CS 6375 Machine Learning Final Project Report

Handwriting recognition

Kainan Xu (kxx140830)

Yushen Fang(yxf151330)

Yishi Wang(yxw166030)

Spring 2017

1. Abstract

As we all know, comparing with the hand-written text, the digital text is easy to read, store, copy, edit, transport, etc. The handwriting recognition is a technique that converts the handwritten text into digital text. This technique refers to many fields such as image recognition, image processing, and machine learning. In our project, we had an attempt to build a handwriting recognition program using data we found on public websites and machine learning algorithms we learnt.

1. Task description

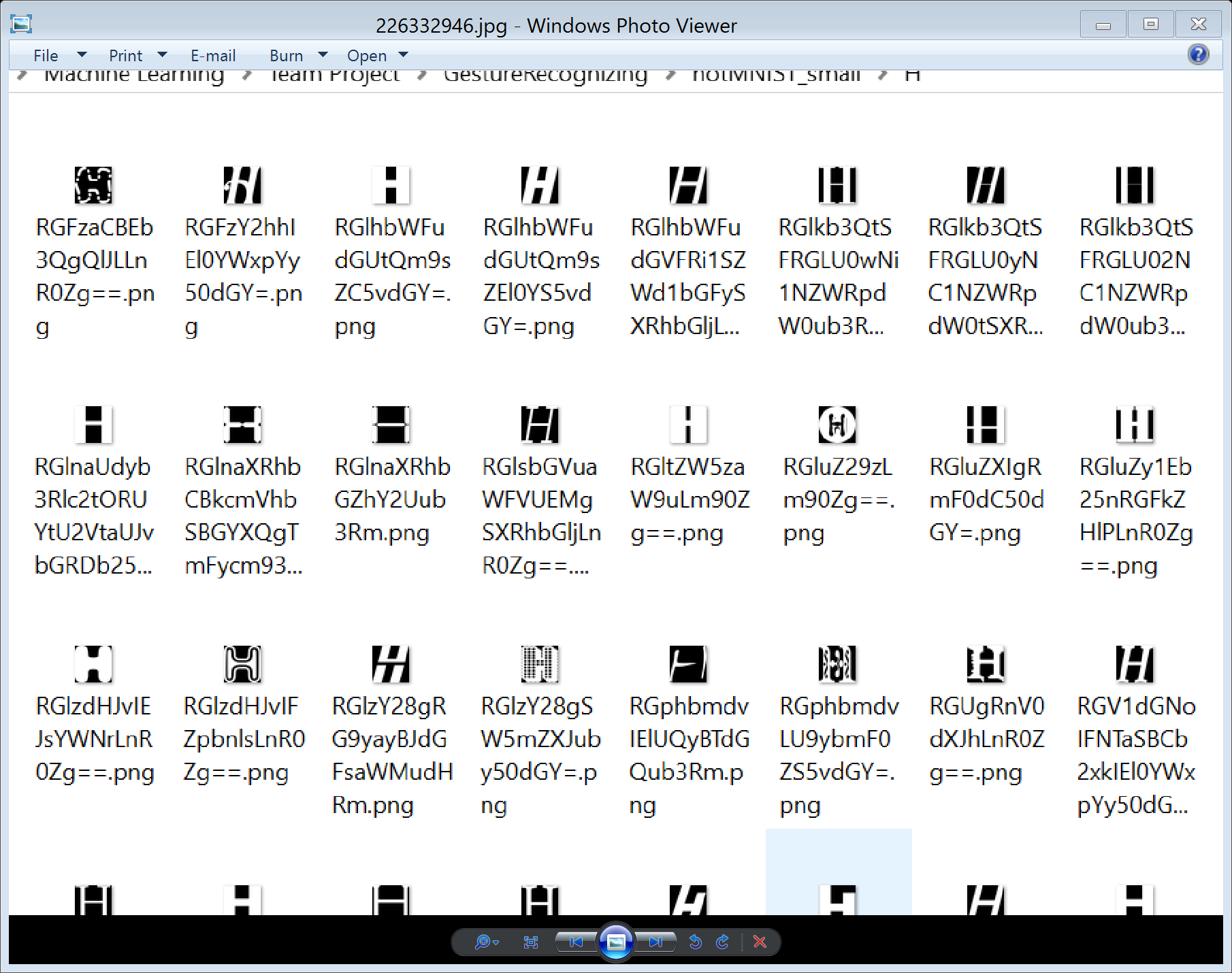
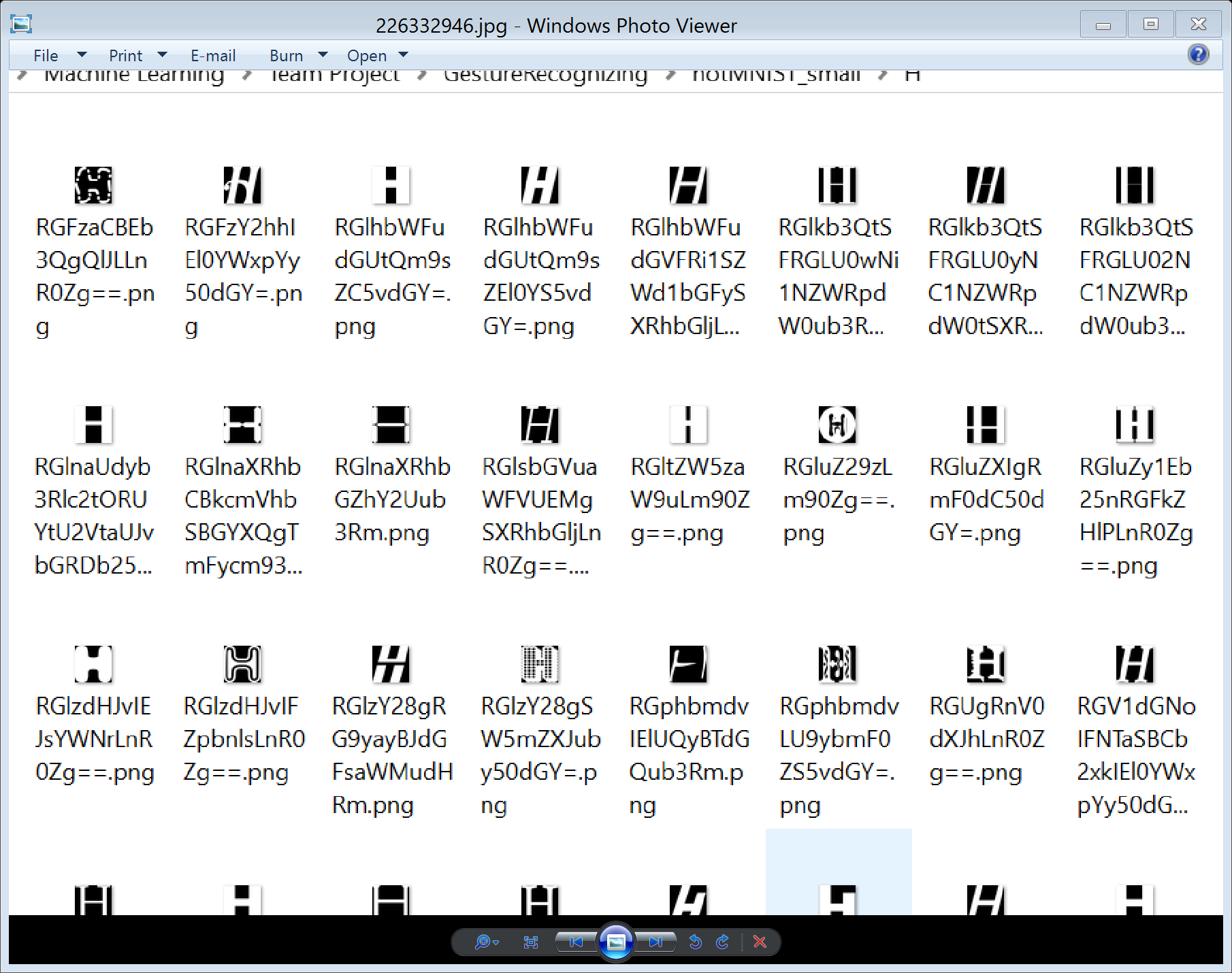
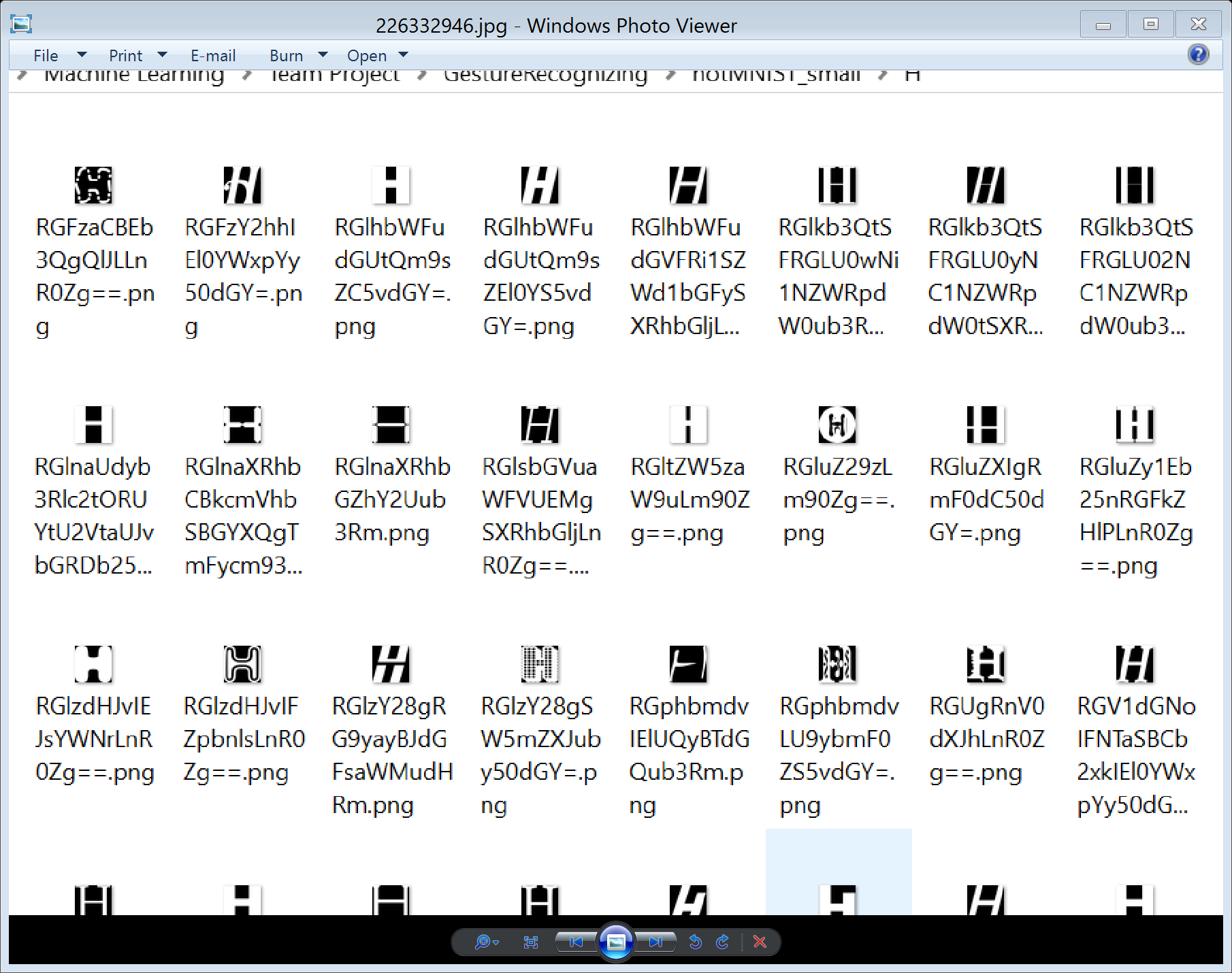
We built a program to convert the hand-written English characters into digital text. This program was written in Java which outputs a GUI first to request users to input a single English character by mouse. Then, our program will convert the input image into a storable text file. After that, our program will use our training data to analyze the storable text file and use machine learning algorithm to decide which character it is.

