深度視覺 HW2

M113040064 資工碩 李冠宏

執行結果分析

1. Data Preparation and Visualization

首先將 colab 連結到 google drive，新增路徑並引入相關套件，最後下載用來訓練和測試的 train data 和 test data 以及他們對應的 label

```
from google.colab import drive
drive.mount('/content/drive')

import sys
sys.path.append('/content/drive/MyDrive/HW2/')

from _utils import load_data
import numpy as np
import matplotlib.pyplot as plt

train_data, train_label, test_data, test_label, classes = load_data()
```

印出所需資料的 shape 及 class 長度

```
print(f'Shape of training data: {train_data.shape}')
print(f'Shape of training labels: {train_label.shape}')
print(f'Shape of test data: {test_data.shape}')
print(f'Shape of test labels: {test_label.shape}')
print(f'Number of classes: {len(classes)}')
```

執行結果

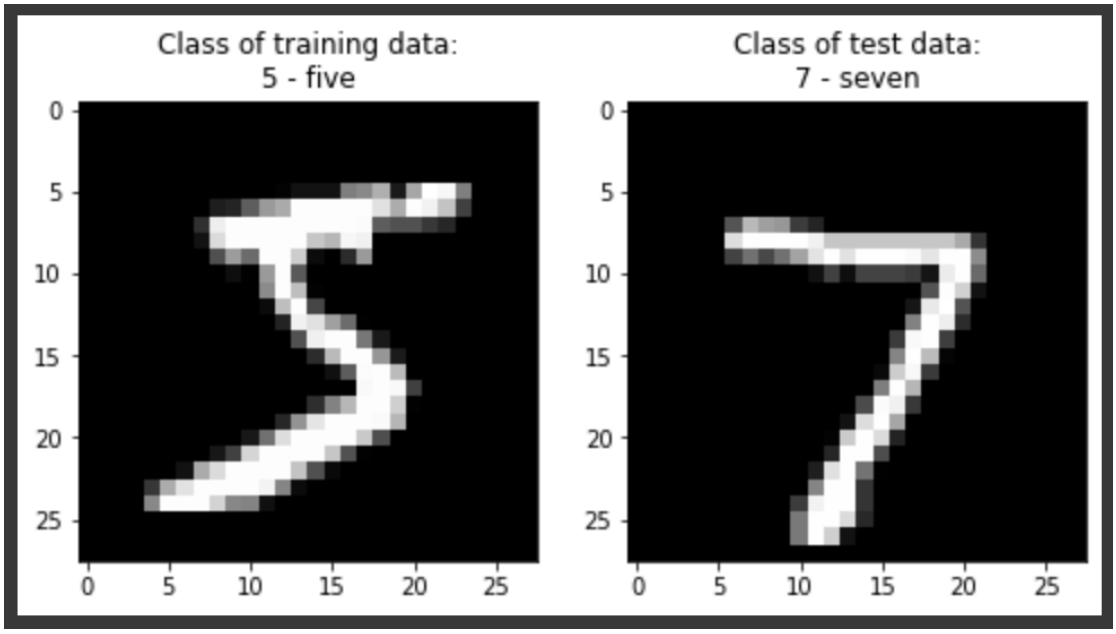
```
Shape of training data: (60000, 28, 28)
Shape of training labels: (60000,)
Shape of test data: (10000, 28, 28)
Shape of test labels: (10000,)
Number of classes: 10
```

2. 測試 training data 和 testing data 各一筆

這階段透過 matplotlib.pyplot 套件來畫子圖，測試我們的資料是否有正確地被引入進來

```
# Display samples of training data and test data with their classes
plt.rcParams['figure.figsize'] = (8, 5)
plt.rcParams['image.cmap'] = 'gray'
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(train_data[0]); ax1.set_title(f'Class of training data:\n{classes[train_label[0]]}')
ax2.imshow(test_data[0]); ax2.set_title(f'Class of test data:\n{classes[test_label[0]]}')
fig.show()
```

