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 : ____________________________

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

LIST OF FIGURES

LIST OF TABLES

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 sufficient 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, payeeś name field, courtesy amount field, legal amount field, and signature pane. This project focuses on handwritten character recognition on courtesy amount field.

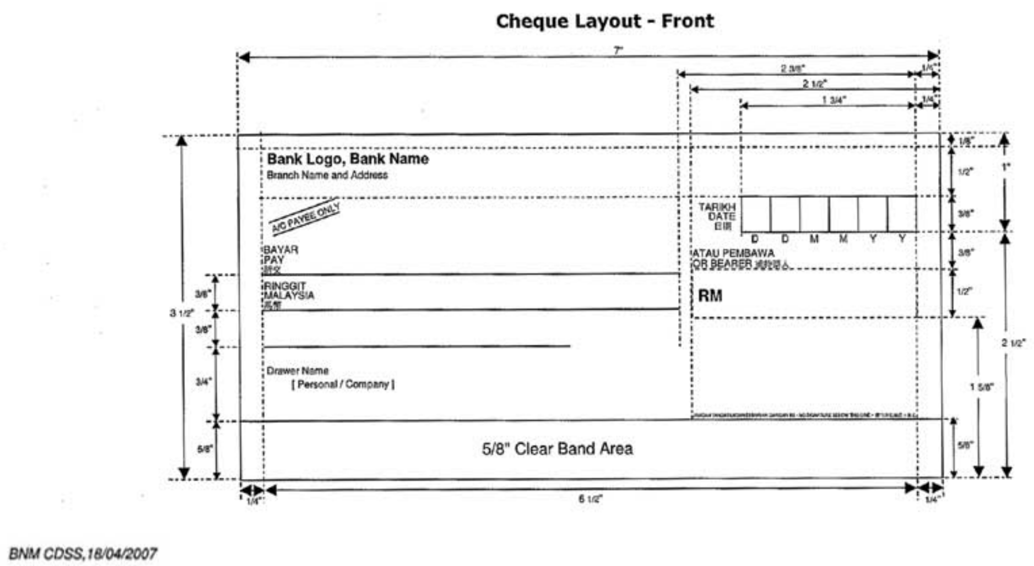


Figure 1.1: Cheque layout specified by Bank Negara Malaysia, 2007

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:

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

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:

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

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.

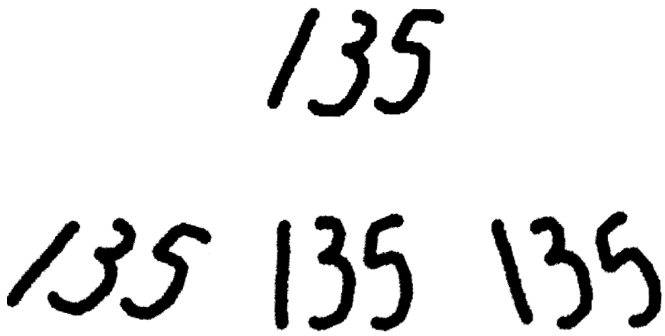


Figure 2.1: An example of slant correction. Picture adopted from (Cheung 1998).

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 extraction 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.

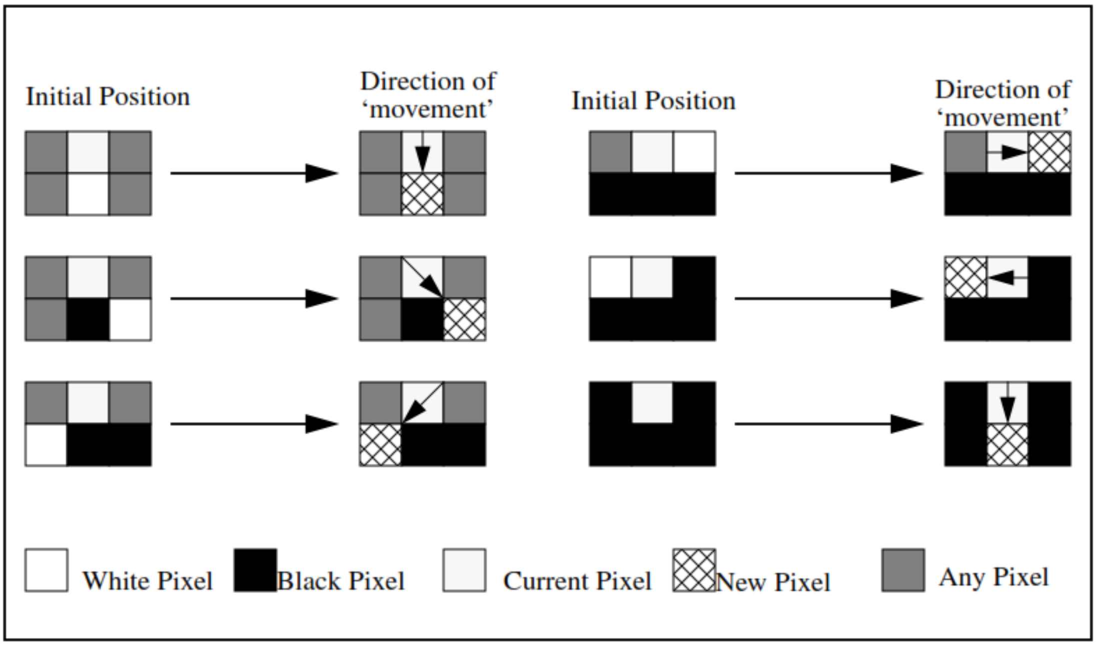


Figure 2.2: Pixel-to-pixel path construction based on the current position. Picture adopted from (Khan 1998).

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).

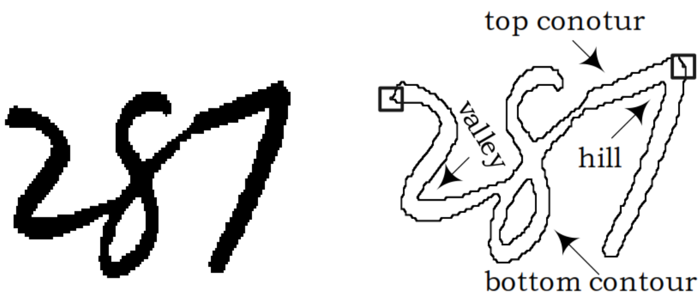


Figure 2.3: An example of touching numerals and its contours. Picture adopted from citepyunlei04.

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%.

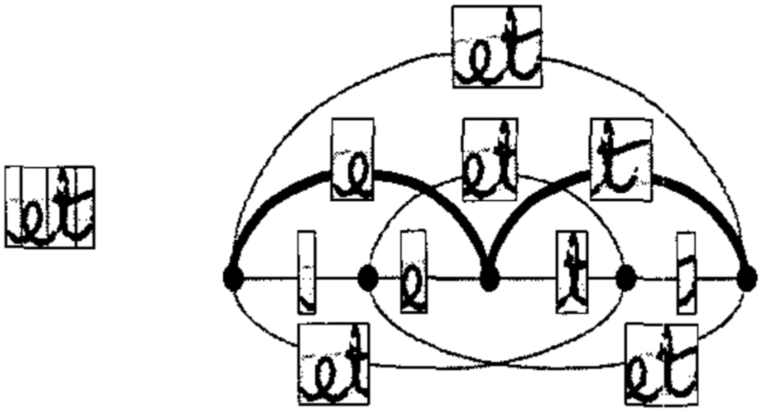


Figure 2.4: Applying over-segmentation on a connected component and find the best segmentation path based on word likelihood. Picture adopted from (Y.H. et al. 2003).

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)

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

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

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

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)

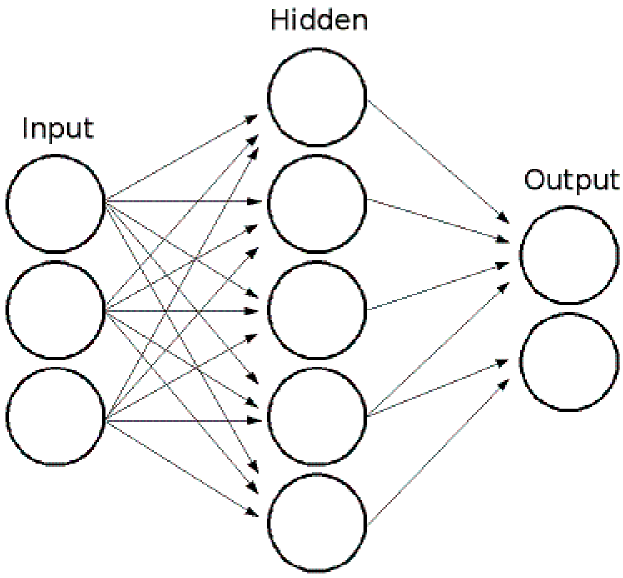


Figure 2.5: A typical feed-forward neural network architecture used in backpropagation. Picture adopted from (*Neural Networks* 2011).

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 $x(i), i = 1, 2, ..., N$, compute all the predictions and compare with the actual class $y(i)$ it belongs. The error is $(prediction - actual)$.
- *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 jth neuron in the rth layer. For $r = 1, 2, ..., L$ (layers of neurons) and $j = 1, 2, ..., kr$ (number of neurons in the rth layer), $\mathbf{w}_j^r(new) = \mathbf{w}_j^r(old) + \Delta\mathbf{w}_j^r$

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?

Kanellopoulus et al [Kanellopoulus, 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 $x_i = 1, 2, ...N$, be the feature vectors of the training set, X. The goal is to find a hyperplane, that is

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

where $w = [w_1, w_2, ..., w_l]^T$ 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.

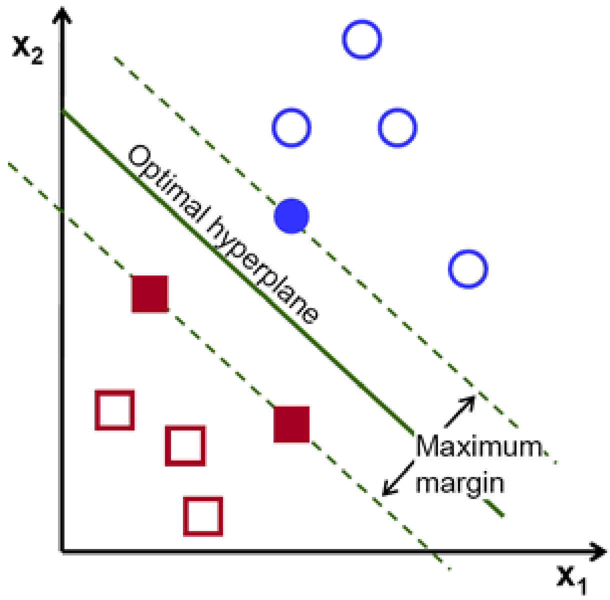


Figure 2.6: The optimal hyperplane that gives the maximum margin between classes. Picture adopted from (*Introduction to SVM* 2011).

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

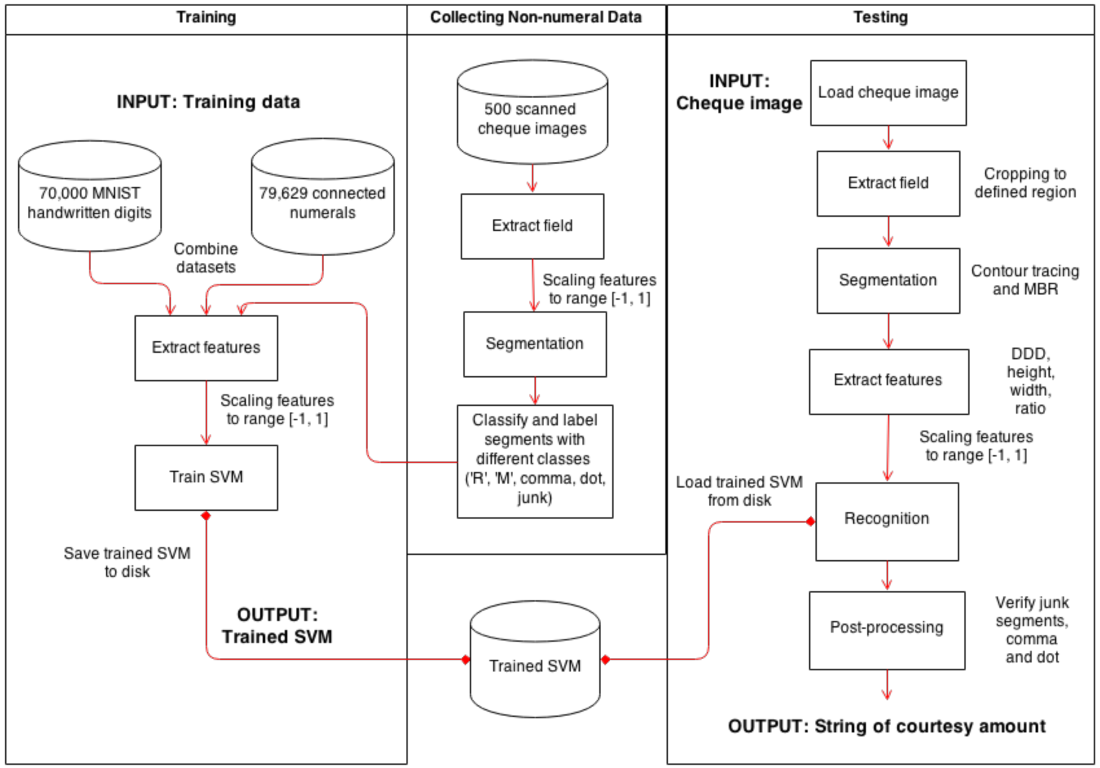


Figure 3.1: The overall process and the components involved in training and testing module.

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:

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

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

where length of bounding box $= 0.5x$

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

where 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 (Chaaban 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

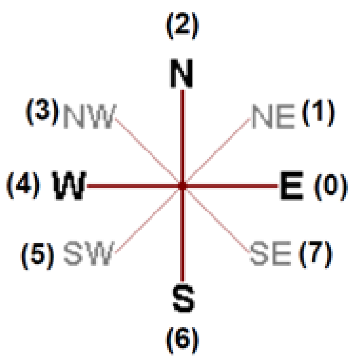


Figure 3.3: The 8-directions and its corresponding digit that indicates each direction.

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.

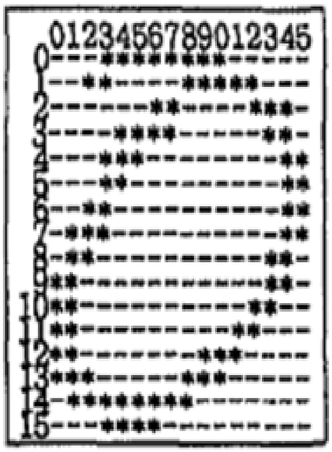


Figure 3.4: The pixels of a zero digit image in binary form.

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 16x16x16 = 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.

$$x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.4}$$

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

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.

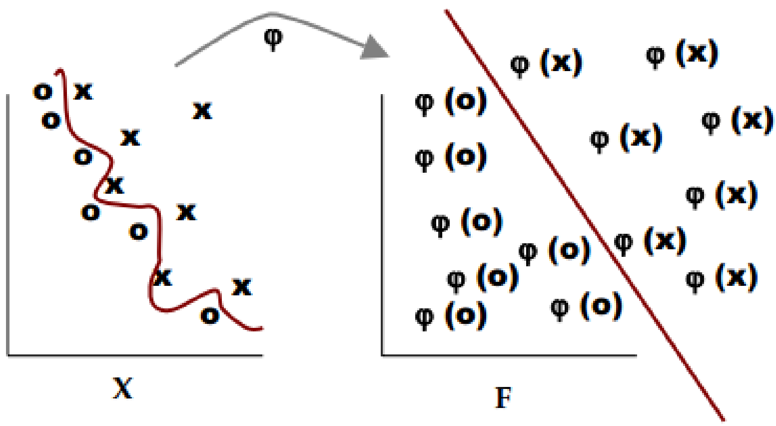


Figure 3.5: Transforming x to $\phi(x)$ to become linearly separable.

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

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

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

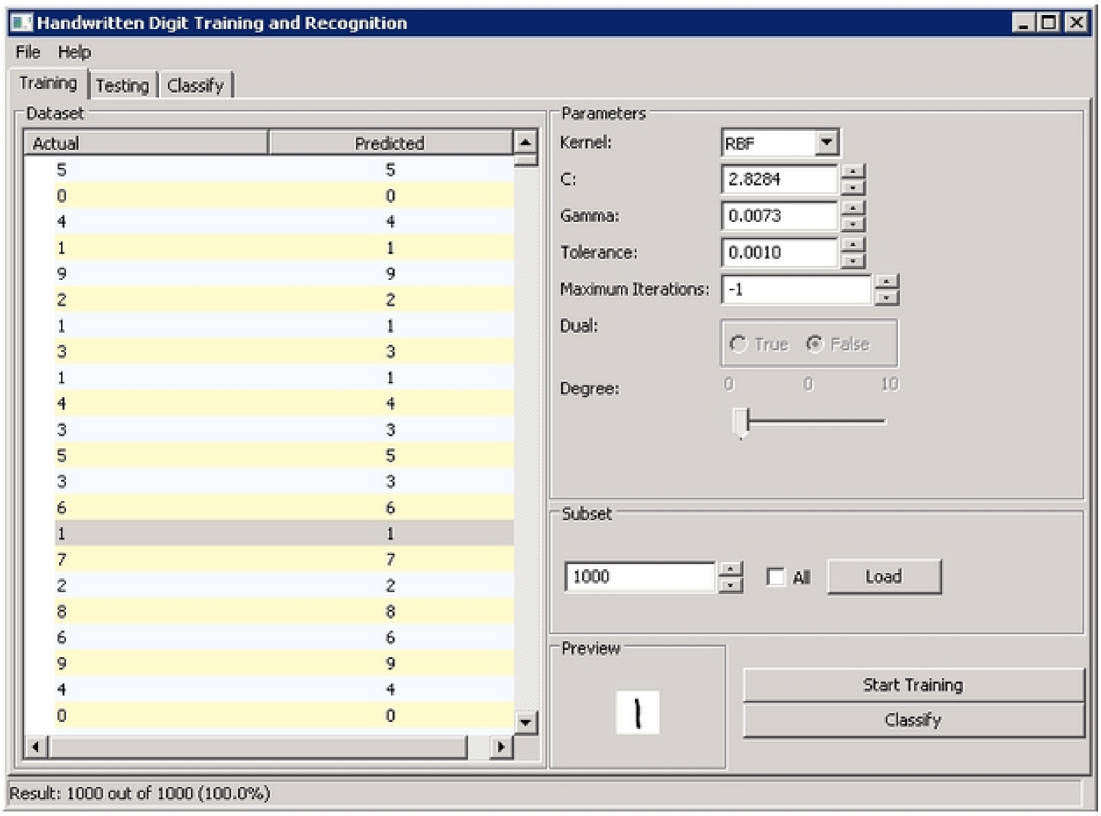


Figure 3.6: Screenshot of the *Training* tab.

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 SVMâĂę and the file will be saved in the selected directory.

