Report

1. Introduction

This project aims at realizing handwritten digits recognition using OpenCV in C++. The program read the digit images and corresponding labels, which indicates the number the writer had intended to write, from the training dataset to train a machine learning model. Then, the trained model together with an input program could recognize the hand written digits.

The handwritten digit recognition techonology plays an important role in people's daily lives. It is widely implemented in scenaarios such as automated data entry in banks, automated grading system in schools... Needless to say, it is very significant.

2. Run the codes

Since setting the environment of OpenCV for C++ is quite troublesome in VS Code and Google Colab etc., Visual Studio (2022) is more convenient. So I set the maximum iteration to be 10 times in order to successfully train locally.

(1) Change directory to "opencvtest", double click "opencvtest.sln". Remember to substitute the path to training dataset and testing dataset to your own path! ctrl+F5 to run.

```
string train_image_path = "your_disc:/your_path/opencvtest/train&test/train-
images.idx3-ubyte";
(same for the following 3 paths)
```

(2) The training process costs about half an hour on my laptop. So, thank you for your patience!

(3) After the training, you will find "dnn_model.xml" in the current directory, which is "opencvtest". You will also find the test result in the terminal.

(4) If you want to further try the recognition & visualization features, move "dnn_model.xml" to directory "predict". Double click "predict.sln".

(5) If you want to extract the images from the source database and convert them into .jpg, please uncomment the extract() function at the beginning of the main() function, and change the parameter into your local path to test images and test labels. **(I have already extracted all the testing images into the directory "images", if you don't want to do this.)**

Also, remember to change the path to where you want to save the images into your own path, which you can find at the end of extract() function "saveFile=...".

```
string saveFile = "path_to_the_directory_where_you_want_to_save_the_images" +
to_string((unsigned int)label) + "_" + to_string(i) + ".jpg";
```

(6) Put the handwritten digit image into the same directory of "predict.cpp", which is the directory "predict". Remember to change the path to the image in the "predict.cpp".

```
Mat image = imread("image_file_name.extension", 0);
```

(7) ctrl+F5 to run

3. Interpretation of the codes

Part 1: train.cpp

In the main function, there are 3 parts: preparing data, train the DNN, and do testing.

(1) preparing data: here DNN method is adopted for a higher effectiveness

Mat one_hot(Mat label, int num): to transform label data into one-hot type.

e.g. 5 --> [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

```
train_labels = one_hot(train_labels, 10);
```

Normalization: to map the value in the image matrix into [0, 1] for subsequent process.

```
train_images = train_images / 255.0;
```

(2) build an DNN (by creating an ANN and adding 3 hidden layers) and train the model

layerN: the input layer has 784 (28x28) neurons, the output layer has 10 neurons (for digit 0 to digit 9), and the 3 hidden layers have 128, 64, 32 (just because it looks professional) neurons relatively.

setTrainMethod: here we use backpropagation method for gradient descent

setActivationFunction: here we use sigmoid funtion for mapping

setTermCriteria: here we set the maximum iteration to be 10 times and the epison (accuracy) to be 0.0001

```
Mat layerN = (Mat_<int>(1, 5) << 784, 128, 64, 32, 10);
ann->setLayerSizes(layerN);
ann->setTrainMethod(ml::ANN_MLP::BACKPROP, 0.001, 0.1);
ann->setActivationFunction(ml::ANN_MLP::SIGMOID_SYM, 1.0, 1.0);
ann->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER + TermCriteria::EPS, 10,
0.0001));
```

(3) test the model

prediction: it is a matrix with each row representing one image and each column indicating the likelihood of being the digit (e.g. column 0 represents digit 0).

minMaxLoc: to find minimum and maximum, here we only find maximum.

```
Mat prediction;
ann->predict(test_images, prediction);
//compute accuracy
int correctNum = 0;
for (int i = 0; i < prediction.rows; i++)
{
    //get i_th image's prediction
    Mat tmp = prediction.rowRange(i, i + 1);
    double max_value = 0;
    Point max_point;
    // find the max probability
    minMaxLoc(tmp, NULL, &max_value, NULL, &max_point);
    int predict_label = max_point.x;
    int true_label = test_labels.at<int32_t>(i, 0);
    if (predict_label == true_label)
    {
        correctNum++;
    }
}
double accuracy = double(correctNum) / double(prediction.rows);Finally, we save the
trained model
```