1. Training Dataset (dataset)

We used the EMNIST data set as our training data set. [1] In this dataset, there are different characters` images. Each of these images was converted to 28\*28 pixels image and stored in one readable .txt file. We will use the characters “H” and “A” as samples to show you our dataset.



The images above show how character “H” originally look like. Taken a close look at these images, we can find that some of them are general written “H” while some of them are not. For example, the image (a), (b), and (c) are general “H”. However, there are many vague (or we can say strange) “H” such as image (d), (e). All of them will be converted into 28\*28 pixels images and finally be used as training data.



(c)

(a)

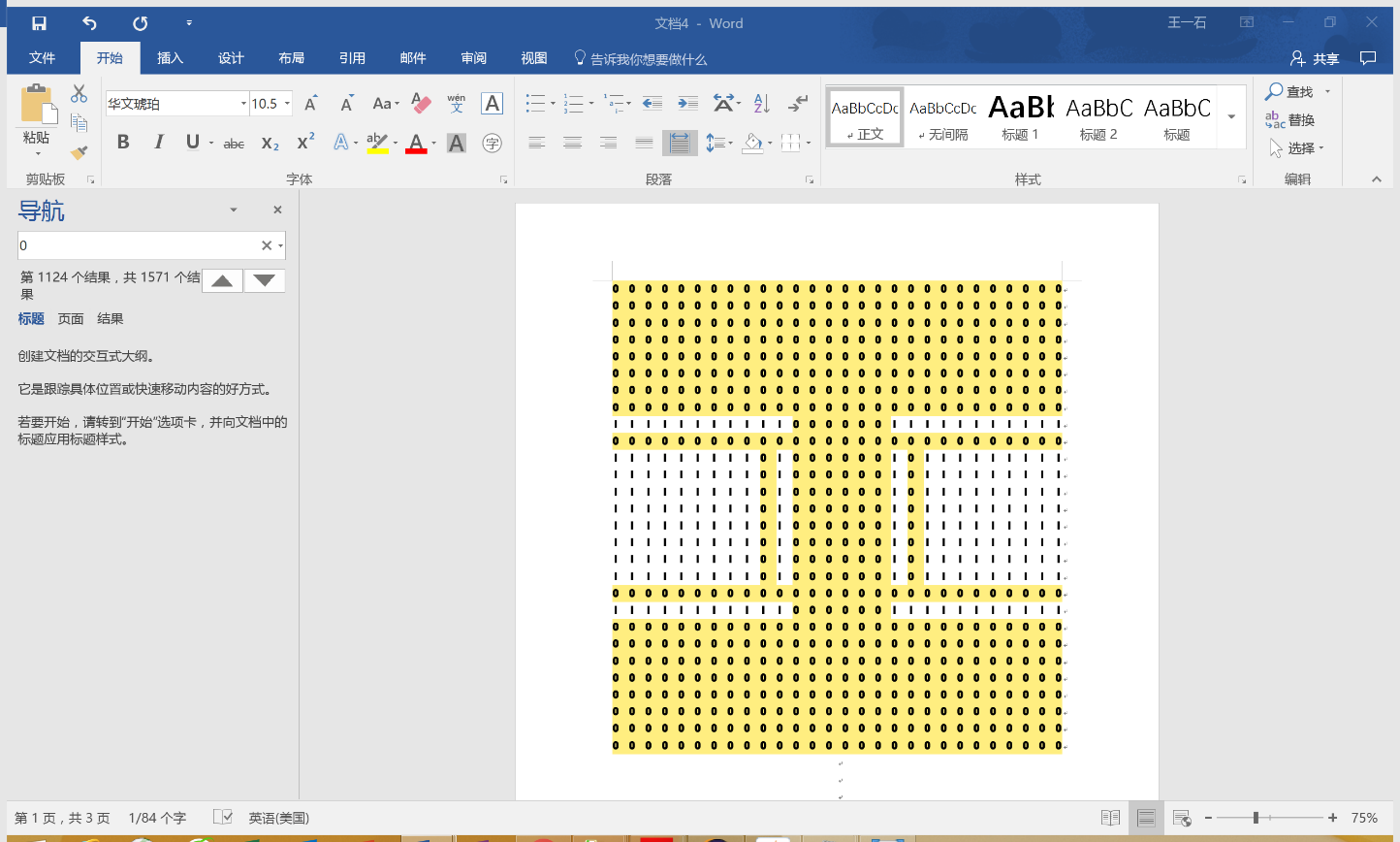
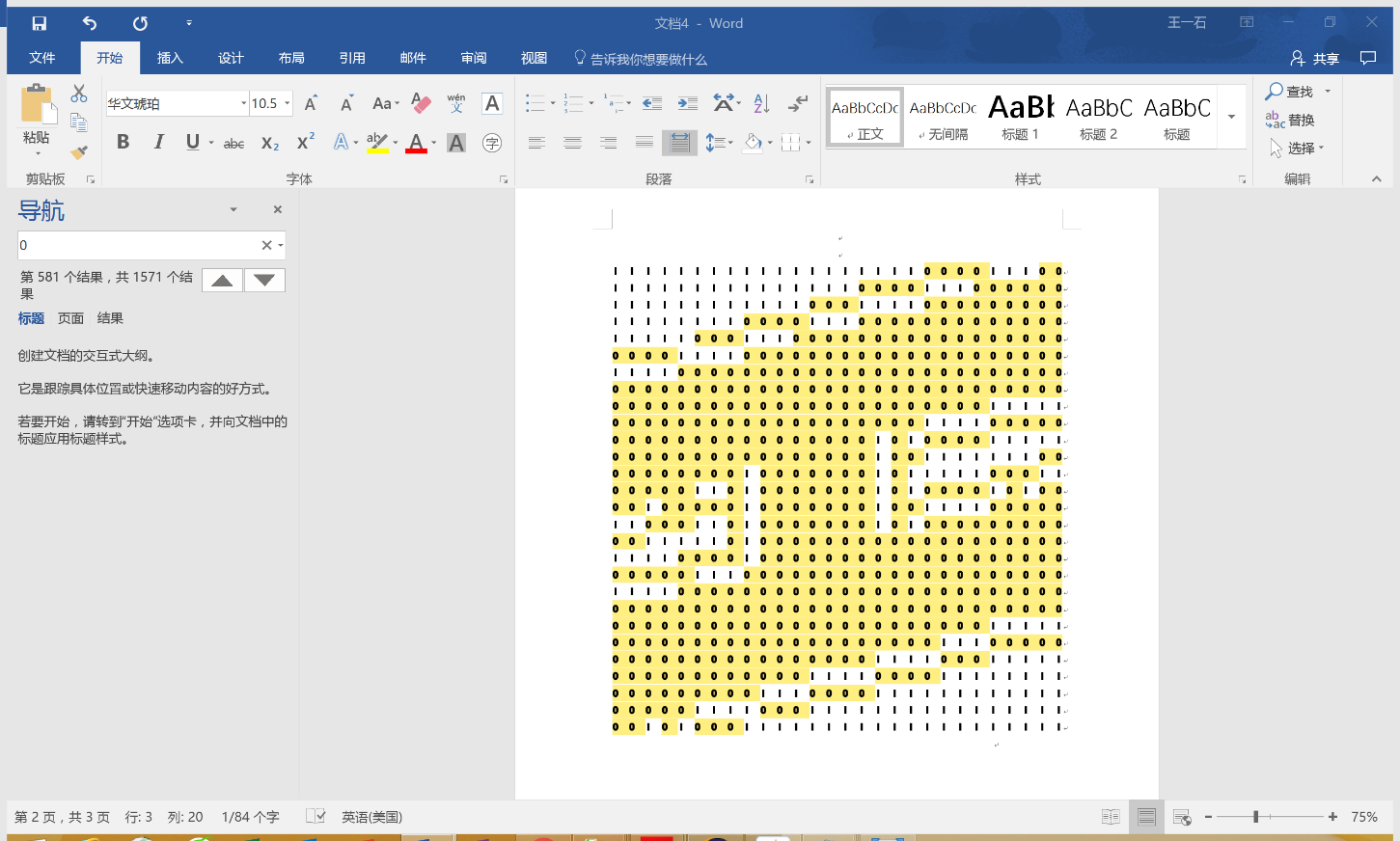
(b)

