資料預處理步驟：

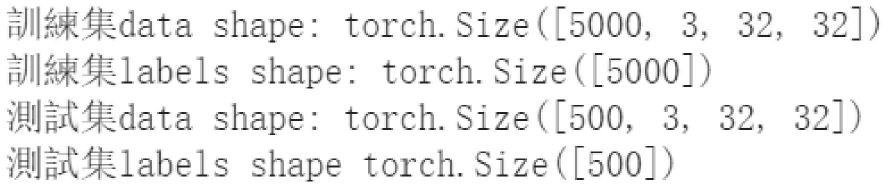
使用 data.py 裡面的 cifar10 分別準備 x_train, y_train, x_test, y_test 四個資料集(subsampled)，其中：

x_train：訓練集內容，設定為 5000 筆資料

y_train：訓練集標籤內容，設定為 5000 筆資料

x_test：測試集內容，設定為 500 筆資料

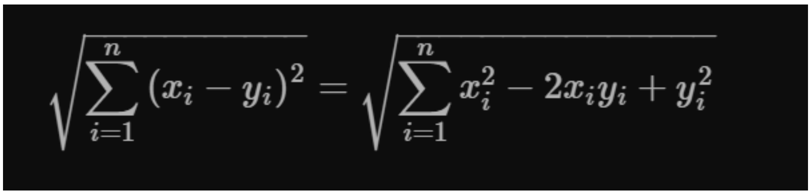
y_test：測試集標籤內容，設定為 500 筆資料



```
訓練集data shape: torch.Size([5000, 3, 32, 32])
訓練集labels shape: torch.Size([5000])
測試集data shape: torch.Size([500, 3, 32, 32])
測試集labels shape torch.Size([500])
```

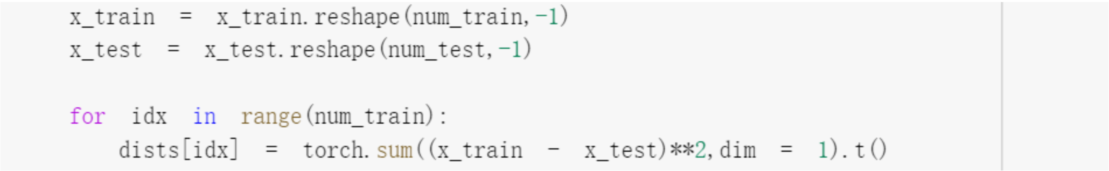
1. compute_distances_two_loops(x_train, x_test)

說明：KNN 利用 Euclidean distance，公式如下：



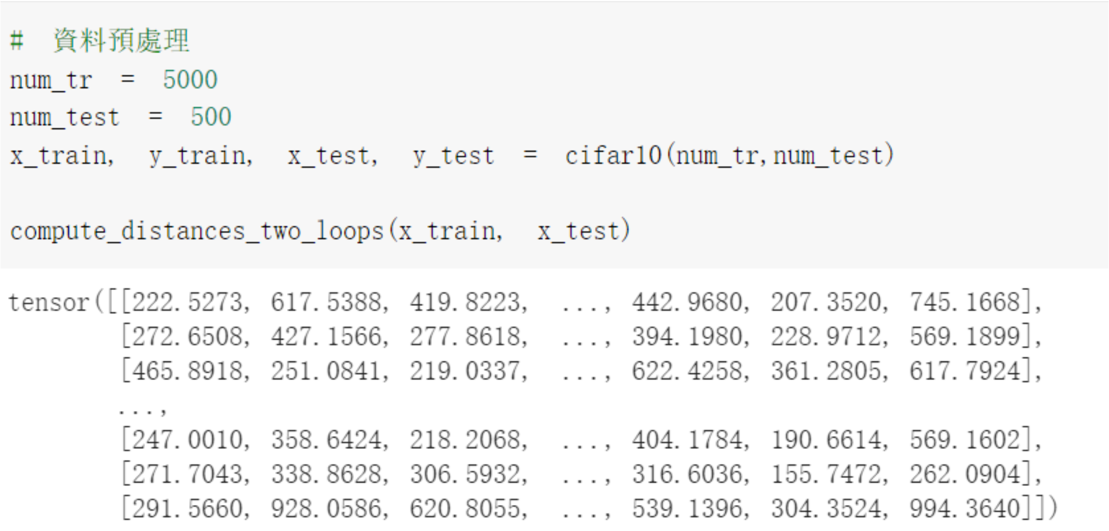
$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} = \sqrt{\sum_{i=1}^{n} x_i^2 - 2x_iy_i + y_i^2}$$

使用兩個迴圈搭配 sum()、torch.square()去手刻公式，計算 squared Euclidean distance



```
x_train = x_train.reshape(num_train,-1)
x_test = x_test.reshape(num_test,-1)

for idx in range(num_train):
    dists[idx] = torch.sum((x_train - x_test)**2,dim = 1).t()
```