執行結果：

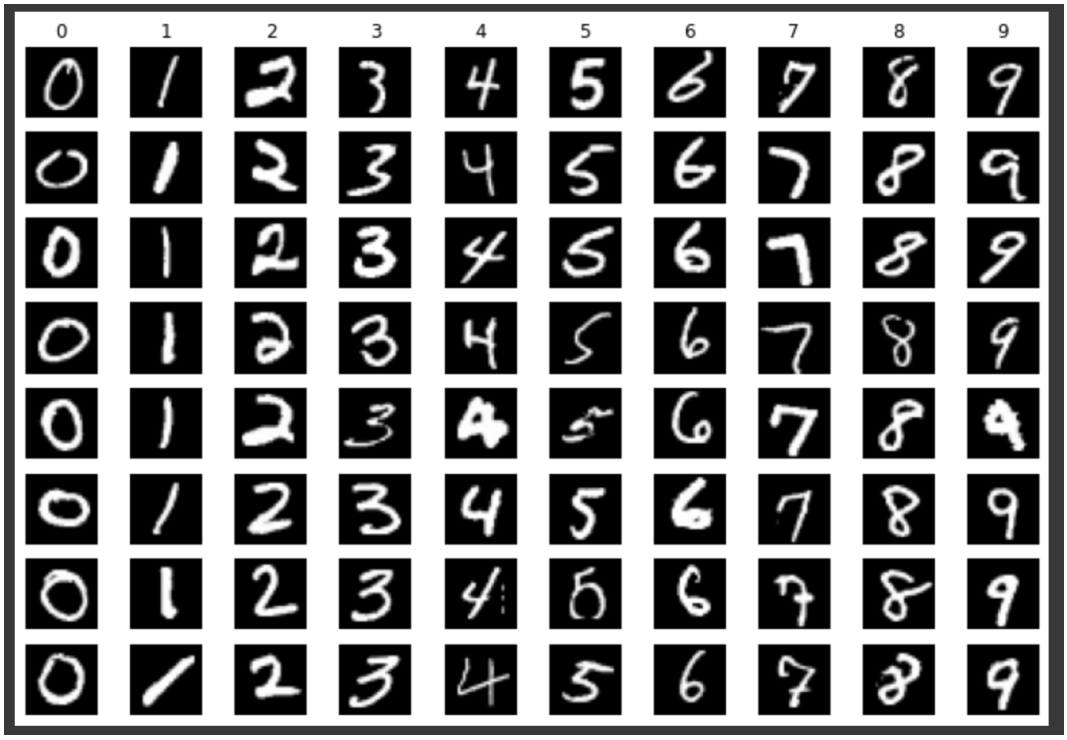


3. 測試所有 training data

這階段將所有 train data 進行測試，並印出每個 class 中屬於此類別的 train data

```
# Display samples of training data from every classes
plt.rcParams['figure.figsize'] = (12, 8)
num_classes = len(classes)
num_example = 8 #每一個類別的數量
for label, _ in enumerate(classes):
  idxs = np.flatnonzero(train_label == label) #flatnonzero 會回傳非 0 元素的 index
  idxs = np.random.choice(idxs, size=num_example, replace=False) # 代表從 idxs 中隨機抽取 num_example 的"不重複"數字
  for i, idx in enumerate(idxs):
    plt_idx = i * num_classes + label + 1
    plt.subplot(num_example, num_classes, plt_idx) #plt.subplot(row, column, index)
    plt.imshow(train_data[idx]); plt.axis('off')
    if i == 0:
      plt.title(label) #印出類別的 index
plt.show()
```

