoadTest, ())
641         event.Skip()
642
643     def OnSelectTestItem(self, event):
644         self.bitmap_test_prev.SetBitmap(wx.EmptyBitmap(28, 28))
645         item = self.list_ctrl_test.GetSelectedObject()
646         self.bitmap_test_prev.SetBitmap(item.image)
647         self.bitmap_test_prev.GetParent().GetSizer().Layout()
648         self.Refresh()
649         event.Skip()
650
651     def OnButtonTrain(self, event):
652         self.SetStatusText("Extracting_features...")
```

```
653     self.button_train.Enable(False)
654     msg = "Extracting_features..."
655     self.busyDlg = wx.BusyInfo(msg)
656     thread.start_new_thread(self.OnStartExtract, ())
657     event.Skip()
658
659     def OnButtonClass(self, event):
660         self.SetStatusText("Classifying_images...")
661         self.button_class.Enable(False)
662         msg = "Please_wait_while_the_dataset_is_being_
663             classified..."
664         self.busyDlg = wx.BusyInfo(msg)
665         thread.start_new_thread(self.OnStartClass, ())
666         event.Skip()
667
668     def OnButtonTest(self, event):
669         try:
670             self.clf
671         except AttributeError:
672             dial = wx.MessageDialog(None, "The_SVM_must_be_
673                 trained_before_testing!", 'Error', wx.OK | wx.
674                 ICON_ERROR)
675             dial.ShowModal()
676         return
677
678     self.SetStatusText("Extracting_features...")
679     self.button_test.Enable(False)
680     msg = "Extracting_features..."
681     self.busyDlg = wx.BusyInfo(msg)
682     thread.start_new_thread(self.OnStartExtractTest, ())
683
684     event.Skip()
685
686     def OnButtonAnalyse(self, event):
687         self.analyseDlg = AnalyseDialog(self)
688         g = self.analyseDlg.grid_analyse
689
690         for i in range(10):
691             for j in range(10):
692                 g.SetCellValue(i, j, '0')
693
694         for entry in self.data_test:
695             for i in range(10):
696                 for j in range(10):
697                     if entry.predicted==str(i) and entry.
698                         actual==str(j):
699                             val = int(g.GetCellValue(i, j)) + 1
700                             g.SetCellValue(i, j, str(val))
701
702                         """
703                         for_entry_in_self.data_test:
704                             for_i_in_range(10):
```

```

700         cnt=0
701         if entry.actual==str(i):
702             for j in range(10):
703                 if entry.predicted==str(j):
704                     cnt+=1
705             """
706             #g.SetValue(row,col,s)
707             self.analyseDlg.ShowModal()
708             #analyse = AnalyseFrame(self)
709             #analyse.Show()
710             event.Skip()
711
712     def OnButtonBrowse(self, event):
713         try:
714             self.clf
715         except AttributeError:
716             dial = wx.MessageDialog(None, "The SVM must be"
717                                     "trained before using!", 'Error', wx.OK | wx.
718                                     ICON_ERROR)
719             dial.ShowModal()
720             return
721         wildcard = "JPG_file (*.jpg)|*.jpg|JPEG_file (*.jpeg)"
722         dialog = wx.FileDialog(None, "Choose an image", os.
723                              .getcwd(),
724                               "", wildcard, wx.OPEN)
725         if dialog.ShowModal() == wx.ID_OK:
726             path = dialog.GetPath()
727
728             Img = wx.Image(path, wx.BITMAP_TYPE_JPEG)
729             # scale the image, preserving the aspect ratio
730             W = Img.GetWidth()
731             H = Img.GetHeight()
732             self.MaxImageSize = 28
733             if W > H:
734                 NewW = self.MaxImageSize
735                 NewH = self.MaxImageSize * H / W
736             else:
737                 NewH = self.MaxImageSize
738                 NewW = self.MaxImageSize * W / H
739             Img = Img.Scale(NewW, NewH)
740             self.bitmap_input.SetBitmap(wx.BitmapFromImage(Img
741                                         ))
742             img = cv2.imread(path, cv2.CV_LOAD_IMAGE_GRAYSCALE
743                             )
744             img = 255-img
745
746             self.binary = img
747
748             self.setStatusText("Extracting features...")
749             self.button_test.Enable(False)

```

```

745             msg = "Extracting_features..."
746             self.busyDlg = wx.BusyInfo(msg)
747             thread.start_new_thread(self.OnStartExtractInput,
748                                     ())
749             event.Skip()
750
751     def OnPageTest(self, event):
752         #print self.notebook_2.GetSelection()
753         event.Skip()
754
755     def OnStartLoad(self):
756         img, labels = read_mnist(range(10), 'training', path=
757                                 self.mnist_path)
758         self.images = img
759         labels = labels.flatten()
760         self.labels = labels.astype(float)
761
761     if self._subset!=60000:
762         i=0
763         temp=[]
764         for im in self.images:
765             #im = 255-im
766             temp.append(im)
767             i+=1
768             if i>=self._subset:
769                 break
770         self.images = temp
771
772         i=0
773         temp=[]
774         for lb in self.labels:
775             temp.append(lb)
776             i+=1
777             if i>=self._subset:
778                 break
779         self.labels = temp
780
781         self.PopulateList(self.list_ctrl_train)
782
783         wx.CallAfter(self.OnDoneLoad)
784
785     def OnStartLoadTest(self):
786         img, labels = read_mnist(range(10), 'testing', path=
787                                 self.mnist_path)
788         self.images_test = img
789         labels = labels.flatten()
790         self.labels_test = labels.astype(float)
791
791     if self._subset_test!=60000:
792         i=0

```

```
793         temp=[ ]
794     for im in self.images_test:
795         #im = 255-im
796         temp.append(im)
797         i+=1
798         if i>=self._subset_test:
799             break
800     self.images_test = temp
801
802     i=0
803     temp=[ ]
804     for lb in self.labels_test:
805         temp.append(lb)
806         i+=1
807         if i>=self._subset_test:
808             break
809     self.labels_test = temp
810
811     self.PopulateListTest(self.list_ctrl_test)
812
813     wx.CallAfter(self.OnDoneLoadTest)
814
815     def OnDoneLoad(self):
816         self.busyDlg = None
817         #self.Show()
818         #self.Layout()
819         self.button_load.Enable(True)
820         self.button_train.Enable(True)
821         self.SetStatusText("Dataset_completely_loaded.")
822
823     def OnDoneLoadTest(self):
824         self.busyDlg = None
825         #self.Show()
826         #self.Layout()
827         self.button_load_test.Enable(True)
828         self.button_test.Enable(True)
829         self.SetStatusText("Dataset_completely_loaded.")
830
831     def OnDoneExtract(self):
832         self.busyDlg = None
833         self.SetStatusText("Extraction_completed.")
834         try:
835             self.clf
836             self.button_class.Enable(True)
837         except AttributeError:
838             self.SetStatusText("Training_dataset...")
839             msg = "Training_dataset..._This_might_take_a_
840             significant_amount_of_time..."
841             self.busyDlg = wx.BusyInfo(msg)
842             thread.start_new_thread(self.OnStartTrain, ())
843             #wx.CallAfter(self.OnStartTrain)
```

