CS512: Hand-written Digit Recognition System

Kuo-wei Chung
Rutgers University
Piscataway, NJ, USA
Email: kc1026@scarletmail.rutgers.edu

Shubham Sinha
Rutgers University
Piscataway, NJ, USA
Email: ss3076@scarletmail.rutgers.edu

Rong Zhang
Rutgers University
Piscataway, NJ, USA
Email: rz214@scarletmail.rutgers.edu

***Abstract*— This project proposed a handwritten digit recognition system which can use a image as an input to get the numerical value of the image. In this system, we use kNN algorithm to train the images from MNIST, then classify the input image using the trained classifier.**

I. Project Description

This project proposed a handwritten digit recognition system using KNN algorithm which is not covered in class. We can use an image as input and figure out what the numerical value is using the KNN classifier. Stumbling blocks: The user input image may be noisy and if we cannot denoise the image thoroughly, this algorithm will not work well.

The project has four stages: Gathering, Design, Infrastructure Implementation, and User Interface.

A. Stage1 - The Requirement Gathering Stage.

- The general system description:
  We plan to design a Handwritten Digit Recognition system. For determining what digit the user has written, we will first allow the user to enter a digit as an image of the handwritten image. This image will be fed to a K Nearest Neighbour clustering algorithm which will compare it to the images of handwritten digits from MNIST dataset. We will be using K Nearest Neighbour algorithm which has not been covered in class and is one the most widely known machine learning algorithms. Using this system, we can allow post offices to have an automated distribution system by recognizing the handwritten zip code on the envelopes or the packages. Base on this system, the post office can improve their work efficiency and reduce their working loads.
- The three types of users (grouped by their data access/update rights): There are three types of users: senders, postman, and post office staffs.
- The sender's interaction modes: The senders will write a zip code on the envelope or package. Then he will scan the zip code on the package to the machine in the post office, then the machine will recognize the zip code to distribute this package into different boxes. The staffs and administrators will collect these packages to the next stop (another post office).
- The real world scenarios:
  - Scenario1 description: Sender A write a zip code on the package and scan it , the machine accepts the package and distributes it to a corresponding box.
  - System Data Input for Scenario1: Image of the handwritten zipcode
  - Input Data Types for Scenario1: RGB matrix
  - System Data Output for Scenario1: The recognition result of the zipcode and the corresponding box index, accepted.
  - Output Data Types for Scenario1: Integer and boolean
  - Scenario2 description: Sender B write a zip code on the package and scan it, but the machine refuses the package and tells the sender that zipcode is wrong as well as no matching box.
  - System Data Input for Scenario1: Image of the handwritten zipcode
  - Input Data Types for Scenario1: RGB matrix
  - System Data Output for Scenario1: The recognition result of the zipcode, not accepted.
  - Output Data Types for Scenario1: Integer and boolean
- The Postman's interaction modes: The postman scans an envelope for the handwritten phone number on the envelope, to either place a call or send a text to the receiver.
- The real world scenarios: Please insert the real world scenarios in here, as follows.
  - Scenario1 description: The postman scans the handwritten phone number on envelope 1, then the machine send a text automatically to the deliver.
  - System Data Input for Scenario1: Image of the handwritten phone number
  - Input Data Types for Scenario1: RGB matrix
  - System Data Output for Scenario1: The phone number
  - Output Data Types for Scenario1: Integer
  - Scenario2 description: The postman scans the handwritten phone number on envelope 2, then make a phone call to the deliver.
  - System Data Input for Scenario2: Image of the handwritten phone number
  - Input Data Types for Scenario2: RGB matrix
  - System Data Output for Scenario2: The phone number
  - Output Data Types for Scenario2: Integer
- The post office staffs interaction modes: The post office staff will carry the mails and packages which have already been classified to different logistics centers. Then, the staff in each post office will use the machine in the post office to distribute these packages again to next stop, or