答案：

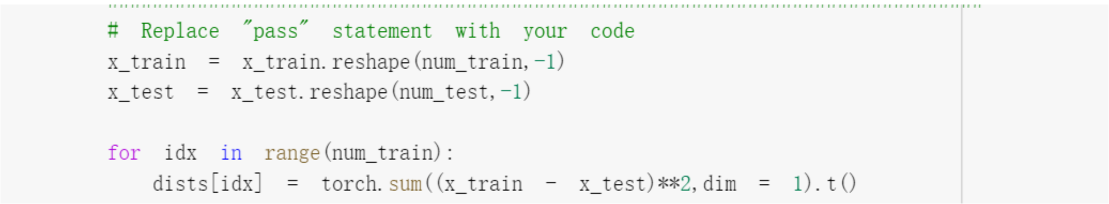


```
# 資料預處理
num_tr = 5000
num_test = 500
x_train, y_train, x_test, y_test = cifar10(num_tr,num_test)

compute_distances_two_loops(x_train, x_test)
```

```
tensor([[222.5273, 617.5388, 419.8223,  ..., 442.9680, 207.3520, 745.1668],
        [272.6508, 427.1566, 277.8618,  ..., 394.1980, 228.9712, 569.1899],
        [465.8918, 251.0841, 219.0337,  ..., 622.4258, 361.2805, 617.7924],
        ...,
        [247.0010, 358.6424, 218.2068,  ..., 404.1784, 190.6614, 569.1602],
        [271.7043, 338.8628, 306.5932,  ..., 316.6036, 155.7472, 262.0904],
        [291.5660, 928.0586, 620.8055,  ..., 539.1396, 304.3524, 994.3640]])
```

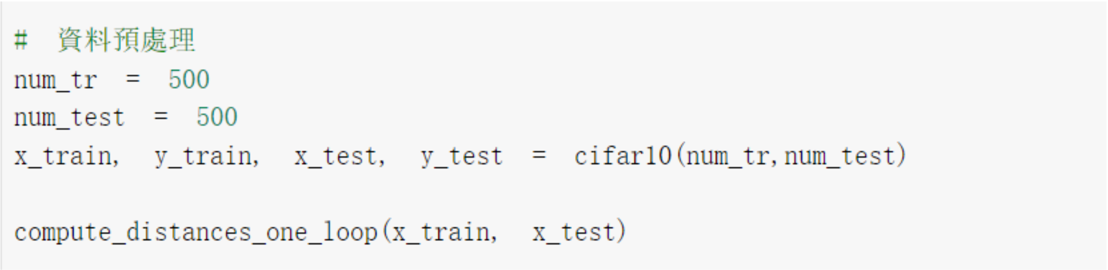
2. compute_distances_one_loop(x_train, x_test)

說明：將兩個資料集都設為 500 筆，先將 x_train, x_test 分別用 reshape()操作，再用一個迴圈依公式手刻。

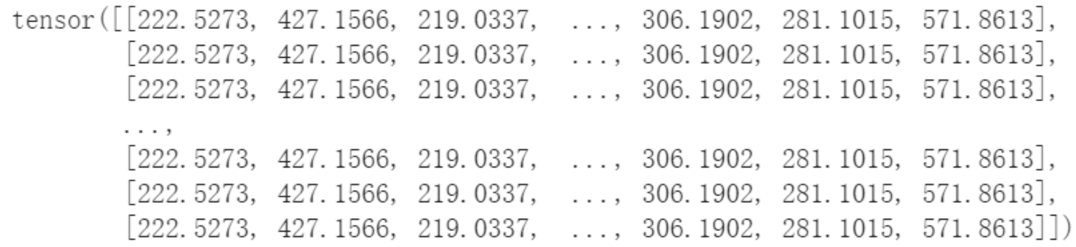


```
        # Replace "pass" statement with your code
        x_train = x_train.reshape(num_train,-1)
        x_test = x_test.reshape(num_test,-1)

        for idx in range(num_train):
            dists[idx] = torch.sum((x_train - x_test)**2,dim = 1).t()
```

答案：



```
# 資料預處理
num_tr = 500
num_test = 500
x_train, y_train, x_test, y_test = cifar10(num_tr,num_test)

compute_distances_one_loop(x_train, x_test)
```

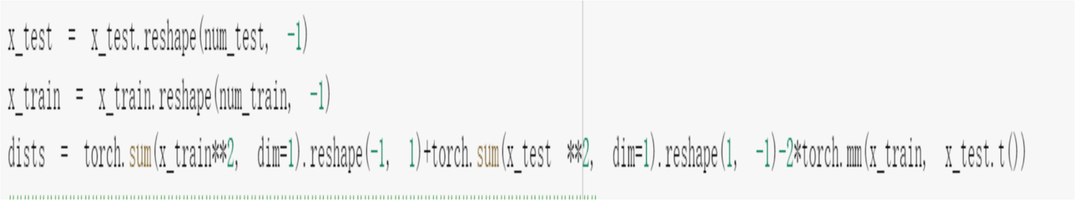


```
tensor([[222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613],
        [222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613],
        [222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613],
        ...,
        [222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613],
        [222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613],
        [222.5273, 427.1566, 219.0337,  ..., 306.1902, 281.1015, 571.8613]])
```