```

843
844     def OnDoneExtractTest(self):
845         self.busyDlg = None
846         self.SetStatusText("Testing_dataset...")
847         msg = "Please_wait_while_the_dataset_is_being_tested
848             ..."
849         self.busyDlg = wx.BusyInfo(msg)
850         thread.start_new_thread(self.OnStartTest, ())
851         #wx.CallAfter(self.OnStartTrain)
852
853     def OnStartTrain(self):
854         if self._kernel=='linear':
855             clf = svm.LinearSVC(C=self._c, dual=self._dual,
856                                 tol=self._tol)
857         elif self._kernel=='poly':
858             clf = svm.SVC(kernel='poly',C=self._c, gamma=self.
859                         _gamma, tol=self._tol, max_iter=self._max_iter,
860                         degree=self._deg, probability=True)
861         elif self._kernel=='rbf':
862             clf = svm.SVC(kernel='rbf', C=self._c, gamma=self.
863                         _gamma, tol=self._tol, max_iter=self._max_iter,
864                         probability=True)
865         elif self._kernel=='sigmoid':
866             clf = svm.SVC(kernel='sigmoid', C=self._c, gamma=
867                           self._gamma, tol=self._tol, max_iter=self.
868                           _max_iter, probability=True)
869         scaler = StandardScaler()
870         self.clf = Pipeline([("scaler", scaler), ("svm", clf)
871                           ])
872         self.clf.fit(self.features, self.labels)
873         wx.CallAfter(self.OnDoneTrain)
874
875     def OnDoneTrain(self):
876         self.busyDlg = None
877         self.button_train.Enable(True)
878         self.button_class.Enable(True)
879         #self.Layout()
880         self.SetStatusText("Training_completed.")
881
882     def OnStartClass(self):
883         prediction = self.clf.predict(self.features)
884         self.accuracy = self.clf.score(self.features, self.
885                                         labels)
886         for entry,pred in zip(self.data, prediction):
887             entry.predicted = str(int(round(pred)))
888             # Condition testing: if entry.predicted != label
889             # then red colour
890         wx.CallAfter(self.OnDoneClass)
891
892     def OnDoneClass(self):
893         self.busyDlg = None

```

```

883         self.button_class.Enable(True)
884         self.classed = True
885         FastObjectListView.RefreshObjects(self.list_ctrl_train
886             )
886     #self.Layout()
887     self.SetStatusText("Result: "+str(int(self.accuracy*
888         self._subset))+ " out of "+str(self._subset)+" ("+
889         str(self.accuracy*100)+"%)")
890
891     def OnStartTest(self):
892         prediction = self.clf.predict(self.features_test)
893         self.accuracy_test = self.clf.score(self.features_test
894             ,self.labels_test)
895         for entry,pred in zip(self.data_test, prediction):
896             entry.predicted = str(int(round(pred)))
897         wx.CallAfter(self.OnDoneTest)
898
899     def OnDoneTest(self):
900         self.busyDlg = None
901         self.button_test.Enable(True)
902         self.button_analyse.Enable(True)
903         self.tested = True
904         FastObjectListView.RefreshObjects(self.list_ctrl_test)
905         #self.Layout()
906         self.SetStatusText("Result: "+str(int(self.
907             accuracy_test*self._subset_test))+ " out of "+str(
908             self._subset_test)+" ("+str(self.accuracy_test*100)
909             +"%)")
910
911     def PopulateList(self, list_ctrl):
912         self.data = []
913         for label,im in zip(self.labels,self.images):
914             bmp = GetBitmap(im)
915             self.data.append(Entry(image=bmp,actual=str(int(
916                 label)), predicted=''))
917             list_ctrl.SetObjects(self.data)
918
919     def PopulateListTest(self, list_ctrl):
920         self.data_test = []
921         for label,im in zip(self.labels_test,self.images_test):
922             :
923             bmp = GetBitmap(im)
924             self.data_test.append(Entry(image=bmp,actual=str(
925                 int(label)), predicted=''))
926             list_ctrl.SetObjects(self.data_test)
927
928     def OnStartExtractOld(self):
929         features = []
930         i=0

```

```

924      # Classification module: Test if size is not 28x28
925      # make sure it goes through resizing and
926      # morphological operations
927      for img in self.images:
928          #f = cv2.imread(img, cv2.CV_LOAD_IMAGE_GRAYSCALE)
929          #f = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY)
930          #f = 255-f
931          #io.imsave("train_b4.jpg", img)
932          #f = cv2.imread("train_b4.jpg", cv2.
933          #                CV_LOAD_IMAGE_GRAYSCALE)
934          #f = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
935          #cv2.imwrite("train.jpg", f)
936          feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2,
937                                      cv2.cv.CV_DIST_MASK_PRECISE)
938          feat = tuple(itertools.chain(*feat))
939          features.append(feat)
940          i+=1
941          self.setStatusText("Extracting_features..."+str(i)
942                  +"_out_of_" +str(self._subset))
943
944      self.features = np.array(features)
945
946      thread.start_new_thread(self.OnStartRed, ())
947      #wx.CallAfter(self.OnStartRed)
948
949      def OnStartExtract(self):
950          features = []
951          i=0
952          # Classification module: Test if size is not 28x28
953          # make sure it goes through resizing and
954          # morphological operations
955          for img in self.images:
956              f = Image.fromarray(img)
957              f = trim(f)
958              f = f.convert('L')
959              f = np.array(f)
960              f = img_as_ubyte(f)
961              f = cv2.resize(f, (16,16), interpolation=cv2.
962                            INTER_LANCZOS4)
963              feat = ExtractDDD(f)
964              feat = tuple(itertools.chain(*feat))
965              feat = tuple(itertools.chain(*feat))
966              features.append(feat)
967              i+=1
968              self.setStatusText("Extracting_features..."+str(i)
969                  +"_out_of_" +str(self._subset))
970
971      self.features = np.array(features)
972
973      #thread.start_new_thread(self.OnStartRed, ())

```

