Implementation of a Character Recognition Software using Neural Networks

Project#2

Akiki Clara, Attieh Joseph

Abstract—Nowadays, Artificial Neural Networks (ANNs) are relatively new computing systems that are being extensively used for solving real world problems. They process information in a way similar to humans by simulating functions and activities from the biological brain. ANNs are currently being used to solve many problems that require a certain kind of intelligence. One of the problems that could be solved using neural networks is Optical Character Recognition.

Index Terms— artificial neural networks, diagonal, feature, histogram, image, input, neuron, optical character recognition, perceptron, zoning,

CONTENTS

1 INTRODUCTION

THIS project objective is to build an Optical Character Recognition (OCR) system building on the neural network already implemented in project1. Our system will be able to extract multiple features from each image of a data set of characters in order to train five aggregated neural networks. After the end of the training phase, these networks will be able to evaluate new characters outside of the data set provided, return confidence scores of different possible characters it could represent and identify the character(s) with the highest weight.

In this project, we worked on improving the graphical interface and used the functionalities of Project1 in order to make the training process easier and faster. After training and validating the neural networks, each one of them will be saved then loaded into Project 2 so that the user could be able to test new characters and the result will be conducted based on multiple features in the image.

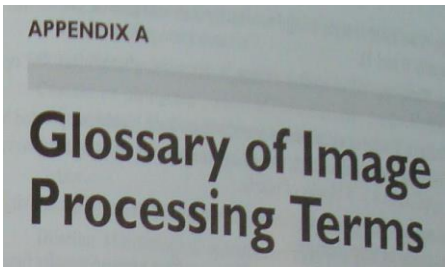
This project will be coded in JAVA and will be visualized graphically with a GUI interface so users can interact with the main functionalities. This friendly interface will be built using Javafx and CSS. It will mainly allow the user to test any letter by either drawing in the application or by importing an image.

2 BACKGROUND

2.1 Definition

Optical Character Recognition knows as OCR is the process of identifying characters from a graphic file to convert them into a regular text file.

For example, an image like in Fig. 1 can be provided to the system supporting OCR, and the text would be outputted.



APPENDIX A

Glossary of Image Processing Terms

Fig. 1. Example of OCR

2.2 History

The first version of OCR was implemented in 1940. Ten years later, the first OCR machines start appearing. Then, multiple generations started to show:

- First generation OCR (NOF, Farrington 360, IBM 1418) recognizing only special fonts (1960 - 1965).
- Second generation OCR (IBM 1287, NEC, Toshiba) recognizing constrained hand-printed alpha-numerals (1965 – 1975).
- Third generation OCR (IBM 1975) recognizing 275 fonts including handwritten ones

2.3 Types of OCR

OCR systems have mainly three types:

- Fixed-font where the system is able to recognize one font
- Multi-font where the system is able to recognize multiple fonts
- Omni-font where the system is able to recognize most fonts without the need of maintaining databases of those specific fonts

2.4 Uses of OCR

OCR systems are mainly applicable when a certain system needs to read important handwritten & machine-printed characters such as passport number, bank checks, postal zip codes, … It is also a first step in the transition from traditional file-based systems to more advanced systems such as database-based systems. Furthermore, it plays a huge rule in the advancements towards a world where machines are smart enough to be able to read handwritten characters and understand them (natural language processing and Internet of Things).

3 APPROACH USED

In order for our system to recognize characters, we had to

- Pre-process the image containing the letter
- Retrieve different features
- Build and train multiple ANN agents (developed in project 1) taking as input the different features extracted
- Aggregate the output of the multiple agents built

Fig. 2. Approach used in this project

3.1 Pre-processing

First, the graphic file goes through a series of preprocessing methods to produce data that is easier and more efficient for the OCR to recognize. The main steps implemented in our project are:

1-Binarization (also known as thresholding): It is done by converting a gray-scale image into a binary image where a 0 corresponds to a white pixel and a 1 corresponds to a black pixel.

2- Minimum bounded rectangle: It is done by identifying the useful part of our image. It is done by finding the

exact area covered by the character.

3- Skeletonizing: This is done by first applying thinning algorithms to the character in order to reduce the data size and normalize characters so that they are processed regardless of their width.

4- Resizing: we resized the character depending on the feature that we needed to extract.

