Pattern Recognition

ASSIGNMENT REPORT

Date of submission: 2019/07/03

Name: 劉　立坤

Student ID: 2IE19337P

Contents

[1. Abstract 2](#_Toc13049481)

[2. Implementation details 2](#_Toc13049482)

[2.1 data\_loader 2](#_Toc13049483)

[2.2 intensity\_attenuator 2](#_Toc13049484)

[2.3 binarizer 2](#_Toc13049485)

[2.4 k\_nearest\_neighbor 2](#_Toc13049486)

[2.5 CNN 3](#_Toc13049487)

[3. Results 4](#_Toc13049488)

[3.1 Step 01/ data loader 4](#_Toc13049489)

[3.2 k\_nearest\_neighbor 4](#_Toc13049490)

[3.3 CNN 6](#_Toc13049491)

[4. Analysis 7](#_Toc13049492)

[4.1 Data loading 7](#_Toc13049493)

[4.2 Image processors 7](#_Toc13049494)

[4.3 KNN classifier 7](#_Toc13049495)

[4.4 Neural Nets 7](#_Toc13049496)

[5. References 8](#_Toc13049497)

# Abstract

In this assignment, we’ve reimplemented step 01,02,04 and 05 provided by sample program as well as a simple CNN classifier in python.

In step 01, we’ve changed the original read one process one into read in all the images including both training samples and test samples into a Numpy array and then reshape the single (28,28) image array into a single row vector so that the whole dataset became a (n,784) matrix to increase recognition speed. Also, to reduce the loading time, an ‘npz’ file was created for quick data loading.

For step 02 and step 04 we’ve combined them into a single python package ‘image\_processors’, in this package there’re 2 simple functions: intensity attenuator and binarizer which both directly return the result as a NumPy array.

In step 05, we’ve changed the distance from Manhattan distance to Euclidian distance which greatly increase calculation complexity (To measure the time in same condition, the sample code’s distance calculation method has also changed to Euclidian distance via Numpy’s linear algebra package). The result is stored in a distance matrix and then sort each sample’s distance array to find the nearest neighbor or k nearest neighbor.

In addition, a simple CNN was implemented to increase the accuracy and recognition speed.

# Implementation details

## data\_loader

In this module, we firstly read each image in the datasets and reshape the image into a (1,784) Numpy array which then form a (datasize,784) matrix through append method. Meanwhile, we’ve also split the file name and extract the label from file name then store it in the label array. This module returns 4 Numpy arrays: Xtr for training images, Ytr for training images’ labels, Xte for test images and Yte for test images’ labels.

## intensity\_attenuator

This module accepts a Numpy array and an integer ranges from 0 to 100 as strength, then multiply to the Numpy array.

## binarizer

This module accepts a Numpy array and a threshold value. For any point in data array, if the value is greater than threshold then the value is set to 255 else set to 0.

## k\_nearest\_neighbor [1]

This module takes data array and label array and give out a distance matrix by using simple linear algebra method: in which both A and B are matrixes.



## 2.5 CNN

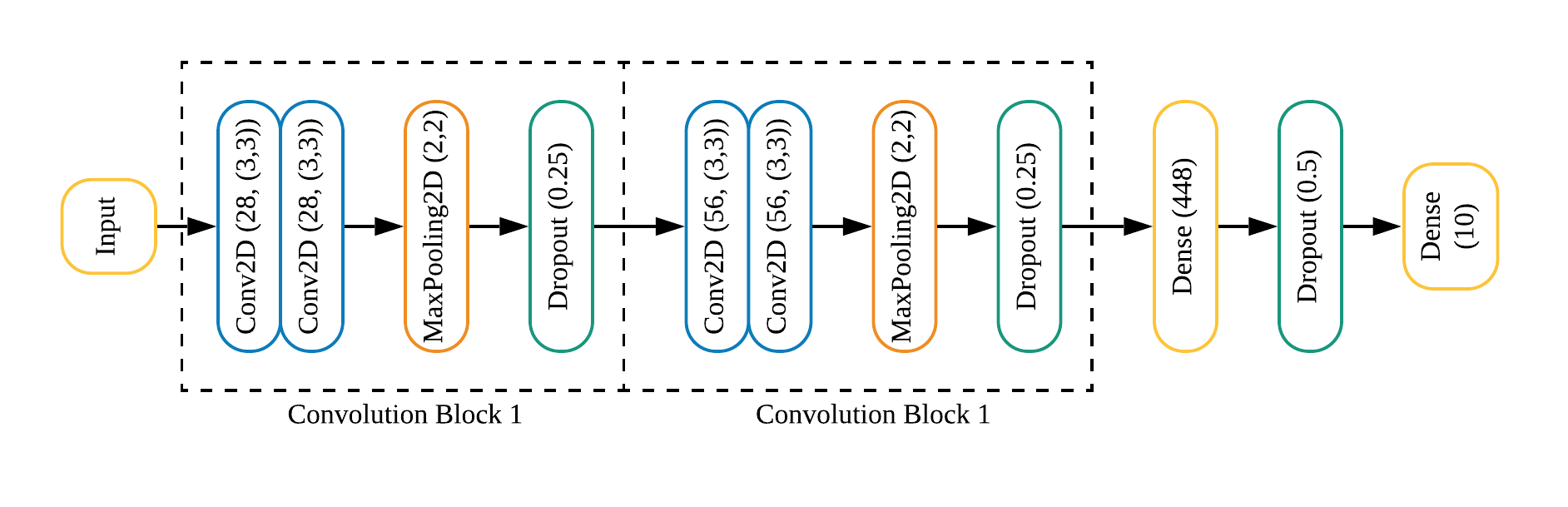
In this part we’ve Implemented a neural network with 2 convolution blocks (Consists of 2 convolution layers, 1 2X2 MaxPooling layer and 1 dropout layer) with a 3X3 window size and a dense layer with activation function of “ReLU”, and an output layer with “Softmax” as activation function. As ordinary neural networks, this model utilizes cross entropy as loss function and Adam as optimizer. The entire architecture is shown in figure 1.