```

965         wx.CallAfter(self.OnDoneExtract)
966
967     def OnStartRed(self):
968         # fit pca and std_scaler - save globally
969         #self.SetStatusText("Initialising PCA and standard
970         #scaler...")
971         #self.pca = RandomizedPCA(whiten=True)
972         self.SetStatusText("Scaling_the_feature_vector...")
973         try:
974             self.features = self.clf.transform(self.features)
975         except AttributeError:
976             self.clf = MinMaxScaler(feature_range=(-1, 1))
977             self.features = self.min_max_scaler.fit_transform(
978                 self.features)
979
980             #self.SetStatusText("Applying PCA...")
981             #self.features = self.pca.fit_transform(self.features)
982
983             self.SetStatusText("Extraction_completed.")
984
985             wx.CallAfter(self.OnDoneExtract)
986
987     def OnStartExtractTestOld(self):
988         features = []
989         i=0
990         # Classification module: Test if size is not 28x28
991         # make sure it goes through resizing and
992         # morphological operations
993         for img in self.images_test:
994             #f = cv2.imread(img, cv2.CV_LOAD_IMAGE_GRAYSCALE)
995             #f = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
996             #io.imsave("test_b4.jpg", img)
997             #f = cv2.imread("test_b4.jpg", cv2.
998             #               CV_LOAD_IMAGE_GRAYSCALE)
999             feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2,
1000                                         cv2.cv.CV_DIST_MASK_PRECISE)
1001             feat = tuple(itertools.chain(*feat))
1002             features.append(feat)
1003             i+=1
1004             self.SetStatusText("Extracting_features..."+str(i)
1005                               +"_out_of_" +str(self._subset_test))
1006
1007             self.features_test = np.array(features)
1008
1009             thread.start_new_thread(self.OnStartRedTest, ())
1010             #wx.CallAfter(self.OnStartRed)
1011
1012     def OnStartExtractTest(self):
1013         features = []
1014         i=0

```

```

1008     # Classification module: Test if size is not 28x28
1009     # make sure it goes through resizing and
1010     # morphological operations
1011     for img in self.images_test:
1012         f = Image.fromarray(img)
1013         f = trim(f)
1014         f = f.convert('L')
1015         f = np.array(f)
1016         f = img_as_ubyte(f)
1017         f = cv2.resize(f, (16,16), interpolation=cv2.
1018                         INTER_LANCZOS4)
1019         feat = ExtractDDD(f)
1020         feat = tuple(itertools.chain(*feat))
1021         feat = tuple(itertools.chain(*feat))
1022         features.append(feat)
1023         i+=1
1024         self.SetStatusText("Extracting_features..."+str(i)
1025             +"_out_of_" +str(self._subset_test))
1026
1027
1028     def OnStartRedTest(self):
1029         # fit pca and std_scaler - save globally
1030         self.SetStatusText("Scaling_the_feature_vector...")
1031         self.features_test = self.min_max_scaler.transform(
1032             self.features_test)
1033         #self.SetStatusText("Applying PCA...")
1034         #self.features_test = self.pca.transform(self.
1035             features_test)
1036         self.SetStatusText("Extraction_completed.")
1037
1038     def OnStartExtractInputOld(self): #modify
1039         #features = []
1040         # Classification module: Test if size is not 28x28
1041         # make sure it goes through resizing and
1042         # morphological operations
1043         f = self.binary
1044         #f = cv2.cvtColor(f, cv2.COLOR_BGR2GRAY)
1045         #io.imsave("input_b4.jpg",f)
1046         #f = cv2.imread("input_b4.jpg", cv2.
1047                         CV_LOAD_IMAGE_GRAYSCALE)
1048         f = cv2.resize(f, (28,28), interpolation=cv2.
1049                         INTER_LANCZOS4)
1050         #cv2.imwrite("input.jpg",f)
1051         f = cv2.cv.fromarray(f)
1052         cv2.cv.Erode(f,f)

```

```

1049         f = np.asarray(f)
1050         #cv2.imwrite("after_erosion.jpg", f)
1051
1052         feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2, cv2
1053             .cv.CV_DIST_MASK_PRECISE)
1054         feat = tuple(itertools.chain(*feat))
1055         #features.append(feat)
1056
1057         #self.features_input = np.array(features)
1058         self.features_input = np.array(feat)
1059
1060         thread.start_new_thread(self.OnStartRedInput, ())
1061         #wx.CallAfter(self.OnStartRed)
1062
1063     def OnStartExtractInput(self): #modify
1064         #features = []
1065         # Classification module: Test if size is not 28x28
1066         # make sure it goes through resizing and
1067         # morphological operations
1068         f = self.binary
1069         f = Image.fromarray(f)
1070         f = trim(f)
1071         f = f.convert('L')
1072         f = np.array(f)
1073         f = img_as_ubyte(f)
1074         f = cv2.resize(f, (16,16), interpolation=cv2.
1075             INTER_LANCZOS4)
1076         #cv2.imwrite("input.jpg", f)
1077         #f = cv2.cv.fromarray(f)
1078         #cv2.cv.Erode(f, f)
1079         #f = np.asarray(f)
1080         #cv2.imwrite("after_erosion.jpg", f)
1081
1082         feat = ExtractDDD(f)
1083         feat = tuple(itertools.chain(*feat))
1084         feat = tuple(itertools.chain(*feat))
1085         #features.append(feat)
1086
1087         #self.features_input = np.array(features)
1088         self.features_input = np.array(feat)
1089
1090         #thread.start_new_thread(self.OnStartRedInput, ())
1091         wx.CallAfter(self.OnDoneExtractInput)
1092
1093     def OnStartRedInput(self): #modify
1094         # fit pca and std_scaler - save globally
1095         self.SetStatusText("Scaling_the_feature_vector...")
1096         self.features_input = self.min_max_scaler.transform(
1097             self.features_input)
1098         #self.SetStatusText ("Applying PCA...")
```