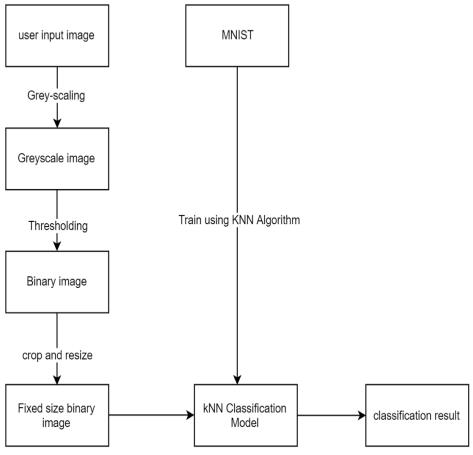
take them out for delivery.

- The real world scenarios: Please insert the real world scenarios in here, as follows.
  - Scenario1 description: Staff A scan a package from previous stop, the machine accepts the package and distributes it to a new corresponding box.
  - System Data Input for Scenario1: Image of the hand-written zipcode
  - Input Data Types for Scenario1: RGB matrix
  - System Data Output for Scenario1: The recognition result of the zipcode and the corresponding box index, accepted.
  - Output Data Types for Scenario1: Integer and boolean
  - Scenario2 description: Staff B scan a zip code on the package, but the machine refuses the package and tells the staff that this package had reached the destination.
  - System Data Input for Scenario2: Image of the hand-written zipcode
  - Input Data Types for Scenario2: RGB matrix
  - System Data Output for Scenario2: The recognition result of the zipcode, not accepted.
  - Output Data Types for Scenario2: Integer and boolean
- Project Time line and Divison of Labor. Timeline:
  - 03/20 - 04/02 : Data collection, test image pre-processing
  - 04/03 - 04/09: KNN
  - 04/10 - 04/23: Visualization,Integrated system Division of Labor
  - Rong Zhang: KNN Algorithm, Visualization, Documentation, Data Collection, Analysis
  - Kuo-Wei Chung: KNN Algorithm, User Interface, Documentation, Visualization, Analysis
  - Shubham Sinha: KNN Algorithm, Documentation, Data Processing, Analysis, Visualization

B. Stage2 - The Design Stage.

- Description:
  Our project will be consists of several parts: Training data (MNIST dataset), kNN algorithm, test data (user input image), input image-preprocessing, classification result.
  - Training data: The training data in our system will be the MNIST dataset. This dataset consists of many images of hand-written digits and the labels of these digits.
  - kNN algorithm: kNN algorithm is the key algorithm that we will use this algorithm to train the classifier using MNIST dataset. By training the classifier using this algorithm, we can use this model to classify the test data to figure what the numerical value is.
  - Test data(user input image): test data will be the user input image, which can be a photo of the handwritten digit.
  - Input image-preprocessing: For the test data, we need to pre-process the image to make it easier for the classifier to make decision. The preprocessing will include grey scaling, thresholding, noise reduction, crop, compression and resizing of the image. After image-preprocessing, the input photo will become a binary and square photo to be classified. Apply the processed image to the classifier and we will get the classification result.
  - Space complexity: MNIST has a training set of 60,000 examples, and a test set of 10,000 examples. Each of these examples is a 2D matrix of size 28*28 and as it is binary image it will take 1 bit for each pixel. We will also have the user input image which can be of any size, lets say m*n*3 as it will be a RGB image and each pixel will be a integer so each pixel is 8 bits(as values range from 0 to 255). So total space complexity is O((70000*28*28)+(m*n*3*8)) bits.
  - Time complexity: For preprocessing step the time complexity is O(mn) for grayscaling and binarising, O(m+n) for bounding box. So total time complexity for preprocessing is O(mn) where m is the pixel width and n is the pixel height of the user image. For the training phase we run kNN while varying k from 1 to 9. For kNN the time complexity is O(n'd+nk) where k is the hyperparameter, n is the cardinality of the training set and d is the dimension of each sample. So for our case n'=60000, d=28*28 and k is varying from 1 to 9. Total time complexity is O(60000*28*28+60000*9). For testing phase we have fixed k=K. So the time complexity will be O(60000*28*28+60000*K). Therefore, The total time complexity: O(mn)+O(n'd+nK)