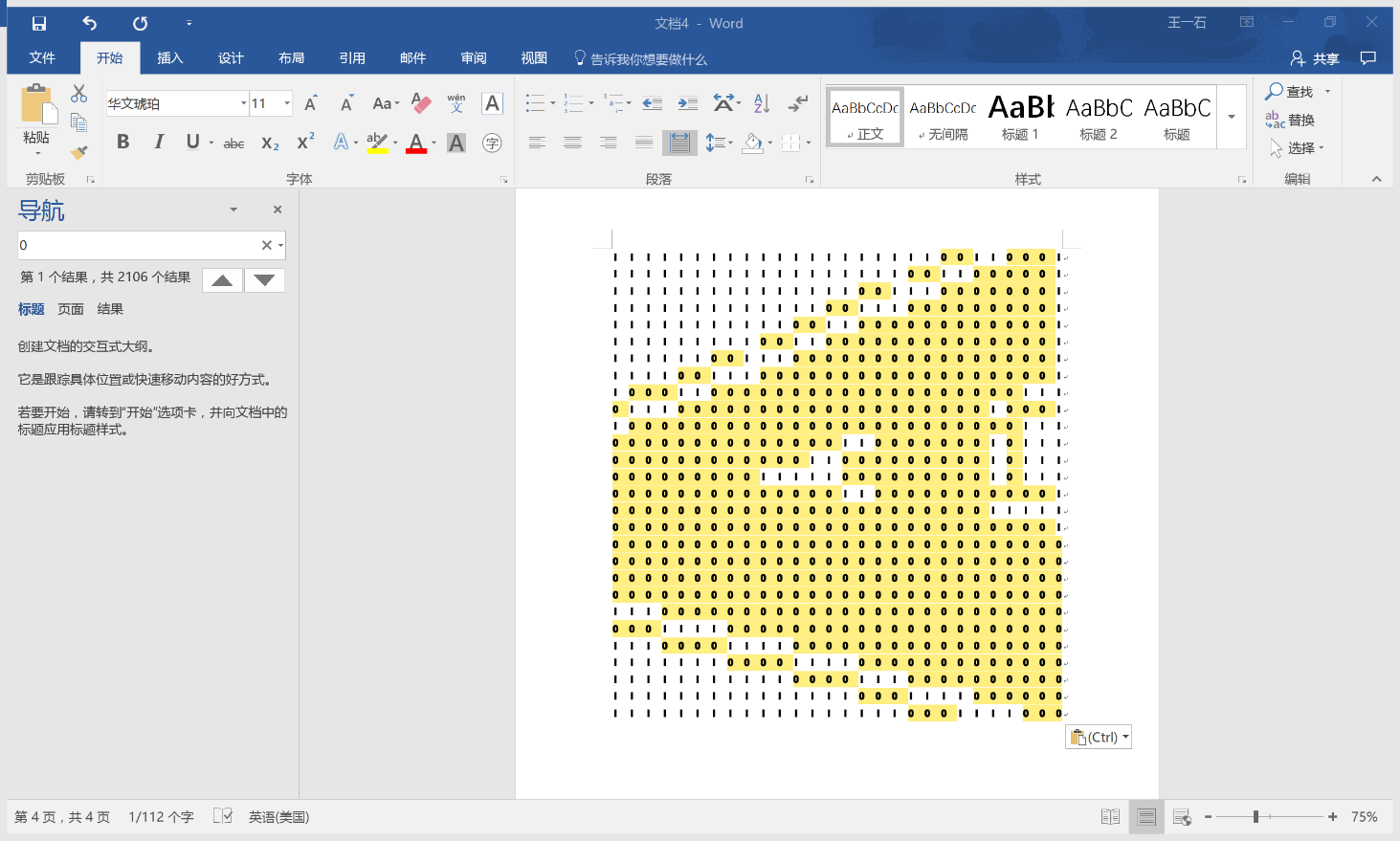


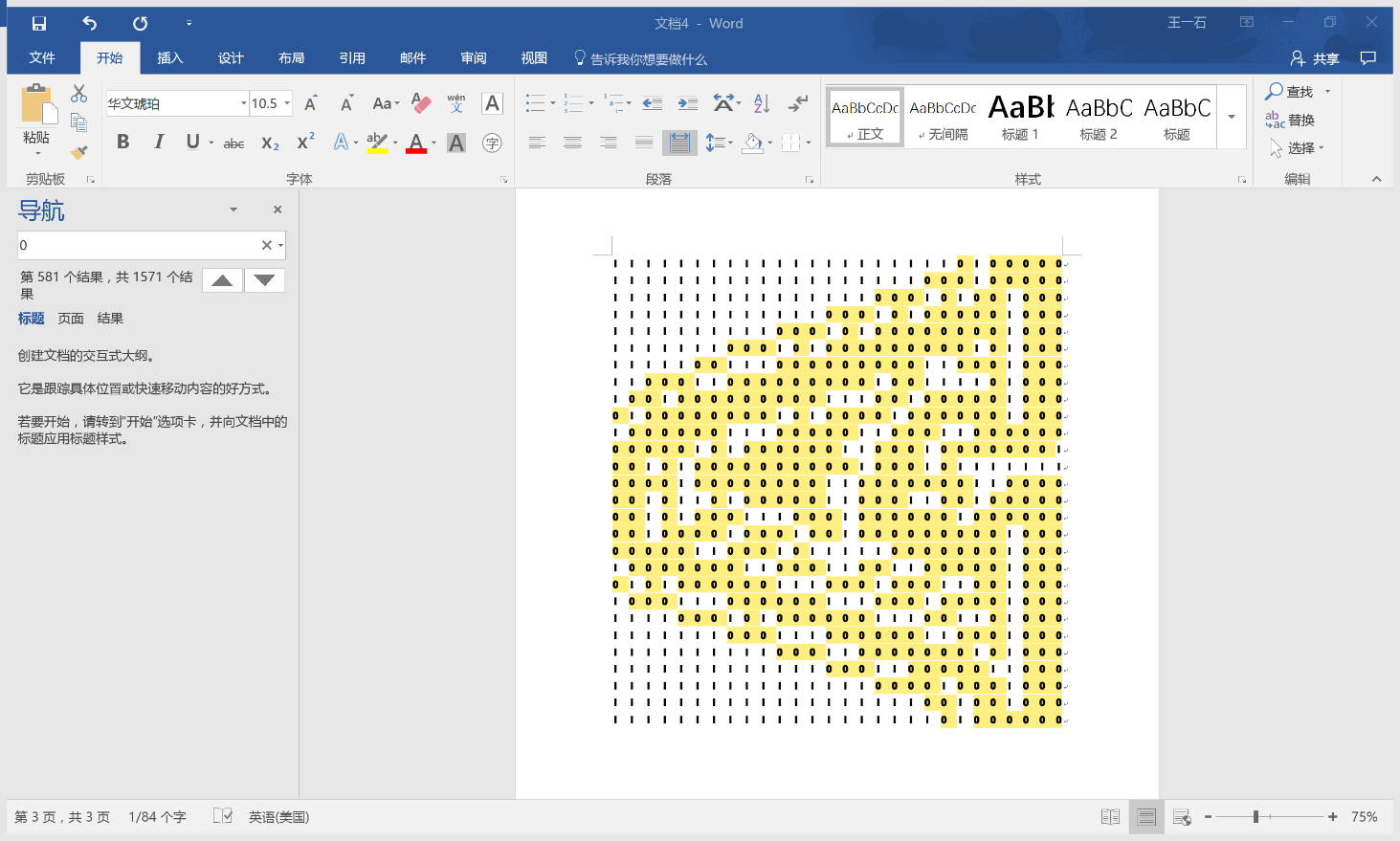
(e)

(d)

In the following images, we will see the sample of how characters “H” or “A” are stored in .txt files after being converted into 28\*28 pixels images. Every single pixel is stored by 0 or 1 in one single line in .txt file and every pixel is represented by 0 or 1 (part of the hand-written character or not). Thus, each image is corresponding to one 28\*28 pixels image and one .txt file with 784 lines (each line has one digit to represent one pixel). In fact, to clearly display how they work, I copied the content from .txt file, made them colorful and put them in .docx before making screenshots.