執行結果：



4. KNN Implementation

這階段是建構模型，在 knn 類別中，我們首先在建構式定義好 train data 的大小、資料本身和 label。

接下來定義 predict 方法，這裡分為 L1 和 L2，其中公式定義如下：

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

在 L1 和 L2 分別計算好 test data 和 train data 的距離後，存入名為 dists 的 numpy 2d array 中，有了這個 array 後，對於每一筆 test data，我們就可以在這裡面找 5 個距離最小的點、取得他們的 index，最後取得他們的 label 並找出現頻率最高的 label 作為我們預測的 label，再存回 preds 中

```
class knn(object):
  def __init__(self, data, label):
    self.num_train_data = data.shape[0] #5000
    self.train_data = data.reshape(self.num_train_data, -1) #訓練資料
    self.train_label = label #訓練解答

  def predict(self, test_data, dist_metric='l1', k=1):
    num_test_data = test_data.shape[0] #500
    dists = np.zeros((num_test_data, self.num_train_data)) #dist 為 num_test_data * num_train_data 大小的二維矩陣
    if dist_metric == 'l1':
      ####################
      # TODO:
      # 1. Flatten test_data, that is, change its shape from (num_test_data, test_data_height, test_data_width) to (num_test_data, flatten_test_data)
      # 2. Caculate L1 distances between test data and all the training data for each test data, and then store distances in variable `dists`
      ####################

      # -----START OF YOUR CODE-----
      test_data.reshape(num_test_data, -1) #flatten the numpy array
      for test_idx in range(num_test_data):
        for train_idx in range(self.num_train_data):
```

```
        dists[test_idx][train_idx] =
np.sum(np.abs(test_data[test_idx] - train_data[train_idx]))
      # ------END OF YOUR CODE------

    elif dist_metric == 'l2':
      ####################
      # TODO:
      # 1. Flatten test_data, that is, change its shape from
(num_test_data, test_data_height, test_data_width) to (num_test_data,
flatten_test_data)
      # 2. Caculate L2 distances between test data and all the training
data for each test data, and then store distances in variable `dists`
      ####################

      # -----START OF YOUR CODE-----
      test_data.reshape(num_test_data, -1) #flatten the numpy array
      for test_idx in range(num_test_data):
        for train_idx in range(self.num_train_data):
          dists[test_idx][train_idx] =
np.sqrt(np.sum(np.square(test_data[test_idx] -
train_data[train_idx])))
      # ------END OF YOUR CODE------

    else:
      raise ValueError("dist_metric can only be 'l1' or 'l2'")

    preds = np.zeros(num_test_data)
    ####################
    # TODO:
    # 1. Take majority vote from k closest data to assign each test
data a label, and then store labels in variable `preds`
    ####################

    # -----START OF YOUR CODE-----
    # min_index = np.where(dists[0] == min(dists[0]))
    for test_idx in range(len(test_data)):
      indexs = np.argpartition(dists[test_idx], k) #最近 k 個點的 index
      labels = []
```

```
      for idx in indexs[:k]:
        labels.append(train_label[idx])
      max_value = np.bincount(labels).argmax() #找頻率最高的
      # min_index = np.where(dists[test_idx] == min(dists[test_idx]))
      preds[test_idx] = max_value
    # ------END OF YOUR CODE------

    return preds
```

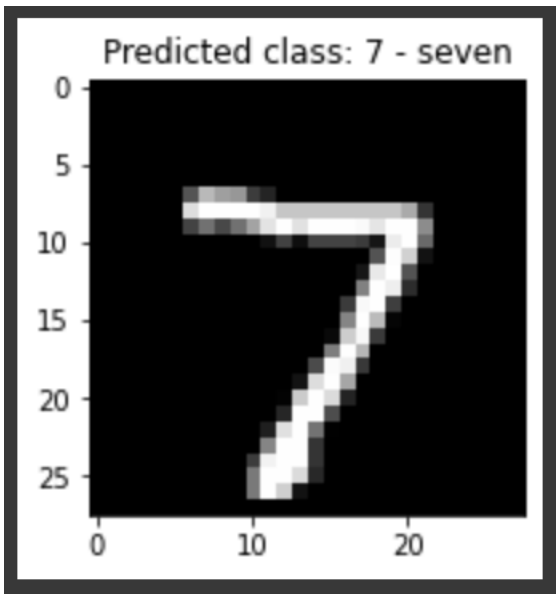
5. 執行單筆資料測試

```
# Run a single test
single_test_data = test_data[:1] #test_data 的第一筆資料
num_test_data, test_data_height, test_data_width =
single_test_data.shape

#建立 model
classifier = knn(train_data, train_label)
#猜測
predicted_label = classifier.predict(single_test_data, k=1,
dist_metric='l2')[0].astype(np.int32)

#畫圖
plt.rcParams['figure.figsize'] = (3, 3)
plt.imshow(single_test_data.reshape(test_data_height,
test_data_width)); plt.title(f'Predicted class:
{classes[predicted_label]}')
plt.show()
```

執行結果：
(預測正確)