```
1094     #self.features_input = self.pca.transform(self.
1095         features_input)
1096     self.SetStatusText("Extraction_completed.")
1097     wx.CallAfter(self.OnDoneExtractInput)
1098
1099     def OnDoneExtractInput(self): #modify
1100         self.busyDlg = None
1101         self.SetStatusText("Classifying_image...")
1102         msg = "Please_wait_while_the_image_is_being_classified
1103             ..."
1104         self.busyDlg = wx.BusyInfo(msg)
1105         thread.start_new_thread(self.OnStartTestInput, ())
1106         #wx.CallAfter(self.OnStartTrain)
1107
1108     def OnStartTestInput(self):
1109         #self.pred_scaler = MinMaxScaler()
1110         self.pred_input = self.clf.predict(self.features_input
1111             )
1112         try:
1113             self.data = self.clf.predict_proba(self.
1114                 features_input)
1115             temp = None
1116             for item in self.data:
1117                 temp = item
1118             except AttributeError:
1119                 self.data = self.clf.decision_function(self.
1120                     features_input)
1121                 temp = None
1122                 for item in self.data:
1123                     temp = item
1124                 temp1 = None
1125                 for item in self.pred_input:
1126                     temp1 = str(int(item))
1127                     #self.pred_input = self.pred_scaler.fit_transform(
1128                         temp1)
1129                     self.pred_input = temp1
1130
1131                     self.data = temp
1132                     self.data = self.data.tolist()
1133
1134                     wx.CallAfter(self.OnDoneTestInput)
1135
1136     def OnDoneTestInput(self):
1137         self.label_num.SetLabel(self.pred_input)
1138         #print self.data
1139         #print self.pred_input
1140         #self.window_plot.data = self.data
1141
1142         #self.window_plot.GetParent().GetSizer().Show(self.
1143             window_plot)
```

```
1138     self.window_plot.GetParent().GetSizer().Layout()
1139     self.window_plot.draw_plot()
1140     self.busyDlg = None
1141     self.SetStatusText("Classification_done.")
1142
1143     def rowFormatterTrain(self, listItem, digit):
1144         if digit.predicted != digit.actual and self.classed ==
1145             True:
1146                 listItem.SetTextColour(wx.RED)
1147
1148     def rowFormatterTest(self, listItem, digit):
1149         if digit.predicted != digit.actual and self.tested ==
1150             True:
1151                 listItem.SetTextColour(wx.RED)
1152
1153 class WindowPlot(wx.ScrolledWindow):
1154
1155     def __init__(self, parent):
1156         wx.ScrolledWindow.__init__(self, parent)
1157         self.parent = parent
1158         self.gparent = self.parent.GetParent()
1159         self.ggparent = self.gparent.GetParent()
1160         #self.data =
1161             [0.02, 0.3, 0.45, 0.56, 0.67, 0.78, 0.89, 0.95, 0.21, 0.1]
1162         self.__set_properties()
1163         #self.__do_layout()
1164
1165     def __set_properties(self):
1166         self.SetScrollbars(20, 20, 50, 50)
1167         self.dpi = 100
1168         self.fig = Figure((5.0, 4.0), dpi=self.dpi,
1169                         tight_layout=True)
1170         self.canvas = FigureCanvas(self, -1, self.fig)
1171         self.axes = self.fig.add_subplot(111)
1172         #self.axes = plt
1173         self.axes.set_ylabel('Classes')
1174         self.axes.set_xlabel('Relative_Probability_of_Classes')
1175
1176     def draw_plot(self):
1177         self.axes.clear()
1178         self.axes.set_ylabel('Classes')
1179         self.axes.set_xlabel('Relative_Probability_of_Classes')
1180         self.axes.barrh(self.y_pos, self.ggparent.data, alpha
1181                         =0.4)
```

```

1181         self.canvas.draw()
1182
1183     """
1184     def _rowFormatterTrain(listItem, _customer):
1185         if _customer.amountOwed >_0:
1186             listItem.SetTextColour(wx.RED)
1187
1188     def _rowFormatterTest(listItem, _customer):
1189         if _customer.amountOwed >_0:
1190             listItem.SetTextColour(wx.RED)
1191     """
1192
1193 # end of class MyFrame

1 # entry.py
2
3 class Entry(object):
4     def __init__(self, image, actual, predicted):
5         self.image = image
6         self.actual = actual
7         self.predicted = predicted

1 import wx
2
3 class AnalyseDialog(wx.Dialog):
4     def __init__(self, parent):
5         # begin wxGlade: MyDialog.__init__
6         #kwds["style"] = wx.DEFAULT_DIALOG_STYLE
7         wx.Dialog.__init__(self, parent, style=wx.
                           DEFAULT_DIALOG_STYLE|wx.RESIZE_BORDER)
8         self.label_desc = wx.StaticText(self, wx.ID_ANY, _(""
                           Each_cell_in_the_table_below_shows_the_number_of_
                           images_classified._\n_Row_indicates_the_predicted_
                           class,_column_indicates_the_actual_class."), style=
                           wx.ALIGN_CENTRE)
9         self.grid_analyse = wx.grid.Grid(self, wx.ID_ANY, size
                           =(1, 1))
10        self.grid_analyse.DisableCellEditControl()
11        self.grid_analyse.AutoSize()
12
13        self.__set_properties()
14
15        self.__do_layout()
16        # end wxGlade
17
18    def __set_properties(self):
19        # begin wxGlade: MyDialog.__set_properties
20        self.SetTitle(_("Result_Analysis"))
21        self.label_descSetFont(wx.Font(12, wx.DEFAULT, wx.
                           NORMAL, wx.NORMAL, 0, ""))
22        self.grid_analyse.CreateGrid(10, 10)
23        self.grid_analyse.EnableEditing(0)

```

```
24     self.grid_analyse.EnableDragColSize(0)
25     self.grid_analyse.EnableDragRowSize(0)
26     self.grid_analyse.EnableDragGridSize(0)
27     self.grid_analyse.SetColLabelValue(0, _("0"))
28     self.grid_analyse.SetColSize(0, 5)
29     self.grid_analyse.SetColLabelValue(1, _("1"))
30     self.grid_analyse.SetColSize(1, 5)
31     self.grid_analyse.SetColLabelValue(2, _("2"))
32     self.grid_analyse.SetColSize(2, 5)
33     self.grid_analyse.SetColLabelValue(3, _("3"))
34     self.grid_analyse.SetColSize(3, 5)
35     self.grid_analyse.SetColLabelValue(4, _("4"))
36     self.grid_analyse.SetColSize(4, 5)
37     self.grid_analyse.SetColLabelValue(5, _("5"))
38     self.grid_analyse.SetColSize(5, 5)
39     self.grid_analyse.SetColLabelValue(6, _("6"))
40     self.grid_analyse.SetColSize(6, 5)
41     self.grid_analyse.SetColLabelValue(7, _("7"))
42     self.grid_analyse.SetColSize(7, 5)
43     self.grid_analyse.SetColLabelValue(8, _("8"))
44     self.grid_analyse.SetColSize(8, 5)
45     self.grid_analyse.SetColLabelValue(9, _("9"))
46     self.grid_analyse.SetColSize(9, 5)
47
48     self.grid_analyse.SetRowLabelValue(0, _("0"))
49     self.grid_analyse.SetRowSize(0, 5)
50     self.grid_analyse.SetRowLabelValue(1, _("1"))
51     self.grid_analyse.SetRowSize(1, 5)
52     self.grid_analyse.SetRowLabelValue(2, _("2"))
53     self.grid_analyse.SetRowSize(2, 5)
54     self.grid_analyse.SetRowLabelValue(3, _("3"))
55     self.grid_analyse.SetRowSize(3, 5)
56     self.grid_analyse.SetRowLabelValue(4, _("4"))
57     self.grid_analyse.SetRowSize(4, 5)
58     self.grid_analyse.SetRowLabelValue(5, _("5"))
59     self.grid_analyse.SetRowSize(5, 5)
60     self.grid_analyse.SetRowLabelValue(6, _("6"))
61     self.grid_analyse.SetRowSize(6, 5)
62     self.grid_analyse.SetRowLabelValue(7, _("7"))
63     self.grid_analyse.SetRowSize(7, 5)
64     self.grid_analyse.SetRowLabelValue(8, _("8"))
65     self.grid_analyse.SetRowSize(8, 5)
66     self.grid_analyse.SetRowLabelValue(9, _("9"))
67     self.grid_analyse.SetRowSize(9, 5)
68     # end wxGlade
69
70 def __do_layout(self):
71     # begin wxGlade: MyDialog.__do_layout
72     sizer_23 = wx.BoxSizer(wx.VERTICAL)
73     sizer_23.Add(self.label_desc, 1, wx.ALL | wx.EXPAND |
```

