8690 Final Report

-Digit Recognizer

github.com/wxqk3/Digit-Recognizer\_Deep\_Learning



**Weiliang Xia**

5.15.2019

# 

# 1. Team Project Topic

Digit Recognizer

# 2. Team members

Weiliang Xia

# 3. Input Data (with some illustration figures)

**Two csv file**

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

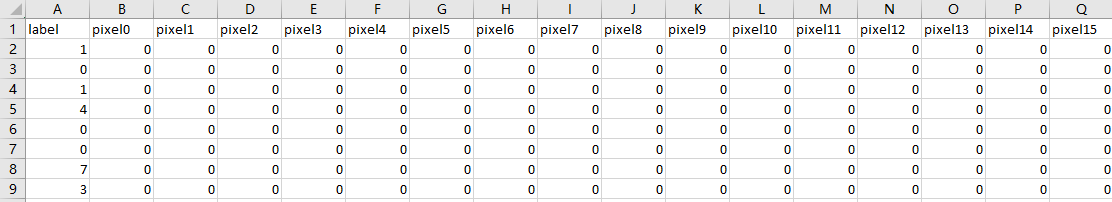
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

**Train.csv (42000, 784)**

42000 examples, each example has 784 pixels(28\*28)



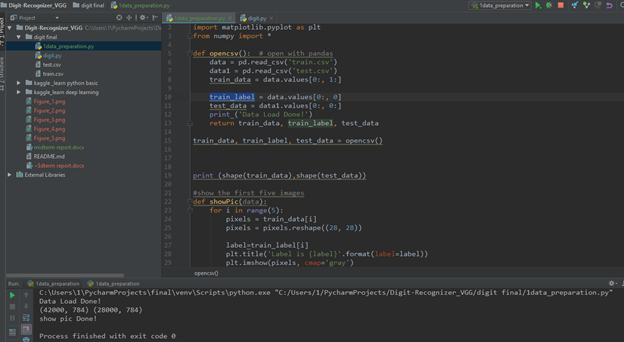
Each row is a picture example contain 28\*28 pixels

And there are 42000 rows/examples in this csv.

**Test.csv (28000, 784)**

28000 examples, each example has 784 pixels(28\*28)

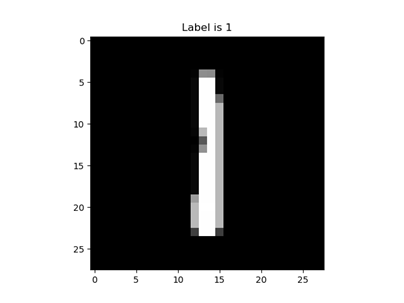
After my part1 coding:



1.data preparation

pandas to processing csv

pyplot to visualize the raw data



# 4. Output data (with some illustration figures)

According to testing data (28000,784),output data should be (28000,1) like following

ImageId,Label

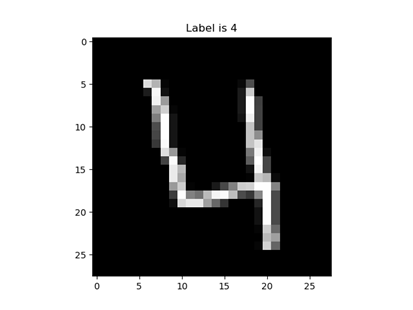
1,3

2,7

3,8

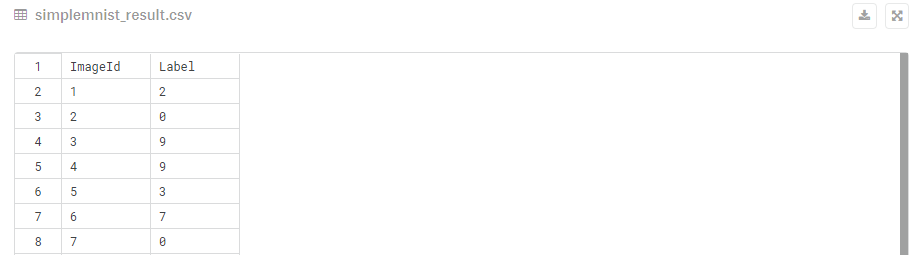
(27997 more lines)

If show in figure



# 5. State-of-art-results which will serve as baseline methods for comparison.

I Choose the best results from kaggle LeaderBoard Top1 team as the baseline comparison: which has 100% accuracy



# 6. Approach such as Network architecture.

After a few draft approaches below 90%...

First approach:

In -> [Conv2D,relu]\*2 -> Flatten -> [Dense,relu] -> [Dense,softmax] ->Out

**

Rank 1893 and I have the accuracy of 97.4% (1893 out of 2943, top 64%)

Final approach:

In -> [Conv2D,relu]\*2 -> MaxPool2D -> Dropout(0.25)

-> [Conv2D,relu]\*2 -> MaxPool2D -> Dropout(0.25)

-> [Dense,relu] -> [Dense,softmax] ->Out

For the final approach, I have the accuracy of 98%