6. Applied in Small Dataset

測試 train data 和 test data 分別的數量，並建立 knn model

```
small_train_data = train_data[:5000]
small_train_label = train_label[:5000]
small_test_data = test_data[:500]
small_test_label = test_label[:500]
classifier = knn(small_train_data, small_train_label)
print(f'Number of training data: {small_train_data.shape[0]}')
print(f'Number of test data: {small_test_data.shape[0]}')
```

執行結果：

```
Number of training data: 5000
Number of test data: 500
```

在較小的 dataset 中進行訓練及測試

```
num_test_data = small_test_data.shape[0]
for dm in ['l1', 'l2']:
  print(f'Using {dm.upper()} distance metric:')
  for kv in [2, 3, 4, 10, 20]:
```

```
    preds = classifier.predict(small_test_data, k=kv,
dist_metric=dm)
    num_correct = np.sum(preds == small_test_label)
    accuracy = float(num_correct) / num_test_data
    print(f'k = {kv}, accuracy = {accuracy}')
  print('')
```

執行結果：
在 k 值相同的情況下，L1 的準確度比 L2 高很多，而在 k 相異的情況下，k 一般來說不要太大結果會比較好

```
Using L1 distance metric:
k = 2, accuracy = 0.666
k = 3, accuracy = 0.692
k = 4, accuracy = 0.69
k = 10, accuracy = 0.688
k = 20, accuracy = 0.672

Using L2 distance metric:
k = 2, accuracy = 0.204
k = 3, accuracy = 0.204
k = 4, accuracy = 0.21
k = 10, accuracy = 0.196
k = 20, accuracy = 0.172
```

7. Applied in Whole Dataset

為了要找到最好的算距離方法以及 k，在此用了一段程式去訓練就上述而言所有的可能性，並在每一回合更新最好高準確率的 k 值，最後有最高準確率的組合為使用 L1 且 k 值為 4

```
num_test_data = test_data.shape[0]
best_k = 0
best_method = ""
highest_accuracy = 0

for dm in ['l1', 'l2']:
  for kv in [2, 3, 4, 10, 20]:
    preds = classifier.predict(test_data, k=kv, dist_metric=dm)
```

```
    num_correct = np.sum(preds == test_label)
    accuracy = float(num_correct) / num_test_data
    print(f'Using {dm.upper()} distance metric, k = {kv}\nAccuracy =
{accuracy*100}%')
    if highest_accuracy < accuracy:
      highest_accuracy = accuracy
      best_method = dm
      best_k = kv
    print(f"目前最好的組合為: {best_method} + k={best_k}, 其準確率為:
{highest_accuracy*100}%\n")
    print("-----------------------------------------------\n")
```

執行結果：

L1 + k=4 有最高的準確率

```
Using L1 distance metric, k = 2
Accuracy = 71.00999999999999%
目前最好的組合為: l1 + k=2, 其準確率為: 71.00999999999999%

-----------------------------------------------

Using L1 distance metric, k = 3
Accuracy = 72.85000000000001%
目前最好的組合為: l1 + k=3, 其準確率為: 72.85000000000001%

-----------------------------------------------

Using L1 distance metric, k = 4
Accuracy = 73.71%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L1 distance metric, k = 10
Accuracy = 73.69%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L1 distance metric, k = 20
Accuracy = 72.81%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%
```

```
Using L2 distance metric, k = 2
Accuracy = 19.89%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L2 distance metric, k = 3
Accuracy = 20.580000000000002%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L2 distance metric, k = 4
Accuracy = 20.36%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L2 distance metric, k = 10
Accuracy = 18.19%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%

-----------------------------------------------

Using L2 distance metric, k = 20
Accuracy = 16.24%
目前最好的組合為: l1 + k=4, 其準確率為: 73.71%
```

最後將 kv 值改為 4，dm 改為 L1，得到最佳解

```
# -----You may change values here-----
kv = 4
dm = 'l1'
# ----------------------------------------


num_test_data = test_data.shape[0]
preds = classifier.predict(test_data, k=kv, dist_metric=dm)
num_correct = np.sum(preds == test_label)
```

```
accuracy = float(num_correct) / num_test_data
print(f'Using {dm.upper()} distance metric, k = {kv}\nAccuracy =
{accuracy}')
```

執行結果：

```
Using L1 distance metric, k = 4
Accuracy = 0.7371
```