```
    ALIGN_CENTER_VERTICAL, 5)
74 sizer_23.Add(self.grid_analyse, 3, wx.ALL | wx.EXPAND,
    5)
75 self.SetSizer(sizer_23)
76 sizer_23.Fit(self)
77 self.Layout()
78 # end wxGlade
79
80 class AboutDialog(wx.Dialog):
81     def __init__(self, *args, **kwds):
82         # begin wxGlade: ResultDialog.__init__
83         kwds["style"] = wx.DEFAULT_DIALOG_STYLE
84         wx.Dialog.__init__(self, *args, **kwds)
85         self.label_1 = wx.StaticText(self, wx.ID_ANY, _("This
    program is written by Chong Yong Shean\nCompiled
    using Python\nGUI created using wxPython with
    the aid of wxGlade"))
86         self.button_1 = wx.Button(self, wx.ID_OK, "")
87
88         self.__set_properties()
89         self.__do_layout()
90
91         self.Bind(wx.EVT_BUTTON, self.OnClick, self.button_1)
92
93     # end wxGlade
94
95     def __set_properties(self):
96         # begin wxGlade: ResultDialog.__set_properties
97         self.setTitle(_("About the Author"))
98         _icon = wx.EmptyIcon()
99         _icon.CopyFromBitmap(wx.Bitmap("C:\\Documents and
            Settings\\YongShean\\My Documents\\icon.png", wx.
            BITMAP_TYPE_ANY))
100        #self.setIcon(_icon)
101        self.label_1SetFont(wx.Font(14, wx.MODERN, wx.NORMAL,
            wx.BOLD, 0, ""))
102    # end wxGlade
103
104    def __do_layout(self):
105        # begin wxGlade: ResultDialog.__do_layout
106        sizer_3 = wx.BoxSizer(wx.VERTICAL)
107        sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
108        sizer_4.Add(self.label_1, 0, wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
            0)
109        sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
110        sizer_3.Add(self.button_1, 0, wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
            0)
111        self.SetSizer(sizer_3)
112        sizer_3.Fit(self)
```

```

113         self.Layout()
114         # end wxGlade
115
116     def OnClick(self, event):    # wxGlade: ResultDialog.<
117         event_handler>
117         self.Destroy()
118         event.Skip()

```

A-0-8 Training and Testing of Combined Dataset

```

1 # train_dataset_improved.py
2 # Training and testing with stratified k-fold cross validation
3
4 from sklearn.preprocessing import MinMaxScaler
5 from sklearn.cross_validation import StratifiedKFold
6 import numpy as np
7 import time
8 from sklearn.pipeline import Pipeline
9 from sklearn import svm
10 from sklearn.externals import joblib
11 from sklearn.metrics import classification_report,
11     accuracy_score, confusion_matrix, precision_recall_curve
12 from time import strftime
13 import matplotlib.pyplot as plt
14
15 _kernel = 'rbf'
16 _c = 2.8
17 _gamma = 0.0073
18 _tol = 0.001
19 _max_iter = -1
20 _dual = False
21 _deg = 5
22
23 print "Loading_training_set..."
24 features = np.load('features.npy')
25 labels = np.load('labels.npy').flatten()
26
27 start_run = time.time()
28 min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
29
30 class_weights = np.bincount(labels)
31 d = dict()
32 for i,w in zip(range(len(class_weights)+1),class_weights):
33     d[i] = w
34 # Set weights here #
35 if _kernel=='linear':
36     clf = svm.LinearSVC(C=_c, dual=_dual, tol=_tol,
36         class_weight=d)
37 elif _kernel=='poly':
38     clf = svm.SVC(kernel='poly',C=_c, gamma=_gamma, tol=_tol,
38         max_iter=_max_iter, degree=_deg, probability=True,
38         class_weight=d)

```

```

39 elif _kernel=='rbf':
40     clf = svm.SVC(kernel='rbf', C=_c, gamma=_gamma, tol=_tol,
41                     max_iter=_max_iter, probability=True, class_weight=d)
41 elif _kernel=='sigmoid':
42     clf = svm.SVC(kernel='sigmoid', C=_c, gamma=_gamma, tol=
43                     _tol, max_iter=_max_iter, probability=True,
44                     class_weight=d)
43
44 print "Splitting_training_and_testing_set_using_Stratified_K-
45 fold..."
45 skf = StratifiedKFold(labels, n_folds=3)
46 k=1
47 for train_index, test_index in skf:
48     features_train, features_test = features[train_index],
49                     features[test_index]
50     labels_train, labels_test = labels[train_index], labels[
51         test_index]
52
53     print "Training_dataset..."
54     scaling_svm = Pipeline([('scaler', min_max_scaler), ('svm',
55                             clf)])
56     scaling_svm.fit(features_train, labels_train)
57     joblib.dump(scaling_svm, 'scaling_svm_new_fold'+str(k)+'.
58                 pkl')
59     end_run = time.time()
60     print "Training_completed."
61     print "Fold_no."+str(k)+":_Time_elapsed", round(end_run-
62         start_run, 4), "seconds"
63
64     print "Testing_dataset..."
65     print("Detailed_classification_report_(Training):")
66     print()
67     print("The_model_is_trained_on_the_full_development_set.")
68     print("The_scores_are_computed_on_the_full_evaluation_set.")
69     print()
70     print("Confusion_matrix")
71     conf_arr = confusion_matrix(y_true,y_pred)
72     norm_conf = []
73     for i in conf_arr:
74         a = 0
75         tmp_arr = []
76         a = sum(i, 0)
77         for j in i:
78             tmp_arr.append(float(j)/float(a))

```