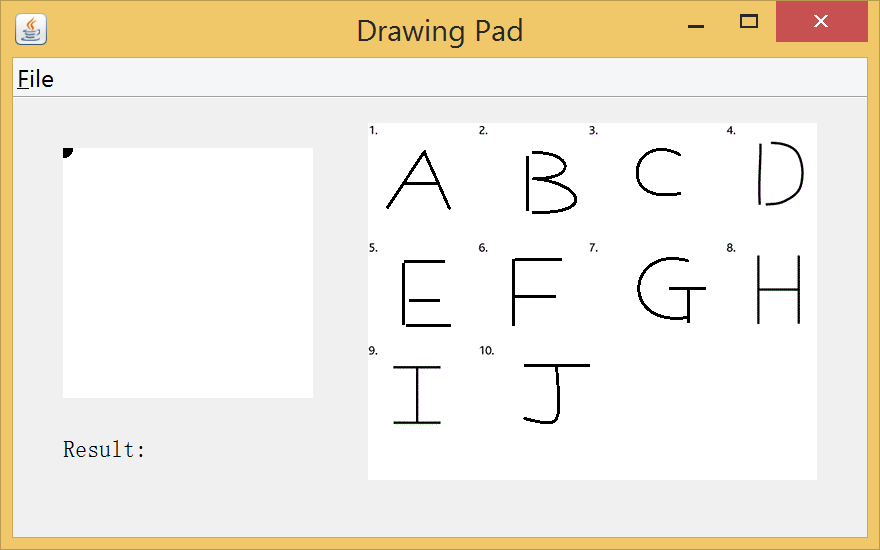




Every transferred data (.txt files in folder “AtoJ” and its subfolders) is stored in the parent folders with the corresponding character`s name as name of its parent folder. We only have training set for character A-J with limited data found. Thus, our program can only recognize written character A-J. However, if we can get enough data (images for other characters or even symbols), we just need to use one of our method to transfer them to .txt files first and put them into corresponding parent folders, then these characters will be recognizable.

1. Interface

Our interface is very simple and clear. After running the Java file, a window will pop up. On the left side of this window, there is a rectangle area for user to input a character by holding down a mouse button and moving.



Input area

Options:

1. New input
2. Start recognition
3. Exit

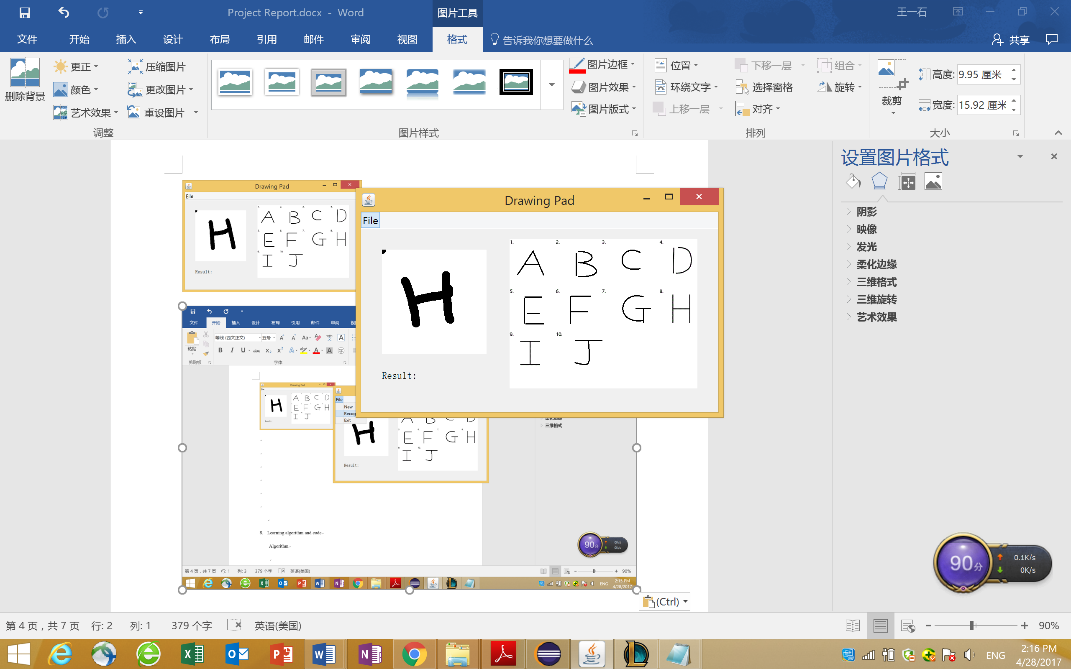
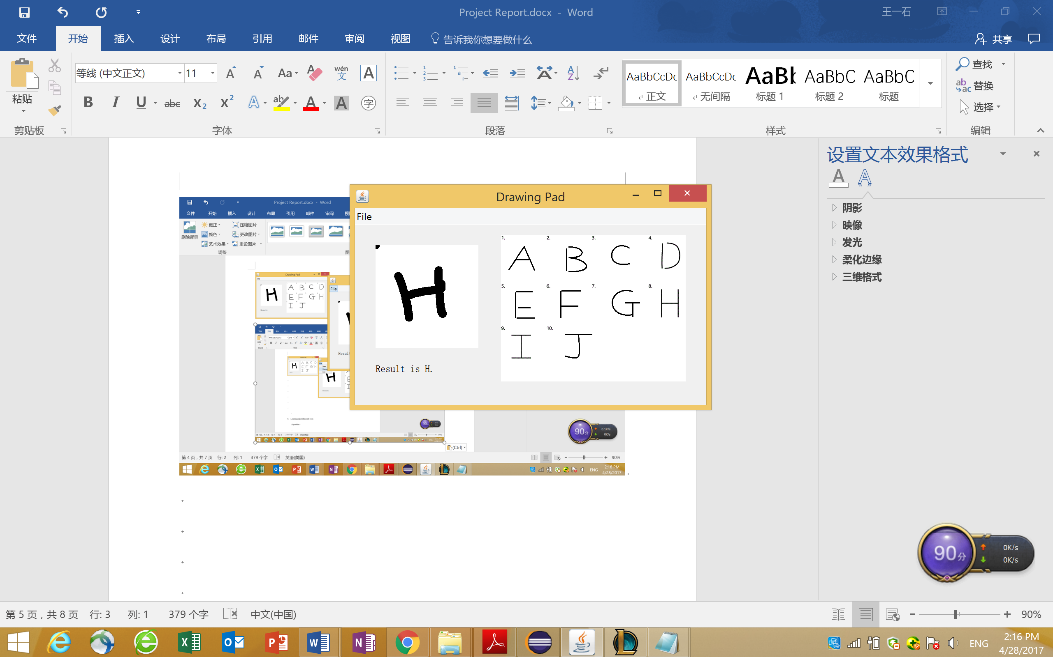
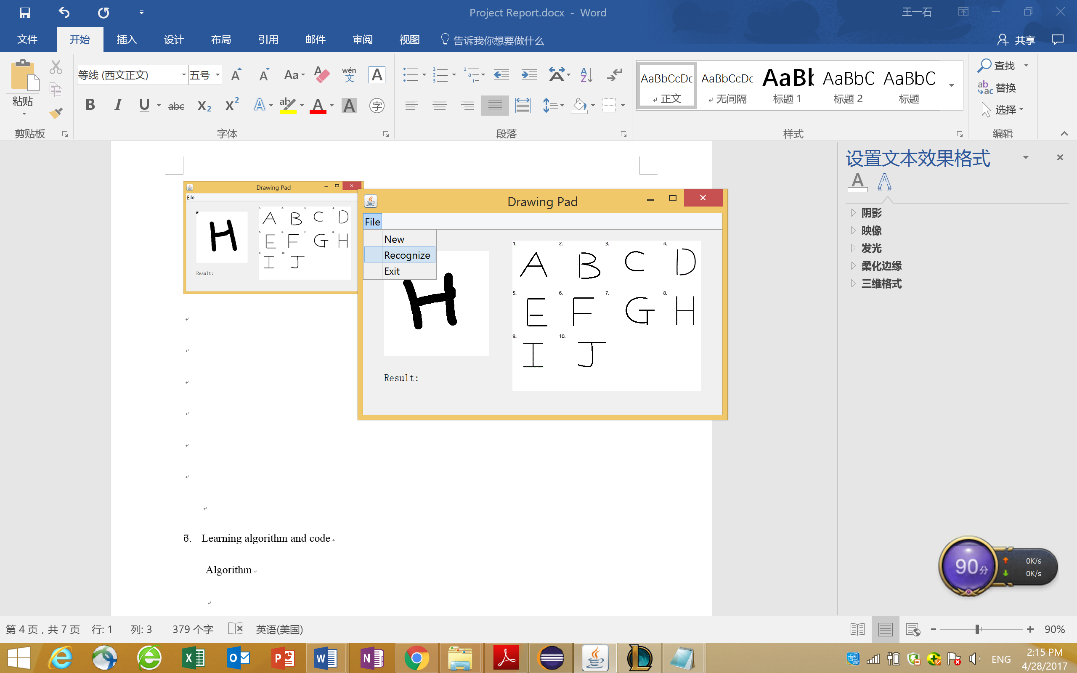
Input samples