Testing Tab

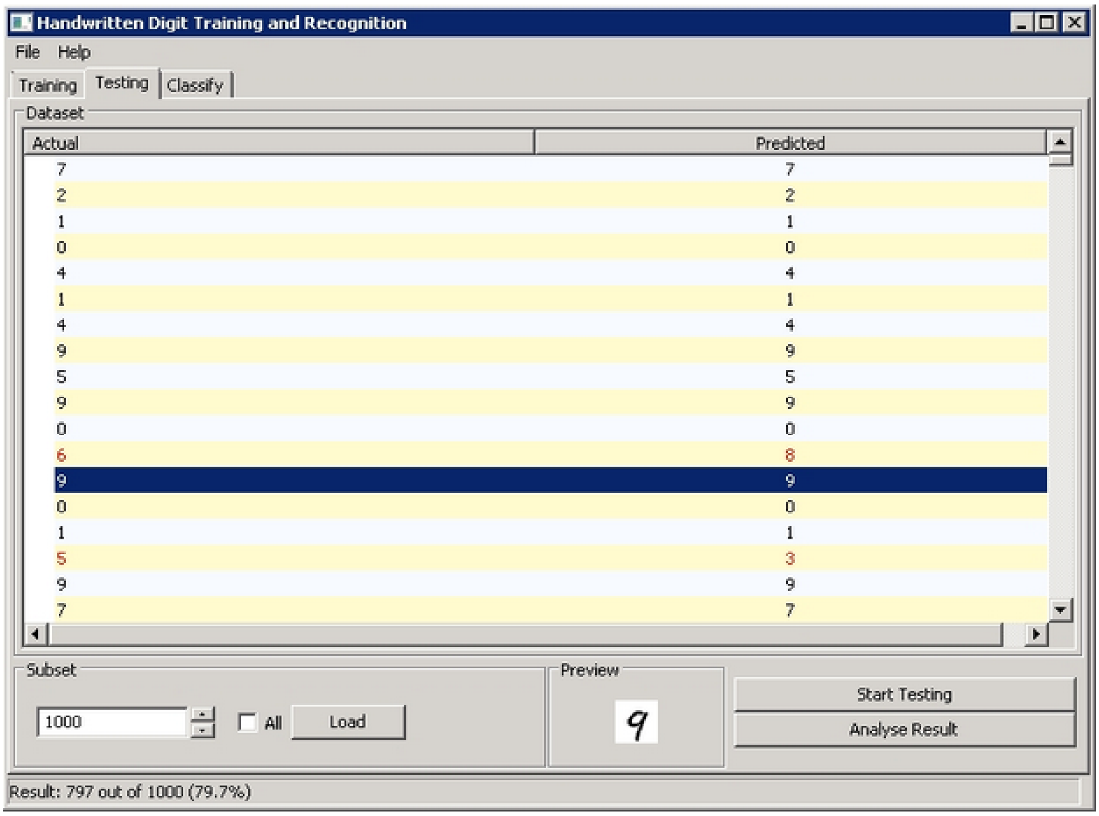
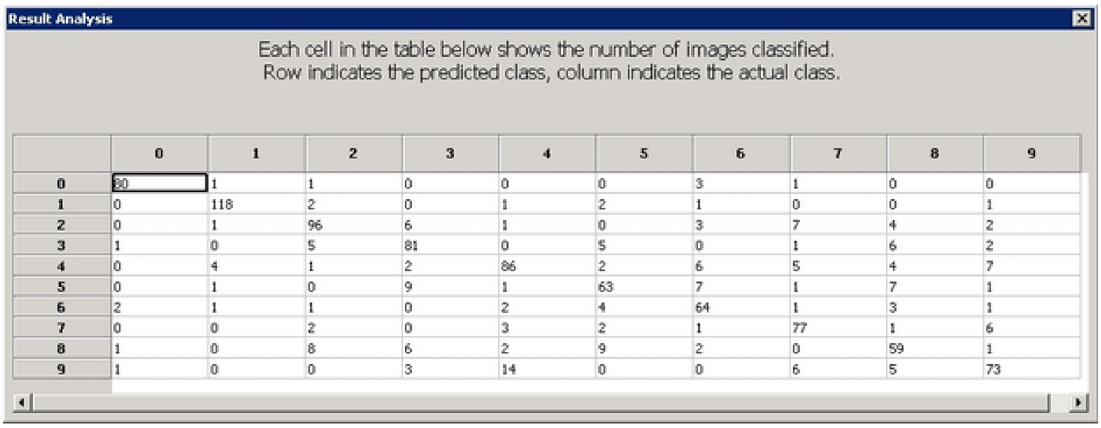


Figure 3.7: Screenshot of the *Testing* tab.

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.



Result Analysis

Each cell in the table below shows the number of images classified.
Row indicates the predicted class, column indicates the actual class.

	0	1	2	3	4	5	6	7	8	9
0	80	1	1	0	0	0	3	1	0	0
1	0	118	2	0	1	2	1	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

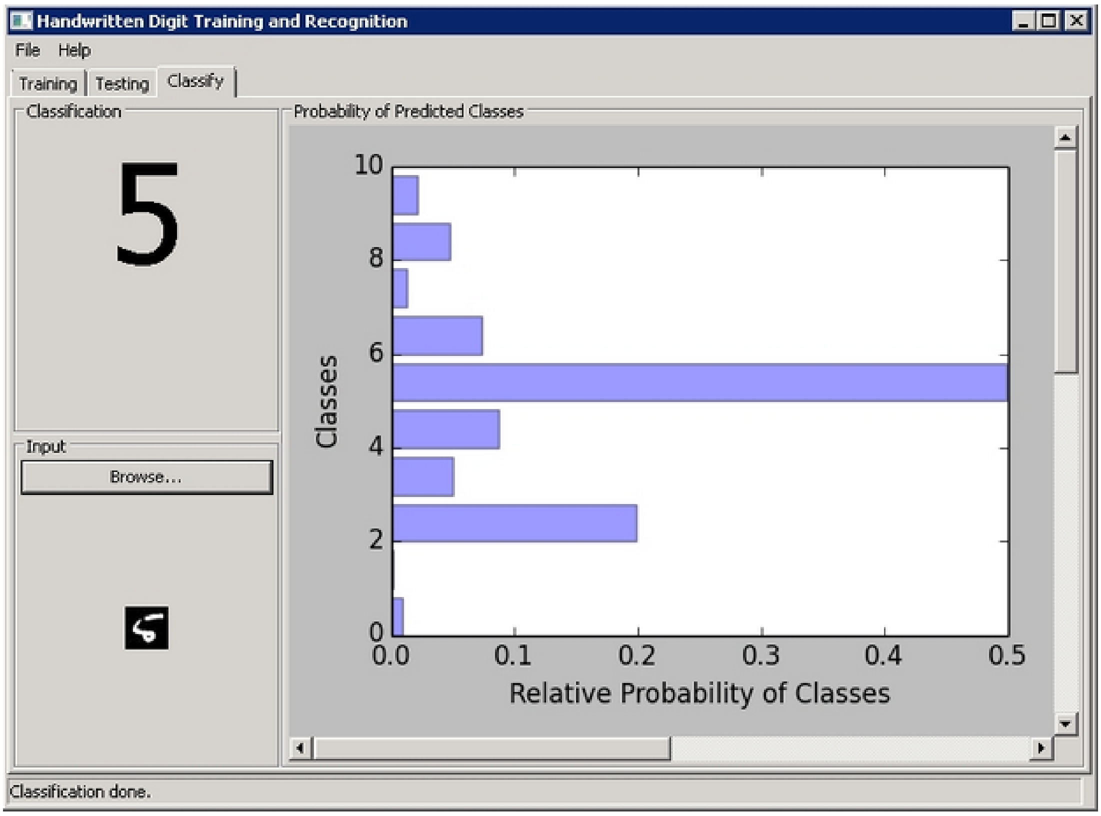


Figure 3.9: Screenshot of the *Classify* tab.

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

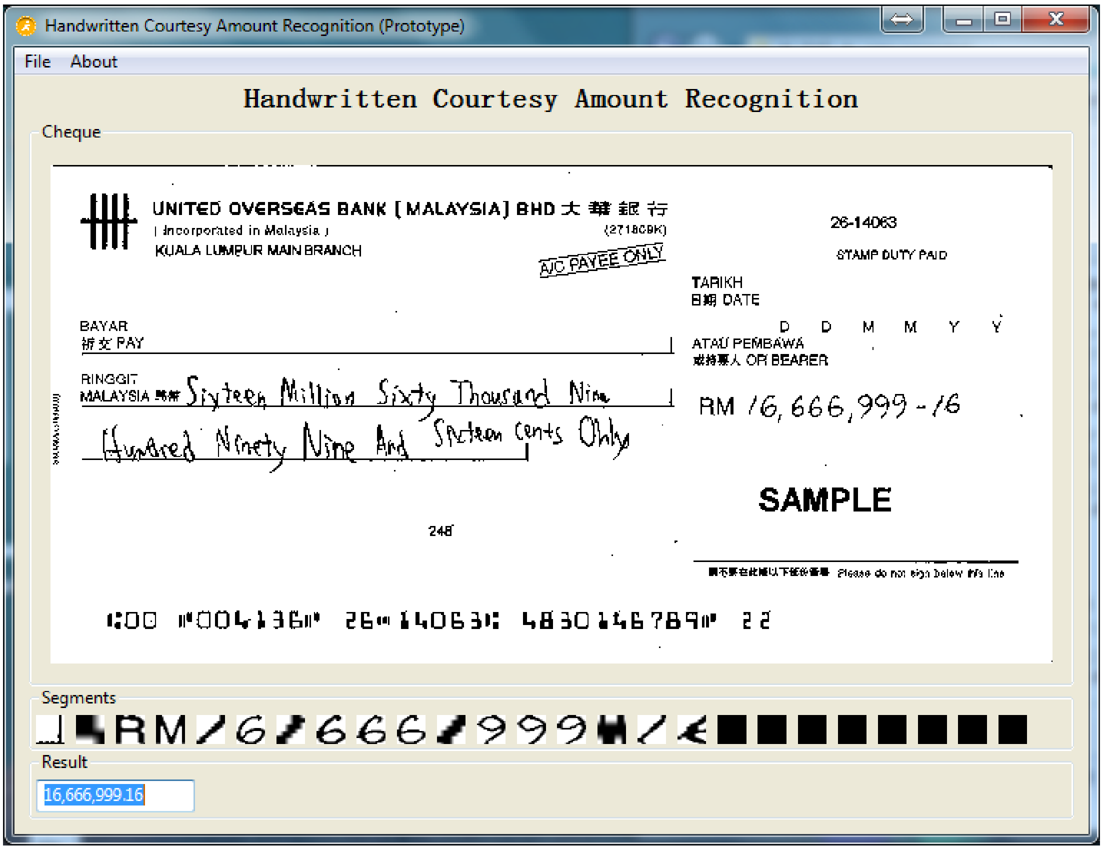


Figure 3.10: Screenshot of the *Handwritten Courtesy Amount Recognition* demonstration GUI.

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.

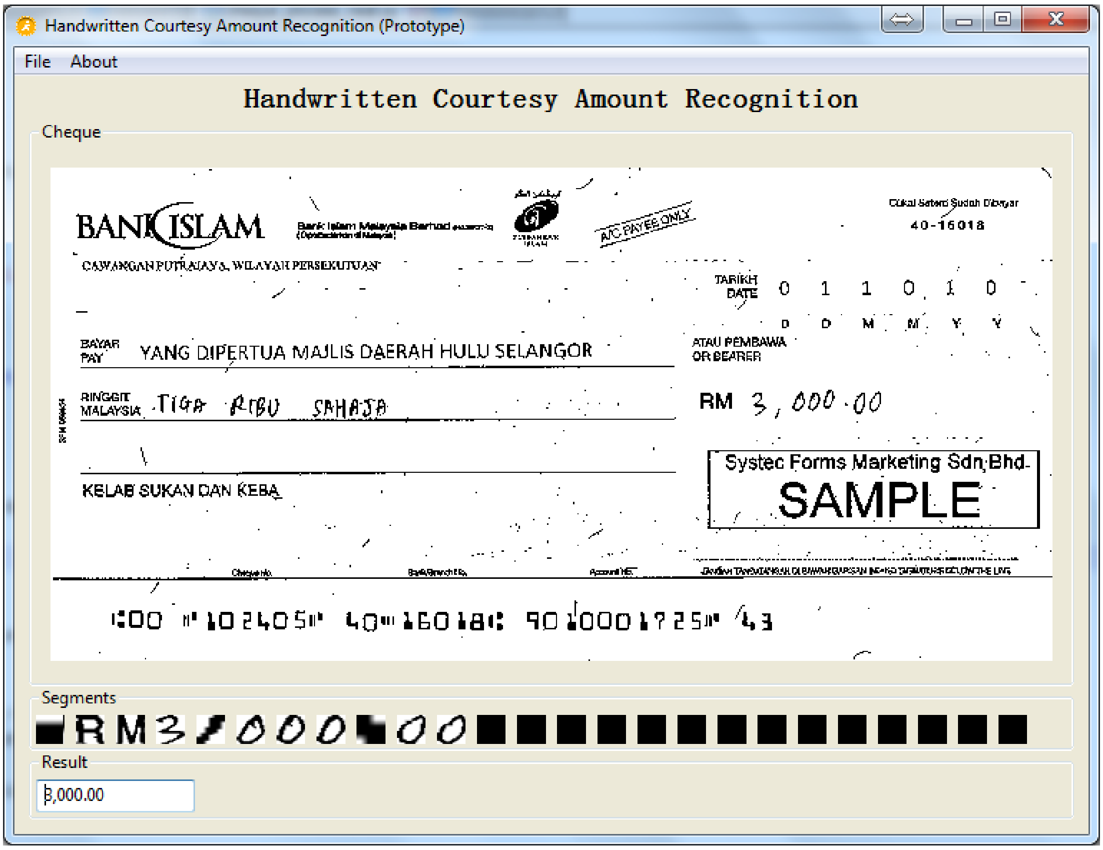


Figure 3.11: A successful example of recognition.

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.

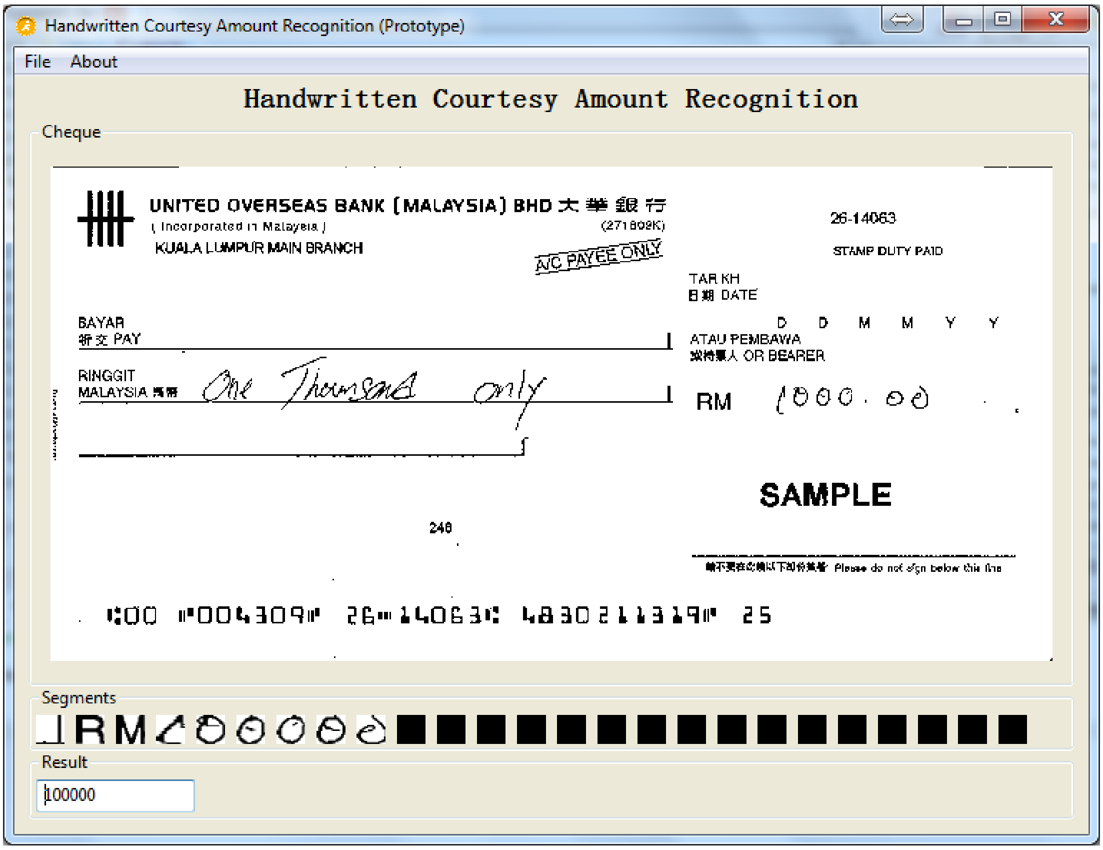


Figure 3.12: Recognition error: A decimal point is missed.

- 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.

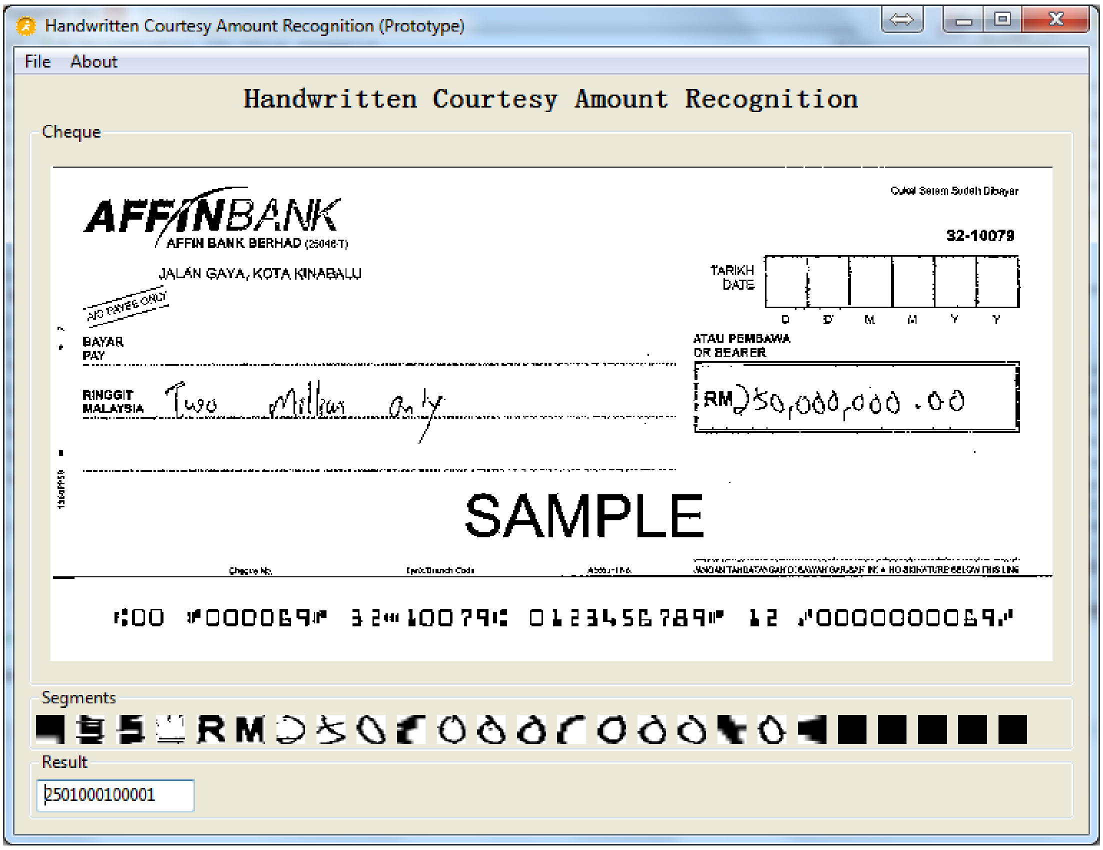


Figure 3.13: Recognition error: Classify ',' as '1'.

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.

Attempt 1	Attempt 2
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.