3. compute_distances_no_loops(x_train, x_test)

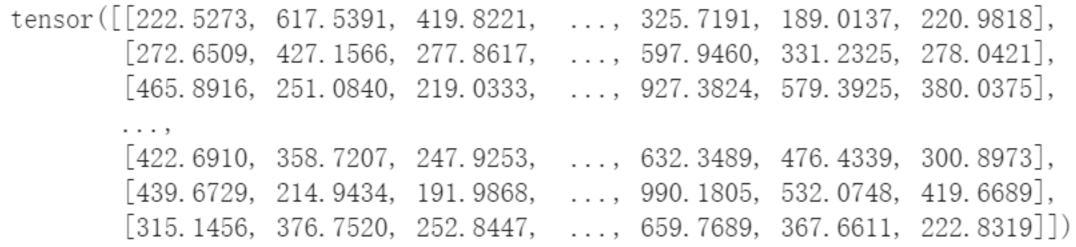
說明：

根據上述公式對等的右式，使用 sum()、reshape()、torch.mm()去手刻公式

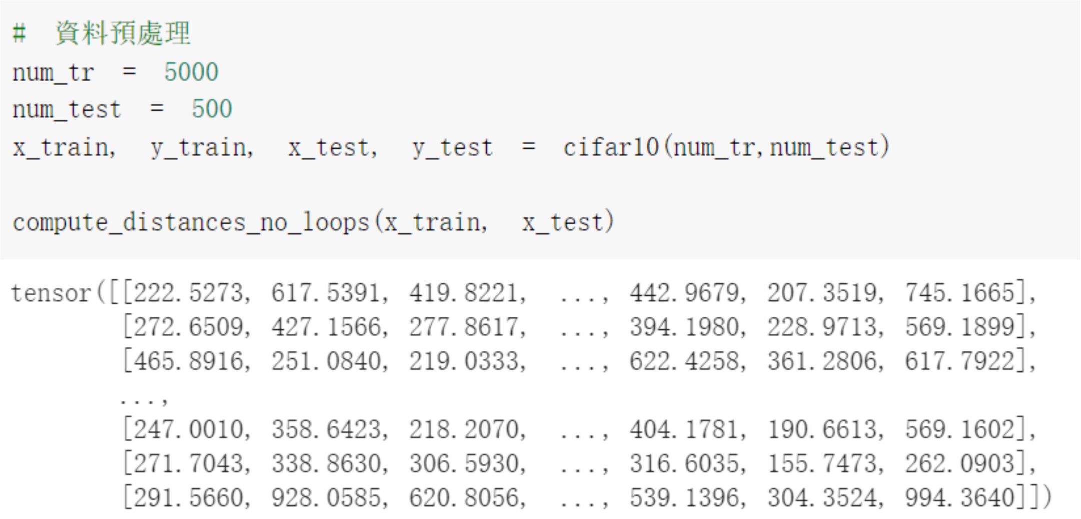


```
x_test = x_test.reshape(num_test, -1)
x_train = x_train.reshape(num_train, -1)
dists = torch.sum(x_train**2, dim=1).reshape(-1, 1)+torch.sum(x_test **2, dim=1).reshape(1, -1)-2*torch.mm(x_train, x_test.t())
```

答案：



```
tensor([[222.5273, 617.5391, 419.8221,  ..., 325.7191, 189.0137, 220.9818],
        [272.6509, 427.1566, 277.8617,  ..., 597.9460, 331.2325, 278.0421],
        [465.8916, 251.0840, 219.0333,  ..., 927.3824, 579.3925, 380.0375],
        ...,
        [422.6910, 358.7207, 247.9253,  ..., 632.3489, 476.4339, 300.8973],
        [439.6729, 214.9434, 191.9868,  ..., 990.1805, 532.0748, 419.6689],
        [315.1456, 376.7520, 252.8447,  ..., 659.7689, 367.6611, 222.8319]])
```