Results display

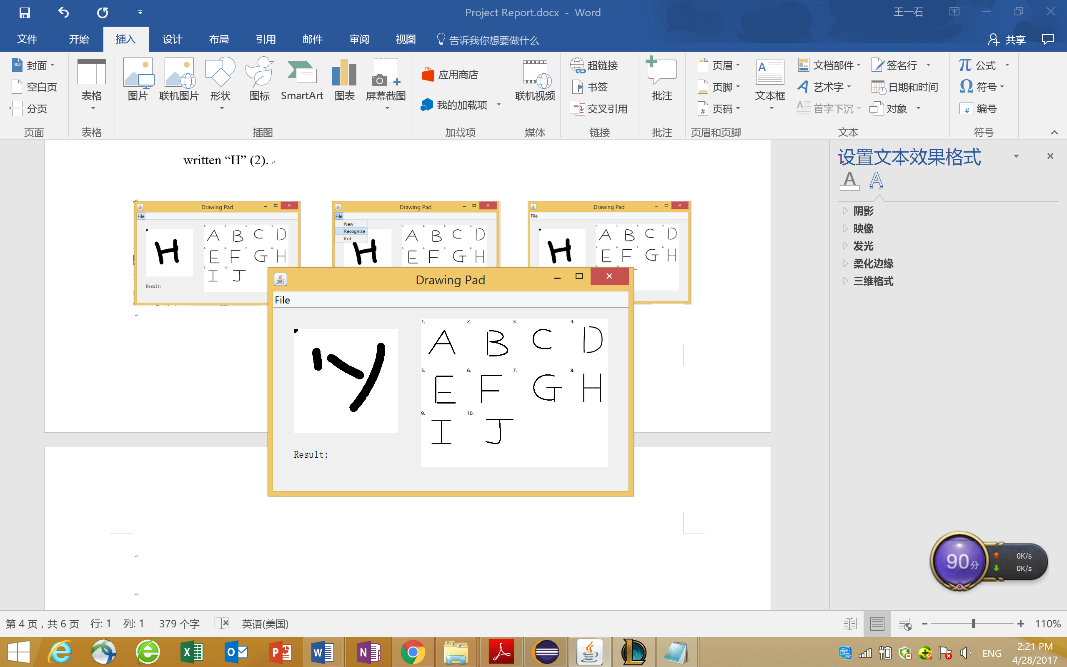
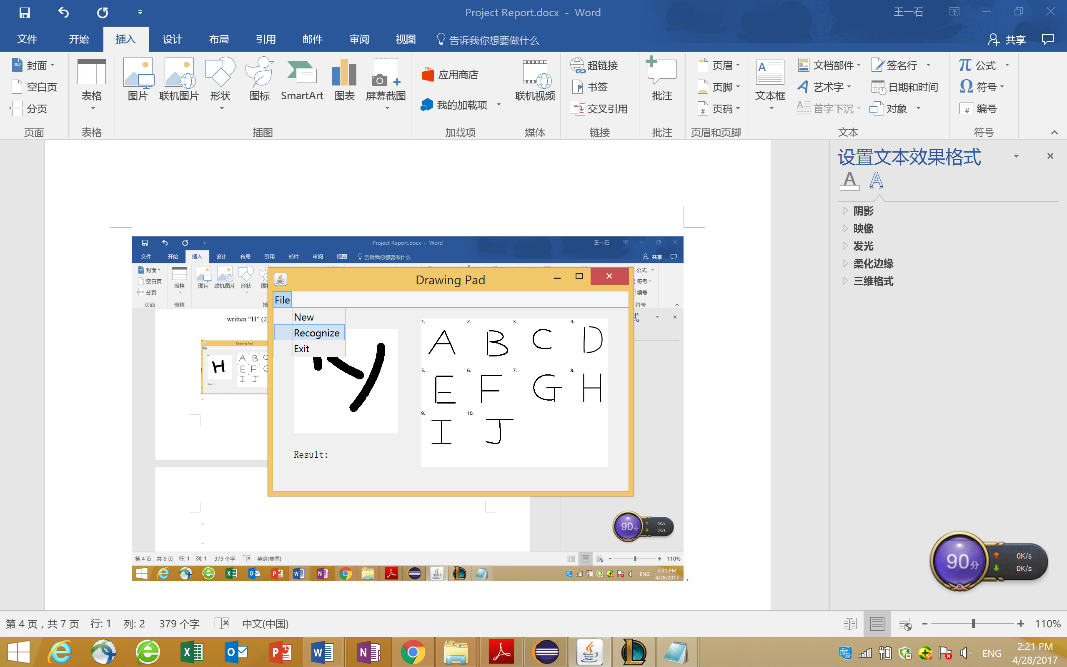
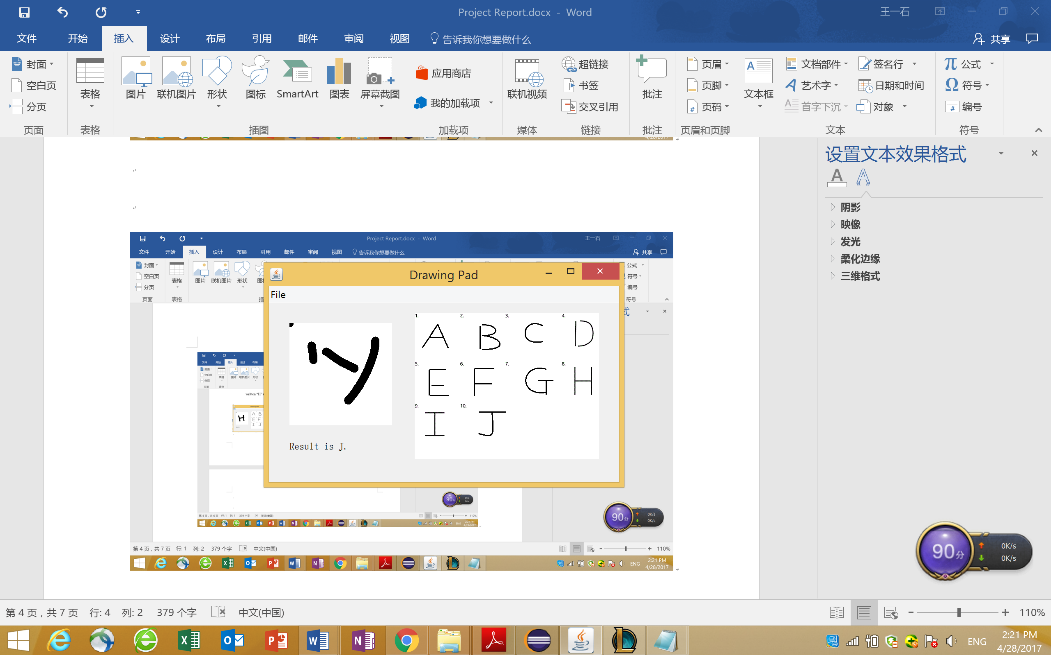
After finishing the input step, the user should click on the button in menu bar and select new to re-input or recognize to recognize the input character. The result will be showed right blow the input area. The right part of this window displays the sample of input and it shows all the valid character that can be recognized. We don`t have the training data for all the English characters. The valid characters for recognition are from A-J.