The images below show this process:

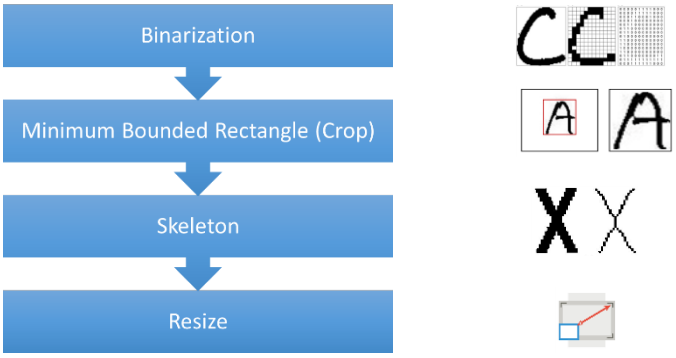


Fig. 3. Preprocessing of the image

3.2 Feature extraction

3.2.1 Diagonal Feature

First, the graphic file is resized to be of a size 90 x 60 pixel. Then, it is divided into 54 equal zones, each of size 10x10 pixels. We implemented a variation of the algorithm. The original one traversed each zone diagonally, computed the sum of pixels and got the average of each zone. In order for this feature to be more decisive and more meaningful, the features are extracted from the pixels of each zone by getting the average of each diagonal, then computing the average for all the zone.

This procedure is repeated for all the zones leading to extraction of 54 features for each character. Then, the average of each feature column and row is computed.

Those extracted features are used to train the first two neural network that will be used in our design.

Input: preprocessed image of size 90*60.
Output: Feature vector of size 69
1- Resize the image to 90 x 60 pixels
2- Divide into 54 equal zones each of size 10 x 10 pixels
3- For each region
Get the average value of pixels among each diagonal out of the 19 diagonals
Average them out to get one single value
Attribute value to region
4- For each row
Get average values row wise
5- For each column
Get average values column wise

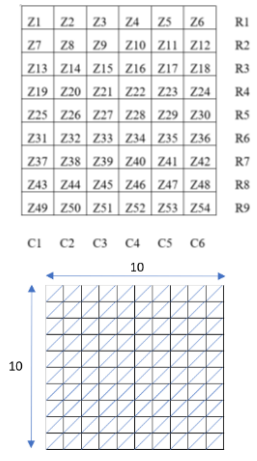


Fig. 4. Diagonal Feature extraction

3.2.2 Zoning Feature

The zoning feature takes as input the pre-processed binary image and produces as output a feature vector of size 98. This feature resizes first the image to 32x32 pixels and it divides it into 64 equal zones (each zone has a size of 4x4 pixels). Then for each zone, it calculates the density of the on pixels (black pixels) over the total pixels in that region and it inserts the 64 average values in the output vector. It computes also the total number of on pixels in the up region (zone1-zone32) as well as the total number of on pixels in the down region (zone33-zone64) and it subtracts them from

each other, the result (difference1) is divided by the total number of pixels in the image and it is saved in the output vector. It does the same for the left region (zones in the first 4 columns) and the right region (zones in the last 4 columns) and the result (difference2) is also saved in the output vector. Finally, the zoning feature takes the density of each of the two consecutive zones and add them together, the 32 values will be added to the output vector.

Input: pre-processed image of size 32x32.
Output: Feature vector of size 98
1-Divide into 64 equal zones each of size 4x4 pixels
2-Compute density for each zone:
Get the average value of on pixels(black) over all the pixels in that zone
3-Compute Up-Down, Right-Left density:
Calculate the total number of on pixels in the up, down, left and right regions
Diff1=(up-down)/total number of pixels
Diff2=(left-right)/total number of pixels
4-Compute the consecutive density:
Calculate the density of two consecutive zones

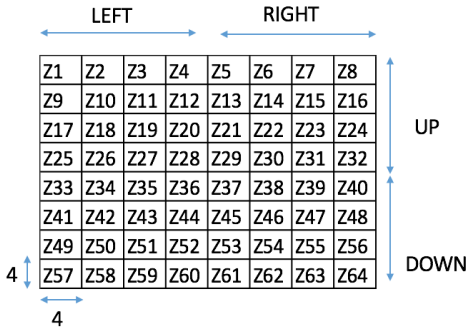


Fig. 5. Zoning Feature extraction

3.2.3 Projection Histogram

Projection histograms calculates foreground pixels in different directions. The basic idea behind them is that character images (2D signals) are represented as a 1D signal. This feature is scale-invariant but depend on rotation and skew.