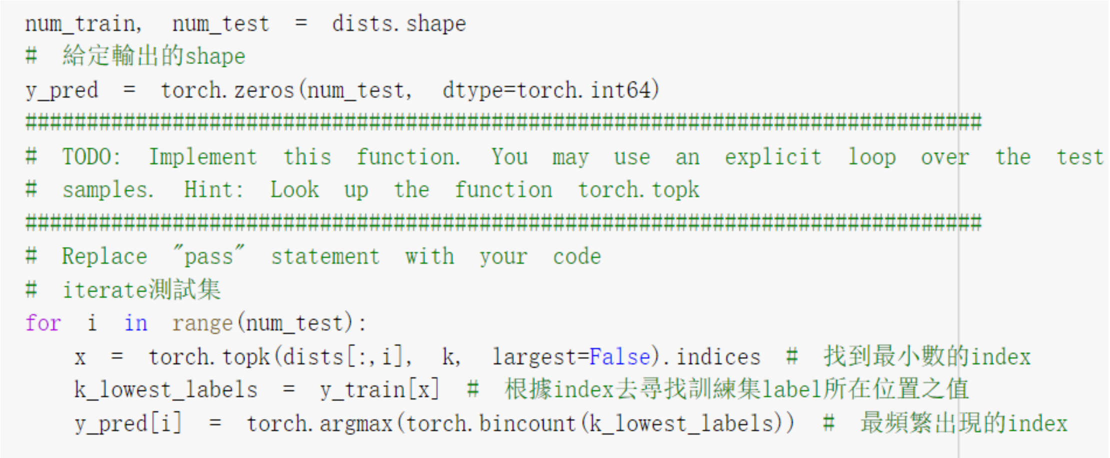
```
# 資料預處理
num_tr = 5000
num_test = 500
x_train, y_train, x_test, y_test = cifar10(num_tr,num_test)

compute_distances_no_loops(x_train, x_test)
```

```
tensor([[222.5273, 617.5391, 419.8221,  ..., 442.9679, 207.3519, 745.1665],
        [272.6509, 427.1566, 277.8617,  ..., 394.1980, 228.9713, 569.1899],
        [465.8916, 251.0840, 219.0333,  ..., 622.4258, 361.2806, 617.7922],
        ...,
        [247.0010, 358.6423, 218.2070,  ..., 404.1781, 190.6613, 569.1602],
        [271.7043, 338.8630, 306.5930,  ..., 316.6035, 155.7473, 262.0903],
        [291.5660, 928.0585, 620.8056,  ..., 539.1396, 304.3524, 994.3640]])
```

4. predict_labels(dists, y_train, k=1)

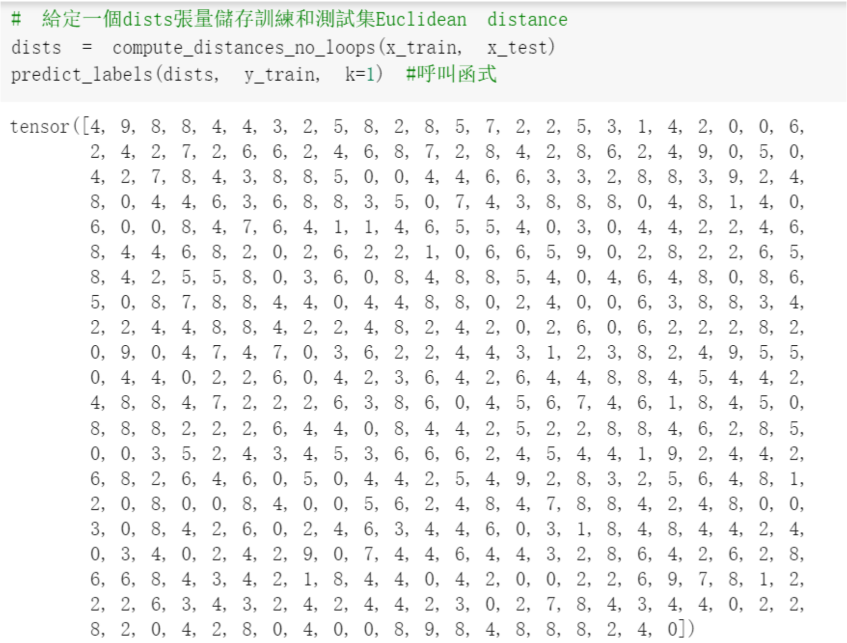
說明：

給定輸出的 shape->使用一個迴圈 iterate 測試集->找到最小數的 index->根據 index 去尋找訓練集 label 所在位置之值->將最頻繁出現的 index 給予 y_pred[i]



```
num_train, num_test = dists.shape
# 給定輸出的shape
y_pred = torch.zeros(num_test, dtype=torch.int64)
##############################################################################
# TODO: Implement this function. You may use an explicit loop over the test
# samples. Hint: Look up the function torch.topk
##############################################################################
# Replace "pass" statement with your code
# iterate測試集
for i in range(num_test):
    x = torch.topk(dists[:,i], k, largest=False).indices  # 找到最小數的index
    k_lowest_labels = y_train[x]  # 根據index去尋找訓練集label所在位置之值
    y_pred[i] = torch.argmax(torch.bincount(k_lowest_labels))  # 最頻繁出現的index
```