1. Example

Let us try two examples. I tried to input a clear hand-written “H” (1) and one very vague hand written “H” (2).



(1)



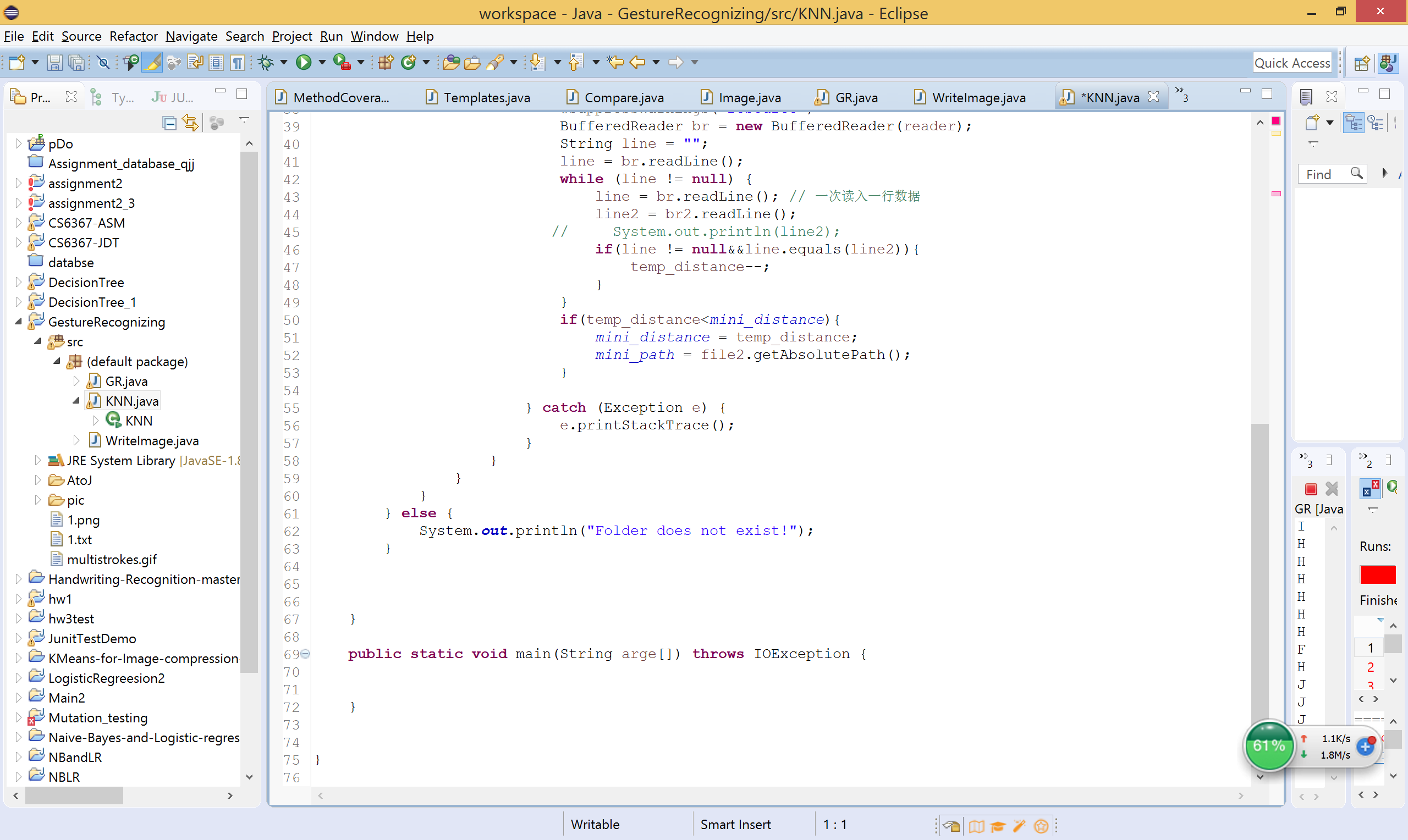
(2)

We can see that a general hand-written “H” can be recognized successfully. However, if a hand-written “H” is too vague, our program will judge it as the most similar words by its algorithm.

The most recent hand-written input will be saved in the “GestureRecognizing” folder as “1.png” and “1.txt” will be the corresponding .txt file for transferred 250\*250 pixels image. (the initial input will be 250\*250. It will be transferred to 28\*28 and finally become .txt file with 784 lines)