In this project, we have used three type of projection histogram in one feature:

- In horizontal histogram, the foreground pixels were counted row wise.
- In vertical histogram, the foreground pixels were counted column wise.
- In diagonal left histogram are counted left diagonal wise.

The 32 vertical, 32 horizontal and 63 diagonal histogram features are used to form a feature vector of size 127.

Input: pre-processed image of size 32x32.
Output: Feature vector of size 127
1- Horizontal projection
    For each row:
        if pixel is black then countR++
2- Vertical projection
    For each column:
        if pixel is black then countC++
3- Diagonal projection
    For each diagonal:
        if pixel is black then countD++

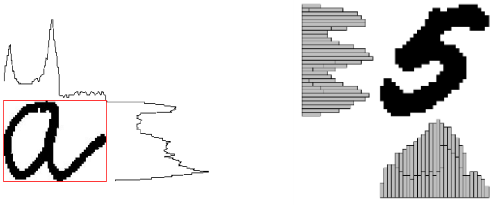


Fig. 6. Projection Histogram extraction

3.3 Building the structure of the neural networks

Our system uses five neural networks. Details of those networks are found in the table below:

	Feature Handled	Training set (approx.)	Number of inputs	Nbr of Cells in hidden layer	Nbr of cells in outlayer
ANN1	Diagonal Feature	600 pictures	70	200	26
ANN2	Diagonal Feature	1000 pictures	70	100	26
ANN3	Zoning Feature	600 pictures	98	100	26
ANN4	Zoning Feature	1000 pictures	98	200	26
ANN5	Projection Histogram	600 pictures	127	200	26

Fig. 7. Neural Network Architectures

As we can see, the number of inputs of each neural network depend on the feature used. On the other hand, we chose the output layer to have 26 cells since we have 26 letters in the alphabet. If the first cell gave as an output a 1 and the others gave a 0, that means that the letter is an "A". This number of neurons is more robust against errors and give more accurate results compared to using 5 neurons to handle the 26 characters. We will elaborate this concept in the next section.

The other design choices were chosen after rigorous testing. In fact, we proceeded as follow: for each feature, we started with a neural network having a hidden layer of 50 neurons. We tried to train each network with our dataset. The training was made using validation algorithms implemented in project1. We mainly used K-Fold validation having k=1 in order to train the data set and test what the network has learned from this dataset. We keep on training until the ratios of the validations converges to 1. If after a certain time the ratio starts dropping, that means that the network cannot hold such knowledge and is forgetting some data. In this case, we increase the number of neurons in the hidden layer and we start again. A part of our training set is shown below:

Fig. 8. Training set of the networks

After choosing the architectures that maximizes the knowledge of our network, we proceed by training each neural network. We build the network using project 1 then select the feature to be fed to the input neurons. The next step is to import the images in our data set and train. After successfully training the network and reaching a value of 1.0 for the validations, we save the neural network. This saved network will be loaded later ons in project 2.

Furthermore, we decided to have multiple neural networks working on the same feature to get more precision. After a lot of testing, we remarked that networks with different architectures trained on the same features tend to identify different set of characters successfully. To take advantage of that, we increased our training set and changed the architecture of the network to get a better result in the end.

The whole process is showed in this picture:

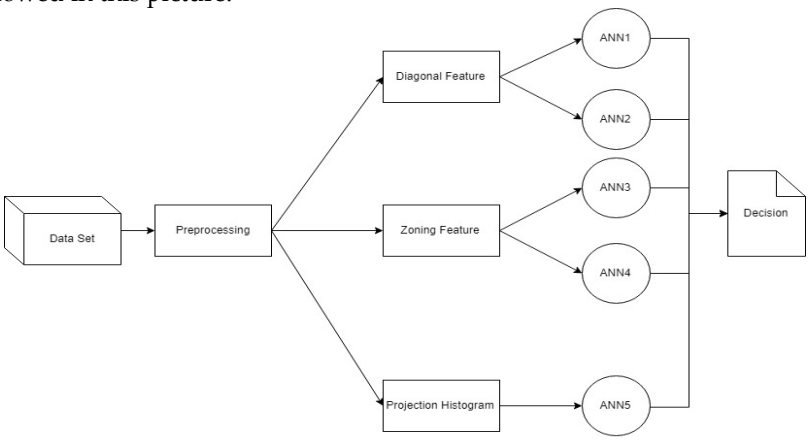


Fig. 9. Architecture of our OCR system

3.4 Aggregating the outputs of the neural networks

After building our networks and training them, we are now ready to use our OCR system. In order to identify a certain character, it is first preprocessed. Then, the diagonal feature, zoning feature and projection histogram are calculated. Each feature is then provided as an input to its corresponding neural network(s).

Since we have 26 neurons in the output layer and the unit step as activation function, many outputs might result from each neural network. We identify three major cases:

- Case 1: only one cell in the output layer outputs a one. This means that the neural network identified a certain character. For example, if cell 1 in the output layer fires a 1, that means that the character is recognized as an A with a 100% confidence score. If the second cell fires a one, the character is recognized as a B with a 100% confidence score ….
- Case 2: no cell in the output layers outputs a zero. In this case, the neural network was unable to recognize the inputted character.
- Case 3: more than one cell fires a one. That means that our neural network got confused. It could not take a decision regarding the letter recognized. Thus, we monitor each cell that fired a one and we give a confidence score relative to each character. For example, if cell 1, cell 4 and cell 5 fired a 1, we can say that the neural network identified the letter A with a 33.33% confidence, the letter D with a 33.33% confidence and the letter E with a 33.33% confidence.

Each network calculated its own confidence values. Then, we aggregate the output of each network in a way that all of the networks have equal weights. We disregard networks that could not identify any character from the aggregation (Case 2). For example, if ANN1 resulted in having the letter A with a 100% confidence score while ANN2 could not recognize the letter, the aggregated result would be that the character is a letter A. Furthermore, if ANN1 recognized the character as the letter B with a 100% confidence score while ANN2 recognized it as a P with a 100% confidence score, the aggregated result would be that the character B has a 50% confidence score while character P has a 50% confidence score.

4 GRAPHICAL USER INTERFACE

After building our ANN agent that recognizes handwritten letter and testing it multiple times, we created an interface for the user to try testing different letters by either drawing them in the application or by importing an image. The five different ANN networks were saved in the application's package and whenever the user wants to test a letter (drawing or importing), it will call different image pre-processing methods and extraction methods from another package (com.proj package).

First, the user runs the program and see a welcome screen.

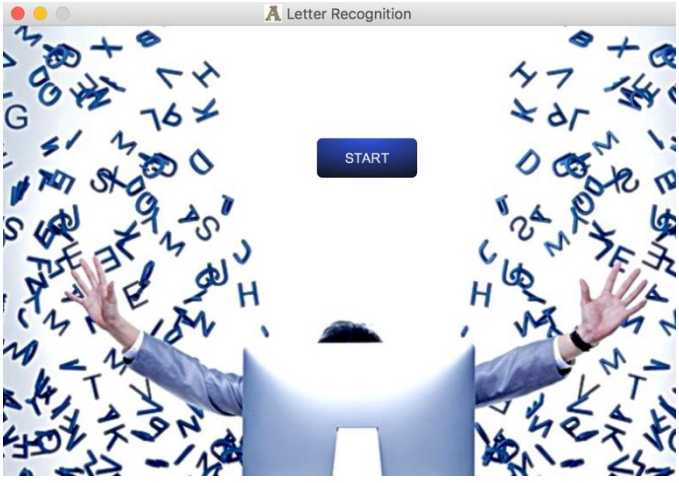


Fig. 10. Welcome Screen

Then the user has the choice to either draw a letter or import an image of a letter.