```

79         norm_conf.append(tmp_arr)
80
81     fig = plt.figure()
82     plt.clf()
83     ax = fig.add_subplot(111)
84     ax.set_aspect(1)
85     res = ax.imshow(np.array(norm_conf), cmap=plt.cm.jet,
86                     interpolation='nearest')
87
88     width = len(conf_arr)
89     height = len(conf_arr[0])
90
91     cb = fig.colorbar(res)
92     plt.savefig('confusion_matrix_train'+str(k)+'.png', format=
93                 'png')
94     print()
95     print("Precision-Recall_Curve")
96     from sklearn import preprocessing
97     lb = preprocessing.LabelBinarizer()
98     y_true = lb.fit_transform(y_true)
99     y_pred = lb.fit_transform(y_pred)
100    precision, recall, thresholds = precision_recall_curve(
101        y_true.flatten(), scaling_svm.predict_proba(
102            features_train), pos_label=1)
103    plt.plot(thresholds, precision, 'r', thresholds, recall, 'b')
104    plt.xlabel('Threshold')
105    plt.ylabel('Precision/Recall')
106    plt.title('Precision/Recall_vs_Threshold')
107    plt.savefig('precision_recall_threshold_train'+str(k)+'.png',
108                format='png')
109    print()
110    np.save('labels_train_predicted_new_fold'+str(k)+'.npy',
111            y_pred)
112
113    print("Detailed_classification_report_(Testing) :")
114    print()
115    print("The_model_is_trained_on_the_full_development_set.")
116    print("The_scores_are_computed_on_the_full_evaluation_set."
117          )
118    print()
119    y_true, y_pred = labels_test, scaling_svm.predict(
120        features_test)
121    print(classification_report(y_true, y_pred))
122    print()
123    print "Accuracy:", accuracy_score(y_true, y_pred)
124    print()
125    print("Confusion_matrix")
126    conf_arr = confusion_matrix(y_true, y_pred)
127    norm_conf = []
128    for i in conf_arr:

```

```

122     a = 0
123     tmp_arr = []
124     a = sum(i, 0)
125     for j in i:
126         tmp_arr.append(float(j)/float(a))
127     norm_conf.append(tmp_arr)
128
129     fig = plt.figure()
130     plt.clf()
131     ax = fig.add_subplot(111)
132     ax.set_aspect(1)
133     res = ax.imshow(np.array(norm_conf), cmap=plt.cm.jet,
134                     interpolation='nearest')
135
136     width = len(conf_arr)
137     height = len(conf_arr[0])
138
139     cb = fig.colorbar(res)
140     plt.savefig('confusion_matrix_test'+str(k)+'.png', format='
141     png')
142     print()
143     print("Precision-Recall_Curve")
144     from sklearn import preprocessing
145     lb = preprocessing.LabelBinarizer()
146     y_true = lb.fit_transform(y_true)
147     y_pred = lb.fit_transform(y_pred)
148     precision, recall, thresholds = precision_recall_curve(
149         y_true.flatten(), scaling_svm.predict_proba(features_test)
150         , pos_label=1)
151     plt.plot(thresholds, precision, 'r', thresholds, recall, 'b
152         ')
153     plt.xlabel('Threshold')
154     plt.ylabel('Precision/Recall')
155     plt.title('Precision/Recall_vs_Threshold')
156     plt.savefig('precision_recall_threshold_test'+str(k)+'.png'
157         , format='png')
158     print()
159     np.save('labels_test_predicted_new_fold'+str(k)+'.npy',
160             y_pred)
161     k+=1

```

A-0-9 Shared scripts - required for the above scripts

Directional Distance Distribution

```

1 from skimage import io
2 import numpy as np
3
4 def ExtractDDD(img):
5     wb = [[[0 for i in range(16)] for j in range(16)] for k in
6           range(16)]

```

```

7   for n in range(16):
8       for m in range(16):
9           if img[n][m]<128:
10              for k in range(16): # East
11                  wb[n][m][0]+=1
12                  if m+k<=15:
13                      if img[n][m+k]>=128:
14                          break
15                  else:
16                      if img[n][m+k-16]>=128:
17                          break
18              for k in range(16): # West
19                  wb[n][m][4]+=1
20                  if m-k<=15:
21                      if img[n][m-k]>=128:
22                          break
23                  else:
24                      if img[n][m-k-16]>=128:
25                          break
26              for k in range(16): # South
27                  wb[n][m][6]+=1
28                  if n+k<=15:
29                      if img[n+k][m]>=128:
30                          break
31                  else:
32                      if img[n+k-16][m]>=128:
33                          break
34              for k in range(16): # North
35                  wb[n][m][2]+=1
36                  if n-k<=15:
37                      if img[n-k][m]>=128:
38                          break
39                  else:
40                      if img[n-k-16][m]>=128:
41                          break
42              for k in range(16): # South East
43                  wb[n][m][7]+=1
44                  if m+k<=15 and n+k<=15:
45                      if img[n+k][m+k]>=128:
46                          break
47                  if m+k<=15 and n+k>=16:
48                      if img[n+k-16][m+k]>=128:
49                          break
50                  if m+k>=16 and n+k<=15:
51                      if img[n+k][m+k-16]>=128:
52                          break
53                  elif m+k>=16 and n+k>=16:
54                      if img[n+k-16][m+k-16]>=128:
55                          break
56              for k in range(16): # North West
57                  wb[n][m][3]+=1

```

```

58             if m-k<=15 and n-k<=15:
59                 if img[n-k][m-k]>=128:
60                     break
61             if m-k<=15 and n-k>=16:
62                 if img[n-k-16][m-k]>=128:
63                     break
64             if m-k>=16 and n-k<=15:
65                 if img[n-k][m-k-16]>=128:
66                     break
67         elif m-k>=16 and n-k>=16:
68             if img[n-k-16][m-k-16]>=128:
69                 break
70     for j,k in zip(range(16),range(16)): # South
71         West
72         wb[n][m][5] +=1
73         if n+k<=15:
74             if m-k<=15:
75                 if img[n+k][m-k]>=128:
76                     break
77             else:
78                 if img[n+k][m-k-16]>=128:
79                     break
80         else:
81             if m-k<=15:
82                 if img[n+k-16][m-k]>=128:
83                     break
84             else:
85                 if img[n+k-16][m-k-16]>=128:
86                     break
87     for j,k in zip(range(16),range(16)): # North
88         East
89         wb[n][m][1] +=1
90         if n-k<=15:
91             if m+k<=15:
92                 if img[n-k][m+k]>=128:
93                     break
94             else:
95                 if img[n-k][m+k-16]>=128:
96                     break
97             else:
98                 if m+k<=15:
99                     if img[n-k-16][m+k]>=128:
100                     break
101             else:
102                 if img[n-k-16][m+k-16]>=128:
103                     break
104         if img[n][m]>=128:
105             for k in range(16): # East
106                 wb[n][m][8] +=1
107                 if m+k<=15:
108                     if img[n][m+k]<128:

```