- Flow Diagram:



- Pseudo Code:
  PREPROCESSING PHASE
  Input: The scanned image of a handwritten digit
  Output: Pre-processed image of the scanned image

1) Read the image file from the user as im having size m*n

2) Convert im to RGB image
3) Apply grayscaling algorithm to im to get g
4) T = threshold of binarising
5) For all pixels of g :
   if g(x,y) $>$ T: bw(x,y) = 1
   else: bw(x,y)=0
6) For row u in bw starting from start to end:
   if(bw has no 1 in u): continue
   else upperbound = u then break
7) For row v in bw starting from end to start:
   if(bw has no 1 in v): continue
   else lowerbound = v then break
8) For column x in bw starting from start to end:
   if(bw has no 1 in x): continue
   else leftbound = v then break
9) For column y in bw starting from end to start:
   if(bw has no 1 in y): continue
   else rightbound = v then break
10) bw = get tighter bounding box around the handwritten digit using lowerbound, upperbound, leftbound, rightbound
11) resize bw to 28*28
12) store bw for classification using k-NN classification algorithm

TRAINING PHASE(fixing value of k)

1) Load the MNIST training set and test set
2) Divide MNIST test set into validation and testing sets
3) P(accuracy) = 0% ; K=1
4) For k = 1 to 9
   a) Train the kNN algorithm on MNIST training set
   b) Feed the validation set images to the trained kNN
   c) kNN returns the numerical value of the digit it guesses it to be
   d) P'= prediction accuracy from this model
   e) If P' $>$ P then K = k

TESTING PHASE

1) Train the kNN algorithm with k=K on MNIST training set
2) Feed bw that has been stored for classification to the trained kNN
3) kNN returns the predicted numerical value of the digit

- Algorithms:
  - k-nearest neighbors algorithm: kNN is one kind of algorithms that can classify the data to different sets. Each digit has its own feature, so we can input all these features in kNN to train the classifier model. The training examples, i.e. digits, are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.
  - Grey scaling algorithm: We need to transfer the RGB matrix of the input image to the grayscale picture. After the grey-scaling, the image can be taken to the next step which is thresholding. The value of the grayscale image can be calculated as: Gray(x,y) = (Red(x,y) * 0.3 + Green(x,y) * 0.59 + Blue(x,y) * 0.11)
  - Thresholding algorithm: By thresholding, we can get a binary image, which is nearly the same form as the training data in MNIST. Binary image will be easier to be classified. The formula will show as follows: if Grey(x, y) $>$ T value(x, y) = 1, otherwise value(x, y) = 0
  - Crop empty border algorithm: Load an image into memory first. Second, Starting from the picture at position (0,0), and then do some unsafe row-by-row / column-by-column going through all pixels until I meet a pixel with another color, then cut away the border. And again, starting from the picture at position (end, end) and then do some unsafe row-by-row / column-by-column going through all pixels until I meet a pixel with another color, then cut away the border.
  - Data Structures: Matrix is a two-dimensional vector, which will store an m*n size data. The matrix will be used to store the image information from training data, validation data and test data.

- Constraints:
  - The user input image should include only one digit.
  - The user input image should include only a digit and this digit must be hand-written.
  - The user input image should contain a digit rather than a letter or something else.
  - The user image should not contain too much noise.

C. Stage3 - The Implementation Stage.

To implement this project, we use MATLAB R2016b as our development and write the project in MATLAB code. MATLAB has strong ability in image process, which can help us in image-preprocessing part. KNN algorithm will need a large amount of calculations in matrices and MATLAB can handle these calculations properly. The next stage is user interface of the application, and MATLAB is also can used to develop a GUI. Our application will use the user-input image as input and output the numerical value of the image. A sample data is a user input data in 1440*1080 pixels and requires 1440*1080*3 double space to store.