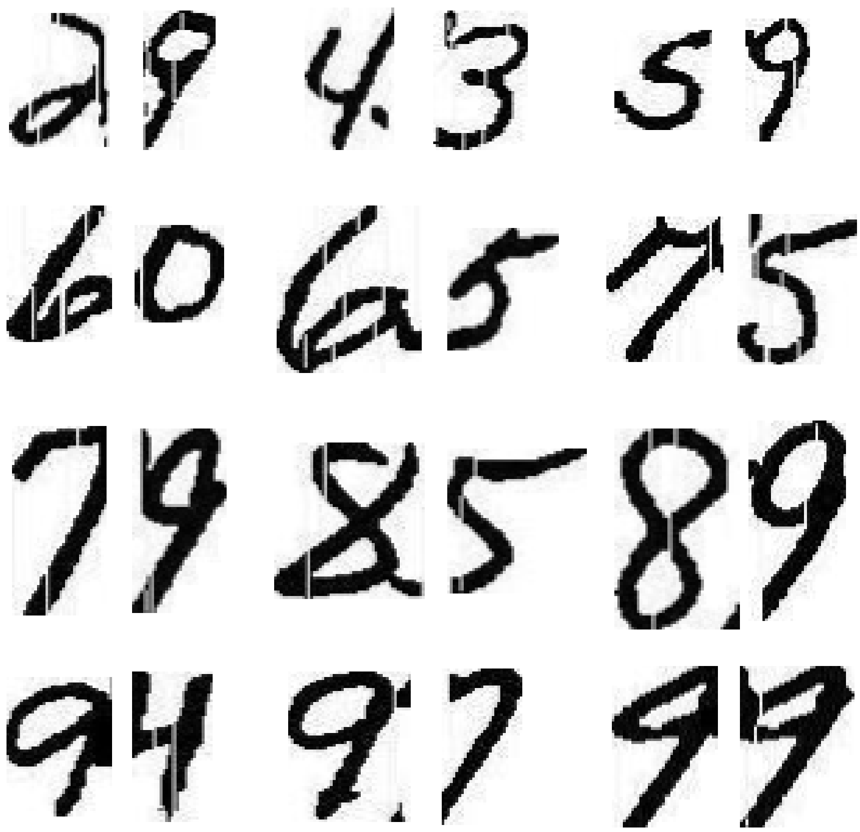


Figure 4.1: Examples of successful dropfall segmentation.

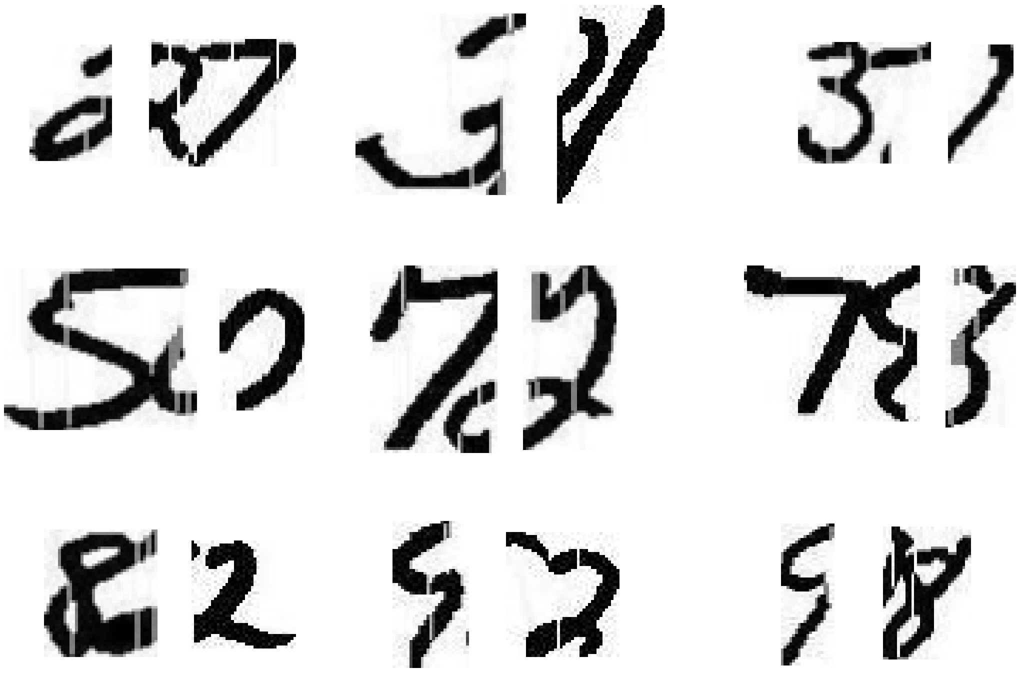


Figure 4.2: Examples of unsuccessful dropfall segmentation.

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 perfomance of the system is evaluated by testing with 100 cheques.

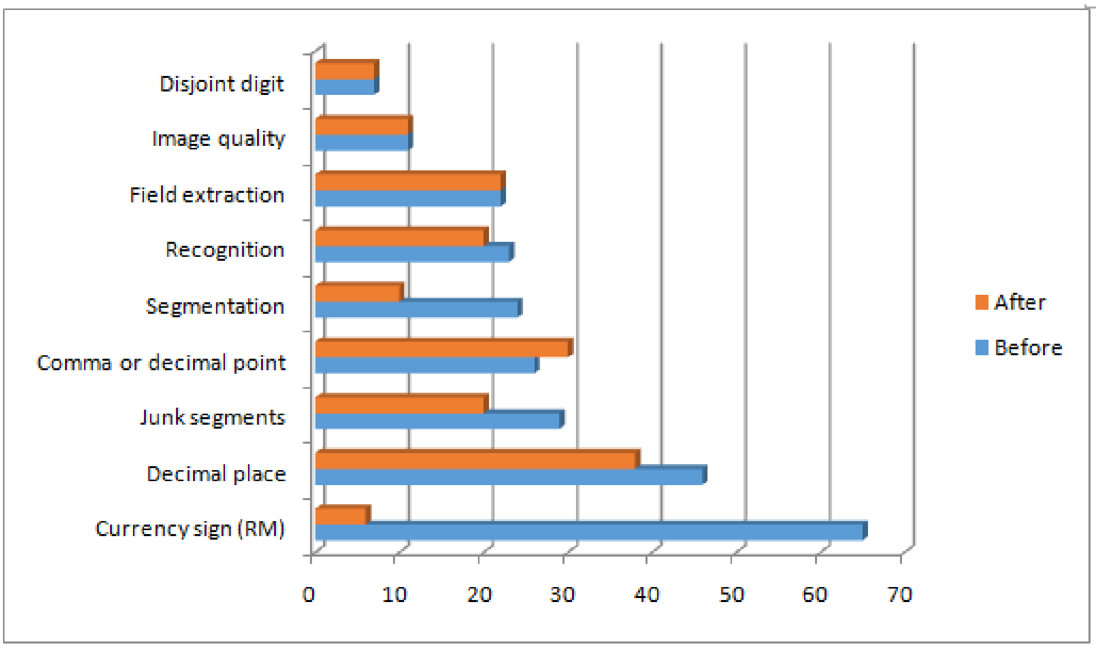


Figure 4.3: Frequency of occurrence of error on the recognition of 100 cheques before and after training.

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.

Chaaban, 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/fsps/en/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 hand-writing 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

```
# grid_search.py

print(__doc__)

import numpy as np

from time import time
from operator import itemgetter
from scipy.stats import randint as sp_randint

from sklearn.preprocessing import MinMaxScaler
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn import svm
from sklearn.externals import joblib

_kernel = ’rbf’
_c = 2.8
_gamma = 0.0073
_tol = 0.001
_max_iter = -1
_dual = False
_deg = 5

# get some data
features = np.load(’features.npy’)
labels = np.load(’labels.npy’)
X, y = features, labels.flatten()

# build a classifier
min_max_scaler = MinMaxScaler(feature_range=(-1, 1))

X = min_max_scaler.fit_transform(X)
samp = np.random.choice(len(X), 5000)
X = X[samp]
y = y[samp]
class_weights = np.bincount(y)
d = dict()
for i,w in zip(range(len(class_weights)+1),class_weights):
   d[i] = w
# Set weights here #
clf = svm.SVC(tol=_tol, max_iter=_max_iter, probability=True,
   class_weight=d)
#scaling_svm = Pipeline([("scaler", min_max_scaler), ("svm",
   clf)])
```

```

# Utility function to report best scores
def report(grid_scores, n_top=10):
    top_scores = sorted(grid_scores, key=itemgetter(1), reverse=True)[:n_top]
    for i, score in enumerate(top_scores):
        print("Model␣with␣rank:␣{0}".format(i + 1))
        print("Mean␣validation␣score:␣{0:.3f}␣(std:␣{1:.3f})".format(
              score.mean_validation_score,
              np.std(score.cv_validation_scores)))
        print("Parameters:␣{0}".format(score.parameters))
        print("")

# use a full grid over all parameters
param_grid = {'kernel':('linear', 'rbf'),
              'C':[2.2, 2.5, 2.8, 3.0],
              'gamma':[0.00675, 0.007, 0.00725, 0.0075]}

# run grid search
if __name__ == '__main__':
   print "Running␣grid␣search..."
   grid_search = GridSearchCV(clf, param_grid=param_grid, n_jobs=-1, verbose=1)
   start = time()
   grid_search.fit(X, y)

   print("GridSearchCV␣took␣%.2f␣seconds␣for␣%d␣candidate␣parameter␣settings."
      % (time() - start, len(grid_search.grid_scores_)))
   report(grid_search.grid_scores_)
```

A-0-3 Experiment 2

```
# Experiment: Dropfall segmentation on connected numerals

from __future__ import print_function
log = open("c:\\dropfall_log2.txt", "w")

from functions import * # file must be in the directory

#img = Image.open('cn('+str(i)+').jpg')
#area = ExtractField(img)  #only for cheque inputs

direct = ['top-left','top-right','bottom-left','bottom-right']
clf = joblib.load("clf_min_max_scaled.pkl")

for i in range(100,199):
   print ("Processing␣image␣number␣..."+str(i+1), file=log)
   img = cv2.imread('test␣('+str(i+1)+').jpg', cv2.CV_LOAD_IMAGE_GRAYSCALE)
   arr_conf = []
```

```
    for k in range(4):
        tile1, tile2 = dropfall(img, direction=direct[k])
        try:
            if tile1.shape[0]<10 or tile1.shape[1]<10 or tile2.shape[1]<10 or tile2.shape[0]<10:
                print ("Dropfall␣error!", file=log)
                continue
        except:
            print ("Dropfall␣error!", file=log)
            continue
        img_lst = []
        img_lst.append(tile1)
        img_lst.append(tile2)
        #img_lst = np.array(img_lst)
        input = FeatureExtraction(img_lst)
        pred = clf.predict(input)
        confidence = clf.predict_proba(input)
        arr_conf.append([confidence[0][pred[0]]+confidence[1][pred[1]]])
    arr_conf = np.array(arr_conf)
    index = arr_conf.argmax()
    tile1, tile2 = dropfall(img, direction=direct[index])
    try:
        if tile1.shape[0]<10 or tile1.shape[1]<10 or tile2.shape[1]<10 or tile2.shape[0]<10:
            print ("Dropfall␣error!", file=log)
            continue
    except:
        print ("Dropfall␣error!", file=log)
        continue
    img_list = []
    img_list.append(tile1)
    img_list.append(tile2)
    input = FeatureExtraction(img_list)
    pred = clf.predict(input)
    print ("Predicted:␣"+str(pred[0])+str(pred[1])+"␣with␣probability␣"+str(clf.predict_proba(input[0])[0][pred[0]])+"␣and␣"+str(clf.predict_proba(input[1])[0][pred[1]]), file=log)
    if i<=10:
        io.imsave('cn_'+str(i)+'_tile1.jpg',tile1)
        io.imsave('cn_'+str(i)+'_tile2.jpg',tile2)
    else:
        io.imsave('cn_'+str(i+1)+'_tile1.jpg',tile1)
        io.imsave('cn_'+str(i+1)+'_tile2.jpg',tile2)
```

A-0-4 Experiment 3

```
# Experiment: Test on 100 cheques

from functions import *
from skimage.io import imread, imsave
```

```
from PIL import Image

#from __future__ import print_function
#log = open("c:\\cheque_exp_log.txt", "w")

total_output = []
for i in range(1,101):
    print "Processing cheque number ", str(i), "..."
    chq = Image.open('chq ('+str(i)+').jpeg')
    img = ExtractField(chq)
    img = pil_to_cv(img)

    height, width = img.shape

    img_lst, mean_height, mean_width, y_arr = Segmentation(img)
    datalr_y = [[y] for y in y_arr]
    datalr_X = [[x] for x in range(1,len(img_lst)+1)]

    datalr_X = np.array(datalr_X)
    datalr_y = np.array(datalr_y)
    from sklearn.linear_model import LinearRegression

    regr = LinearRegression()
    regr.fit(datalr_X, datalr_y)

    clf = joblib.load("scaling_svm_overall.pkl")
    input = FeatureExtraction(img_lst)
    pred = clf.predict(input)
    confidence = clf.predict_proba(input)

    dists = regr.predict(datalr_y)-datalr_y
    dists = abs(dists)
    max_dist = dists.max()

    output = []
    for item in pred:
        if item<=9:
            output.append(item)
        if item>=10 and item<=19:
            output.append('0'+str(item-10))
        if item>=20 and item<=109:
            if item==21:
                output.append(-1)
                #print "Junk"
            else:
                output.append(str(item-10))
        if item>=110 and item<=113:
            if item==110:
                output.append(",")
            if item==111:
                output.append(".")
```

```
        if item==112:
            output.append(-1)
            #print "R"
        if item==113:
            output.append(-1)
            #print "M"

i=0

for dist in dists:
    height, width = img_lst[i].shape
    if i==0 and pred[i]==111:
        #print "Pos", i, ": probably a junk"
        output[i] = -1
    if i>0 and pred[i]==111 and pred[i-1]>=110:
        #print "Pos", i, ": probably a junk"
        output[i] = -1
    if i>0 and i<pred.shape[0]-2 and pred[i]<110 and pred[i+1]==112 and pred[i+2]==113:
        output[i] = -1
    if pred[i]==21:
        if confidence[i][21]<0.7:
            # check if it's a dot or comma
            if confidence[i][111] > confidence[i][110]:
            # if prob of it being a dot is higher than of comma
                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:
                    #print "Pos", i, ": probably a dot"
                    output[i] = '.'
                if i>0 and pred[i-1]<110: # extra condition
                    #print "Pos", i, ": probably a junk due to non-number before dot"
                    output[i] = -1
            else:
                if dist>0.2*max_dist:
                    #print "Pos", i, ": probably being a comma"
                    output[i] = ','
                else:
                    #print "Pos", i, ": probably being a 1"
                    output[i] = 1
    if pred[i]==1 or pred[i]==110:
        if dist>0.2*max_dist:
            #print "Pos", i, ": probably being a comma"
            output[i] = ','
        else:
            #print "Pos", i, ": probably being a 1"
            output[i] = 1
    i+=1
```

```
    output = filter(lambda a: a != -1, output)
    index_of_first_number = -1
    for k in range(len(output)):
        if output[k]<100 and output[k]!=-1:
            index_of_first_number = k
            break
    for n in range(index_of_first_number):
        if output[n]==',' or output[n]=='.':
            output[n] = -1

    output = filter(lambda a: a != -1, output)
    cnt_comma = 0
    cnt_dot = 0
    m = -1
    for k in range(len(output)):
        if output[k]==',':
            cnt_comma += 1
            m = k
        if output[k]=='.':
            cnt_dot += 1
    if cnt_comma==1 and cnt_dot==0:
        output[m] = '.'
    total_output.append(output)
print "Completed."
```

A-0-5 Experiment 4

```
# train_dataset.py

from sklearn.preprocessing import MinMaxScaler
from sklearn.cross_validation import StratifiedKFold
import numpy as np
import time
from sklearn.pipeline import Pipeline
from sklearn import svm
from sklearn.externals import joblib

_kernel = 'rbf'
_c = 2.8
_gamma = 0.0073
_tol = 0.001
_max_iter = -1
_dual = False
_deg = 5

print "Loading␣training␣set..."
features = np.load('features.npy')
labels = np.load('labels.npy')

start_run = time.time()
min_max_scaler = MinMaxScaler()

```

```
class_weights = np.bincount(labels.flatten())
d = dict()
for i,w in zip(range(len(class_weights)+1),class_weights):
   d[i] = w
# Set weights here #
if _kernel=='linear':
    clf = svm.LinearSVC(C=_c, dual=_dual, tol=_tol)
elif _kernel=='poly':
    clf = svm.SVC(kernel='poly',C=_c, gamma=_gamma, tol=_tol,
       max_iter=_max_iter, degree=_deg, probability=True)
elif _kernel=='rbf':
    clf = svm.SVC(kernel='rbf', C=_c, gamma=_gamma, tol=_tol,
       max_iter=_max_iter, probability=True)
elif _kernel=='sigmoid':
    clf = svm.SVC(kernel='sigmoid', C=_c, gamma=_gamma, tol=
       _tol, max_iter=_max_iter, probability=True)

print "Splitting␣training␣and␣testing␣set␣using␣Stratified␣K-
   fold..."
skf = StratifiedKFold(labels.flatten(), n_folds=3)
k=1
for train_index, test_index in skf:
   features_train, features_test = features[train_index],
      features[test_index]
   labels_train, labels_test = labels[train_index], labels[
      test_index]

   print "Training␣dataset..."
   scaling_svm = Pipeline([("scaler", min_max_scaler), ("svm",
       clf)])
   scaling_svm.fit(features_train, labels_train)
   joblib.dump(scaling_svm, 'scaling_svm_fold'+str(k)+'.pkl')
   end_run = time.time()
   print "Training␣completed."
   print "Fold␣no."+str(k)+":␣Time␣elapsed", round(end_run-
      start_run,4), "seconds"

   print "Testing␣dataset..."
   labels_train_predicted = scaling_svm.predict(features_train
      )
   correct = 0
   for i in range(len(labels_train)):
      if labels_train_predicted[i]==labels_train[i]:
         correct+=1
   print "Accuracy:␣", str(correct/float(len(labels_train)))
   np.save('labels_train_predicted_fold'+str(k)+'.npy',
      labels_train_predicted)
   labels_test_predicted = scaling_svm.predict(features_test)
   correct = 0
   for i in range(len(labels_test)):
      if labels_test_predicted[i]==labels_test[i]:
```