Figure 1 Simple neural network’s architecture

Summary of the model is as follows:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 28, 28, 28) 280

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 26, 26, 28) 7084

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 13, 13, 28) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 13, 13, 28) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 13, 13, 56) 14168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 11, 11, 56) 28280

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 5, 5, 56) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 5, 5, 56) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 1400) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 448) 627648

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 448) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 10) 4490

=================================================================

Total params: 681,950

Trainable params: 681,950

Non-trainable params: 0

We then trained this model with mnist dataset’s training data using 60 times passes and shuffle enabled.

# Results

## Step 01/ data loader

As the original method’s execution time could not be measured as a result of it’s read one process one method, we’ve implemented another function data\_loader to read in all data at once, result is as follows:

The shape of training data is: (2000, 784)

The shape of training label is: (2000,)

The shape of test data is: (10000, 784)

The shape of test label is: (10000,)

Data loading took 170.295777 seconds

With the new method which directly read in a Numpy array:

New data loading took 0.111530 seconds

The loading process therefore is around 1500X faster than original method. (The corresponding “. npz” file is approximately 72MB and the testing environment used an SSD with a 3400MB/s maximum sequential read speed).

## k\_nearest\_neighbor

The accuracy and total accuracy after change from Manhattan distance to Euclidian distance is as follows:

0: 0.8430

1: 0.9920

2: 0.5940

3: 0.5770

4: 0.5470

5: 0.4190

6: 0.7540

7: 0.7160

8: 0.4560

9: 0.6590

= Confusion matrix ===========

0843, 0030, 0005, 0003, 0000, 0014, 0078, 0002, 0020, 0005,

0001, 0992, 0001, 0004, 0000, 0000, 0002, 0000, 0000, 0000,

0098, 0147, 0594, 0037, 0009, 0001, 0033, 0020, 0055, 0006,

0021, 0173, 0019, 0577, 0005, 0039, 0036, 0016, 0076, 0038,

0094, 0128, 0002, 0001, 0547, 0001, 0057, 0022, 0019, 0129,

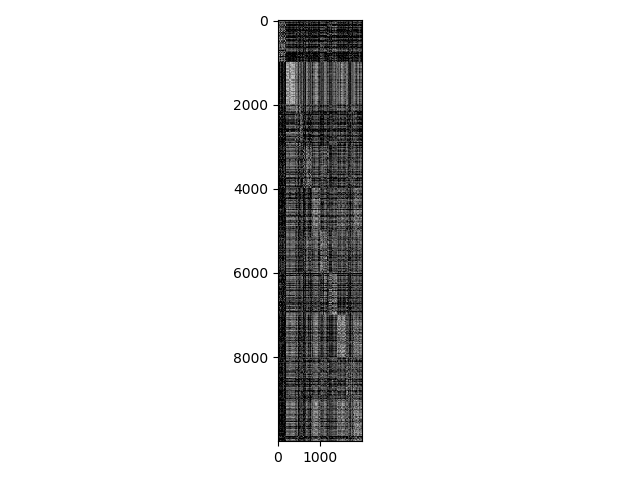
0062, 0173, 0005, 0066, 0013, 0419, 0134, 0002, 0073, 0053,

0102, 0116, 0009, 0000, 0002, 0010, 0754, 0001, 0004, 0002,

0037, 0109, 0006, 0011, 0014, 0000, 0001, 0716, 0000, 0106,

0020, 0271, 0028, 0040, 0008, 0052, 0079, 0013, 0456, 0033,

0032, 0102, 0002, 0009, 0095, 0004, 0006, 0077, 0014, 0659,

= Total Recognition accuracy ===========

TOTAL: 0.6557

Classification took 462.822850 seconds

Note that the 462 seconds here is classification and data loading’s combined time.