- **Scenario1: We have taken a digit 5 written by person 1 on paper**

Training data:
MNIST training images which have 60000 images

Figure2: the first 10 training images and labels

Testing data:
Image of the handwritten digit 5 on paper

Figure3: From left to right the images are: raw RGB image, grayscale image, binary image, resize(28*28) image, color-inverted image

Output:
Classification result (a digit between 0 an 9)

```
>> mnistOnUser
     5
```

Figure4: Execution result from Matlab

- **Scenario2: We have taken a digit 2 written by person 2 on paper**

Training data:
MNIST training images which have 60000 images

Figure4: the first 10 training images and labels

Testing data:
Image of the handwritten digit 2 on paper

Figure5: From left to right the images are: raw RGB image, grayscale image, binary image, resize(28*28) image, color-inverted image

Output:
Classification result (a digit between 0 an 9)

```
>> mnistOnUser
     2
```

Figure6: Execution result from Matlab

- **Scenario3: We have taken a digit 2 written by person 3 on white board**

Training data:
MNIST training images which have 60000 images

Figure7: the first 10 training images and labels

Testing data:
Image of the handwritten digit 2

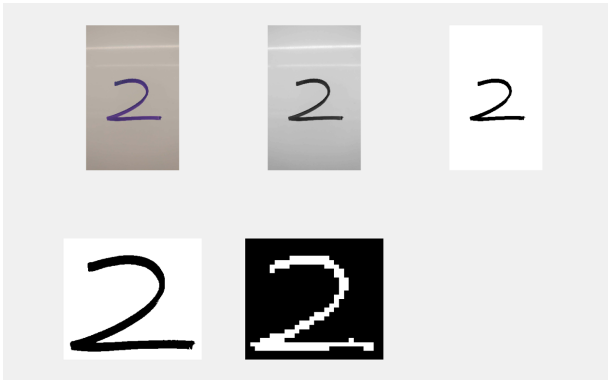


Figure8: From left to right the images are: raw RGB image, grayscale image, binary image, resize(28*28) image, color-inverted image

Output:
Classification result (a digit between 0 an 9)

```
>> mnistOnUser
     2
```

Figure9: Execution result from Matlab

- **Working code**
  Input: A photo of the handwritten image
  Output: Classification result (a number between 0 and 9)

  Step 1: Read the training images (MNIST) and the user input
  Step 2: Preprocess the user input to convert it into binary image and resize
  Step 3: Apply kNN algorithm
  Step 4: Compute the result

  kNN algorithm
  // input:
  // train_set[matrix], train_label[matrix], test[matrix]
  // output:
  // label[integer]

  Step 1: Calculate distances according to Euclidean distance
  Step 2: Set k=3 as it gives the max accuracy on MNIST validation data
  Step 3: Decide the final classification result

- **Demo and sample findings**
  - *Data size*
    For running the application, we use MATLAB R2016b which requires 1947 MB RAM and at least 2GB hard disk.
  - *Sample data input*
    Scenario 1
    Training set: 60000*784 double which takes 376320000 bytes, Test data: 1*784 double which takes 6272 bytes Input data: 1440*1080*3 double which takes 4665600 bytes
    Scenario 2
    Training set: 60000*784 double which takes 376320000 bytes, Test data: 1*784 double which takes 6272 bytes, Input data:1846*2048*3 double which takes 11341824 bytes
    Scenario 3
    Training set: 60000*784 double which takes 376320000 bytes, Test data: 1*784 double which takes 6272 bytes, Input data: 1*1080*3 double which takes 6220800 bytes
  - *Streaming*
    MNIST dataset
    We download MNIST dataset from the website http://yann.lecun.com/exdb/mnist/ and use the functions loadMNISTImage and loadMNISTLabel to load the dataset as matrix.
    User input image
    For the user input image, we just take photos of handwritten digit and use the images as input. In coding, we use the function imread() in MATLAB to load the photo as a matrix.
  - *Findings*
    We divide the MNIST test data with a 3:2 ratio. The larger division will be used as a validation set for fixing the k value. The smaller part will be used for reporting the accuracy of the system. Accuracy on the validation data on varying k value using Euclidean distance