答案：



```
# 給定一個dists張量儲存訓練和測試集Euclidean distance
dists = compute_distances_no_loops(x_train, x_test)
predict_labels(dists, y_train, k=1) #呼叫函式
```

```
tensor([4, 9, 8, 8, 4, 4, 3, 2, 5, 8, 2, 8, 5, 7, 2, 2, 5, 3, 1, 4, 2, 0, 0, 6,
        2, 4, 2, 7, 2, 6, 6, 2, 4, 6, 8, 7, 2, 8, 4, 2, 8, 6, 2, 4, 9, 0, 5, 0,
        4, 2, 7, 8, 4, 3, 8, 8, 5, 0, 0, 4, 4, 6, 6, 3, 3, 2, 8, 8, 3, 9, 2, 4,
        8, 0, 4, 4, 6, 3, 6, 8, 8, 3, 5, 0, 7, 4, 3, 8, 8, 8, 0, 4, 8, 1, 4, 0,
        6, 0, 0, 8, 4, 7, 6, 4, 1, 1, 4, 6, 5, 5, 4, 0, 3, 0, 4, 4, 2, 2, 4, 6,
        8, 4, 4, 6, 8, 2, 0, 2, 6, 2, 2, 1, 0, 6, 6, 5, 9, 0, 2, 8, 2, 2, 6, 5,
        8, 4, 2, 5, 5, 8, 0, 3, 6, 0, 8, 4, 8, 8, 5, 4, 0, 4, 6, 4, 8, 0, 8, 6,
        5, 0, 8, 7, 8, 8, 4, 4, 0, 4, 4, 8, 8, 0, 2, 4, 0, 0, 6, 3, 8, 8, 3, 4,
        2, 2, 4, 4, 8, 8, 4, 2, 2, 4, 8, 2, 4, 2, 0, 2, 6, 0, 6, 2, 2, 2, 8, 2,
        0, 9, 0, 4, 7, 4, 7, 0, 3, 6, 2, 2, 4, 4, 3, 1, 2, 3, 8, 2, 4, 9, 5, 5,
        0, 4, 4, 0, 2, 2, 6, 0, 4, 2, 3, 6, 4, 2, 6, 4, 4, 8, 8, 4, 5, 4, 4, 2,
        4, 8, 8, 4, 7, 2, 2, 2, 6, 3, 8, 6, 0, 4, 5, 6, 7, 4, 6, 1, 8, 4, 5, 0,
        8, 8, 8, 2, 2, 2, 6, 4, 4, 0, 8, 4, 4, 2, 5, 2, 2, 8, 8, 4, 6, 2, 8, 5,
        0, 0, 3, 5, 2, 4, 3, 4, 5, 3, 6, 6, 6, 2, 4, 5, 4, 4, 1, 9, 2, 4, 4, 2,
        6, 8, 2, 6, 4, 6, 0, 5, 0, 4, 4, 2, 5, 4, 9, 2, 8, 3, 2, 5, 6, 4, 8, 1,
        2, 0, 8, 0, 0, 8, 4, 0, 0, 5, 6, 2, 4, 8, 4, 7, 8, 8, 4, 2, 4, 8, 0, 0,
        3, 0, 8, 4, 2, 6, 0, 2, 4, 6, 3, 4, 4, 6, 0, 3, 1, 8, 4, 8, 4, 4, 2, 4,
        0, 3, 4, 0, 2, 4, 2, 9, 0, 7, 4, 4, 6, 4, 4, 3, 2, 8, 6, 4, 2, 6, 2, 8,
        6, 6, 8, 4, 3, 4, 2, 1, 8, 4, 4, 0, 4, 2, 0, 0, 2, 2, 6, 9, 7, 8, 1, 2,
        2, 2, 6, 3, 4, 3, 2, 4, 2, 4, 4, 2, 3, 0, 2, 7, 8, 4, 3, 4, 4, 0, 2, 2,
        8, 2, 0, 4, 2, 8, 0, 4, 0, 0, 8, 9, 8, 4, 8, 8, 8, 2, 4, 0])
```

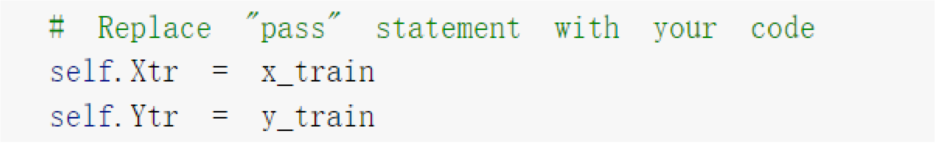
## 5.	KNN classifier

說明：內含__init__、predict、check_accuracy 三種方法

答案：實作如下：

__init__：

初始動作，將 x_train 和 y_train 分別指定給 self.Xtr, self.Ytr



```
# Replace "pass" statement with your code
self.Xtr = x_train
self.Ytr = y_train
```