For new method, we’ve also plotted the distance matrix as shown in figure 2.

The recognition result is:

Got 8971 / 10000 correct => accuracy: 0.897100 @ k=1

Classification took 2.015476 seconds

Take the previous time consumed by the data loading procedure into consideration, the total time is: 172.311253 seconds. The performance was boosted by around 2.7X.

Figure 2 distance matrix

In addition, we’ve also tested a set of K values [1,3,5,7,11,13,17,19,50,100] to show the knn’s performance on the writing digits dataset.

Got 8971 / 10000 correct => accuracy: 0.897100 @ k=1

Got 8945 / 10000 correct => accuracy: 0.894500 @ k=3

Got 8919 / 10000 correct => accuracy: 0.891900 @ k=5

Got 8873 / 10000 correct => accuracy: 0.887300 @ k=7

Got 8784 / 10000 correct => accuracy: 0.878400 @ k=11

Got 8750 / 10000 correct => accuracy: 0.875000 @ k=13

Got 8658 / 10000 correct => accuracy: 0.865800 @ k=17

Got 8626 / 10000 correct => accuracy: 0.862600 @ k=19

Got 8112 / 10000 correct => accuracy: 0.811200 @ k=50

Got 7648 / 10000 correct => accuracy: 0.764800 @ k=100

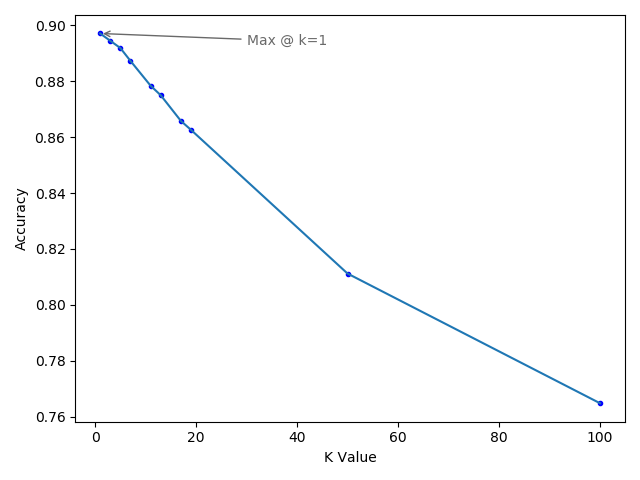
The result indicated that for the writing digits, it’s probably not wise to choose a large k value.

Figure 3 K values verses accuracies

## CNN

This simple model trained by mnist dataset’s training data can achieve 99.99% accuracy on the samples test dataset.

Got 9999 correct, accuracy = 99.990000%

Meanwhile, as for the mnist’s test data, the accuracy has dropped to 99.58% as shown below:

Got 9958 correct, accuracy = 99.580000%

# Analysis

## Data loading

We’ve changed the data loading process from read in a set of small png file into read in an entire Numpy array, which can be quickly loaded into RAM for further processing.

## Image processors

The intensity attenuator reduces the intensity value of the original image and will affect knn’s accuracy due to more indistinguishable distance

The binarizer take the threshold and change the image into a 0/255 sequence, with proper threshold, this method could raise the accuracy of the recognition.

## KNN classifier

By switching form Manhattan distance to Euclidian distance, the accuracy will be higher but with a much greater cost. Also, typically, a traditional 2 loops distance calculation method would take approximately around 200 seconds for the given distance matrix, by using Numpy’s linear algebra method, we’ve successfully cut the time down to around 0.2 seconds (calculation procedure only).

## Neural Nets

The simple neural network has reached an impressive accuracy of 99.99%, however, as the test data from the sample program may contains data from mnist’s training samples, it seems only fair to run with mnist’s test data, and the result show that the implemented neural network’s accuracy on mnist dataset is 99.56%.

# References

[1] K. Hechenbichler and K. Schliep, “Weighted k-Nearest-Neighbor Techniques and Ordinal Classification,” 2004. [Online]. Available: https://epub.ub.uni-muenchen.de/1769/. [Accessed: 03-Jul-2019].

[2] “Stanford University CS231n: Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.stanford.edu/. [Accessed: 03-Jul-2019].