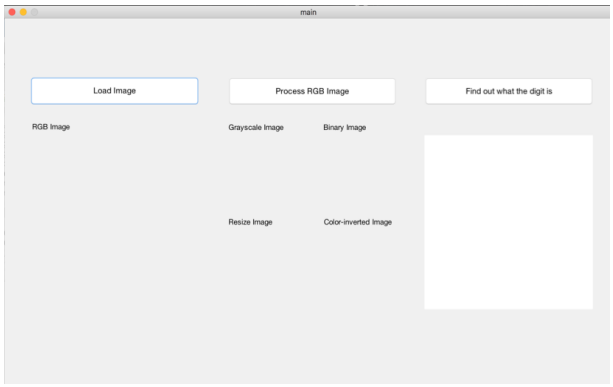
k	Accuracy
1	95.15
2	94.15
3	95.17
4	94.7
5	94.93
6	94.62
7	94.83
8	94.65
9	94.85

    As the accuracy is maximum for k=3 for the validation data, we have fixed k=3 for classifying the user input. After finding the best k value, we test the algorithm using Manhattan distance when k=3 and get the accuracy which is 94.33%. Euclidean distance performs better on the validation data, so we use kNN with Euclidean distance and k=3 to classify the user input. On the test data we created it runs with an accuracy of 97.98 %.
  - *Usefulness and novelty*
    This application can classify testing data based on training data using kNN algorithm, recognizing digits without human being. Plus, in the workplace, productivity, innovation and efficiency are always important. Our application could improve working efficiency of post offices. Instead of classifying mails and sending SMS messages by human, our system could classify mails by zip code and send SMS messages by phone number through scanners. Therefore, post offices could reduce human costs and raise profits.

D. Stage4 - User Interface.

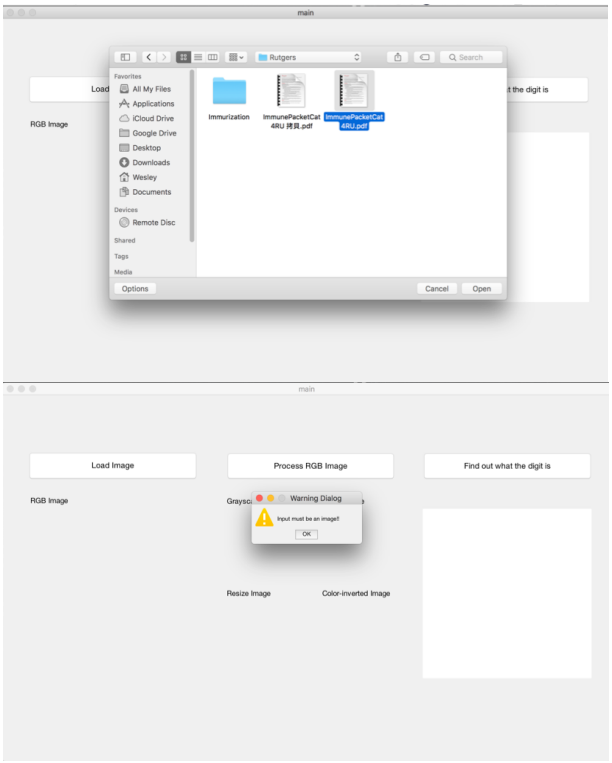
- **The initial statement to activate your application with the corresponding initial UI screenshot**
  To activate the digit recognition system, our recommended display resolution setting on MacOS is 1280x800. If user tries to activate the system on Windows, some resolution problems might happen like missing some text. User could try to reset display resolution to fix this issue. The user interface is based on a mouse and buttons. That is, user can interact with data by hovering and clicking a mouse. There are three sections in our system, and each of section owns a

button. The first section, Load Image, which allows user to load an image file from the file system after clicking the load image button. The second section, Process Image, would show user the image-processing flow step by step in our system. The third section, Find out the Digit, which would show the prediction result by classification.