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

A-0-6 Handwritten Courtesy Amount Recognition System GUI

```
#!/usr/bin/env python
# -*- coding: CP936 -*-
#
# generated by wxGlade 0.6.8 (standalone edition) on Fri Nov 22 10:50:18 2013
#

import wx
import os
from functions import *
from cheque_process_neat import *

# begin wxGlade: dependencies
import gettext
# end wxGlade

# begin wxGlade: extracode
# end wxGlade


class MyFrame(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: MyFrame.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)

        self.output = _("")

        # Menu Bar
        self.frame_1_menubar = wx.MenuBar()
        wxglade_tmp_menu = wx.Menu()
        self.open = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _("Open Cheque.."), "", wx.ITEM_NORMAL)
        wxglade_tmp_menu.AppendItem(self.open)
        wxglade_tmp_menu.AppendSeparator()
        self.Exit = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _("Exit"), "", wx.ITEM_NORMAL)
        wxglade_tmp_menu.AppendItem(self.Exit)
        self.frame_1_menubar.Append(wxglade_tmp_menu, _("File"))
        wxglade_tmp_menu = wx.Menu()
        self.author = wx.MenuItem(wxglade_tmp_menu, wx.ID_ANY, _("Author Information"), "", wx.ITEM_NORMAL)
        wxglade_tmp_menu.AppendItem(self.author)
```

```
        self.frame_1_menubar.Append(wxglade_tmp_menu, _("About"))
        self.SetMenuBar(self.frame_1_menubar)
        # Menu Bar end
        self.label_title = wx.StaticText(self, wx.ID_ANY, _("Handwritten␣Courtesy␣Amount␣Recognition"), style=wx.ALIGN_CENTRE)
        self.MaxImageSize = 700
        Img = wx.Image("C:\Users\YongShean\Desktop\FYP\cars␣corrected\chq␣(1).jpeg", wx.BITMAP_TYPE_ANY)
        #Img = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(700, 700))
        # scale the image, preserving the aspect ratio
        W = Img.GetWidth()
        H = Img.GetHeight()
        if W > H:
            NewW = self.MaxImageSize
            NewH = self.MaxImageSize * H / W
        else:
            NewH = self.MaxImageSize
            NewW = self.MaxImageSize * W / H
        Img = Img.Scale(NewW,NewH)
        self.bitmap_1 = wx.StaticBitmap(self, wx.ID_ANY, wx.BitmapFromImage(Img))
        self.sizer_4_staticbox = wx.StaticBox(self, wx.ID_ANY, _("Cheque"))
        self.bitmap_2 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        #self.bitmap_3 = wx.StaticBitmap(self, wx.ID_ANY, wx.Bitmap("C:\\Documents and Settings\\YongShean\\My Documents\\img 3.jpg", wx.BITMAP_TYPE_ANY))
        self.bitmap_3 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_4 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_5 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_6 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_7 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_8 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_9 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_10 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_11 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_12 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
```

```
        self.bitmap_13 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_14 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_15 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_16 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_17 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_18 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_19 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_20 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_21 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_22 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_23 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_24 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_25 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.bitmap_26 = wx.StaticBitmap(self, wx.ID_ANY, wx.EmptyBitmap(20, 20))
        self.sizer_3_staticbox = wx.StaticBox(self, wx.ID_ANY, _("Segments"))
        self.text_ctrl_1 = wx.TextCtrl(self, wx.ID_ANY, self.output)
        self.sizer_2_staticbox = wx.StaticBox(self, wx.ID_ANY, _("Result"))

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

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

    def __set_properties(self):
        # begin wxGlade: MyFrame.__set_properties
        self.SetTitle(_("Handwritten␣Courtesy␣Amount␣Recognition␣(Prototype)"))
        _icon = wx.EmptyIcon()
        _icon.CopyFromBitmap(wx.Bitmap("icon.ico", wx.BITMAP_TYPE_ANY))
        self.SetIcon(_icon)
```

```
        self.SetBackgroundColour(wx.Colour(236, 233, 216))
        self.label_title.SetFont(wx.Font(15, wx.DEFAULT, wx.
           NORMAL, wx.BOLD, 0, ""))
        self.bitmap_1.SetMinSize((700, 350))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: MyFrame.__do_layout
        sizer_1 = wx.BoxSizer(wx.VERTICAL)
        self.sizer_2_staticbox.Lower()
        sizer_2 = wx.StaticBoxSizer(self.sizer_2_staticbox, wx
           .HORIZONTAL)
        self.sizer_3_staticbox.Lower()
        sizer_3 = wx.StaticBoxSizer(self.sizer_3_staticbox, wx
           .HORIZONTAL)
        grid_sizer_1 = wx.GridSizer(1, 25, 5, 0)
        self.sizer_4_staticbox.Lower()
        sizer_4 = wx.StaticBoxSizer(self.sizer_4_staticbox, wx
           .VERTICAL)
        sizer_1.Add(self.label_title, 0, wx.ALL | wx.
           ALIGN_CENTER_HORIZONTAL, 5)
        sizer_4.Add(self.bitmap_1, 0, wx.ALL | wx.EXPAND | wx.
           ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
            10)
        sizer_1.Add(sizer_4, 0, wx.LEFT | wx.RIGHT | wx.EXPAND
           , 10)
        grid_sizer_1.Add(self.bitmap_2, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_3, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_4, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_5, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_6, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_7, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_8, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_9, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_10, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_11, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_12, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_13, 0, wx.
           ALIGN_CENTER_VERTICAL, 0)
```

```
        grid_sizer_1.Add(self.bitmap_14, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_15, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_16, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_17, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_18, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_19, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_20, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_21, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_22, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_23, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_24, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_25, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        grid_sizer_1.Add(self.bitmap_26, 0, wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_3.Add(grid_sizer_1, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_1.Add(sizer_3, 0, wx.LEFT | wx.RIGHT | wx.EXPAND, 10)
        sizer_2.Add(self.text_ctrl_1, 0, 0, 0)
        sizer_1.Add(sizer_2, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)
        self.SetSizer(sizer_1)
        sizer_1.Fit(self)
        self.Layout()
        # end wxGlade

    def GetBitmap(self, npimage):
        from scipy.misc import imresize
        npimage = imresize(npimage, (20,20))
        height, width = npimage.shape
        image = wx.EmptyImage(width,height)
        pilimage = Image.fromarray(npimage)
        rgb = pilimage.convert('RGB')
        image.SetData(rgb.tostring())
        wxBitmap = wx.BitmapFromImage(image)
        return wxBitmap

```

```
    def OnOpen(self, event):  # wxGlade: MainFrame.<event_handler>
        self.bitmap_2.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_3.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_4.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_5.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_6.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_7.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_8.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_9.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_10.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_11.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_12.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_13.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_14.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_15.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_16.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_17.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_18.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_19.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_20.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_21.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_22.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_23.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_24.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_25.SetBitmap(wx.EmptyBitmap(20, 20))
        self.bitmap_26.SetBitmap(wx.EmptyBitmap(20, 20))
        wildcard = "JPG␣file␣(*.jpg)|*.jpg|\JPEG␣file␣(*.jpeg)|*.jpeg"
        dialog = wx.FileDialog(None, "Choose␣a␣file", os.getcwd(),
                               "", wildcard, wx.OPEN)
        if dialog.ShowModal()==wx.ID_OK:
            cheque = dialog.GetPath()

            Img = wx.Image(cheque, wx.BITMAP_TYPE_ANY)
            # scale the image, preserving the aspect ratio
            W = Img.GetWidth()
            H = Img.GetHeight()
            if W > H:
                NewW = self.MaxImageSize
                NewH = self.MaxImageSize * H / W
            else:
                NewH = self.MaxImageSize
                NewW = self.MaxImageSize * W / H
            Img = Img.Scale(NewW,NewH)
            self.bitmap_1.SetBitmap(wx.BitmapFromImage(Img))
            self.output, segments = ChequeOutput(cheque)

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

```
#Img = Img.SetHeight(20)
#Img = Img.SetWidth(20)
if i==0:
    self.bitmap_2.SetBitmap(Img)
if i==1:
    self.bitmap_3.SetBitmap(Img)
if i==2:
    self.bitmap_4.SetBitmap(Img)
if i==3:
    self.bitmap_5.SetBitmap(Img)
if i==4:
    self.bitmap_6.SetBitmap(Img)
if i==5:
    self.bitmap_7.SetBitmap(Img)
if i==6:
    self.bitmap_8.SetBitmap(Img)
if i==7:
    self.bitmap_9.SetBitmap(Img)
if i==8:
    self.bitmap_10.SetBitmap(Img)
if i==9:
    self.bitmap_11.SetBitmap(Img)
if i==10:
    self.bitmap_12.SetBitmap(Img)
if i==11:
    self.bitmap_13.SetBitmap(Img)
if i==12:
    self.bitmap_14.SetBitmap(Img)
if i==13:
    self.bitmap_15.SetBitmap(Img)
if i==14:
    self.bitmap_16.SetBitmap(Img)
if i==15:
    self.bitmap_17.SetBitmap(Img)
if i==16:
    self.bitmap_18.SetBitmap(Img)
if i==17:
    self.bitmap_19.SetBitmap(Img)
if i==18:
    self.bitmap_20.SetBitmap(Img)
if i==19:
    self.bitmap_21.SetBitmap(Img)
if i==20:
    self.bitmap_22.SetBitmap(Img)
if i==21:
    self.bitmap_23.SetBitmap(Img)
if i==22:
    self.bitmap_24.SetBitmap(Img)
if i==23:
    self.bitmap_25.SetBitmap(Img)
if i==24:
```

```
                    self.bitmap_26.SetBitmap(Img)

            self.text_ctrl_1.ChangeValue(self.output)
        dialog.Destroy()
        self.Layout()

    def OnExit(self, event):  # wxGlade: MainFrame.<event_handler>
        event.Skip()
        self.Destroy()

    def OnAbout(self, event):  # wxGlade: MainFrame.<event_handler>
        self.dialog = AboutDialog(None)
        #self.SetTopWindow(self.frame)
        self.dialog.Show()
        event.Skip()

# end of class MyFrame

class AboutDialog(wx.Dialog):
    def __init__(self, *args, **kwds):
        # begin wxGlade: ResultDialog.__init__
        kwds["style"] = wx.DEFAULT_DIALOG_STYLE
        wx.Dialog.__init__(self, *args, **kwds)
        self.label_1 = wx.StaticText(self, wx.ID_ANY, _(" This program is written by Chong Yong Shean \n Compiled using Python"))
        self.button_1 = wx.Button(self, wx.ID_OK, "")

        self.__set_properties()
        self.__do_layout()

        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button_1)

        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: ResultDialog.__set_properties
        self.SetTitle(_("About the Author"))
        _icon = wx.EmptyIcon()
        _icon.CopyFromBitmap(wx.Bitmap("icon.ico", wx.BITMAP_TYPE_ANY))
        self.SetIcon(_icon)
        self.label_1.SetFont(wx.Font(14, wx.MODERN, wx.NORMAL, wx.BOLD, 0, ""))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: ResultDialog.__do_layout
        sizer_3 = wx.BoxSizer(wx.VERTICAL)
```

```
        sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_4.Add(self.label_1, 0, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
        sizer_3.Add(self.button_1, 0, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        self.SetSizer(sizer_3)
        sizer_3.Fit(self)
        self.Layout()
        # end wxGlade

    def OnClick(self, event):  # wxGlade: ResultDialog.<event_handler>
        self.Destroy()
        event.Skip()

# end of class ResultDialog
class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        frame_1 = MyFrame(None, wx.ID_ANY, "")
        self.SetTopWindow(frame_1)
        frame_1.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    gettext.install("app") # replace with the appropriate catalog name

    app = MyApp(0)
    app.MainLoop()
```

```
# cheque_process_neat.py
from functions import *
"""
#␣Log␣file
from␣__future__␣import␣print_function
log␣=␣open("c:\\cheque_exp_log.txt",␣"w")

#␣Save␣all␣output␣in␣an␣array␣(for␣many␣cheques)
total_output␣=␣[]

#␣Start␣loop␣to␣process␣cheque
for␣i␣in␣range(1,101):
␣␣␣print␣"Processing␣cheque␣number␣",␣str(i),␣"..."
␣␣␣chq_path␣=␣'chq␣('+str(i)+').jpeg'
␣␣␣#␣Insert/create␣a␣function␣that␣processes␣cheque
␣␣␣output␣=␣ChequeOutput(chq_path)[0]
```

```
"""
def ChequeOutput(chq_path):
    # Load cheque image
    # chq = Image.open('chq ('+str(i)+').jpeg')
    chq = Image.open(chq_path)

    # Extract field
    img = ExtractField(chq)
    img = pil_to_cv(img)

    height, width = img.shape

    # Segmentation
    img_lst, mean_height, mean_width, y_arr = Segmentation(img)

    # Construct reference line
    datalr_y = [[y] for y in y_arr]
    datalr_X = [[x] for x in range(1,len(img_lst)+1)]

    datalr_X = np.array(datalr_X)
    datalr_y = np.array(datalr_y)
    from sklearn.linear_model import LinearRegression

    regr = LinearRegression()
    regr.fit(datalr_X, datalr_y)

    dists = regr.predict(datalr_y)-datalr_y
    #dists = abs(dists)
    max_dist = abs(dists).max()

    # Extract features
    input = FeatureExtraction(img_lst)

    # Recognition
    clf = joblib.load("scaling_svm_overall.pkl")
    input = FeatureExtraction(img_lst)
    pred = clf.predict(input)
    confidence = clf.predict_proba(input)

    output = []
    for item in pred:
        if item<=9:
            output.append(item)
        if item>=10 and item<=19:
            output.append('0'+str(item-10))
        if item>=20 and item<=109:
            if item==21:
                output.append(-1)
                #print "Junk"
            else:
                output.append(str(item-10))
```

```
        if item>=110 and item<=113:
            if item==110:
                output.append(",")
            if item==111:
                output.append(".")
            if item==112:
                output.append(-1)
                #print "R"
            if item==113:
                output.append(-1)
                #print "M"

    # Post-processing
    i=0
    for dist in dists:
        height, width = img_lst[i].shape
        if i==0 and pred[i]==111: #Dot is located at the first position
            #print "Pos", i, ": probably a junk"
            output[i] = -1
        if i>0 and pred[i]==111 and pred[i-1]>=110: #The character precede dot must be a number
            #print "Pos", i, ": probably a junk"
            output[i] = -1
        if i>0 and i<pred.shape[0]-2 and pred[i]<112 and pred[i+1]==112 and pred[i+2]==113: #There are some characters precede RM
            output[i] = -1
        if pred[i]==21: #It’s a junk
            if confidence[i][21]<0.7: #Confidence of it being a junk is less than 0.7
                # check if it’s a dot or comma
                if confidence[i][111] > confidence[i][110]:
                # if prob of it being a dot is higher than of comma
                    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: #Next two characters are numbers
                        #print "Pos", i, ": probably a dot"
                        output[i] = ’.’
                    if i>0 and pred[i-1]>110: # The character precede this segment is not a number
                        #print "Pos", i, ": probably a junk due to non-number before dot"
                        output[i] = -1
                #else:
                    #if dist<0 :
                        #print "Pos", i, ": probably being a comma"
                        #output[i] = ’,’
                    #else:
                        #print "Pos", i, ": probably being a 1"
```

```
                        #output[i] = 1
        if pred[i]==1 or pred[i]==110: # Differentiate between '1' and ','
            if dist<-0.4*max_dist: # Absolute distance between the regression line and the maximum is larger than 0.4 and the maximum is below the regression line
                #print "Pos", i, ": probably being a comma"
                output[i] = ','
            else:
                #print "Pos", i, ": probably being a 1"
                output[i] = 1
        i+=1

    output = filter(lambda a: a != -1, output)
    index_of_first_number = -1
    for k in range(len(output)):
        if output[k]<100 and output[k]!=-1:
            index_of_first_number = k
            break
    for n in range(index_of_first_number): # Eliminate any comma or dot before the first number
        if output[n]==',' or output[n]=='.':
            output[n] = -1

    output = filter(lambda a: a != -1, output)
    cnt_comma = 0
    cnt_dot = 0
    m = -1
    for k in range(len(output)):
        if output[k]==',':
            cnt_comma += 1
            m = k
        if output[k]=='.':
            cnt_dot += 1
    if cnt_comma==1 and cnt_dot==0 and m<=len(output)-3: #If there is no dot and only one comma and next three characters are not all numbers
        if output[m+1]<100 and output[m+1]!=-1 and output[m+2]<100 and output[m+2]!=-1:
            output[m] = '.'

    output = ''.join(map(str,output))
    return output, img_lst
```

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

```
import wx
from gui import *
from popups import *
#import databases
#import functions

```

```
# begin wxGlade: dependencies
import gettext
# end wxGlade

class MyApp(wx.App):
    def OnInit(self):
        wx.InitAllImageHandlers()
        frame_main = MyFrame(None, wx.ID_ANY, "")
        self.SetTopWindow(frame_main)
        frame_main.Show()
        return 1

# end of class MyApp

if __name__ == "__main__":
    gettext.install("app") # replace with the appropriate
        catalog name

    app = MyApp(0)
    app.MainLoop()
```

```
#!/usr/bin/env python
# -*- coding: CP936 -*-
#
# generated by wxGlade 0.6.8 (standalone edition) on Thu Dec
   12 14:35:02 2013
#

import wx
import wx.grid
from wx.lib.agw import floatspin as FS
import thread
from functions import *
from ObjectListView import FastObjectListView, ColumnDefn
from entry import *
from popups import *
from ddd import *

import numpy as np
from PIL import Image, ImageChops
from skimage import io
from skimage.filter import threshold_otsu
import cv2
from skimage import img_as_float, img_as_ubyte
import itertools
from sklearn import preprocessing
from sklearn.decomposition import RandomizedPCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import gc
from sklearn.externals import joblib
from sklearn import svm
from sklearn.pipeline import Pipeline
```

```

import matplotlib
import matplotlib.pyplot as plt; plt.rcdefaults()
from matplotlib.figure import Figure
from matplotlib.backends.backend_wxagg import
   FigureCanvasWxAgg as FigureCanvas

from skimage import img_as_ubyte

class MyFrame(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: MyFrame.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)

        self.notebook_2 = wx.Notebook(self, wx.ID_ANY, style
           =0)
        self.notebook_2_pane_1 = wx.Panel(self.notebook_2, wx.
           ID_ANY)
        self.list_ctrl_train = FastObjectListView(self.
           notebook_2_pane_1, wx.ID_ANY, style=wx.LC_REPORT |
           wx.SUNKEN_BORDER)

        #self.list_ctrl_train.rowFormatter = self.
           rowFormatterTrain
        self.tested = False
        self.classed = False

        self.sizer_4_staticbox = wx.StaticBox(self.
           notebook_2_pane_1, wx.ID_ANY, _("Dataset"))
        self.label_1 = wx.StaticText(self.notebook_2_pane_1,
           wx.ID_ANY, _("Kernel:"))
        self.choice_kernel = wx.Choice(self.notebook_2_pane_1,
            wx.ID_ANY, choices=[_("Linear"), _("Polynomial"),
           _("RBF"), _("Sigmoid")])
        self.label_2 = wx.StaticText(self.notebook_2_pane_1,
           wx.ID_ANY, _("C:"))
        self.spin_ctrl_c = FS.FloatSpin(self.notebook_2_pane_1
           , -1, min_val=0, max_val=20, increment=0.01)

        self.label_3 = wx.StaticText(self.notebook_2_pane_1,
           wx.ID_ANY, _("Gamma:"))
        self.spin_ctrl_gamma = FS.FloatSpin(self.
           notebook_2_pane_1, -1, min_val=0, max_val=10,
           increment=0.0001)

        self.label_4 = wx.StaticText(self.notebook_2_pane_1,
           wx.ID_ANY, _("Tolerance:"))
        self.spin_ctrl_tol = FS.FloatSpin(self.
           notebook_2_pane_1, -1, min_val=0, max_val=1,
           increment=0.0001)
```

```

        self.label_5 = wx.StaticText(self.notebook_2_pane_1, wx.ID_ANY, _("Maximum Iterations:"))
        self.spin_ctrl_max = wx.SpinCtrl(self.notebook_2_pane_1, wx.ID_ANY, "-1", min=-1, max=1000)
        self.label_6 = wx.StaticText(self.notebook_2_pane_1, wx.ID_ANY, _("Dual:"))
        self.radio_box_dual = wx.RadioBox(self.notebook_2_pane_1, wx.ID_ANY, "", choices=[_("True"), _("False")], majorDimension=2, style=wx.RA_SPECIFY_COLS)
        self.label_7 = wx.StaticText(self.notebook_2_pane_1, wx.ID_ANY, _("Degree:"))
        self.slider_deg = wx.Slider(self.notebook_2_pane_1, wx.ID_ANY, 0, 0, 10, style=wx.SL_HORIZONTAL | wx.SL_LABELS)
        self.sizer_6_staticbox = wx.StaticBox(self.notebook_2_pane_1, wx.ID_ANY, _("Parameters"))
        self.spin_ctrl_set = wx.SpinCtrl(self.notebook_2_pane_1, wx.ID_ANY, "60000", min=0, max=60000)
        self.checkbox_all = wx.CheckBox(self.notebook_2_pane_1, wx.ID_ANY, _("All"))
        self.button_load = wx.Button(self.notebook_2_pane_1, wx.ID_ANY, _("Load"))
        self.sizer_7_staticbox = wx.StaticBox(self.notebook_2_pane_1, wx.ID_ANY, _("Subset"))
        self.bitmap_train_prev = wx.StaticBitmap(self.notebook_2_pane_1, wx.ID_ANY, wx.EmptyBitmap(28,28))
        self.sizer_10_staticbox = wx.StaticBox(self.notebook_2_pane_1, wx.ID_ANY, _("Preview"))
        self.button_train = wx.Button(self.notebook_2_pane_1, wx.ID_ANY, _("Start Training"))
        self.button_class = wx.Button(self.notebook_2_pane_1, wx.ID_ANY, _("Classify"))
        self.notebook_2_pane_2 = wx.Panel(self.notebook_2, wx.ID_ANY)
        self.list_ctrl_test = FastObjectListView(self.notebook_2_pane_2, wx.ID_ANY, style=wx.LC_REPORT | wx.SUNKEN_BORDER)