```

107                         break
108
109             else:
110                 if img[n][m+k-16]<128:
111                     break
112
113             for k in range(16): # West
114                 wb[n][m][12]+=1
115                 if m-k<=15:
116                     if img[n][m-k]<128:
117                         break
118
119             else:
120                 if img[n][m-k-16]<128:
121                     break
122
123             for k in range(16): # South
124                 wb[n][m][14]+=1
125                 if n+k<=15:
126                     if img[n+k][m]<128:
127                         break
128
129             else:
130                 if img[n+k-16][m]<128:
131                     break
132
133             for k in range(16): # North
134                 wb[n][m][10]+=1
135                 if n-k<=15:
136                     if img[n-k][m]<128:
137                         break
138
139             else:
140                 if img[n-k-16][m]<128:
141                     break
142
143             for k in range(16): # South East
144                 wb[n][m][15]+=1
145                 if m+k<=15:
146                     if n+k<=15:
147                         if img[n+k][m+k]<128:
148                             break
149
150             else:
151                 if img[n+k-16][m+k]<128:
152                     break
153
154             else:
155                 if img[n+k-16][m+k-16]<128:
156                     break
157

```

```

158                     break
159             for j,k in zip(range(16),range(16)): # South
160                 West
161                 wb[n][m][13] +=1
162                 if n+k<=15:
163                     if m-k<=15:
164                         if img[n+k][m-k]<128:
165                             break
166                         else:
167                             if img[n+k][m-k-16]<128:
168                                 break
169                         else:
170                             if m-k<=15:
171                                 if img[n+k-16][m-k]<128:
172                                     break
173                                 else:
174                                     if img[n+k-16][m-k-16]<128:
175                                         break
176             for j,k in zip(range(16),range(16)): # North
177                 East
178                 wb[n][m][9] +=1
179                 if n-k<=15:
180                     if m+k<=15:
181                         if img[n-k][m+k]<128:
182                             break
183                         else:
184                             if img[n-k][m+k-16]<128:
185                                 break
186                             else:
187                                 if m+k<=15:
188                                     if img[n-k-16][m+k]<128:
189                                         break
190                                         else:
191                                             if img[n-k-16][m+k-16]<128:
192                                                 break
193
194     wb_new = [[0 for i in range(4)] for j in range(4)]
195
196     wb = np.array(wb)
197
198     n=0
199     m=0
200     for i in range(0,13,4):
201         for j in range(0,13,4):
202             temp = wb[i:i+4,j:j+4]
203             temp = temp.mean(0)
204             wb_new[n][m] = temp.mean(0)
205             m+=1
206             n+=1
207             m=0
208     wb_new = np.array(wb_new)

```

```
207  
208     return wb_new
```

Function Definitions

```
1 import numpy as np  
2 from PIL import Image, ImageChops  
3 from skimage import io  
4 from skimage.filter import threshold_otsu  
5 import cv2  
6 from skimage import img_as_float, img_as_ubyte  
7 import itertools  
8 from sklearn import preprocessing  
9 from sklearn.decomposition import RandomizedPCA  
10 from sklearn.preprocessing import StandardScaler  
11 import gc  
12 from sklearn.externals import joblib  
13 from sklearn import svm  
14 from ddd import *  
15  
16 def cv_to_pil(im):  
17     #im = img_as_float(im)  
18     im = Image.fromarray(im)  
19  
20     return im  
21  
22 def pil_to_cv(im):  
23     im = im.convert('L')  
24     im = np.array(im)  
25     #im = img_as_ubyte(im)  
26  
27     return im  
28  
29 def cv_to_skimage(im):  
30     return img_as_float(im)  
31  
32 def skimage_to_cv(im):  
33     return img_as_ubyte(im)  
34  
35 def pil_to_skimage(im):  
36     im = im.convert('L')  
37     im = np.array(im)  
38  
39     return im  
40  
41 def skimage_to_pil(im):  
42     return Image.fromarray(im)  
43  
44 def binarize(img):  
45     thresh = threshold_otsu(img)  
46     binary = img > thresh  
47     binary = binary.astype(int)*255
```

```
48
49     return binary
50     # Skimage image
51
52 def window(iterable, size=2):
53     i = iter(iterable)
54     win = []
55     for e in range(0, size):
56         win.append(next(i))
57     yield win
58     for e in i:
59         win = win[1:] + [e]
60     yield win
61
62 def trim(im):
63     bg = Image.new(im.mode, im.size, im.getpixel((0,0)))
64     diff = ImageChops.difference(im, bg)
65     diff = ImageChops.add(diff, diff, 2.0, -100)
66     bbox = diff.getbbox()
67     if bbox:
68         return im.crop(bbox)
69
70 def ExtractField(im):
71     pw, ph = im.size
72     cw = int(0.6*pw) #0.64
73     ch = int(0.4*ph) #0.4
74     rw = int(0.5*cw) #0.52
75     rh = int(0.4*ch) #0.44
76     region = (cw, ch, cw+rw, ch+rh)
77     area = im.crop(region)
78     area = area.rotate(90)
79
80     area = pil_to_cv(area)
81
82     i=0
83     min_i = 999
84     max_i = -1
85     cnt_b=0
86     cnt_w=0
87
88     for f in window(area):
89         i+=1
90         f = np.array(f)
91         black = False
92         for num in f:
93             for j in num:
94                 if j.any()==0:
95                     black = True
96                     break
97             if black==True:
98                 if i<min_i:
```