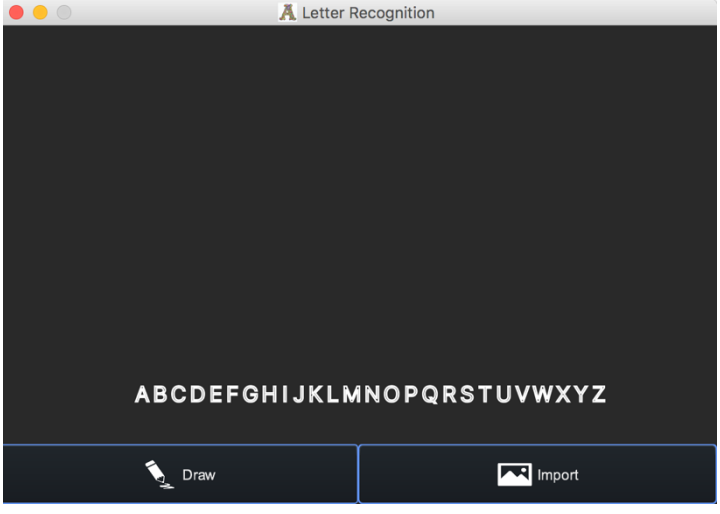


Fig. 11. Interface for choosing input method

The user can import an image from its desktop by pressing on import. If the user decides to draw, another window pops-up, allowing him to write on the screen. He/she is provided with the option to erase or clear the whole image in case he/she did any mistake. By pressing on the save button, the image drawn or imported will be preprocessed and then tested in the network.

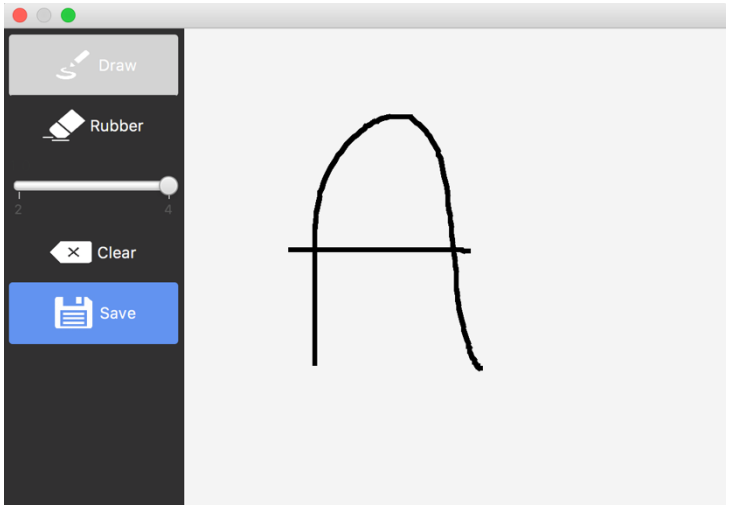


Fig. 12. Screen allowing the user to draw images

After saving, the first window will be updated with the new image, with confidence scores of each network with the aggregated confident score. Also, the character with the highest score is highlighted in blue.