        #self.list_ctrl_test.rowFormatter = self.rowFormatterTest



        self.sizer_13_staticbox = wx.StaticBox(self.notebook_2_pane_2, wx.ID_ANY, _("Dataset"))

```

```
        self.spin_ctrl_set_test = wx.SpinCtrl(self.notebook_2_pane_2, wx.ID_ANY, "10000", min=1, max=10000)
        self.checkbox_all_test = wx.CheckBox(self.notebook_2_pane_2, wx.ID_ANY, _("All"))
        self.button_load_test = wx.Button(self.notebook_2_pane_2, wx.ID_ANY, _("Load"))
        self.sizer_2_staticbox = wx.StaticBox(self.notebook_2_pane_2, wx.ID_ANY, _("Subset"))
        self.bitmap_test_prev = wx.StaticBitmap(self.notebook_2_pane_2, wx.ID_ANY, wx.EmptyBitmap(28,28))
        self.sizer_15_staticbox = wx.StaticBox(self.notebook_2_pane_2, wx.ID_ANY, _("Preview"))
        self.button_test = wx.Button(self.notebook_2_pane_2, wx.ID_ANY, _("Start Testing"))
        self.button_analyse = wx.Button(self.notebook_2_pane_2, wx.ID_ANY, _("Analyse Result"))
        #self.bitmap_test_prev = wx.StaticBitmap(self.notebook_2_pane_2, wx.ID_ANY, wx.Bitmap("C:\\Documents and Settings\\YongShean\\My Documents\\img3.jpg", wx.BITMAP_TYPE_ANY))
        #self.sizer_15_staticbox = wx.StaticBox(self.notebook_2_pane_2, wx.ID_ANY, _("Preview"))
        #self.button_test = wx.Button(self.notebook_2_pane_2, wx.ID_ANY, _("Start Testing"))
        #self.button_analyse = wx.Button(self.notebook_2_pane_2, wx.ID_ANY, _("Analyse Result"))
        self.notebook_2_pane_3 = wx.Panel(self.notebook_2, wx.ID_ANY)
        self.label_num = wx.StaticText(self.notebook_2_pane_3, wx.ID_ANY, _("7"), style=wx.ALIGN_CENTRE)
        self.sizer_19_staticbox = wx.StaticBox(self.notebook_2_pane_3, wx.ID_ANY, _("Classification"))
        self.button_browse = wx.Button(self.notebook_2_pane_3, wx.ID_ANY, _("Browse..."))
        self.bitmap_input = wx.StaticBitmap(self.notebook_2_pane_3, wx.ID_ANY, wx.EmptyBitmap(28,28), style=wx.ALIGN_CENTRE)
        self.sizer_20_staticbox = wx.StaticBox(self.notebook_2_pane_3, wx.ID_ANY, _("Input"))
        self.window_plot = WindowPlot(self.notebook_2_pane_3)
        self.sizer_21_staticbox = wx.StaticBox(self.notebook_2_pane_3, wx.ID_ANY, _("Probability of Predicted Classes"))


        # Menu Bar
        self.frame_main_menubar = wx.MenuBar()
        self.file = wx.Menu()
```

```
        self.selectdir = wx.MenuItem(self.file, wx.ID_ANY, _("Select database directory..."), _("Select the directory of MNIST files"), wx.ITEM_NORMAL)
        self.loadsvm = wx.MenuItem(self.file, wx.ID_ANY, _("A trained SVM..."), _("Select a trained SVM pipelined with standard scaler"), wx.ITEM_NORMAL)
        self.loadpca = wx.MenuItem(self.file, wx.ID_ANY, _("A fitted PCA..."), _("Unimplemented"), wx.ITEM_NORMAL)
        #self.loadstd = wx.MenuItem(self.file, wx.ID_ANY, _("A fitted Standard Scaler..."), "", wx.ITEM_NORMAL)
        self.savesvm = wx.MenuItem(self.file, wx.ID_ANY, _("The trained SVM..."),_("Save the trained SVM pipelined with standard scaler into disk"), wx.ITEM_NORMAL)
        self.savepca = wx.MenuItem(self.file, wx.ID_ANY, _("The fitted PCA..."), _("Unimplemented"), wx.ITEM_NORMAL)
        #self.savestd = wx.MenuItem(self.file, wx.ID_ANY, _("The fitted Standard Scaler..."), "", wx.ITEM_NORMAL)
        self.file.AppendItem(self.selectdir)
        self.submenu_load = wx.Menu()
        self.submenu_load.AppendItem(self.loadsvm)
        self.submenu_load.AppendItem(self.loadpca)
        #self.submenu_load.AppendItem(self.loadstd)
        self.submenu_save = wx.Menu()
        self.submenu_save.AppendItem(self.savesvm)
        self.submenu_save.AppendItem(self.savepca)
        #self.submenu_save.AppendItem(self.savestd)
        self.file.AppendMenu(wx.ID_ANY, _("Load"), self.submenu_load)
        self.file.AppendMenu(wx.ID_ANY, _("Save"), self.submenu_save)
        self.file.AppendSeparator()
        self.exit = wx.MenuItem(self.file, wx.ID_ANY, _("Exit"), "", wx.ITEM_NORMAL)
        self.file.AppendItem(self.exit)
        self.frame_main_menubar.Append(self.file, _("File"))
        self.help = wx.Menu()
        self.view_m = wx.MenuItem(self.help, wx.ID_ANY, _("View manual"), "", wx.ITEM_NORMAL)
        self.help.AppendItem(self.view_m)
        self.help.AppendSeparator()
        self.about = wx.MenuItem(self.help, wx.ID_ANY, _("About the author"), "", wx.ITEM_NORMAL)
        self.help.AppendItem(self.about)
        self.frame_main_menubar.Append(self.help, _("Help"))
        self.SetMenuBar(self.frame_main_menubar)
        # Menu Bar end

```

```
        self.frame_main_statusbar = self.CreateStatusBar(5, 0)

        self.__set_properties()
        self.__do_layout()

        self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.
           OnSelectTrainItem, self.list_ctrl_train)
        self.Bind(wx.EVT_CHOICE, self.OnChoiceKernel, self.
           choice_kernel)
        self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinC, self.
           spin_ctrl_c)
        self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinGamma, self
           .spin_ctrl_gamma)
        self.Bind(wx.EVT_SPINCTRL, self.OnFloatSpinTol, self.
           spin_ctrl_tol)
        self.Bind(wx.EVT_SPINCTRL, self.OnSpinMax, self.
           spin_ctrl_max)
        self.Bind(wx.EVT_RADIOBOX, self.OnRadioDual, self.
           radio_box_dual)
        self.Bind(wx.EVT_COMMAND_SCROLL, self.OnSliderDeg,
           self.slider_deg)
        self.Bind(wx.EVT_SPINCTRL, self.OnSpinSet, self.
           spin_ctrl_set)
        self.Bind(wx.EVT_CHECKBOX, self.OnCheckAll, self.
           checkbox_all)
        self.Bind(wx.EVT_BUTTON, self.OnButtonLoad, self.
           button_load)
        self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.
           OnSelectTestItem, self.list_ctrl_test)
        self.Bind(wx.EVT_BUTTON, self.OnButtonTrain, self.
           button_train)
        self.Bind(wx.EVT_BUTTON, self.OnButtonClass, self.
           button_class)
        self.Bind(wx.EVT_BUTTON, self.OnButtonTest, self.
           button_test)
        self.Bind(wx.EVT_BUTTON, self.OnButtonAnalyse, self.
           button_analyse)
        self.Bind(wx.EVT_BUTTON, self.OnButtonBrowse, self.
           button_browse)
        self.Bind(wx.EVT_NOTEBOOK_PAGE_CHANGED, self.
           OnPageTest, self.notebook_2)
        self.Bind(wx.EVT_SPINCTRL, self.OnSpinTest, self.
           spin_ctrl_set_test)
        self.Bind(wx.EVT_CHECKBOX, self.OnCheckAllTest, self.
           checkbox_all_test)
        self.Bind(wx.EVT_BUTTON, self.OnButtonLoadTest, self.
           button_load_test)
        ############## Menu items #################
        self.Bind(wx.EVT_MENU, self.OnAbout, self.about)
        self.Bind(wx.EVT_MENU, self.OnSelectDir, self.
           selectdir)
```

```
        self.Bind(wx.EVT_MENU, self.OnLoadSVM, self.loadsvm)
        #self.Bind(wx.EVT_MENU, self.OnLoadPCA, self.loadpca)
        #self.Bind(wx.EVT_MENU, self.OnLoadSTD, self.loadstd)
        self.Bind(wx.EVT_MENU, self.OnSaveSVM, self.savesvm)
        #self.Bind(wx.EVT_MENU, self.OnSavePCA, self.savepca)
        #self.Bind(wx.EVT_MENU, self.OnSaveSTD, self.savestd)
        self.Bind(wx.EVT_MENU, self.OnExit, self.exit)
        self.Bind(wx.EVT_MENU, self.OnViewManual, self.view_m)

    def __set_properties(self):
        # begin wxGlade: MyFrame.__set_properties
        self.SetTitle(_("Handwritten Digit Training and 
           Recognition"))

        # ObjectListView
        actualColumn = ColumnDefn("Actual","center",50,
           valueGetter="actual",minimumWidth=30)
        predictedColumn = ColumnDefn("Predicted","center",70,
           valueGetter="predicted",minimumWidth=50)
        actualColumn.isSpaceFilling = True
        predictedColumn.isSpaceFilling = True

        self.list_ctrl_train.SetColumns([actualColumn,
           predictedColumn])
        self.list_ctrl_test.SetColumns([actualColumn,
           predictedColumn])

        # Floatspin options
        self.spin_ctrl_c.SetFormat("%f")
        self.spin_ctrl_c.SetDigits(4)
        self.spin_ctrl_c.SetDefaultValue(2.8)
        self.spin_ctrl_c.SetToDefaultValue()
        self.spin_ctrl_gamma.SetFormat("%f")
        self.spin_ctrl_gamma.SetDigits(4)
        self.spin_ctrl_gamma.SetDefaultValue(0.0073)
        self.spin_ctrl_gamma.SetToDefaultValue()
        self.spin_ctrl_tol.SetFormat("%f")
        self.spin_ctrl_tol.SetDigits(4)
        self.spin_ctrl_tol.SetDefaultValue(0.001)
        self.spin_ctrl_tol.SetToDefaultValue()

        # Default selection
        self.radio_box_dual.SetSelection(1)
        self.choice_kernel.SetSelection(2)
        self.radio_box_dual.Enable(False)
        self.slider_deg.Enable(False)

        # Default values
        self._kernel = 'rbf'
        self._c = 2.8
        self._gamma = 0.0073
```

```
        self._tol = 0.001
        self._max_iter = -1
        self._dual = False
        self._deg = 3
        self._subset = 60000
        self.button_train.Enable(False)
        self.button_class.Enable(False)
        self.button_test.Enable(False)
        self.button_analyse.Enable(False)
        self.data =
            [0.02,0.3,0.45,0.56,0.67,0.78,0.89,0.95,0.21,0.1]
        self.mnist_path = '.'

        self._subset_test = 10000

        self.label_num.SetFont(wx.Font(68, wx.DEFAULT, wx.
            NORMAL, wx.NORMAL, 0, ""))
        self.frame_main_statusbar.SetStatusWidths([-1, 0, 0,
            0, 0])
        # statusbar fields
        frame_main_statusbar_fields = [_("Ready")]
        for i in range(len(frame_main_statusbar_fields)):
            self.frame_main_statusbar.SetStatusText(
                frame_main_statusbar_fields[i], i)
        # end wxGlade

        self.list_ctrl_train.rowFormatter = self.
            rowFormatterTrain
        self.list_ctrl_test.rowFormatter = self.
            rowFormatterTest

    def __do_layout(self):
        # begin wxGlade: MyFrame.__do_layout
        sizer_1 = wx.BoxSizer(wx.VERTICAL)
        sizer_17 = wx.BoxSizer(wx.HORIZONTAL)
        self.sizer_21_staticbox.Lower()
        sizer_21 = wx.StaticBoxSizer(self.sizer_21_staticbox,
            wx.HORIZONTAL)
        sizer_18 = wx.BoxSizer(wx.VERTICAL)
        self.sizer_20_staticbox.Lower()
        sizer_20 = wx.StaticBoxSizer(self.sizer_20_staticbox,
            wx.HORIZONTAL)
        sizer_22 = wx.BoxSizer(wx.VERTICAL)
        self.sizer_19_staticbox.Lower()
        sizer_19 = wx.StaticBoxSizer(self.sizer_19_staticbox,
            wx.HORIZONTAL)
        """
␣␣␣␣␣␣␣␣sizer_12␣=␣wx.BoxSizer(wx.VERTICAL)
␣␣␣␣␣␣␣␣sizer_14␣=␣wx.BoxSizer(wx.HORIZONTAL)
␣␣␣␣␣␣␣␣sizer_16␣=␣wx.BoxSizer(wx.VERTICAL)
␣␣␣␣␣␣␣␣self.sizer_15_staticbox.Lower()
```

```
␣␣␣␣␣␣␣␣sizer_15␣=␣wx.StaticBoxSizer(self.sizer_15_staticbox,␣wx.HORIZONTAL)
␣␣␣␣␣␣␣␣self.sizer_13_staticbox.Lower()
␣␣␣␣␣␣␣␣"""

        sizer_12 = wx.BoxSizer(wx.VERTICAL)
        sizer_14 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_16 = wx.BoxSizer(wx.VERTICAL)
        self.sizer_15_staticbox.Lower()
        sizer_15 = wx.StaticBoxSizer(self.sizer_15_staticbox, wx.HORIZONTAL)
        self.sizer_2_staticbox.Lower()
        sizer_2 = wx.StaticBoxSizer(self.sizer_2_staticbox, wx.HORIZONTAL)

        sizer_13 = wx.StaticBoxSizer(self.sizer_13_staticbox, wx.HORIZONTAL)
        sizer_3 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_5 = wx.BoxSizer(wx.VERTICAL)
        sizer_9 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_11 = wx.BoxSizer(wx.VERTICAL)
        self.sizer_10_staticbox.Lower()
        sizer_10 = wx.StaticBoxSizer(self.sizer_10_staticbox, wx.HORIZONTAL)
        self.sizer_7_staticbox.Lower()
        sizer_7 = wx.StaticBoxSizer(self.sizer_7_staticbox, wx.HORIZONTAL)
        sizer_8 = wx.BoxSizer(wx.HORIZONTAL)
        self.sizer_6_staticbox.Lower()
        sizer_6 = wx.StaticBoxSizer(self.sizer_6_staticbox, wx.HORIZONTAL)
        grid_sizer_1 = wx.FlexGridSizer(7, 2, 3, 3)
        self.sizer_4_staticbox.Lower()
        sizer_4 = wx.StaticBoxSizer(self.sizer_4_staticbox, wx.HORIZONTAL)
        sizer_4.Add(self.list_ctrl_train, 1, wx.EXPAND, 0)
        sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
        grid_sizer_1.Add(self.label_1, 0, wx.ALL | wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.choice_kernel, 0, 0, 0)
        grid_sizer_1.Add(self.label_2, 0, wx.ALL | wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.spin_ctrl_c, 0, 0, 0)
        grid_sizer_1.Add(self.label_3, 0, wx.ALL | wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.spin_ctrl_gamma, 0, 0, 0)
        grid_sizer_1.Add(self.label_4, 0, wx.ALL | wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.
```

```
            ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.spin_ctrl_tol, 0, 0, 0)
        grid_sizer_1.Add(self.label_5, 0, wx.ALL | wx.EXPAND |
             wx.ALIGN_CENTER_HORIZONTAL | wx.
            ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.spin_ctrl_max, 0, 0, 0)
        grid_sizer_1.Add(self.label_6, 0, wx.ALL | wx.EXPAND |
             wx.ALIGN_CENTER_HORIZONTAL | wx.
            ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.radio_box_dual, 0, wx.EXPAND, 0)
        grid_sizer_1.Add(self.label_7, 0, wx.ALL | wx.EXPAND |
             wx.ALIGN_CENTER_HORIZONTAL | wx.
            ALIGN_CENTER_VERTICAL, 3)
        grid_sizer_1.Add(self.slider_deg, 0, wx.EXPAND, 0)
        sizer_6.Add(grid_sizer_1, 1, wx.EXPAND, 0)
        sizer_5.Add(sizer_6, 3, wx.EXPAND, 0)
        sizer_8.Add(self.spin_ctrl_set, 0, wx.ALL | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             5)
        sizer_8.Add(self.checkbox_all, 0, wx.LEFT | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             10)
        sizer_8.Add(self.button_load, 0, wx.LEFT | wx.
            ALIGN_RIGHT | wx.ALIGN_CENTER_HORIZONTAL | wx.
            ALIGN_CENTER_VERTICAL, 10)
        sizer_7.Add(sizer_8, 1, wx.EXPAND | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             0)
        sizer_5.Add(sizer_7, 1, wx.EXPAND | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             0)
        sizer_10.Add(self.bitmap_train_prev, 1, wx.EXPAND | wx
            .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL
            , 0)
        sizer_9.Add(sizer_10, 1, wx.EXPAND | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             0)
        sizer_11.Add(self.button_train, 0, wx.EXPAND | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             0)
        sizer_11.Add(self.button_class, 0, wx.EXPAND | wx.
            ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL,
             0)
        sizer_9.Add(sizer_11, 2, wx.LEFT | wx.ALIGN_RIGHT | wx
            .ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL
            , 10)
        sizer_5.Add(sizer_9, 1, wx.EXPAND, 0)
        sizer_3.Add(sizer_5, 1, wx.EXPAND, 0)
        self.notebook_2_pane_1.SetSizer(sizer_3)
        sizer_13.Add(self.list_ctrl_test, 1, wx.EXPAND, 0)
        sizer_12.Add(sizer_13, 5, wx.EXPAND, 0)
```