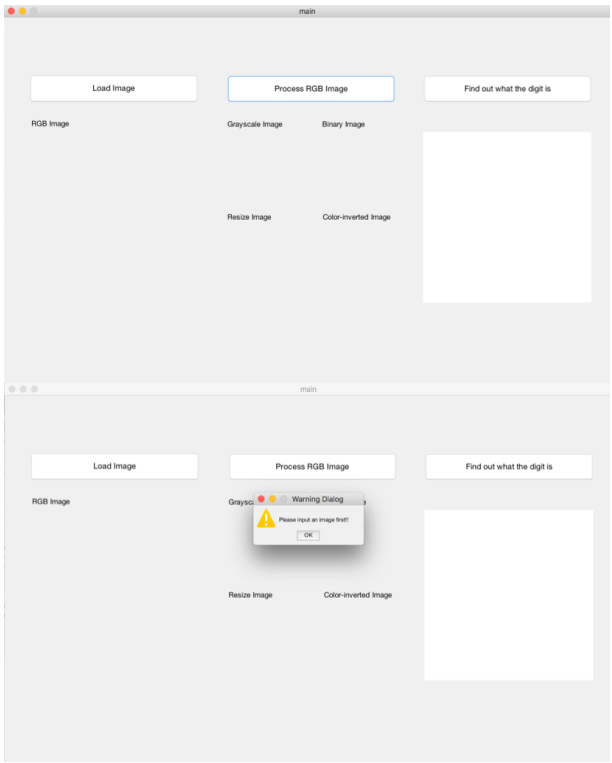


- **The error messages popping-up when users access and/or updates are denied**

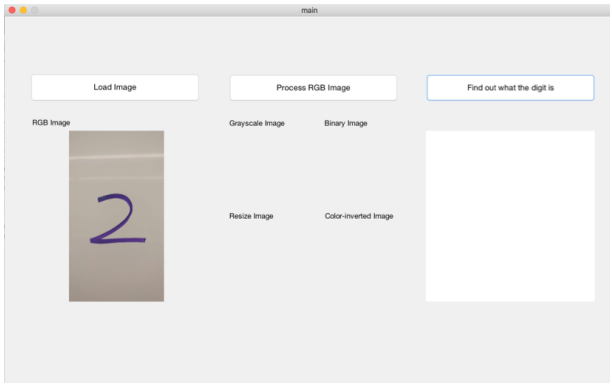
1) The error message: Input must be a image!!
When user try to load an file from the file system, the file must be an image, like jpg or png file. If the input is not an image file, the system would show this error image.
Example: user try to import a pdf file, the system would pop out this error message.

