Appendix B

User's Guide

This chapter provides the user a step-by-step guide in using the programs that was developed to fulfill the objectives of the study.

B.1 Prerequisites

B.1.1 Technologies and Software

In able to run the programs included in this study, some technologies, libraries, and software must be installed. Table B.1 lists these needed technologies.

Technology/Software	Link	Description
GNU C++	`https://gcc.gnu.org/`	Needed for most programs
OpenCV 3	`http://opencv.org/`	For image processing
Weka	`http://www.cs.waikato.ac.nz/ml/weka/`	For Naive Bayes

Table B.1: List of Technologies/Software that should be installed

B.1.2 SigComp2009 Dataset

The dataset from ICDAR SigComp2009[1] that was used in this study is publicly available online. The files in the downloaded zip file is described in Table B.2.

[1] `http://tc11.cvc.uab.es/datasets/SigComp2009_1`

File Name	Description
6b_NFIforgeries.zip	Contains 624 image (PNG) files of offline forged signatures
6b_NFIgenuines.zip	Contains 940 image (PNG) files of offline genuine signatures
NFI_ONforgeries.zip	Contains 620 hwr files of online forged signatures
NFI_ONgenuines.zip	Contains 932 hwr files of online genuine signatures

Table B.2: Description of files included in SigComp2009-evaluation.zip

The 6b_NFIgenuines.zip contains the images the was used in training and validating the models for the offline signature recognition. While the 6b_NFIforgeries.zip contains the validation images for offline signature verification. Extract these files to your work folder.

B.2 Preprocessing

B.2.1 Image Padding and Resize

The *image_padding_resize.cpp* file is the program that converts the images from RGB to gray-scale, image padding, and image resizing.

1. Open the *image_padding_resize.cpp* file.
2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Save the file.
4. Make sure the *compileCV.sh* is in the same folder as the source file.
5. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh image_padding_resize.cpp
   ```

   The resulting images will be saved with a $1024x512$ prefix in the same folder as defined in $PATH$.

B.2.2 Histogram Equalization

The $histogram_eq.cpp$ file is the program that applies histogram equalization in the input images.

1. Open the $histogram_eq.cpp$ file.
2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Save the file.
4. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh histogram_eq.cpp
   ```

   The resulting images will be saved with a $histogramEq$ prefix in the same folder as defined in $PATH$.

B.2.3 Image centering

The $center_image.cpp$ file is the program that repositions the signature to the middle of the canvas.

1. Open the $center_image.cpp$ file.
2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Save the file.
4. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh center_image.cpp
   ```

B.2.4 Generating additional images

The $shift_pixels.cpp$ file is the program that is capable of shifting the pixels of an input image.

1. Open the $shift_pixels.cpp$ file.
2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Change the number of pixels to be shifted.

   ```
   translateImg(src,10,0);
   ```

   The second parameter is for the x-axis shift, a positive number shifts it to the right while a negative number shifts it to the left. The third parameter is for the y-axis shift, a positive number shifts the image upward while a negative number shifts it downward.

4. Save the file.
5. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh shift_pixels.cpp
   ```

B.2.5 Image resizing

The $resize_images.cpp$ file is the program that resizes the input images to four sizes: 8×4, 16×8, 32×16, and 64×32.

1. Open the $resize_images.cpp$ file.
2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Change the definition of $PATHDST8$ to the folder path where the 8×4 resized images will be saved.