```
        """
␣␣␣␣␣␣␣␣sizer_15.Add(self.bitmap_test_prev,␣1,␣wx.ALIGN_CENTER_HORIZONTAL␣|␣wx.ALIGN_CENTER_VERTICAL,␣0)
␣␣␣␣␣␣␣␣sizer_14.Add(sizer_15,␣1,␣wx.EXPAND␣|␣wx.ALIGN_CENTER_HORIZONTAL␣|␣wx.ALIGN_CENTER_VERTICAL,␣0)
␣␣␣␣␣␣␣␣sizer_16.Add(self.button_test,␣0,␣wx.EXPAND␣|␣wx.ALIGN_CENTER_HORIZONTAL␣|␣wx.ALIGN_CENTER_VERTICAL,␣0)
␣␣␣␣␣␣␣␣sizer_16.Add(self.button_analyse,␣0,␣wx.EXPAND␣|␣wx.ALIGN_CENTER_HORIZONTAL␣|␣wx.ALIGN_CENTER_VERTICAL,␣0)
␣␣␣␣␣␣␣␣sizer_14.Add(sizer_16,␣1,␣wx.ALL␣|␣wx.ALIGN_RIGHT␣|␣wx.ALIGN_CENTER_HORIZONTAL␣|␣wx.ALIGN_CENTER_VERTICAL,␣5)
␣␣␣␣␣␣␣␣sizer_12.Add(sizer_14,␣1,␣wx.EXPAND,␣0)
␣␣␣␣␣␣␣␣self.notebook_2_pane_2.SetSizer(sizer_12)
␣␣␣␣␣␣␣␣"""
        sizer_2.Add(self.spin_ctrl_set_test, 0, wx.ALL | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 10)
        sizer_2.Add(self.checkbox_all_test, 0, wx.ALL | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 5)
        sizer_2.Add(self.button_load_test, 0, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 5)
        sizer_14.Add(sizer_2, 3, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_15.Add(self.bitmap_test_prev, 1, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_14.Add(sizer_15, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_16.Add(self.button_test, 0, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_16.Add(self.button_analyse, 0, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_14.Add(sizer_16, 2, wx.ALL | wx.ALIGN_RIGHT | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 5)
        sizer_12.Add(sizer_14, 1, wx.EXPAND, 0)
        self.notebook_2_pane_2.SetSizer(sizer_12)

        sizer_19.Add(self.label_num, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 5)
        sizer_18.Add(sizer_19, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_22.Add(self.button_browse, 0, wx.EXPAND, 0)
```

```
        sizer_22.Add(self.bitmap_input, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_20.Add(sizer_22, 1, wx.EXPAND, 0)
        sizer_18.Add(sizer_20, 1, wx.EXPAND, 0)
        sizer_17.Add(sizer_18, 1, wx.EXPAND, 0)
        sizer_21.Add(self.window_plot, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_21.SetSizeHints(self)
        sizer_17.Add(sizer_21, 3, wx.EXPAND, 0)
        self.notebook_2_pane_3.SetSizer(sizer_17)
        self.notebook_2.AddPage(self.notebook_2_pane_1, _("Training"))
        self.notebook_2.AddPage(self.notebook_2_pane_2, _("Testing"))
        self.notebook_2.AddPage(self.notebook_2_pane_3, _("Classify"))
        sizer_1.Add(self.notebook_2, 1, wx.EXPAND, 0)
        self.SetSizer(sizer_1)
        sizer_1.Fit(self)

        #self.window_plot.GetParent().GetSizer().Hide(self.window_plot)

        self.Layout()
        # end wxGlade

        # Events definition

    def OnSelectTrainItem(self, event):
        self.bitmap_train_prev.SetBitmap(wx.EmptyBitmap(28,28))
        item = self.list_ctrl_train.GetSelectedObject()
        self.bitmap_train_prev.SetBitmap(item.image)
        self.bitmap_train_prev.GetParent().GetSizer().Layout()
        self.Refresh()
        event.Skip()

    def OnChoiceKernel(self, event):
        # Depending on the choice of kernel, certain parameter options would be disabled or changed.
        sel = self.choice_kernel.GetSelection()
        if sel==0: #Linear
            self._kernel = 'linear'
            #change tolerance to 0.0001
            self.spin_ctrl_tol.SetValue(0.0001)
            #enable dual option
            self.radio_box_dual.Enable(True)
            #disable gamma option
            self.spin_ctrl_gamma.Enable(False)
```

```
            #disable max_iter option
            self.spin_ctrl_max.Enable(False)
            #disable degree option
            self.slider_deg.Enable(False)
        if sel==1: #Polynomial
            self._kernel = 'poly'
            #enable degree option
            self.slider_deg.Enable(True)
            self.slider_deg.SetValue(3)
            #disable dual option
            self.radio_box_dual.Enable(False)
            #change tolerance to 0.001
            self.spin_ctrl_tol.SetValue(0.0010)
            #enable gamma option
            self.spin_ctrl_gamma.Enable(True)
            #enable max_iter option
            self.spin_ctrl_max.Enable(True)
        if sel==2: #RBF or Sigmoid
            self._kernel = 'rbf'
            #change tolerance to 0.001
            self.spin_ctrl_tol.SetValue(0.0010)
            #disable dual option
            self.radio_box_dual.Enable(False)
            #disable degree option
            self.slider_deg.Enable(False)
            #enable gamma option
            self.spin_ctrl_gamma.Enable(True)
            #enable max_iter option
            self.spin_ctrl_max.Enable(True)
        if sel==3: #RBF or Sigmoid
            self._kernel = 'sigmoid'
            #change tolerance to 0.001
            self.spin_ctrl_tol.SetValue(0.0010)
            #disable dual option
            self.radio_box_dual.Enable(False)
            #disable degree option
            self.slider_deg.Enable(False)
            #enable gamma option
            self.spin_ctrl_gamma.Enable(True)
            #enable max_iter option
            self.spin_ctrl_max.Enable(True)
        event.Skip()

    def OnSelectDir(self, event):
        dialog = wx.DirDialog(None, "Choose the directory of MNIST database", '.', style=wx.DD_DIR_MUST_EXIST)
        if dialog.ShowModal()==wx.ID_OK:
            self.mnist_path = dialog.GetPath()
        event.Skip()

    def OnLoadSVM(self, event):
```

```
        wildcard = "PKL␣file␣(*.pkl)|*.pkl"
        dialog = wx.FileDialog(None, "Choose␣a␣saved␣SVM␣file", os.getcwd(),
                               "", wildcard, wx.OPEN)

        result = dialog.ShowModal()
        if result==wx.ID_OK:
            path = dialog.GetPath()
            self.clf = joblib.load(path)
            dialog.Destroy()
            self.button_train.SetLabel('Start␣Extracting')
            return
        elif result==wx.ID_CANCEL:
            dialog.Destroy()
            return

        event.Skip()

    def OnLoadPCA(self, event):
        wildcard = "PKL␣file␣(*.pkl)|*.pkl"
        dialog = wx.FileDialog(None, "Choose␣a␣saved␣PCA␣file", os.getcwd(),
                               "", wildcard, wx.OPEN)
        result = dialog.ShowModal()
        if result==wx.ID_OK:
            path = dialog.GetPath()
            self.pca = joblib.load(path)
            dialog.Destroy()
            return
        elif result==wx.ID_CANCEL:
            dialog.Destroy()
            return
        event.Skip()

    def OnLoadSTD(self, event):
        wildcard = "NPY␣file␣(*.npy)|*.npy"
        dialog = wx.FileDialog(None, "Choose␣a␣saved␣scaler␣file", os.getcwd(),
                               "", wildcard, wx.OPEN)
        result = dialog.ShowModal()
        if result==wx.ID_OK:
            path = dialog.GetPath()
            self.min_max_scaler = np.load(path)
            dialog.Destroy()
            return
        elif result==wx.ID_CANCEL:
            dialog.Destroy()
            return
        event.Skip()

    def OnSaveSVM(self, event):
```

```
        try:
            self.clf
        except AttributeError:
            dial = wx.MessageDialog(None, "The␣SVM␣must␣be␣trained␣before␣saving!", 'Error', wx.OK | wx.ICON_ERROR)
            dial.ShowModal()
            return

        dialog = wx.DirDialog(None, "Choose␣the␣directory␣you␣want␣to␣save␣SVM␣to", '.', style=wx.DD_DEFAULT_STYLE)
        if dialog.ShowModal()==wx.ID_OK:
            path = dialog.GetPath()
            joblib.dump(self.clf, str(path)+"trained_svm.pkl")
            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)
        event.Skip()

    def OnSavePCA(self, event):
        try:
            self.clf
        except AttributeError:
            dial = wx.MessageDialog(None, "The␣SVM␣must␣be␣trained␣before␣saving!", 'Error', wx.OK | wx.ICON_ERROR)
            dial.ShowModal()
            return

        dialog = wx.DirDialog(None, "Choose␣the␣directory␣you␣want␣to␣save␣PCA␣to", '.', style=wx.DD_DEFAULT_STYLE)
        if dialog.ShowModal()==wx.ID_OK:
            path = dialog.GetPath()
            joblib.dump(self.pca, str(path)+"fitted_pca.pkl")
            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)
        event.Skip()

    def OnSaveSTD(self, event):
        try:
            self.clf
        except AttributeError:
            dial = wx.MessageDialog(None, "The␣SVM␣must␣be␣trained␣before␣saving!", 'Error', wx.OK | wx.ICON_ERROR)
            dial.ShowModal()
```

```
            return

        dialog = wx.DirDialog(None, "Choose the directory you want to save standard scaler to", '.', style=wx.DD_DEFAULT_STYLE)
        if dialog.ShowModal()==wx.ID_OK:
            path = dialog.GetPath()
            joblib.dump(self.min_max_scaler, str(path)+"fitted_std.pkl")
            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)
        event.Skip()

    def OnViewManual(self, event):
        link = "https://docs.google.com/document/d/1zOiEKk2vD_UMxxCtaSbat0bcW-_i9-pdyttQRP6kQLg/edit?usp=sharing"
        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)
        if dial.ShowModal()==wx.ID_OK:
            wx.LaunchDefaultBrowser(link)
        dial = None
        event.Skip()

    def OnExit(self, event):
        event.Skip()
        self.Destroy()

    def OnAbout(self, event):
        #self.dialog = AboutDialog(None)
        #self.dialog.Show()
        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)
        dial.ShowModal()
        event.Skip()

    def OnFloatSpinC(self, event):
        floatspin = event.GetEventObject()
        fmt = floatspin.GetFormat()
        dgt = floatspin.GetDigits()
        strs = ("%100." + str(dgt) + fmt[1])%floatspin.GetValue()
        self._c = float(strs.strip())
        event.Skip()

```

```
    def OnFloatSpinGamma(self, event):
        floatspin = event.GetEventObject()
        fmt = floatspin.GetFormat()
        dgt = floatspin.GetDigits()
        strs = ("%100." + str(dgt) + fmt[1])%floatspin.GetValue()
        self._gamma = float(strs.strip())
        event.Skip()

    def OnFloatSpinTol(self, event):
        floatspin = event.GetEventObject()
        fmt = floatspin.GetFormat()
        dgt = floatspin.GetDigits()
        strs = ("%100." + str(dgt) + fmt[1])%floatspin.GetValue()
        self._tol = float(strs.strip())
        event.Skip()

    def OnSpinMax(self, event):
        self._max_iter = int(self.spin_ctrl_max.GetValue())
        event.Skip()

    def OnRadioDual(self, event):
        if self.radio_box_dual.GetSelection()==0:
            self._dual = True
        else:
            self._dual = False
        event.Skip()

    def OnSliderDeg(self, event):
        self._deg = int(self.slider_deg.GetValue())
        event.Skip()

    def OnSpinSet(self, event):
        isChecked = self.checkbox_all.GetValue()
        if isChecked:
            self._subset = 60000
        else:
            self._subset = int(self.spin_ctrl_set.GetValue())
        event.Skip()

    def OnCheckAll(self, event):
        isChecked = self.checkbox_all.GetValue()
        if isChecked:
            self._subset = 60000
            self.spin_ctrl_set.SetValue(60000)
            self.spin_ctrl_set.Enable(False)
        else:
            self.spin_ctrl_set.Enable(True)
            self._subset = int(self.spin_ctrl_set.GetValue())
        event.Skip()
```

```

    def OnSpinTest(self, event):
        isChecked = self.checkbox_all_test.GetValue()
        if isChecked:
            self._subset_test = 10000
        else:
            self._subset_test = int(self.spin_ctrl_set_test.GetValue())
        event.Skip()

    def OnCheckAllTest(self, event):
        isChecked = self.checkbox_all_test.GetValue()
        if isChecked:
            self._subset_test = 10000
            self.spin_ctrl_set_test.SetValue(10000)
            self.spin_ctrl_set_test.Enable(False)
        else:
            self.spin_ctrl_set_test.Enable(True)
            self._subset_test = int(self.spin_ctrl_set_test.GetValue())
        event.Skip()

    def OnButtonLoad(self, event):
        #self.Hide()
        self.SetStatusText("Loading dataset...")
        self.button_load.Enable(False)
        msg = "Please wait while the dataset is being loaded..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartLoad, ())
        event.Skip()

    def OnButtonLoadTest(self, event):
        self.SetStatusText("Loading dataset...")
        self.button_load_test.Enable(False)
        msg = "Please wait while the dataset is being loaded..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartLoadTest, ())
        event.Skip()

    def OnSelectTestItem(self, event):
        self.bitmap_test_prev.SetBitmap(wx.EmptyBitmap(28,28))
        item = self.list_ctrl_test.GetSelectedObject()
        self.bitmap_test_prev.SetBitmap(item.image)
        self.bitmap_test_prev.GetParent().GetSizer().Layout()
        self.Refresh()
        event.Skip()

    def OnButtonTrain(self, event):
        self.SetStatusText("Extracting features...")
```

```
        self.button_train.Enable(False)
        msg = "Extracting␣features␣..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartExtract, ())
        event.Skip()

    def OnButtonClass(self, event):
        self.SetStatusText("Classifying␣images...")
        self.button_class.Enable(False)
        msg = "Please␣wait␣while␣the␣dataset␣is␣being␣classified..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartClass, ())
        event.Skip()

    def OnButtonTest(self, event):
        try:
            self.clf
        except AttributeError:
            dial = wx.MessageDialog(None, "The␣SVM␣must␣be␣trained␣before␣testing!", 'Error', wx.OK | wx.ICON_ERROR)
            dial.ShowModal()
            return

        self.SetStatusText("Extracting␣features...")
        self.button_test.Enable(False)
        msg = "Extracting␣features␣..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartExtractTest, ())

        event.Skip()

    def OnButtonAnalyse(self, event):
        self.analyseDlg = AnalyseDialog(self)
        g = self.analyseDlg.grid_analyse

        for i in range(10):
                for j in range(10):
                    g.SetCellValue(i,j,'0')

        for entry in self.data_test:
            for i in range(10):
                for j in range(10):
                    if entry.predicted==str(i) and entry.actual==str(j):
                        val = int(g.GetCellValue(i,j)) + 1
                        g.SetCellValue(i,j,str(val))
        """
␣␣␣␣␣␣␣␣for␣entry␣in␣self.data_test:
␣␣␣␣␣␣␣␣␣␣␣␣for␣i␣in␣range(10):
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣cnt=0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣entry.actual==str(i):
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣j␣in␣range(10):
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣entry.predicted==str(j):
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣cnt+=1
␣␣␣␣␣␣␣␣"""
        #g.SetCellValue(row,col,s)
        self.analyseDlg.ShowModal()
        #analyse = AnalyseFrame(self)
        #analyse.Show()
        event.Skip()

    def OnButtonBrowse(self, event):
        try:
            self.clf
        except AttributeError:
            dial = wx.MessageDialog(None, "The␣SVM␣must␣be␣trained␣before␣using!", 'Error', wx.OK | wx.ICON_ERROR)
            dial.ShowModal()
            return
        wildcard = "JPG␣file␣(*.jpg)|*.jpg|JPEG␣file␣(*.jpeg)|*.jpeg"
        dialog = wx.FileDialog(None, "Choose␣an␣image", os.getcwd(),
                               "", wildcard, wx.OPEN)
        if dialog.ShowModal()==wx.ID_OK:
            path = dialog.GetPath()

            Img = wx.Image(path, wx.BITMAP_TYPE_JPEG)
            # scale the image, preserving the aspect ratio
            W = Img.GetWidth()
            H = Img.GetHeight()
            self.MaxImageSize = 28
            if W > H:
                NewW = self.MaxImageSize
                NewH = self.MaxImageSize * H / W
            else:
                NewH = self.MaxImageSize
                NewW = self.MaxImageSize * W / H
            Img = Img.Scale(NewW,NewH)
            self.bitmap_input.SetBitmap(wx.BitmapFromImage(Img))
            img = cv2.imread(path, cv2.CV_LOAD_IMAGE_GRAYSCALE)
            img = 255-img

            self.binary = img

            self.SetStatusText("Extracting␣features...")
            self.button_test.Enable(False)
```

```
                msg = "Extracting␣features␣..."
                self.busyDlg = wx.BusyInfo(msg)
                thread.start_new_thread(self.OnStartExtractInput,
                    ())

            event.Skip()

        def OnPageTest(self, event):
            #print self.notebook_2.GetSelection()
            event.Skip()

        def OnStartLoad(self):
            img, labels = read_mnist(range(10),'training', path=
                self.mnist_path)
            self.images = img
            labels = labels.flatten()
            self.labels = labels.astype(float)

            if self._subset!=60000:
                i=0
                temp=[]
                for im in self.images:
                    #im = 255-im
                    temp.append(im)
                    i+=1
                    if i>=self._subset:
                        break
                self.images = temp

                i=0
                temp=[]
                for lb in self.labels:
                    temp.append(lb)
                    i+=1
                    if i>=self._subset:
                        break
                self.labels = temp

            self.PopulateList(self.list_ctrl_train)

            wx.CallAfter(self.OnDoneLoad)

        def OnStartLoadTest(self):
            img, labels = read_mnist(range(10),'testing', path=
                self.mnist_path)
            self.images_test = img
            labels = labels.flatten()
            self.labels_test = labels.astype(float)

            if self._subset_test!=60000:
                i=0
```

```
            temp=[]
            for im in self.images_test:
                #im = 255-im
                temp.append(im)
                i+=1
                if i>=self._subset_test:
                    break
            self.images_test = temp

            i=0
            temp=[]
            for lb in self.labels_test:
                temp.append(lb)
                i+=1
                if i>=self._subset_test:
                    break
            self.labels_test = temp

        self.PopulateListTest(self.list_ctrl_test)

        wx.CallAfter(self.OnDoneLoadTest)

    def OnDoneLoad(self):
        self.busyDlg = None
        #self.Show()
        #self.Layout()
        self.button_load.Enable(True)
        self.button_train.Enable(True)
        self.SetStatusText("Dataset completely loaded.")

    def OnDoneLoadTest(self):
        self.busyDlg = None
        #self.Show()
        #self.Layout()
        self.button_load_test.Enable(True)
        self.button_test.Enable(True)
        self.SetStatusText("Dataset completely loaded.")

    def OnDoneExtract(self):
        self.busyDlg = None
        self.SetStatusText("Extraction completed.")
        try:
            self.clf
            self.button_class.Enable(True)
        except AttributeError:
            self.SetStatusText("Training dataset...")
            msg = "Training dataset... This might take a significant amount of time..."
            self.busyDlg = wx.BusyInfo(msg)
            thread.start_new_thread(self.OnStartTrain,())
        #wx.CallAfter(self.OnStartTrain)
```

```python

    def OnDoneExtractTest(self):
        self.busyDlg = None
        self.SetStatusText("Testing␣dataset...")
        msg = "Please␣wait␣while␣the␣dataset␣is␣being␣tested..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartTest,())
        #wx.CallAfter(self.OnStartTrain)

    def OnStartTrain(self):
        if self._kernel=='linear':
            clf = svm.LinearSVC(C=self._c, dual=self._dual, tol=self._tol)
        elif self._kernel=='poly':
            clf = svm.SVC(kernel='poly',C=self._c, gamma=self._gamma, tol=self._tol, max_iter=self._max_iter, degree=self._deg, probability=True)
        elif self._kernel=='rbf':
            clf = svm.SVC(kernel='rbf', C=self._c, gamma=self._gamma, tol=self._tol, max_iter=self._max_iter, probability=True)
        elif self._kernel=='sigmoid':
            clf = svm.SVC(kernel='sigmoid', C=self._c, gamma=self._gamma, tol=self._tol, max_iter=self._max_iter, probability=True)
        scaler = StandardScaler()
        self.clf = Pipeline([("scaler", scaler), ("svm", clf)])
        self.clf.fit(self.features, self.labels)
        wx.CallAfter(self.OnDoneTrain)

    def OnDoneTrain(self):
        self.busyDlg = None
        self.button_train.Enable(True)
        self.button_class.Enable(True)
        #self.Layout()
        self.SetStatusText("Training␣completed.")