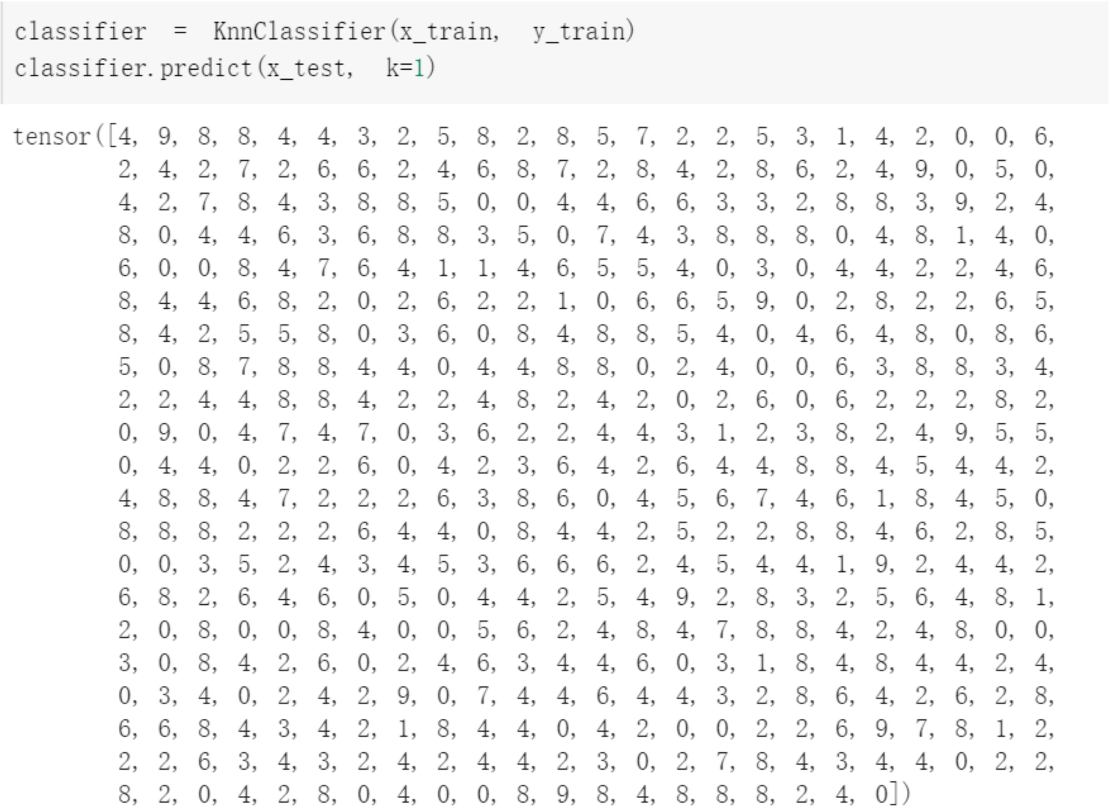
Predict：

說明：

The goal is to return a tensor y_test_pred where the ith index is the assigned label to ith test image by the KNN algorithm.

先使用上面的計算方法計算出歐式距離存放在 dists 中，再呼叫 predict_labels(dists, self.Ytr, k=k)，最後回傳 y_test_pred

答案：



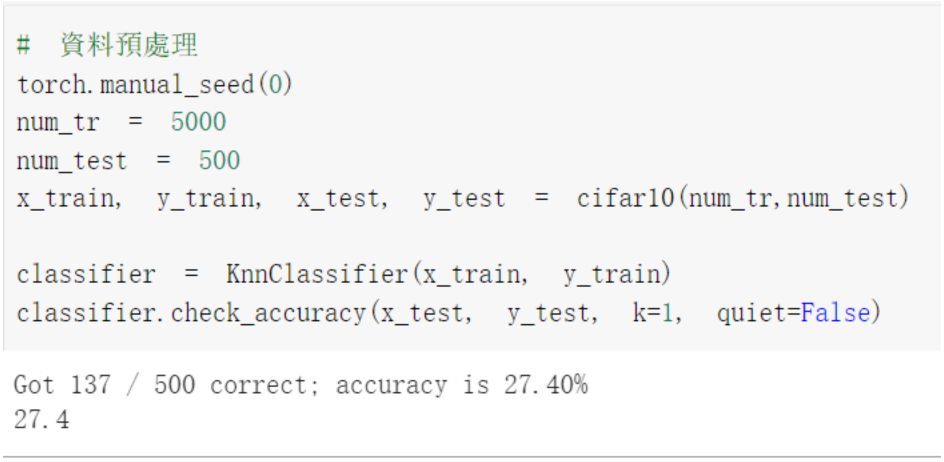
```
classifier = KnnClassifier(x_train, y_train)
classifier.predict(x_test, k=1)
```

```
tensor([4, 9, 8, 8, 4, 4, 3, 2, 5, 8, 2, 8, 5, 7, 2, 2, 5, 3, 1, 4, 2, 0, 0, 6,
        2, 4, 2, 7, 2, 6, 6, 2, 4, 6, 8, 7, 2, 8, 4, 2, 8, 6, 2, 4, 9, 0, 5, 0,
        4, 2, 7, 8, 4, 3, 8, 8, 5, 0, 0, 4, 4, 6, 6, 3, 3, 2, 8, 8, 3, 9, 2, 4,
        8, 0, 4, 4, 6, 3, 6, 8, 8, 3, 5, 0, 7, 4, 3, 8, 8, 8, 0, 4, 8, 1, 4, 0,
        6, 0, 0, 8, 4, 7, 6, 4, 1, 1, 4, 6, 5, 5, 4, 0, 3, 0, 4, 4, 2, 2, 4, 6,
        8, 4, 4, 6, 8, 2, 0, 2, 6, 2, 2, 1, 0, 6, 6, 5, 9, 0, 2, 8, 2, 2, 6, 5,
        8, 4, 2, 5, 5, 8, 0, 3, 6, 0, 8, 4, 8, 8, 5, 4, 0, 4, 6, 4, 8, 0, 8, 6,
        5, 0, 8, 7, 8, 8, 4, 4, 0, 4, 4, 8, 8, 0, 2, 4, 0, 0, 6, 3, 8, 8, 3, 4,
        2, 2, 4, 4, 8, 8, 4, 2, 2, 4, 8, 2, 4, 2, 0, 2, 6, 0, 6, 2, 2, 2, 8, 2,
        0, 9, 0, 4, 7, 4, 7, 0, 3, 6, 2, 2, 4, 4, 3, 1, 2, 3, 8, 2, 4, 9, 5, 5,
        0, 4, 4, 0, 2, 2, 6, 0, 4, 2, 3, 6, 4, 2, 6, 4, 4, 8, 8, 4, 5, 4, 4, 2,
        4, 8, 8, 4, 7, 2, 2, 2, 6, 3, 8, 6, 0, 4, 5, 6, 7, 4, 6, 1, 8, 4, 5, 0,
        8, 8, 8, 2, 2, 2, 6, 4, 4, 0, 8, 4, 4, 2, 5, 2, 2, 8, 8, 4, 6, 2, 8, 5,
        0, 0, 3, 5, 2, 4, 3, 4, 5, 3, 6, 6, 6, 2, 4, 5, 4, 4, 1, 9, 2, 4, 4, 2,
        6, 8, 2, 6, 4, 6, 0, 5, 0, 4, 4, 2, 5, 4, 9, 2, 8, 3, 2, 5, 6, 4, 8, 1,
        2, 0, 8, 0, 0, 8, 4, 0, 0, 5, 6, 2, 4, 8, 4, 7, 8, 8, 4, 2, 4, 8, 0, 0,
        3, 0, 8, 4, 2, 6, 0, 2, 4, 6, 3, 4, 4, 6, 0, 3, 1, 8, 4, 8, 4, 4, 2, 4,
        0, 3, 4, 0, 2, 4, 2, 9, 0, 7, 4, 4, 6, 4, 4, 3, 2, 8, 6, 4, 2, 6, 2, 8,
        6, 6, 8, 4, 3, 4, 2, 1, 8, 4, 4, 0, 4, 2, 0, 0, 2, 2, 6, 9, 7, 8, 1, 2,
        2, 2, 6, 3, 4, 3, 2, 4, 2, 4, 4, 2, 3, 0, 2, 7, 8, 4, 3, 4, 4, 0, 2, 2,
        8, 2, 0, 4, 2, 8, 0, 4, 0, 0, 8, 9, 8, 4, 8, 8, 8, 2, 4, 0])
```

Check_accuracy：

說明：評估 KNN 表現，有使用 manual_seed 生成隨機數的種子，以便重現結果

答案：

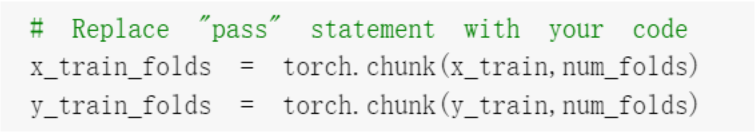


```
# 資料預處理
torch.manual_seed(0)
num_tr = 5000
num_test = 500
x_train, y_train, x_test, y_test = cifar10(num_tr,num_test)

classifier = KnnClassifier(x_train, y_train)
classifier.check_accuracy(x_test, y_test, k=1, quiet=False)
```

```
Got 137 / 500 correct; accuracy is 27.40%
27.4
```

6. KNN 交叉驗證

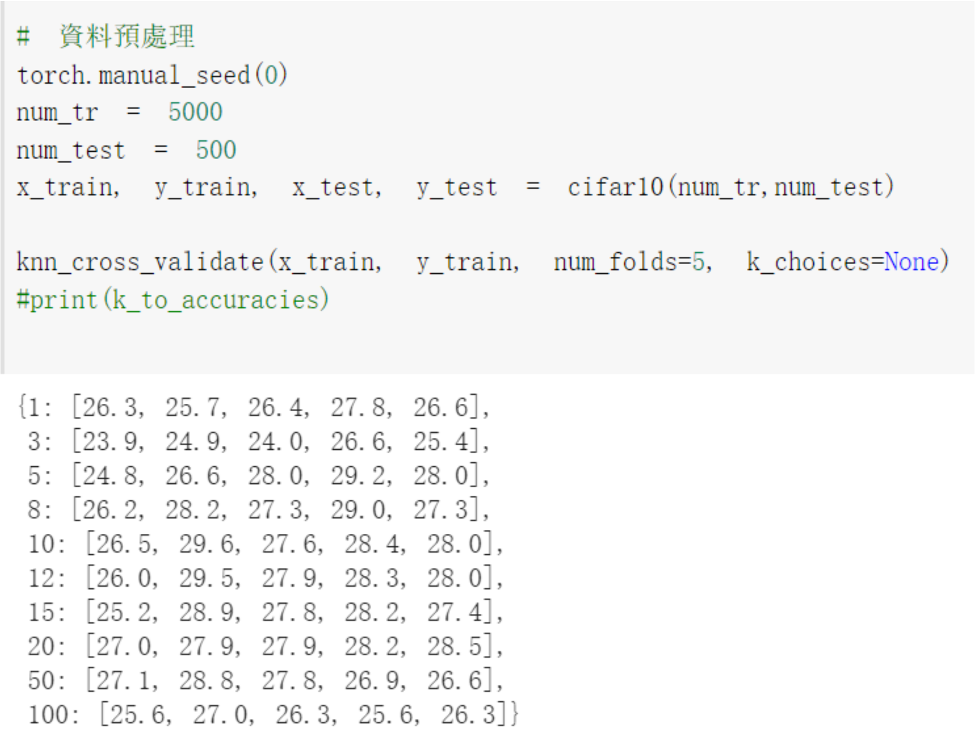
說明：將資料集切成 5 個 chunks，將每個 chunk 都當成 validation set 去驗證，以此來避免 overfitting 的問題。



```
# Replace "pass" statement with your code
x_train_folds = torch.chunk(x_train,num_folds)
y_train_folds = torch.chunk(y_train,num_folds)
```

答案：

不同的 k 值設定下所得到的正確率也不同



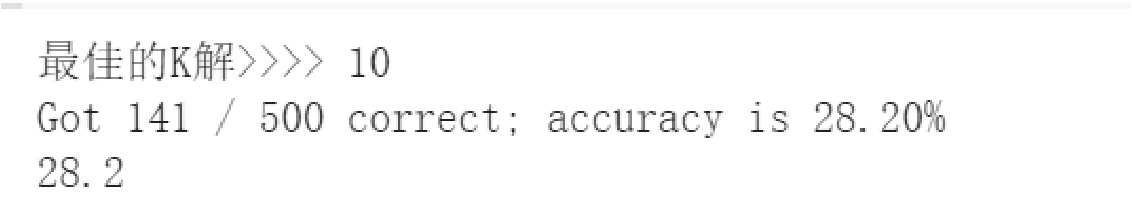
```
#  資料預處理
torch.manual_seed(0)
num_tr  =  5000
num_test  =  500
x_train,  y_train,  x_test,  y_test  =  cifar10(num_tr,num_test)

knn_cross_validate(x_train,  y_train,  num_folds=5,  k_choices=None)
#print(k_to_accuracies)
```

```
{1: [26.3, 25.7, 26.4, 27.8, 26.6],
 3: [23.9, 24.9, 24.0, 26.6, 25.4],
 5: [24.8, 26.6, 28.0, 29.2, 28.0],
 8: [26.2, 28.2, 27.3, 29.0, 27.3],
 10: [26.5, 29.6, 27.6, 28.4, 28.0],
 12: [26.0, 29.5, 27.9, 28.3, 28.0],
 15: [25.2, 28.9, 27.8, 28.2, 27.4],
 20: [27.0, 27.9, 27.9, 28.2, 28.5],
 50: [27.1, 28.8, 27.8, 26.9, 26.6],
 100: [25.6, 27.0, 26.3, 25.6, 26.3]}
```

7. Best K

說明：由交叉驗證可得知，若 k 值設定不同，則訓練正確率也會不同。正確率也並非與 k 值呈現正比，因此使用 best k 從交叉驗證得出的 list 中排序後對正確率分別取平均，就可得知哪一 K 值下，模型表現最好。

答案：



```
最佳的K解>>>> 10
Got 141 / 500 correct; accuracy is 28.20%
28.2
```

參考資料：https://ryli.design/blog/knn