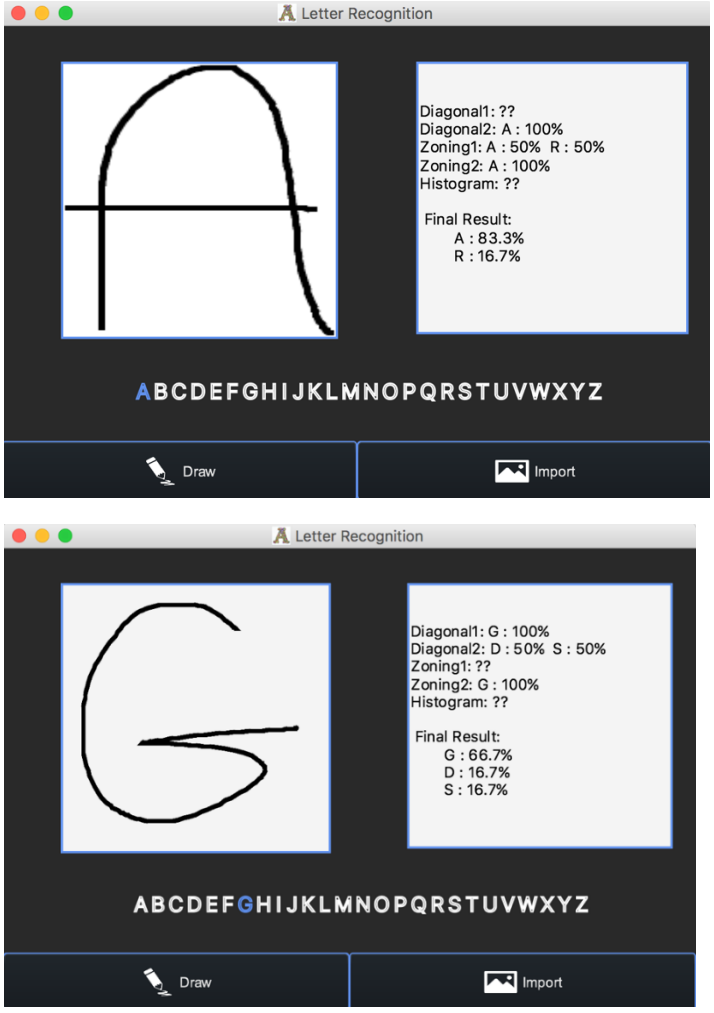


Fig. 13. Confidence scores of the drawn character

5 EXPERIMENTAL TESTING

In this section, we provide sample inputs entered by a user with the sample outputs:

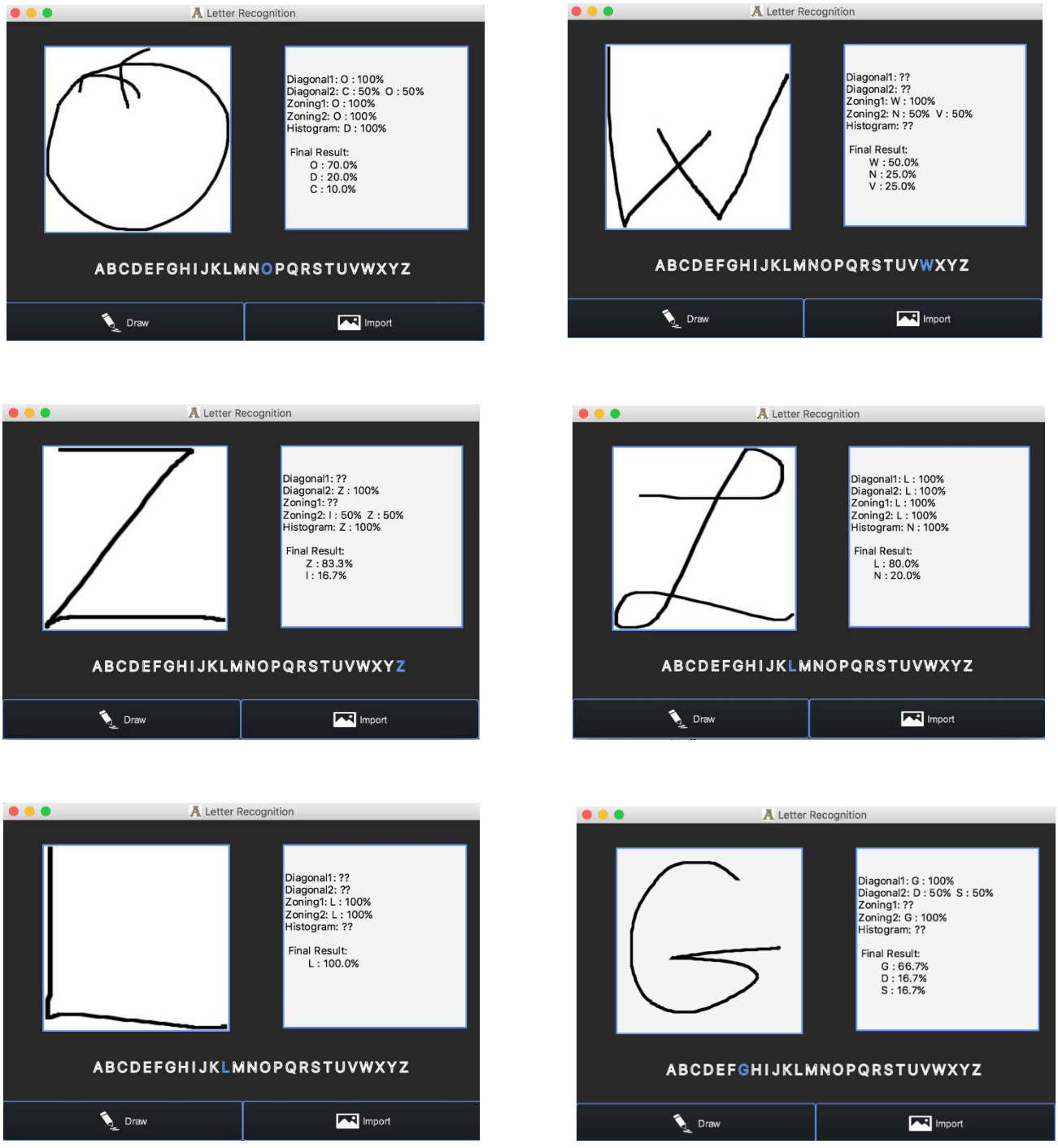


Fig. 13. Characters tested with our application

6 HIGHLIGHTED FEATURES

6.1 Validations algorithms

As discussed in project 1, three algorithms were implemented: the separation validation, k-fold cross validation and the Monte Carlo cross-validation. Those algorithms were implemented in a way that gives the user full control of the neural network. In the separate validation, the user could select the percentage of pairs that he/she need to use for training. In the K-fold cross-validation, the user has the choice to choose the number of subsets that need to hold the data pairs. For the Monte Carlo cross-validation, the user is provided with the choice to set the number of iterations in which the division into random sets happen.
With this amount of flexibility, the user can successfully train his/her network and identify how much knowledge the network is capable of maintaining.

6.2 Individual Decisions

In our implementation, we made sure that the output of each neural network is a set of letters having each a confidence score. This was done by looking at the cells in the output layer that fires a 1. In case all the outputs of the cells are 0, that means that the network did not recognize the character.

6.3 Five neural networks

In addition to the individual decisions taken by the neural network, we provided aggregate confidence scores for the character inputted by the user. In fact, our implementation uses five neural networks. We proceeded by giving each network with equal weights. Each network that does not recognize the character is given a weight of zero. By doing so, we were able to compute aggregate confidence scores.

6.4 Precision

By using 5 neural networks with different architectures, different features and different data sets we are able to provide accurate results (for more details, kindly refer to previous sections).

7 CONCLUSION

In conclusion, an artificial neural network provides a distinctive way to solve the Optical Character Recognition problem. But it requires intuition in order to build the whole network. Decisions should be made regarding the number of layers, number of cells per layer, number of training set, number of neural networks, … With the system that we have provided, we were successfully able to provide the user with confidence score regarding each character provided to the network. This implementation could be expanded to cover more characters such as digits. Also, it can be expanded to recognize multiple handwritings from multiple users and even identify the user writing this character.

References

[1] Ananthram, A. (2018, February 25). Random initialization for neural networks: a thing of the past. Retrieved from https://towardsdatascience.com/random-initialization-for-neural-networks-a-thing-of-the-past-bfcdd806bf9e

[2] Mazzi A., Kaplan F. (2014). Digital Humanities 101. Retrieved from: https://s3.amazonaws.com/ppt-download/dh101-course7-131107073302-phpapp02.pdf?response-contentdisposition=attachment&Signature=EO93pLsEYQVvIyE0BIOohZKdh1I%3D&Expires=1543975897&AWSAccessKeyId=AKIAIA5TS2BVP74IAVEQ

[3] Due Trier, Ø., Jain, A. K., & Taxt, T. (1996). Feature extraction methods for character recognition-A survey. Pattern Recognition, 29(4), 641-662. doi:10.1016/0031-3203(95)00118-2

[4] Feature extraction methods (n.d.). Retrieved from: http://shodhganga.inflibnet.ac.in/bitstream/10603/125229/12/12_chapter%205.pdf

[5] Kumar, G., & Bhatia, P. K. (2014). A Detailed Review of Feature Extraction in Image Processing Systems. 2014 Fourth International Conference on Advanced Computing & Communication Technologies. doi:10.1109/acct.2014.74

[6] Pradeep, J., Srinivasan, E., & Himavathi, S. (2010). Diagonal Feature Extraction Based Handwritten Character System Using Neural Network. International Journal of Computer Applications, 8(9), 17-22. doi:10.5120/1236-1693

[7] Vijayalakshmi, K., Aparna, S., Gopal, G., & Hans, W. J. (2017). Handwritten character recognition using diagonal-based feature extraction. 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). doi:10.1109/wispnet.2017.8299949