    def OnStartClass(self):
        prediction = self.clf.predict(self.features)
        self.accuracy = self.clf.score(self.features,self.labels)
        for entry,pred in zip(self.data, prediction):
            entry.predicted = str(int(round(pred)))
            # Condition testing: if entry.predicted != label then red colour
        wx.CallAfter(self.OnDoneClass)

    def OnDoneClass(self):
        self.busyDlg = None
```

```
        self.button_class.Enable(True)
        self.classed = True
        FastObjectListView.RefreshObjects(self.list_ctrl_train)
        #self.Layout()
        self.SetStatusText("Result: "+str(int(self.accuracy*self._subset))+" out of "+str(self._subset)+" ("+str(self.accuracy*100)+"%)")

    def OnStartTest(self):
        prediction = self.clf.predict(self.features_test)
        self.accuracy_test = self.clf.score(self.features_test,self.labels_test)
        for entry,pred in zip(self.data_test, prediction):
            entry.predicted = str(int(round(pred)))
        wx.CallAfter(self.OnDoneTest)

    def OnDoneTest(self):
        self.busyDlg = None
        self.button_test.Enable(True)
        self.button_analyse.Enable(True)
        self.tested = True
        FastObjectListView.RefreshObjects(self.list_ctrl_test)
        #self.Layout()
        self.SetStatusText("Result: "+str(int(self.accuracy_test*self._subset_test))+" out of "+str(self._subset_test)+" ("+str(self.accuracy_test*100)+"%)")

    def PopulateList(self, list_ctrl):
        self.data = []
        for label,im in zip(self.labels,self.images):
            bmp = GetBitmap(im)
            self.data.append(Entry(image=bmp,actual=str(int(label)), predicted=''))

        list_ctrl.SetObjects(self.data)

    def PopulateListTest(self, list_ctrl):
        self.data_test = []
        for label,im in zip(self.labels_test,self.images_test):
            bmp = GetBitmap(im)
            self.data_test.append(Entry(image=bmp,actual=str(int(label)), predicted=''))

        list_ctrl.SetObjects(self.data_test)

    def OnStartExtractOld(self):
        features = []
        i=0
```

```
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        for img in self.images:
            #f = cv2.imread(img, cv2.CV_LOAD_IMAGE_GRAYSCALE)
            #f = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY)[1]
            #f = 255-f
            #io.imsave("train_b4.jpg",img)
            #f = cv2.imread("train_b4.jpg", cv2.CV_LOAD_IMAGE_GRAYSCALE)
            #f = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            #cv2.imwrite("train.jpg",f)
            feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2, cv2.cv.CV_DIST_MASK_PRECISE)
            feat = tuple(itertools.chain(*feat))
            features.append(feat)
            i+=1
            self.SetStatusText("Extracting␣features..."+str(i)+"␣out␣of␣" +str(self._subset))

        self.features = np.array(features)

        thread.start_new_thread(self.OnStartRed,())
        #wx.CallAfter(self.OnStartRed)

    def OnStartExtract(self):
        features = []
        i=0
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        for img in self.images:
            f = Image.fromarray(img)
            f = trim(f)
            f = f.convert('L')
            f = np.array(f)
            f = img_as_ubyte(f)
            f = cv2.resize(f, (16,16), interpolation=cv2.INTER_LANCZOS4)
            feat = ExtractDDD(f)
            feat = tuple(itertools.chain(*feat))
            feat = tuple(itertools.chain(*feat))
            features.append(feat)
            i+=1
            self.SetStatusText("Extracting␣features..."+str(i)+"␣out␣of␣" +str(self._subset))

        self.features = np.array(features)

        #thread.start_new_thread(self.OnStartRed,())
```

```
        wx.CallAfter(self.OnDoneExtract)

    def OnStartRed(self):
        # fit pca and std_scaler - save globally
        #self.SetStatusText("Initialising PCA and standard scaler...")
        #self.pca = RandomizedPCA(whiten=True)
        self.SetStatusText("Scaling␣the␣feature␣vector...")
        try:
            self.features = self.clf.transform(self.features)
        except AttributeError:
            self.clf = MinMaxScaler(feature_range=(-1, 1))
            self.features = self.min_max_scaler.fit_transform(self.features)

        #self.SetStatusText("Applying PCA...")
        #self.features = self.pca.fit_transform(self.features)

        self.SetStatusText("Extraction␣completed.")

        wx.CallAfter(self.OnDoneExtract)

    def OnStartExtractTestOld(self):
        features = []
        i=0
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        for img in self.images_test:
            #f = cv2.imread(img, cv2.CV_LOAD_IMAGE_GRAYSCALE)
            #f = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #io.imsave("test_b4.jpg",img)
            #f = cv2.imread("test_b4.jpg", cv2.CV_LOAD_IMAGE_GRAYSCALE)
            feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2, cv2.cv.CV_DIST_MASK_PRECISE)
            feat = tuple(itertools.chain(*feat))
            features.append(feat)
            i+=1
            self.SetStatusText("Extracting␣features..."+str(i)+"␣out␣of␣" +str(self._subset_test))

        self.features_test = np.array(features)

        thread.start_new_thread(self.OnStartRedTest,())
        #wx.CallAfter(self.OnStartRed)

    def OnStartExtractTest(self):
        features = []
        i=0
```

```
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        for img in self.images_test:
            f = Image.fromarray(img)
            f = trim(f)
            f = f.convert('L')
            f = np.array(f)
            f = img_as_ubyte(f)
            f = cv2.resize(f, (16,16), interpolation=cv2.INTER_LANCZOS4)
            feat = ExtractDDD(f)
            feat = tuple(itertools.chain(*feat))
            feat = tuple(itertools.chain(*feat))
            features.append(feat)
            i+=1
            self.SetStatusText("Extracting features..."+str(i)+" out of " +str(self._subset_test))

        self.features_test = np.array(features)

        #thread.start_new_thread(self.OnStartRedTest,())
        wx.CallAfter(self.OnDoneExtractTest)

    def OnStartRedTest(self):
        # fit pca and std_scaler - save globally
        self.SetStatusText("Scaling the feature vector...")
        self.features_test = self.min_max_scaler.transform(self.features_test)
        #self.SetStatusText("Applying PCA...")
        #self.features_test = self.pca.transform(self.features_test)
        self.SetStatusText("Extraction completed.")

        wx.CallAfter(self.OnDoneExtractTest)

    def OnStartExtractInputOld(self): #modify
        #features = []
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        f = self.binary
        #f = cv2.cvtColor(f, cv2.COLOR_BGR2GRAY)
        #io.imsave("input_b4.jpg",f)
        #f = cv2.imread("input_b4.jpg", cv2.CV_LOAD_IMAGE_GRAYSCALE)
        f = cv2.resize(f, (28,28), interpolation=cv2.INTER_LANCZOS4)
        #cv2.imwrite("input.jpg",f)
        f = cv2.cv.fromarray(f)
        cv2.cv.Erode(f,f)
```

```
        f = np.asarray(f)
        #cv2.imwrite("after_erosion.jpg",f)

        feat = cv2.distanceTransform(f, cv2.cv.CV_DIST_L2, cv2.cv.CV_DIST_MASK_PRECISE)
        feat = tuple(itertools.chain(*feat))
        #features.append(feat)

        #self.features_input = np.array(features)
        self.features_input = np.array(feat)

        thread.start_new_thread(self.OnStartRedInput,())
        #wx.CallAfter(self.OnStartRed)

    def OnStartExtractInput(self): #modify
        #features = []
        # Classification module: Test if size is not 28x28 make sure it goes through resizing and morphological operations
        f = self.binary
        f = Image.fromarray(f)
        f = trim(f)
        f = f.convert('L')
        f = np.array(f)
        f = img_as_ubyte(f)
        f = cv2.resize(f, (16,16), interpolation=cv2.INTER_LANCZOS4)
        #cv2.imwrite("input.jpg",f)
        #f = cv2.cv.fromarray(f)
        #cv2.cv.Erode(f,f)
        #f = np.asarray(f)
        #cv2.imwrite("after_erosion.jpg",f)

        feat = ExtractDDD(f)
        feat = tuple(itertools.chain(*feat))
        feat = tuple(itertools.chain(*feat))
        #features.append(feat)

        #self.features_input = np.array(features)
        self.features_input = np.array(feat)

        #thread.start_new_thread(self.OnStartRedInput,())
        wx.CallAfter(self.OnDoneExtractInput)

    def OnStartRedInput(self): #modify
        # fit pca and std_scaler - save globally
        self.SetStatusText("Scaling the feature vector...")
        self.features_input = self.min_max_scaler.transform(self.features_input)
        #self.SetStatusText("Applying PCA...")
```

```python
        #self.features_input = self.pca.transform(self.features_input)
        self.SetStatusText("Extraction completed.")

        wx.CallAfter(self.OnDoneExtractInput)

    def OnDoneExtractInput(self): #modify
        self.busyDlg = None
        self.SetStatusText("Classifying image...")
        msg = "Please wait while the image is being classified..."
        self.busyDlg = wx.BusyInfo(msg)
        thread.start_new_thread(self.OnStartTestInput,())
        #wx.CallAfter(self.OnStartTrain)

    def OnStartTestInput(self):
        #self.pred_scaler = MinMaxScaler()
        self.pred_input = self.clf.predict(self.features_input)
        try:
            self.data = self.clf.predict_proba(self.features_input)
            temp = None
            for item in self.data:
                temp = item
        except AttributeError:
            self.data = self.clf.decision_function(self.features_input)
            temp = None
            for item in self.data:
                temp = item
        temp1 = None
        for item in self.pred_input:
            temp1 = str(int(item))
        #self.pred_input = self.pred_scaler.fit_transform(temp1)
        self.pred_input = temp1

        self.data = temp
        self.data = self.data.tolist()

        wx.CallAfter(self.OnDoneTestInput)

    def OnDoneTestInput(self):
        self.label_num.SetLabel(self.pred_input)
        #print self.data
        #print self.pred_input
        #self.window_plot.data = self.data

        #self.window_plot.GetParent().GetSizer().Show(self.window_plot)
```

```
        self.window_plot.GetParent().GetSizer().Layout()
        self.window_plot.draw_plot()
        self.busyDlg = None
        self.SetStatusText("Classification␣done.")

    def rowFormatterTrain(self,listItem,digit):
        if digit.predicted != digit.actual and self.classed == True:
            listItem.SetTextColour(wx.RED)

    def rowFormatterTest(self,listItem,digit):
        if digit.predicted != digit.actual and self.tested == True:
            listItem.SetTextColour(wx.RED)

class WindowPlot(wx.ScrolledWindow):

    def __init__(self, parent):
        wx.ScrolledWindow.__init__(self, parent)
        self.parent = parent
        self.gparent = self.parent.GetParent()
        self.ggparent = self.gparent.GetParent()
        #self.data = [0.02,0.3,0.45,0.56,0.67,0.78,0.89,0.95,0.21,0.1]
        self.__set_properties()
        #self.__do_layout()

    def __set_properties(self):
        self.SetScrollbars(20, 20, 50, 50)
        self.dpi = 100
        self.fig = Figure((5.0, 4.0), dpi=self.dpi, tight_layout=True)
        self.canvas = FigureCanvas(self, -1, self.fig)
        self.axes = self.fig.add_subplot(111)
        #self.axes = plt
        self.axes.set_ylabel('Classes')
        self.axes.set_xlabel('Relative␣Probability␣of␣Classes')
        #classes = ('0','1','2','3','4','5','6','7','8','9')
        classes = np.arange(10)
        self.y_pos = np.arange(len(classes))
        self.axes.set_yticklabels(self.y_pos,classes,minor=True)

    def draw_plot(self):
        self.axes.clear()
        self.axes.set_ylabel('Classes')
        self.axes.set_xlabel('Relative␣Probability␣of␣Classes')
        self.axes.barh(self.y_pos, self.ggparent.data, alpha=0.4)
```

```
        self.canvas.draw()

    """
␣␣␣␣def␣rowFormatterTrain(listItem,␣customer):
␣␣␣␣␣␣␣␣if␣customer.amountOwed␣>␣0:
␣␣␣␣␣␣␣␣␣␣␣␣listItem.SetTextColour(wx.RED)

␣␣␣␣def␣rowFormatterTest(listItem,␣customer):
␣␣␣␣␣␣␣␣if␣customer.amountOwed␣>␣0:
␣␣␣␣␣␣␣␣␣␣␣␣listItem.SetTextColour(wx.RED)
␣␣␣␣"""

# end of class MyFrame
```

```
# entry.py

class Entry(object):
    def __init__(self, image, actual, predicted):
        self.image = image
        self.actual = actual
        self.predicted = predicted
```

```
import wx

class AnalyseDialog(wx.Dialog):
    def __init__(self, parent):
        # begin wxGlade: MyDialog.__init__
        #kwds["style"] = wx.DEFAULT_DIALOG_STYLE
        wx.Dialog.__init__(self, parent, style=wx.DEFAULT_DIALOG_STYLE|wx.RESIZE_BORDER)
        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)
        self.grid_analyse = wx.grid.Grid(self, wx.ID_ANY, size=(1, 1))
        self.grid_analyse.DisableCellEditControl()
        self.grid_analyse.AutoSize()

        self.__set_properties()

        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: MyDialog.__set_properties
        self.SetTitle(_("Result␣Analysis"))
        self.label_desc.SetFont(wx.Font(12, wx.DEFAULT, wx.NORMAL, wx.NORMAL, 0, ""))
        self.grid_analyse.CreateGrid(10, 10)
        self.grid_analyse.EnableEditing(0)
```

```
        self.grid_analyse.EnableDragColSize(0)
        self.grid_analyse.EnableDragRowSize(0)
        self.grid_analyse.EnableDragGridSize(0)
        self.grid_analyse.SetColLabelValue(0, _("0"))
        self.grid_analyse.SetColSize(0, 5)
        self.grid_analyse.SetColLabelValue(1, _("1"))
        self.grid_analyse.SetColSize(1, 5)
        self.grid_analyse.SetColLabelValue(2, _("2"))
        self.grid_analyse.SetColSize(2, 5)
        self.grid_analyse.SetColLabelValue(3, _("3"))
        self.grid_analyse.SetColSize(3, 5)
        self.grid_analyse.SetColLabelValue(4, _("4"))
        self.grid_analyse.SetColSize(4, 5)
        self.grid_analyse.SetColLabelValue(5, _("5"))
        self.grid_analyse.SetColSize(5, 5)
        self.grid_analyse.SetColLabelValue(6, _("6"))
        self.grid_analyse.SetColSize(6, 5)
        self.grid_analyse.SetColLabelValue(7, _("7"))
        self.grid_analyse.SetColSize(7, 5)
        self.grid_analyse.SetColLabelValue(8, _("8"))
        self.grid_analyse.SetColSize(8, 5)
        self.grid_analyse.SetColLabelValue(9, _("9"))
        self.grid_analyse.SetColSize(9, 5)

        self.grid_analyse.SetRowLabelValue(0, _("0"))
        self.grid_analyse.SetRowSize(0, 5)
        self.grid_analyse.SetRowLabelValue(1, _("1"))
        self.grid_analyse.SetRowSize(1, 5)
        self.grid_analyse.SetRowLabelValue(2, _("2"))
        self.grid_analyse.SetRowSize(2, 5)
        self.grid_analyse.SetRowLabelValue(3, _("3"))
        self.grid_analyse.SetRowSize(3, 5)
        self.grid_analyse.SetRowLabelValue(4, _("4"))
        self.grid_analyse.SetRowSize(4, 5)
        self.grid_analyse.SetRowLabelValue(5, _("5"))
        self.grid_analyse.SetRowSize(5, 5)
        self.grid_analyse.SetRowLabelValue(6, _("6"))
        self.grid_analyse.SetRowSize(6, 5)
        self.grid_analyse.SetRowLabelValue(7, _("7"))
        self.grid_analyse.SetRowSize(7, 5)
        self.grid_analyse.SetRowLabelValue(8, _("8"))
        self.grid_analyse.SetRowSize(8, 5)
        self.grid_analyse.SetRowLabelValue(9, _("9"))
        self.grid_analyse.SetRowSize(9, 5)
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: MyDialog.__do_layout
        sizer_23 = wx.BoxSizer(wx.VERTICAL)
        sizer_23.Add(self.label_desc, 1, wx.ALL | wx.EXPAND |
           wx.ALIGN_CENTER_HORIZONTAL | wx.
```

```
            ALIGN_CENTER_VERTICAL, 5)
        sizer_23.Add(self.grid_analyse, 3, wx.ALL | wx.EXPAND, 5)
        self.SetSizer(sizer_23)
        sizer_23.Fit(self)
        self.Layout()
        # end wxGlade

class AboutDialog(wx.Dialog):
    def __init__(self, *args, **kwds):
        # begin wxGlade: ResultDialog.__init__
        kwds["style"] = wx.DEFAULT_DIALOG_STYLE
        wx.Dialog.__init__(self, *args, **kwds)
        self.label_1 = wx.StaticText(self, wx.ID_ANY, _("␣This␣program␣is␣written␣by␣Chong␣Yong␣Shean␣\n␣Compiled␣using␣Python␣\n␣GUI␣created␣using␣wxPython␣with␣the␣aid␣of␣wxGlade"))
        self.button_1 = wx.Button(self, wx.ID_OK, "")

        self.__set_properties()
        self.__do_layout()

        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button_1)

        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: ResultDialog.__set_properties
        self.SetTitle(_("About␣the␣Author"))
        #_icon = wx.EmptyIcon()
        #_icon.CopyFromBitmap(wx.Bitmap("C:\\Documents and Settings\\YongShean\\My Documents\\icon.png", wx.BITMAP_TYPE_ANY))
        #self.SetIcon(_icon)
        self.label_1.SetFont(wx.Font(14, wx.MODERN, wx.NORMAL, wx.BOLD, 0, ""))
        # end wxGlade

    def __do_layout(self):
        # begin wxGlade: ResultDialog.__do_layout
        sizer_3 = wx.BoxSizer(wx.VERTICAL)
        sizer_4 = wx.BoxSizer(wx.HORIZONTAL)
        sizer_4.Add(self.label_1, 0, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        sizer_3.Add(sizer_4, 1, wx.EXPAND, 0)
        sizer_3.Add(self.button_1, 0, wx.ALIGN_CENTER_HORIZONTAL | wx.ALIGN_CENTER_VERTICAL, 0)
        self.SetSizer(sizer_3)
        sizer_3.Fit(self)
```

```
        self.Layout()
        # end wxGlade

    def OnClick(self, event):  # wxGlade: ResultDialog.<
       event_handler>
        self.Destroy()
        event.Skip()
```

A-0-8 Training and Testing of Combined Dataset

```
# train_dataset_improved.py
# Training and testing with stratified k-fold cross validation

from sklearn.preprocessing import MinMaxScaler
from sklearn.cross_validation import StratifiedKFold
import numpy as np
import time
from sklearn.pipeline import Pipeline
from sklearn import svm
from sklearn.externals import joblib
from sklearn.metrics import classification_report,
   accuracy_score, confusion_matrix, precision_recall_curve
from time import strftime
import matplotlib.pyplot as plt

_kernel = 'rbf'
_c = 2.8
_gamma = 0.0073
_tol = 0.001
_max_iter = -1
_dual = False
_deg = 5

print "Loading␣training␣set..."
features = np.load('features.npy')
labels = np.load('labels.npy').flatten()

start_run = time.time()
min_max_scaler = MinMaxScaler(feature_range=(-1, 1))

class_weights = np.bincount(labels)
d = dict()
for i,w in zip(range(len(class_weights)+1),class_weights):
   d[i] = w
# Set weights here #
if _kernel=='linear':
    clf = svm.LinearSVC(C=_c, dual=_dual, tol=_tol,
       class_weight=d)
elif _kernel=='poly':
    clf = svm.SVC(kernel='poly',C=_c, gamma=_gamma, tol=_tol,
       max_iter=_max_iter, degree=_deg, probability=True,
       class_weight=d)
```