```

99             min_i = i
100         elif i>max_i:
101             max_i = i
102
103     region = (0,min_i,rh,max_i)
104     area = cv_to_pil(area)
105     area = area.crop(region)
106     area = area.rotate(-90)
107     #area = trim(area)
108     #area = pil_to_cv(area)
109
110     return area
111
112 def dropfall(tile, direction='top-left'):
113     from numpy import fliplr, flipud
114     height, width = tile.shape
115     cut = np.zeros(height)
116
117     if direction=='bottom-left':
118         tile = flipud(tile)
119     elif direction=='bottom-right':
120         tile = fliplr(tile)
121         tile = flipud(tile)
122     elif direction=='top-right':
123         tile = fliplr(tile)
124
125     for row in range(1, height+1):
126         found_candidate=0
127         found_start=0
128         candidate_x=0
129         candidate_y=0
130         start_x=0
131         start_y=0
132
133         # Check the "right three fourths minus some" of each
134         for col in range(int(round(width/4)), width):
135             pres_pix = tile[row-1, col-1]
136             prev_pix = tile[row-1, col-2]
137             next_pix = tile[row-1, col]
138             if pres_pix==0 and next_pix>0:
139                 found_candidate = 1
140                 candidate_x = col
141                 candidate_y = row
142             if found_candidate==1 and pres_pix==0 and prev_pix>0:
143                 found_start = 1
144                 start_x = candidate_x + 1
145                 start_y = candidate_y
146
147             if found_start==1:
148                 break
149

```

```

150      # Start the drop fall!
151      # Start defining the cut
152      cut = start_x*np.ones(height)
153      row = start_y
154      col = start_x
155      path_tile = tile
156
157      cut_point_x = 0
158      cut_point_y = 0
159
160      while row<height and col<width:
161          if path_tile[row, col-1]==255:
162              row+=1
163          elif path_tile[row, col]==255:
164              row+=1
165              col+=1
166          elif path_tile[row, col-2]==255:
167              row+=1
168              col-=1
169          elif path_tile[row-1, col]==255:
170              col+=1
171          elif path_tile[row-1, col-2]==255:
172              col-=1
173          else:
174              if tile[row-1,col-1]==255:
175                  cut_point_y = row #point at which the cutting into
                           the black begins
176                  cut_point_x = col
177                  row+=1
178                  path_tile[row-1, col-1] = 500 # This marks where the
                           path has been
179                  cut[row-1] = col
180
181      width1 = max(cut)-1
182      min_cut = min(cut)
183      width2 = width - min_cut + 1
184      tile1 = 255 * np.ones((height, width1))
185      tile2 = 255 * np.ones((height, width2))
186
187      for i in range(height):
188          x_cut = int(cut[i])
189          tile1[i, 1:(x_cut-1)] = tile[i, 1:(x_cut-1)]
190          tile2[i, (x_cut - min_cut + 1):width2] = tile[i, x_cut:
                           width]
191
192      if direction=='bottom-left':
193          tile1 = flipud(tile1)
194          tile2 = flipud(tile2)
195      elif direction=='bottom-right':
196          tile1 = fliplr(tile1)
197          tile1 = flipud(tile1)

```

```
198     tile2 = fliplr(tile2)
199     tile2 = flipud(tile2)
200     tile1, tile2 = tile2, tile1
201 elif direction=='top-right':
202     tile1 = fliplr(tile1)
203     tile2 = fliplr(tile2)
204     tile1, tile2 = tile2, tile1
205
206     tile1 = tile1.astype(int)
207     tile2 = tile2.astype(int)
208
209 try:
210     io.imsave('tile1.jpg',tile1)
211     io.imsave('tile2.jpg',tile2)
212
213     tile1 = Image.open('tile1.jpg')
214     tile2 = Image.open('tile2.jpg')
215
216     tile1 = trim(tile1)
217     tile2 = trim(tile2)
218
219     tile1 = pil_to_cv(tile1)
220     tile2 = pil_to_cv(tile2)
221 except ValueError:
222     print "Dropfall_error!"
223     return tile1, tile2
224
225 return tile1, tile2
226
227 def Segmentation(im):
228     #im = pil_to_cv(im)
229
230     #im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
231     ret, img = cv2.threshold(im, 128, 255, cv2.THRESH_OTSU)
232     ret, img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY_INV)
233     contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL
234                                         , cv2.CHAIN_APPROX_SIMPLE)
235     img_lst = []
236     prop_min = []
237
238     for i in range(len(contours)):
239         prop_min.append([i, min([x for [x,y] in contours[i]])])
240
241     prop_min = np.array(prop_min)
242     # sort array with regards to 2nd column
243     prop_min = prop_min[prop_min[:,1].argsort()]
244     img_list = [contours[x] for [x,y] in prop_min]
245     img_list_pos = prop_min
246
247     i=0
248     y_arr = []
```

```

248     for currentContour in img_list:
249         x,y,w,h = cv2.boundingRect(currentContour)
250         #cv2.rectangle(im, (x,y), (x+w,y+h), (0,255,0),2)
251         letter = im[y:y+h,x:x+w]
252         if len(letter)>3 and len(letter[0])<0.7*len(im[0]): # CONSIDER REMOVE THIS CONDITION
253             cv2.imwrite('img'+str(i)+'.jpg',letter)
254             img_lst.append(letter)
255             y_arr.append(y)
256             i+=1
257
258     heights = []
259     widths = []
260
261     for f in img_lst:
262         height, width = f.shape
263         heights.append(height)
264         widths.append(width)
265
266     mean_height = np.mean(heights)
267     mean_width = np.mean(widths)
268
269     return img_lst, mean_height, mean_width, y_arr
270
271 def FeatureExtraction(img_lst):
272     features = []
273
274     for f in img_lst:
275         height, width = f.shape
276         f = cv2.resize(f, (16,16), interpolation=cv2.INTER_LANCZOS4)
277         feat = ExtractDDD(f)
278         feat = list(itertools.chain(*feat))
279         feat.append([height,width,width/height])
280         feat = tuple(itertools.chain(*feat))
281         features.append(feat)
282
283     #for i in range(92):
284     #    features.append([0 for x in range(784)])
285     features = np.array(features)
286     #features = np.nan_to_num(features)
287     #features = features[0:8]
288     return features
289
290 def Recognise(input):
291     clf_single = joblib.load("clf_min_max_scaled.pkl")
292
293     single = clf_single.predict(input)
294     try:
295         confidence = clf_single.predict_proba(input)
296     except AttributeError:

```

```
297     confidence = clf_single.decision_function(input)
298
299     output = []
300     """
301     for [num_single, num_double] in zip(single, double):
302         if num_single not in range(10):
303             if num_double >= 21:
304                 output.append(num_double - 11)
305             else:
306                 output.append(num_double - 10)
307         else:
308             output.append(num_single)
309     """
310     return single.astype(int), confidence
311
312 def GetPILBitmap(npimage):
313     height, width = npimage.shape
314     pilimage = Image.fromarray(npimage)
315     rgb = pilimage.convert('RGB')
316     return rgb
```