   ```
   #define PATHDST8 "C:/resized/8x4/"
   ```

4. Change the definition of $PATHDST16$ to the folder path where the 16 × 8 resized images will be saved.

   ```
   #define PATHDST16 "C:/resized/16x8/"
   ```

5. Change the definition of $PATHDST32$ to the folder path where the 32 × 16 resized images will be saved.

   ```
   #define PATHDST32 "C:/resized/32x16/"
   ```

6. Change the definition of $PATHDST64$ to the folder path where the 64 × 32 resized images will be saved.

   ```
   #define PATHDST64 "C:/resized/64x32/"
   ```

7. Save the file.

8. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh resize_images.cpp
   ```

B.3 Feature Extraction

Before generating the models, the signature must be represented by extracting features in the images and saving it as a text files with scaled or binary values.

1. Open the $create_file_data.cpp$ file.

2. Change the definition of $PATH$ to the folder path of the images.

   ```
   #define PATH "genuines/"
   ```

3. Save the file.

4. Run the program by executing the following command in terminal.

   ```
   $ bash compileCV.sh create_file_data.cpp
   ```

   The scaled and binary output files will be saved to the where $PATH$ is.

B.4 Data Preparation

The *genData.cpp* program randomly splits the datasets into a training and validation set.

1. Run the program by typing the following code in the terminal:

   ```
   $ g++ genData.cpp -o genData
   $ ./genData [FILENAME]
   ```

   where $[FILENAME]$ should be replaced with the file name of the dataset to be used. This will generate two files: *training.in* and *test.in*.

2. The program will then ask for the number of columns. Input the number of features of the dataset plus one for the target attribute (e.g. 512 for 32×16 datasets).

B.5 ANN training

The *backp.cpp* program trains the weights of an Artificial Neural Network given a specified input. The source file must first be configured first before running the program.

1. Configure the program by opening the *backp.cpp* using a text editor or emacs/vim/nano in the terminal.

2. Modify the neural network design by configuring the number of nodes in the source file.

   ```
   // Change these according to preferred ANN design
   #define N 3174
   #define NT 101
   #define NI 2049
   #define NH 1024
   ```

   The nodes in the input layer can be changed using NI, this must be equal to the number of features of a dataset plus a bias node. For the hidden layer, NH is usually half of the NI nodes. The NT should not be changed unless additional authors is added to the dataset. Add all these nodes and replace N with the sum.

3. Change the definition of *numRecs* to the number of records in the training set.

```
// Change this to the number of instances in training set
#define numRecs 2820
```

4. Change the path of the output weight file by changing the definition of $PATH$.

```
// Change this to the desired path of output weights
#define PATH "64x32_sca_shi15_weight.in"
```

5. Save the source file.

6. Run the program by typing the following code in the terminal:

```
$ g++ backp.cpp -o backp
$ ./backp [FILEPATH]
```

where $[FILEPATH]$ is the path of the file name of the training set. This program generates an output weight file that is used in model evaluation phase.

B.6 ANN validation

The *recognition.cpp* evaluates a model using the weights of an ANN as input file. The source file must first be configured first before running the program.

1. Configure the program by opening the *recognition.cpp* using a text editor or emacs/vim/nano in the terminal.

2. Modify the neural network design by configuring the number of nodes in the source file.

```
// Change these according to preferred ANN design
#define N 3174
#define NT 101
#define NI 2049
#define NH 1024
```

The nodes in the input layer can be changed using NI, this must be equal to the number of features of a dataset plus a bias node. For the hidden layer, NH is usually half of the NI

nodes. The NT should not be changed unless additional authors is added to the dataset. Add all these nodes and replace N with the sum.

3. Change the definition of *numRecs* to the number of records in the training set.

```
// Change these to the number of instances in validation set
#define nTests 624
```

4. Change the path of the input weight file by changing the definition of $WGHTSPATH$.

```
// Change this to the path of the weight file
#define WGHTSPATH "weights2/64x32_bin_shi05_weight.in"
```

5. Change the definition of $RESPATH$ to the desired file name of the output results log.

```
// Change this to the desired path where the results will be saved
#define RESPATH "64x32_bin_shi00_res.out"
```

6. Save the source file.

7. Run the program by typing the following code in the terminal:

```
$ g++ res.cpp -o res
$ ./res [FILEPATH]
```

where $[FILEPATH]$ is the path of the file name of the validation set. This program generates an output log file that contains the results of model evaluation phase.

B.7 Weka

B.7.1 Converting the datasets to ARFF

Before Weka could use the datasets produced in feature extraction, it must first be converted into an ARFF file.

1. Change the definition of $ARFF$ to the desired file name of the ARFF file.

```
// Change this to the desired file name of the ARFF file
#define ARFF "08x04_bin_shi00.arff"
```

2. Change the definition of $PATH$ to the file name of the dataset to be converted.

```
// Change this to the path of the dataset file
#define PATH "64x32_bin_forged.in"
```

3. Change the definition of $feats$ to the number of features in the dataset.

```
// Change this to the number of features of the dataset
#define feats 2048
```

4. Save the source file.

5. Run the program by typing the following code in the terminal:

```
$ g++ convARFF.cpp -o convARFF
$ ./convARFF
```

   This program generates the ARFF file that Weka can use for model generation and evaluation.

B.7.2 Using the Weka Program

Weka is used to do the model generation and evaluation of Naive Bayes.

1. Open Weka.
2. Click Explorer.
3. Click open file button and select the desired ARFF file.
4. Click the Classify tab.
5. Click the choose button, then click the bayes folder, then select the NaiveBayes.
6. Change the test option to percentage split with a value of 60%
7. Click start.
8. The result of the training and validation is presented in the results buffer.