1. Code and Learning algorithm

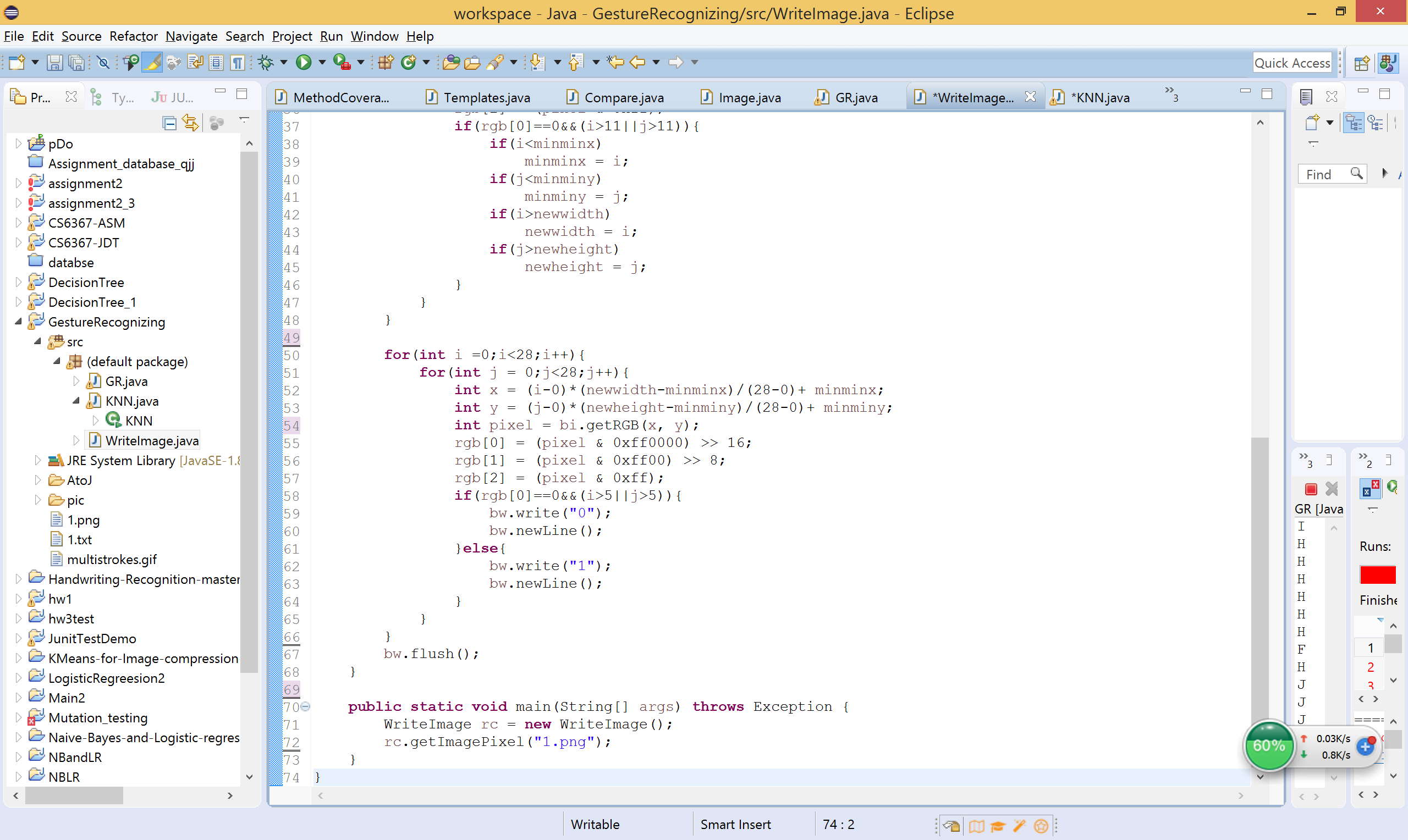
Code for KNN (from KNN.java),



Algorithm,

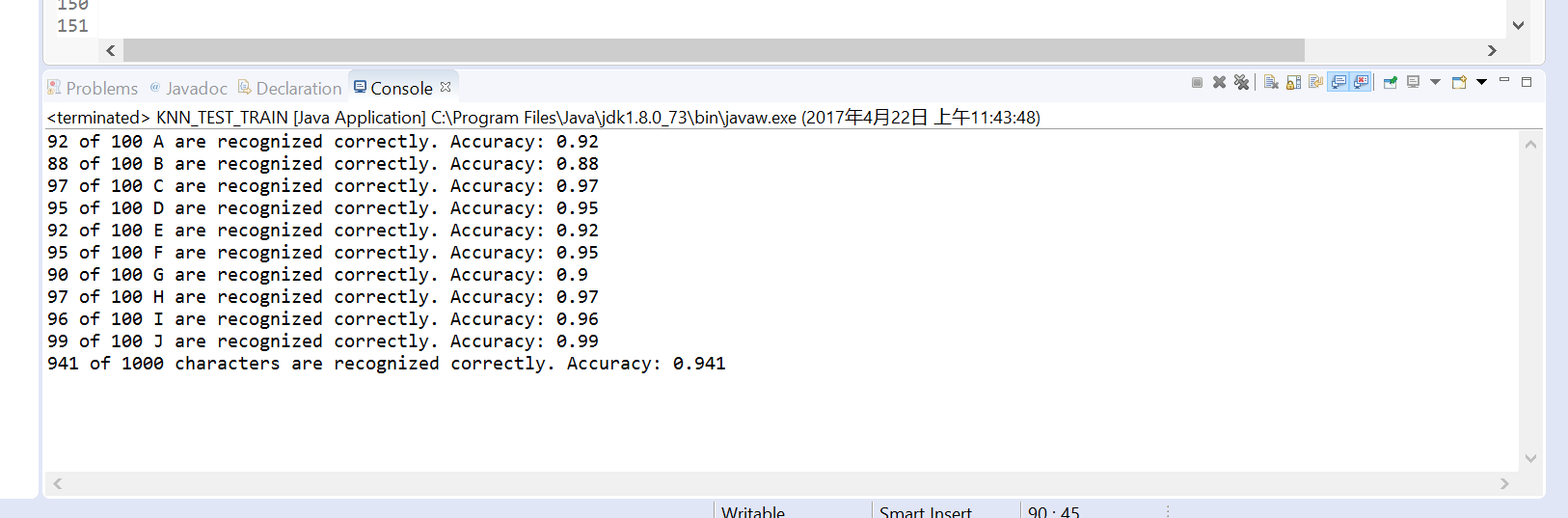
Our program saves the hand-written characters entered by users (test set) and compares them with every single training data (every .txt file we converted from original images) in our dataset. While comparing them, our program calculates the difference between each pixel in test set and corresponding pixel in each training data. In fact, it compares the value saved in the certain lines of testing .txt file and the value in the same line of dataset (all the training .txt files). Every time when these two values are different, the program adds one to the “distance” between these testing file and training file. The progress will go through every line in testing file and each training file to find out which training file has the minimum “distance” to the testing file. If many training files have the same minimum value for “distance”, our program will just use the first one found among them as the file (target) with minimum “distance”. The target file`s path will be recorded and we use its parent folder`s name (just the name of character, mentioned in section 3) as the final output.

Part of code for transferring images to storable .txt (from WriteImage.java)



This part transfer the 28\*28 pixel images into readable .txt file with only 1 and 0 in it. The quantification of the pixel image is helpful for calculating “distance”.

1. Accuracy, Comparison, and Hypothesis

We also made a program to test the accuracy of our program. We firstly generated 100 hand-written images for every character from A-J (totally 1000 images) and then we transferred them into 1000 .txt as test set. After that, we use our KNN algorithm to test all these 1000 test files (.txt files in folder “test+training” and its subfolders). The output is shown below. (generated by KNN\_TEST\_TRAIN.java)

As we can see in the image above, for character from A-J, the lowest accuracy is about 88%, the highest is about 99% and the overall accuracy is 94.1%. Compared with other professionals` work done in other algorithms (tested using the same database as ours), our accuracy is not that high but it is still better than some of other algorithms. [2] We have a hypothesis for their higher accuracies. They used the same dataset as ours but they transferred every image into 20\*20 pixels image instead of 25\*25. Thus, their testing and training dataset may have higher generality with less pixels which might be one of the reasons for their general higher accuracy.

1. Conclusion

Our program shows a good accuracy, we can see how reliable the KNN algorithm is (94.1% overall accuracy). However, according to our accuracy test, we found it is time consuming. It cost more than 3 quarters to finish the test for 1000 files. It is possible to shrink our dataset to make the program faster, however, it will decrease our accuracy. On the contrary, if we enlarge our dataset, our accuracy will get improved but its running time will be slowed. Also, for different characters, our program has different accuracies. Thus, we conclude that the validity, representativeness, and size can be the key for improving this algorithm.

1. Future work

Our priority for future work is to enlarge our dataset to cover all 26 English characters, some punctuations, and some special symbols. The other work will be finding out those misclassified test files, explore their unique characteristics, and add new training files with these characteristics into our training set to improve our accuracy.

**Reference:**

1. NIST Special Database 19: NIST Handprinted Forms and Characters Database. Created August 27, 2010, Updated April 05, 2017.

Web: https://www.nist.gov/srd/nist-special-database-19

2. THE MNIST DATABASE of handwritten digits，Yann LeCun, Courant Institute, NYU；Corinna Cortes, Google Labs, New York; Christopher J.C. Burges, Microsoft Research, Redmond.

Web: http://yann.lecun.com/exdb/mnist/