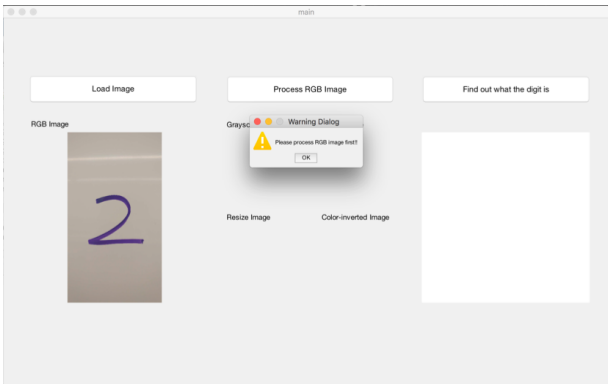


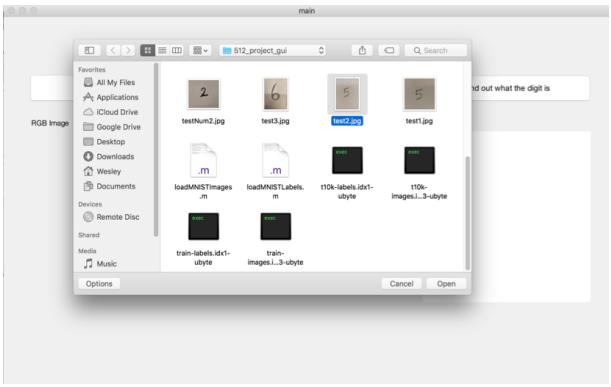
2) The error message: Please input an image first!!
When user try to process an image, he or she must load an image first. If user don't load an image, the system would show this error message while clicking this button.
Example: user try to process an image without loading it first.



3) The error message: Please process RGB image first!!
When user try to find out what the digit is, he or she must process an image first. If user don't process image first, the system would show the message.
Example: user try to find a digit without processing it first.

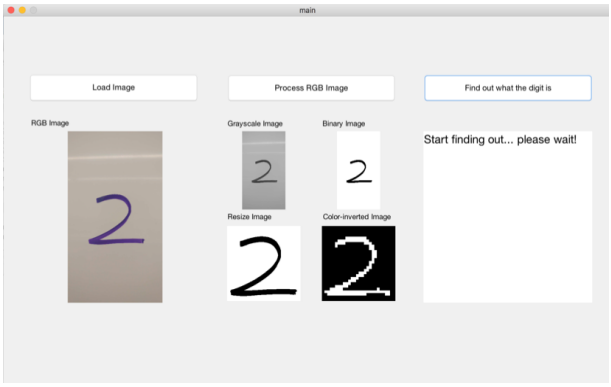




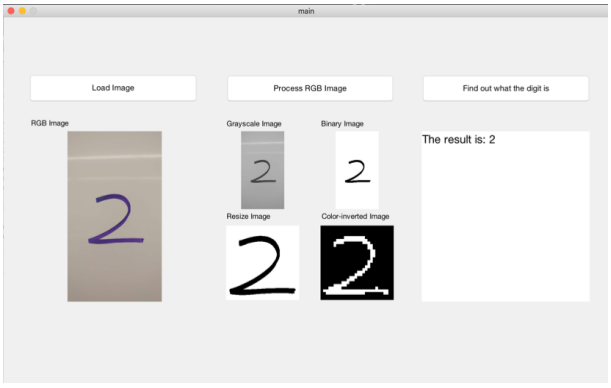


- **The information messages or results that pop-up in response to user interface events**

1) The waiting information message: Start finding out ... please wait!
   The system requires some time to classify what the digit is, while the classification function still works, the result box would show this message.



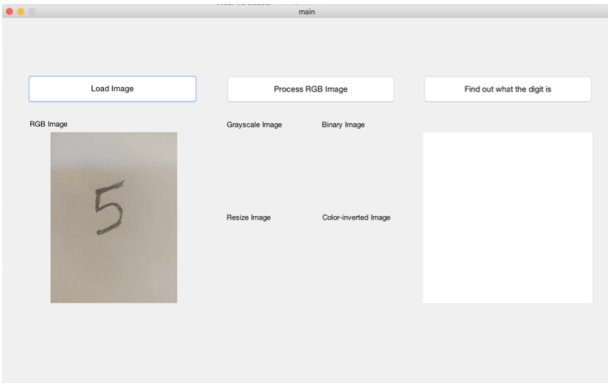
2) The result information message: The result is ...
   After classification done, the system would show the result in the result box.