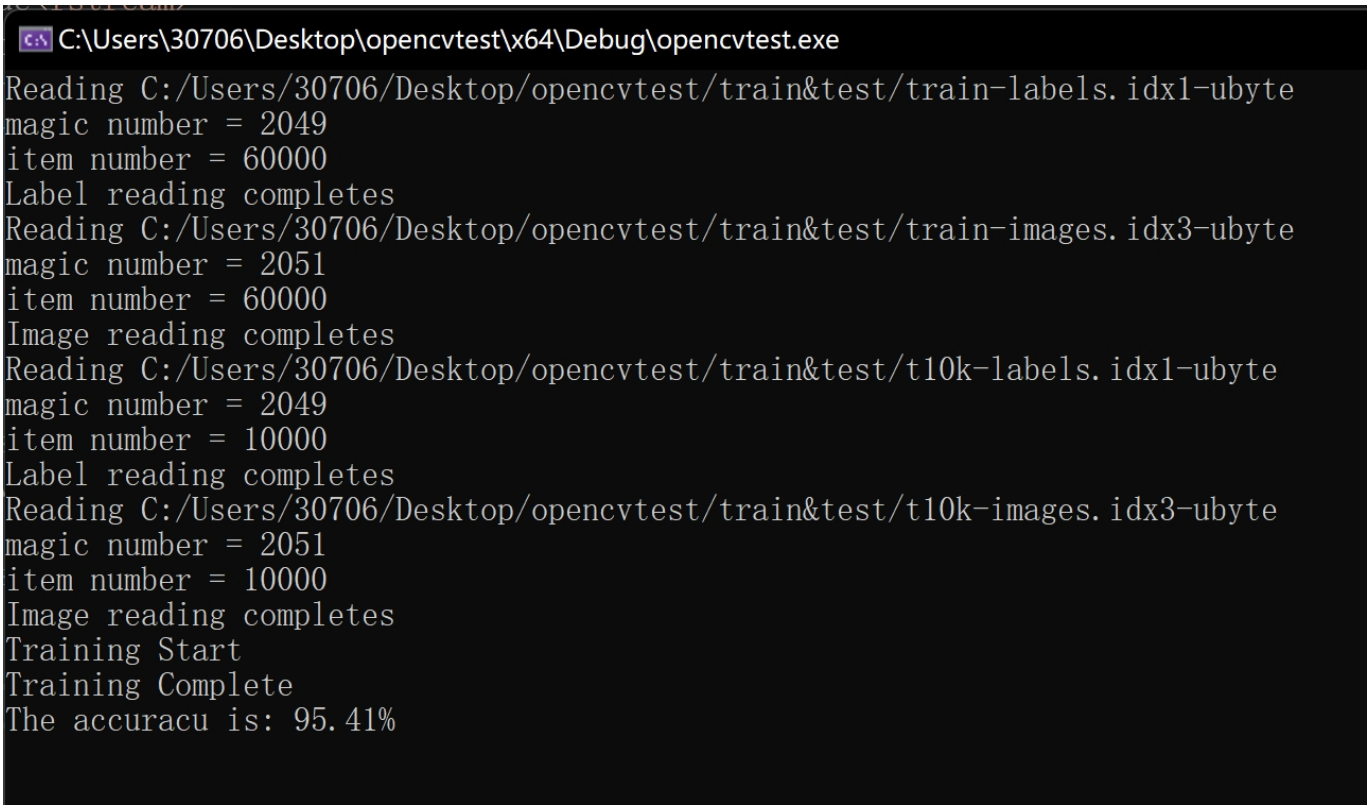
Part 2: predict.cpp

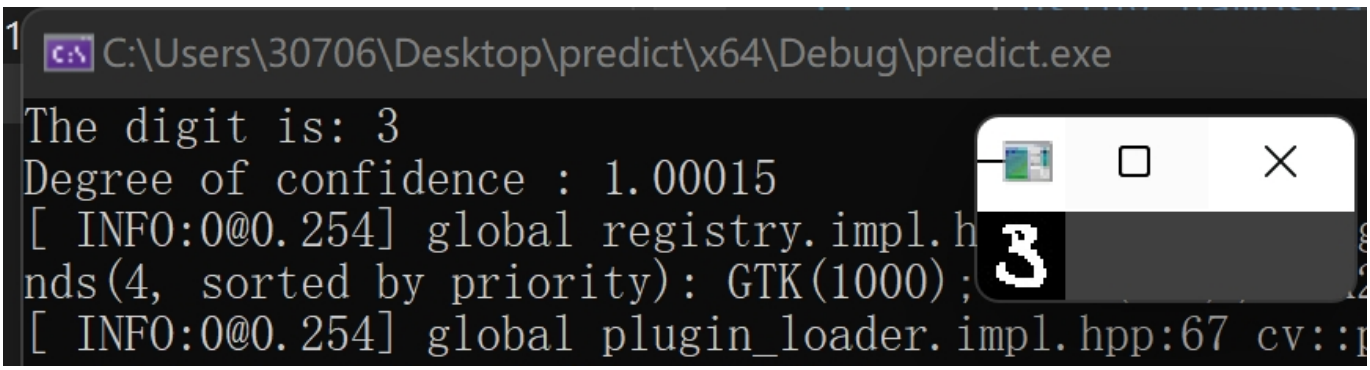
The core procedure is very similar to the testing part.

(For more interpretation, please refer to the comments in the codes)

4. Results

The test result of *train.cpp* showed that the accuracy rate was 95.41%, which is higher then what was expected, because it was trained on a laptap. Also, the recognition result of *predict.cpp* is successful.





5. Conclusion

By using the DNN method, the accuracy is high. If we increase the max iterations in *ann->setTermCriteria*, probably it will perform better. Overall, this experiment is successful and it does correctly recognize handwritten digits.