```
elif _kernel=='rbf':
     clf = svm.SVC(kernel='rbf', C=_c, gamma=_gamma, tol=_tol, max_iter=_max_iter, probability=True, class_weight=d)
elif _kernel=='sigmoid':
     clf = svm.SVC(kernel='sigmoid', C=_c, gamma=_gamma, tol=_tol, max_iter=_max_iter, probability=True, class_weight=d)

print "Splitting training and testing set using Stratified K-fold..."
skf = StratifiedKFold(labels, n_folds=3)
k=1
for train_index, test_index in skf:
   features_train, features_test = features[train_index], features[test_index]
   labels_train, labels_test = labels[train_index], labels[test_index]

   print "Training dataset..."
   scaling_svm = Pipeline([("scaler", min_max_scaler), ("svm", clf)])
   scaling_svm.fit(features_train, labels_train)
   joblib.dump(scaling_svm, 'scaling_svm_new_fold'+str(k)+'.pkl')
   end_run = time.time()
   print "Training completed."
   print "Fold no."+str(k)+": Time elapsed", round(end_run-start_run,4), "seconds"

   print "Testing dataset..."
   print("Detailed classification report (Training):")
   print()
   print("The model is trained on the full development set.")
   print("The scores are computed on the full evaluation set.")
   print()
   y_true, y_pred = labels_train, scaling_svm.predict(features_train)
   print(classification_report(y_true, y_pred))
   print()
   print "Accuracy: ", accuracy_score(y_true, y_pred)
   print()
   print("Confusion matrix")
   conf_arr = confusion_matrix(y_true,y_pred)
   norm_conf = []
   for i in conf_arr:
      a = 0
      tmp_arr = []
      a = sum(i, 0)
      for j in i:
         tmp_arr.append(float(j)/float(a))
```

```
        norm_conf.append(tmp_arr)

    fig = plt.figure()
    plt.clf()
    ax = fig.add_subplot(111)
    ax.set_aspect(1)
    res = ax.imshow(np.array(norm_conf), cmap=plt.cm.jet,
                    interpolation='nearest')

    width = len(conf_arr)
    height = len(conf_arr[0])

    cb = fig.colorbar(res)
    plt.savefig('confusion_matrix_train'+str(k)+'.png', format='png')
    print()
    print("Precision-Recall␣Curve")
    from sklearn import preprocessing
    lb = preprocessing.LabelBinarizer()
    y_true = lb.fit_transform(y_true)
    y_pred = lb.fit_transform(y_pred)
    precision, recall, thresholds = precision_recall_curve(y_true.flatten(),scaling_svm.predict_proba(features_train),pos_label=1)
    plt.plot(thresholds, precision, 'r', thresholds, recall, 'b')
    plt.xlabel('Threshold')
    plt.ylabel('Precision/Recall')
    plt.title('Precision/Recall␣vs␣Threshold')
    plt.savefig('precision_recall_threshold_train'+str(k)+'.png', format='png')
    print()
    np.save('labels_train_predicted_new_fold'+str(k)+'.npy', y_pred)

    print("Detailed␣classification␣report␣(Testing):")
    print()
    print("The␣model␣is␣trained␣on␣the␣full␣development␣set.")
    print("The␣scores␣are␣computed␣on␣the␣full␣evaluation␣set.")
    print()
    y_true, y_pred = labels_test, scaling_svm.predict(features_test)
    print(classification_report(y_true, y_pred))
    print()
    print "Accuracy:␣", accuracy_score(y_true, y_pred)
    print()
    print("Confusion␣matrix")
    conf_arr = confusion_matrix(y_true,y_pred)
    norm_conf = []
    for i in conf_arr:
```

```
        a = 0
        tmp_arr = []
        a = sum(i, 0)
        for j in i:
            tmp_arr.append(float(j)/float(a))
        norm_conf.append(tmp_arr)

    fig = plt.figure()
    plt.clf()
    ax = fig.add_subplot(111)
    ax.set_aspect(1)
    res = ax.imshow(np.array(norm_conf), cmap=plt.cm.jet,
                    interpolation='nearest')

    width = len(conf_arr)
    height = len(conf_arr[0])

    cb = fig.colorbar(res)
    plt.savefig('confusion_matrix_test'+str(k)+'.png', format='png')
    print()
    print("Precision-Recall Curve")
    from sklearn import preprocessing
    lb = preprocessing.LabelBinarizer()
    y_true = lb.fit_transform(y_true)
    y_pred = lb.fit_transform(y_pred)
    precision, recall, thresholds = precision_recall_curve(y_true.flatten(),scaling_svm.predict_proba(features_test),pos_label=1)
    plt.plot(thresholds, precision, 'r', thresholds, recall, 'b')
    plt.xlabel('Threshold')
    plt.ylabel('Precision/Recall')
    plt.title('Precision/Recall vs Threshold')
    plt.savefig('precision_recall_threshold_test'+str(k)+'.png', format='png')
    print()
    np.save('labels_test_predicted_new_fold'+str(k)+'.npy', y_pred)
    k+=1
```

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

Directional Distance Distribution

```
from skimage import io
import numpy as np

def ExtractDDD(img):
    wb = [[[0 for i in range(16)] for j in range(16)] for k in range(16)]

```

```
    for n in range(16):
        for m in range(16):
            if img[n][m]<128:
                for k in range(16): # East
                    wb[n][m][0]+=1
                    if m+k<=15:
                        if img[n][m+k]>=128:
                            break
                    else:
                        if img[n][m+k-16]>=128:
                            break
                for k in range(16): # West
                    wb[n][m][4]+=1
                    if m-k<=15:
                        if img[n][m-k]>=128:
                            break
                    else:
                        if img[n][m-k-16]>=128:
                            break
                for k in range(16): # South
                    wb[n][m][6]+=1
                    if n+k<=15:
                        if img[n+k][m]>=128:
                            break
                    else:
                        if img[n+k-16][m]>=128:
                            break
                for k in range(16): # North
                    wb[n][m][2]+=1
                    if n-k<=15:
                        if img[n-k][m]>=128:
                            break
                    else:
                        if img[n-k-16][m]>=128:
                            break
                for k in range(16): # South East
                    wb[n][m][7]+=1
                    if m+k<=15 and n+k<=15:
                        if img[n+k][m+k]>=128:
                            break
                    if m+k<=15 and n+k>=16:
                        if img[n+k-16][m+k]>=128:
                            break
                    if m+k>=16 and n+k<=15:
                        if img[n+k][m+k-16]>=128:
                            break
                    elif m+k>=16 and n+k>=16:
                        if img[n+k-16][m+k-16]>=128:
                            break
                for k in range(16): # North West
                    wb[n][m][3]+=1
```

```
                    if m-k<=15 and n-k<=15:
                        if img[n-k][m-k]>=128:
                            break
                    if m-k<=15 and n-k>=16:
                        if img[n-k-16][m-k]>=128:
                            break
                    if m-k>=16 and n-k<=15:
                        if img[n-k][m-k-16]>=128:
                            break
                    elif m-k>=16 and n-k>=16:
                        if img[n-k-16][m-k-16]>=128:
                            break
                for j,k in zip(range(16),range(16)): # South West
                    wb[n][m][5]+=1
                    if n+k<=15:
                        if m-k<=15:
                            if img[n+k][m-k]>=128:
                                break
                        else:
                            if img[n+k][m-k-16]>=128:
                                break
                    else:
                        if m-k<=15:
                            if img[n+k-16][m-k]>=128:
                                break
                        else:
                            if img[n+k-16][m-k-16]>=128:
                                break
                for j,k in zip(range(16),range(16)): # North East
                    wb[n][m][1]+=1
                    if n-k<=15:
                        if m+k<=15:
                            if img[n-k][m+k]>=128:
                                break
                        else:
                            if img[n-k][m+k-16]>=128:
                                break
                    else:
                        if m+k<=15:
                            if img[n-k-16][m+k]>=128:
                                break
                        else:
                            if img[n-k-16][m+k-16]>=128:
                                break
            if img[n][m]>=128:
                for k in range(16): # East
                    wb[n][m][8]+=1
                    if m+k<=15:
                        if img[n][m+k]<128:
```

```
                        break
                else:
                    if img[n][m+k-16]<128:
                        break
            for k in range(16): # West
                wb[n][m][12]+=1
                if m-k<=15:
                    if img[n][m-k]<128:
                        break
                else:
                    if img[n][m-k-16]<128:
                        break
            for k in range(16): # South
                wb[n][m][14]+=1
                if n+k<=15:
                    if img[n+k][m]<128:
                        break
                else:
                    if img[n+k-16][m]<128:
                        break
            for k in range(16): # North
                wb[n][m][10]+=1
                if n-k<=15:
                    if img[n-k][m]<128:
                        break
                else:
                    if img[n-k-16][m]<128:
                        break
            for k in range(16): # South East
                wb[n][m][15]+=1
                if m+k<=15:
                    if n+k<=15:
                        if img[n+k][m+k]<128:
                            break
                    else:
                        if img[n+k-16][m+k]<128:
                            break
                else:
                    if n+k<=15:
                        if img[n+k][m+k-16]<128:
                            break
                    else:
                        if img[n+k-16][m+k-16]<128:
                            break
            for k in range(16): # North West
                wb[n][m][11]+=1
                if m-k<=15:
                    if img[n-k][m-k]<128:
                        break
                else:
                    if img[n-k-16][m-k-16]<128:
```

```
                            break
                for j,k in zip(range(16),range(16)): # South West
                    wb[n][m][13]+=1
                    if n+k<=15:
                        if m-k<=15:
                            if img[n+k][m-k]<128:
                                break
                        else:
                            if img[n+k][m-k-16]<128:
                                break
                    else:
                        if m-k<=15:
                            if img[n+k-16][m-k]<128:
                                break
                        else:
                            if img[n+k-16][m-k-16]<128:
                                break
                for j,k in zip(range(16),range(16)): # North East
                    wb[n][m][9]+=1
                    if n-k<=15:
                        if m+k<=15:
                            if img[n-k][m+k]<128:
                                break
                        else:
                            if img[n-k][m+k-16]<128:
                                break
                    else:
                        if m+k<=15:
                            if img[n-k-16][m+k]<128:
                                break
                        else:
                            if img[n-k-16][m+k-16]<128:
                                break

    wb_new = [[0 for i in range(4)] for j in range(4)]

    wb = np.array(wb)

    n=0
    m=0
    for i in range(0,13,4):
        for j in range(0,13,4):
            temp = wb[i:i+4,j:j+4]
            temp = temp.mean(0)
            wb_new[n][m] = temp.mean(0)
            m+=1
        n+=1
        m=0
    wb_new = np.array(wb_new)
```

```
    return wb_new
```

Function Definitions

```
import numpy as np
from PIL import Image, ImageChops
from skimage import io
from skimage.filter import threshold_otsu
import cv2
from skimage import img_as_float, img_as_ubyte
import itertools
from sklearn import preprocessing
from sklearn.decomposition import RandomizedPCA
from sklearn.preprocessing import StandardScaler
import gc
from sklearn.externals import joblib
from sklearn import svm
from ddd import *

def cv_to_pil(im):
   #im = img_as_float(im)
   im = Image.fromarray(im)

   return im

def pil_to_cv(im):
   im = im.convert('L')
   im = np.array(im)
   #im = img_as_ubyte(im)

   return im

def cv_to_skimage(im):
   return img_as_float(im)

def skimage_to_cv(im):
   return img_as_ubyte(im)

def pil_to_skimage(im):
   im = im.convert('L')
   im = np.array(im)

   return im

def skimage_to_pil(im):
   return Image.fromarray(im)

def binarize(img):
   thresh = threshold_otsu(img)
   binary = img > thresh
   binary = binary.astype(int)*255
```

```

    return binary
    # Skimage image

def window(iterable, size=2):
    i = iter(iterable)
    win = []
    for e in range(0, size):
        win.append(next(i))
    yield win
    for e in i:
        win = win[1:] + [e]
        yield win

def trim(im):
    bg = Image.new(im.mode, im.size, im.getpixel((0,0)))
    diff = ImageChops.difference(im, bg)
    diff = ImageChops.add(diff, diff, 2.0, -100)
    bbox = diff.getbbox()
    if bbox:
        return im.crop(bbox)

def ExtractField(im):
    pw, ph = im.size
    cw = int(0.6*pw) #0.64
    ch = int(0.4*ph) #0.4
    rw = int(0.5*cw) #0.52
    rh = int(0.4*ch) #0.44
    region = (cw,ch,cw+rw,ch+rh)
    area = im.crop(region)
    area = area.rotate(90)

    area = pil_to_cv(area)

    i=0
    min_i = 999
    max_i = -1
    cnt_b=0
    cnt_w=0

    for f in window(area):
        i+=1
        f = np.array(f)
        black = False
        for num in f:
            for j in num:
                if j.any()==0:
                    black = True
                    break
        if black==True:
            if i<min_i:
```

```
                min_i = i
            elif i>max_i:
                max_i = i

    region = (0,min_i,rh,max_i)
    area = cv_to_pil(area)
    area = area.crop(region)
    area = area.rotate(-90)
    #area = trim(area)
    #area = pil_to_cv(area)

    return area

def dropfall(tile, direction='top-left'):
    from numpy import fliplr, flipud
    height, width = tile.shape
    cut = np.zeros(height)

    if direction=='bottom-left':
        tile = flipud(tile)
    elif direction=='bottom-right':
        tile = fliplr(tile)
        tile = flipud(tile)
    elif direction=='top-right':
        tile = fliplr(tile)

    for row in range(1, height+1):
        found_candidate=0
        found_start=0
        candidate_x=0
        candidate_y=0
        start_x=0
        start_y=0

        # Check the "right three fourths minus some" of each
        for col in range(int(round(width/4)), width):
            pres_pix = tile[row-1, col-1]
            prev_pix = tile[row-1, col-2]
            next_pix = tile[row-1, col]
            if pres_pix==0 and next_pix>0:
                found_candidate = 1
                candidate_x = col
                candidate_y = row
            if found_candidate==1 and pres_pix==0 and prev_pix>0:
                found_start = 1
                start_x = candidate_x + 1
                start_y = candidate_y

        if found_start==1:
            break

```

```
    # Start the drop fall!
    # Start defining the cut
    cut = start_x*np.ones(height)
    row = start_y
    col = start_x
    path_tile = tile

    cut_point_x = 0
    cut_point_y = 0

    while row<height and col<width:
        if path_tile[row, col-1]==255:
            row+=1
        elif path_tile[row, col]==255:
            row+=1
            col+=1
        elif path_tile[row, col-2]==255:
            row+=1
            col-=1
        elif path_tile[row-1, col]==255:
            col+=1
        elif path_tile[row-1, col-2]==255:
            col-=1
        else:
            if tile[row-1,col-1]==255:
                cut_point_y = row #point at which the cutting into the black begins
                cut_point_x = col
            row+=1
        path_tile[row-1, col-1] = 500 # This marks where the path has been
        cut[row-1] = col

    width1 = max(cut)-1
    min_cut = min(cut)
    width2 = width - min_cut + 1
    tile1 = 255 * np.ones((height, width1))
    tile2 = 255 * np.ones((height, width2))

    for i in range(height):
        x_cut = int(cut[i])
        tile1[i, 1:(x_cut-1)] = tile[i, 1:(x_cut-1)]
        tile2[i, (x_cut - min_cut + 1):width2] = tile[i, x_cut:width]

    if direction=='bottom-left':
        tile1 = flipud(tile1)
        tile2 = flipud(tile2)
    elif direction=='bottom-right':
        tile1 = fliplr(tile1)
        tile1 = flipud(tile1)
```

```
        tile2 = fliplr(tile2)
        tile2 = flipud(tile2)
        tile1, tile2 = tile2, tile1
    elif direction=='top-right':
        tile1 = fliplr(tile1)
        tile2 = fliplr(tile2)
        tile1, tile2 = tile2, tile1

    tile1 = tile1.astype(int)
    tile2 = tile2.astype(int)

    try:
        io.imsave('tile1.jpg',tile1)
        io.imsave('tile2.jpg',tile2)

        tile1 = Image.open('tile1.jpg')
        tile2 = Image.open('tile2.jpg')

        tile1 = trim(tile1)
        tile2 = trim(tile2)

        tile1 = pil_to_cv(tile1)
        tile2 = pil_to_cv(tile2)
    except ValueError:
        print "Dropfall error!"
        return tile1, tile2

    return tile1, tile2

def Segmentation(im):
    #im = pil_to_cv(im)

    #im = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    ret,img = cv2.threshold(im,128,255,cv2.THRESH_OTSU)
    ret,img = cv2.threshold(img,0,255,cv2.THRESH_BINARY_INV)
    contours,hierarchy = cv2.findContours(img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    img_lst = []
    prop_min = []

    for i in range(len(contours)):
        prop_min.append([i,min(x for [[x,y]] in contours[i])])

    prop_min = np.array(prop_min)
    # sort array with regards to 2nd column
    prop_min = prop_min[prop_min[:,1].argsort()]
    img_list = [contours[x] for [x,y] in prop_min]
    img_list_pos = prop_min

    i=0
    y_arr = []
```

```
    for currentContour in img_list:
        x,y,w,h = cv2.boundingRect(currentContour)
        #cv2.rectangle(im,(x,y),(x+w,y+h),(0,255,0),2)
        letter = im[y:y+h,x:x+w]
        if len(letter)>3 and len(letter[0])<0.7*len(im[0]): #
            CONSIDER REMOVE THIS CONDITION
            cv2.imwrite('img'+str(i)+'.jpg',letter)
            img_lst.append(letter)
            y_arr.append(y)
            i+=1

    heights = []
    widths = []

    for f in img_lst:
        height, width = f.shape
        heights.append(height)
        widths.append(width)

    mean_height = np.mean(heights)
    mean_width = np.mean(widths)

    return img_lst, mean_height, mean_width, y_arr

def FeatureExtraction(img_lst):
    features = []

    for f in img_lst:
        height, width = f.shape
        f = cv2.resize(f, (16,16), interpolation=cv2.
            INTER_LANCZOS4)
        feat = ExtractDDD(f)
        feat = list(itertools.chain(*feat))
        feat.append([height,width,width/height])
        feat = tuple(itertools.chain(*feat))
        features.append(feat)

    #for i in range(92):
        #features.append([0 for x in range(784)])
    features = np.array(features)
    #features = np.nan_to_num(features)
    #features = features[0:8]
    return features

def Recognise(input):
    clf_single = joblib.load("clf_min_max_scaled.pkl")

    single = clf_single.predict(input)
    try:
        confidence = clf_single.predict_proba(input)
    except AttributeError:
```

```
      confidence = clf_single.decision_function(input)

   output = []
   """
␣␣␣for␣[num_single,␣num_double]␣in␣zip(single,␣double):
␣␣␣␣␣␣if␣num_single␣not␣in␣range(10):
␣␣␣␣␣␣␣␣␣if␣num_double>=21:
␣␣␣␣␣␣␣␣␣␣␣␣output.append(num_double-11)
␣␣␣␣␣␣␣␣␣else:
␣␣␣␣␣␣␣␣␣␣␣␣output.append(num_double-10)
␣␣␣␣␣␣else:
␣␣␣␣␣␣␣␣␣output.append(num_single)
␣␣␣"""
   return single.astype(int), confidence

def GetPILBitmap(npimage):
      height, width = npimage.shape
      pilimage = Image.fromarray(npimage)
      rgb = pilimage.convert('RGB')
      return rgb
```