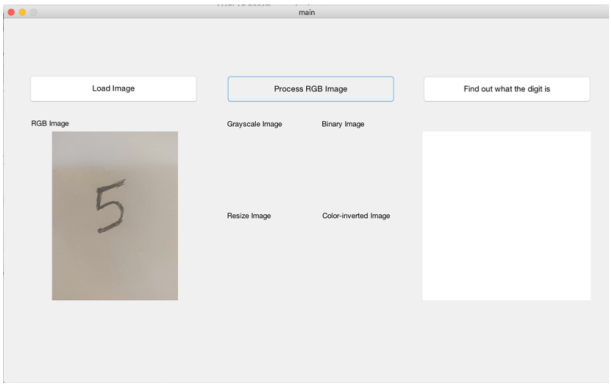
- **The interface mechanisms that activate different views**

1) Load an image from your file system

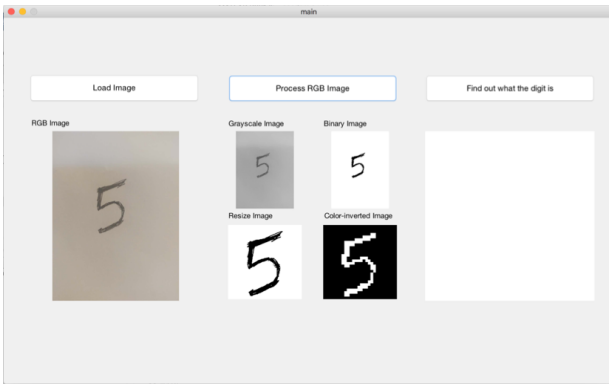
2) After loading, the image would show on the interface



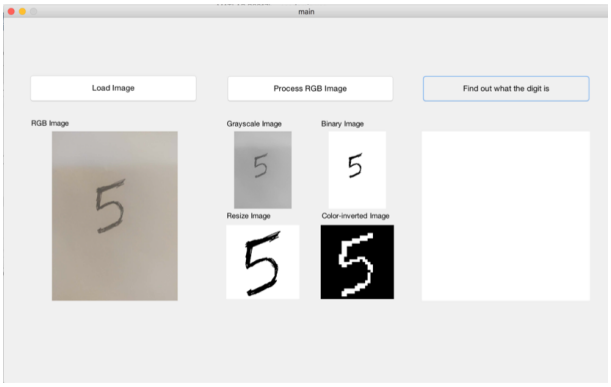
3) Process the image you loaded



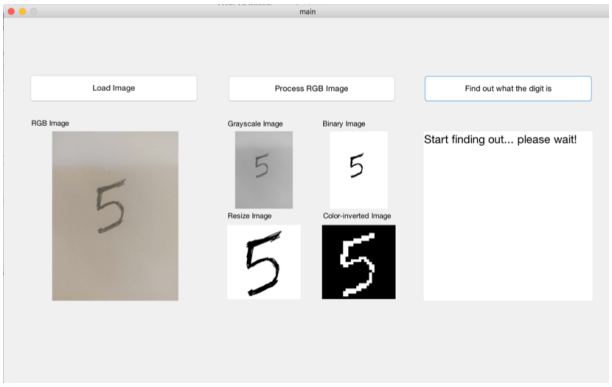
4) After processing, the image process would show on the interface



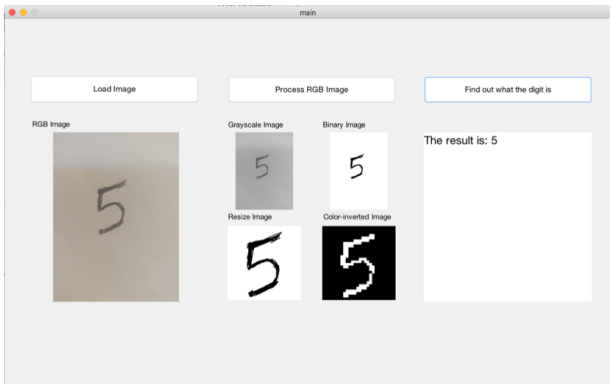
5) When image process done, you could find what the digit is



6) The waiting message would show in the result box while the system still finds



7) After finding out what the digit is, the result would show in the result box



- **Reference**

[LeCun et al., 1998a]Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. ”Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 86(11):2278-2324, November 1998.

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm