深度學習

HW2

學號 : B103012002

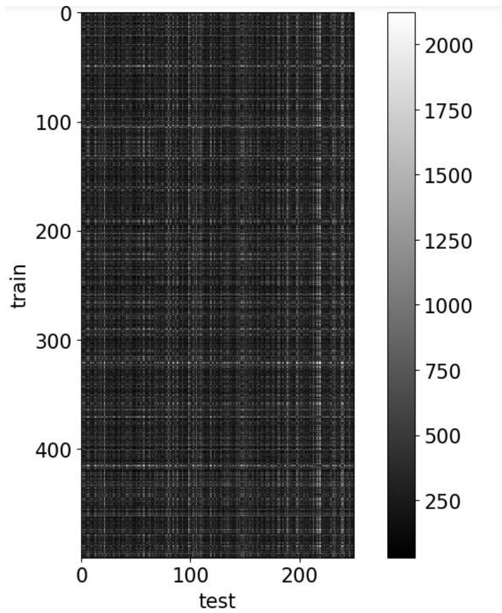
姓名 ： 林凡皓

I. Compute distance : Naïve implementation

- compute_distances_two_loops :

  1. 解題思路 ： 先透過 x.view( )將輸入的兩個矩陣變成二為矩陣，再透過兩個 for loop 計算每個訓練資料與測試資料的歐式距離。

  2. 執行結果 ：



II. Compute distances : vectorization

- compute_distances_one_loop :

  1. 解題思路 ： 先透過 x.view( )將輸入的兩個矩陣變成二為矩陣，透過一個 for loop 迭代訓練資料以及 broadcast 來符合測試資料。

  2. 執行結果 ：

```
Difference:  0.0
Good! The distance matrices match
```

- compute_distances_no_loop :

  1. 解題思路 ： 先透過 x.view( )將輸入的兩個矩陣變成二為矩陣，並利用(x - y)^2 = x^2 + y^2 – dot(x, y)的概念，分別計算出 x^2、y^2、dot(x, y)，即可完成本題。

  2. 執行結果 ：

```
Difference:  1.8969015959204817e-11
Good! The distance matrices match
```

- Two_loops v.s. one_loop v.s. no_loops :

```
Two loop version took 3.16 seconds
One loop version took 0.23 seconds (14.0X speedup)
No loop version took 0.00 seconds (700.2X speedup)
```

由結果可以看出，一個迴圈會比兩個迴圈快上 14 倍左右，而不使用迴圈會比一個迴圈快上 700 倍左右。

III. Predict labels

- predict_labels :

1. 解題思路 ：透過 torch.topk( )取得最近的 k 個鄰居的 index，並透過 index 取得 label。接著計算每個 label 出現的次數並取出現最多的 label。

2. 執行結果 ：

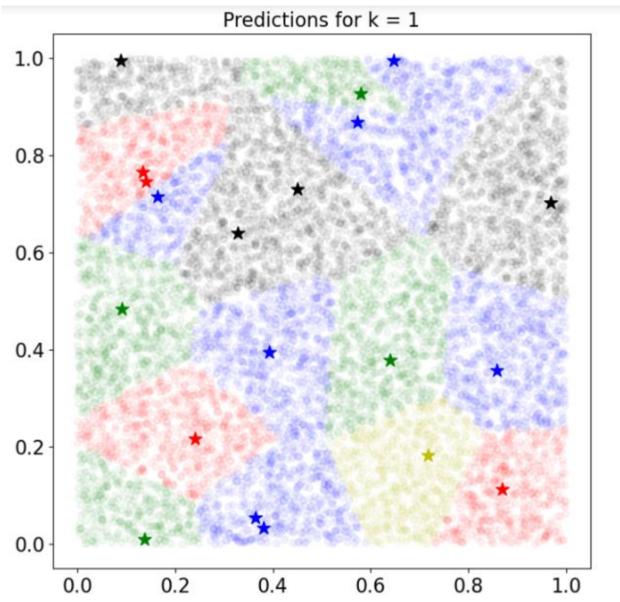
```
Correct:  True
```

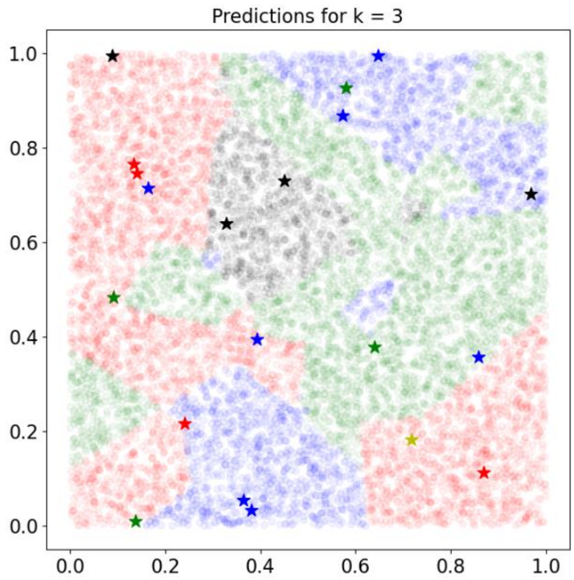
3. 額外討論 ：torch.topk( )傳入三個參數，第一個為 dists，第二個為 k，第三個是 largest，也就是說要在 dists 這個 tensor 中找到 k 個值，如果 largest=False，就會找最小值。

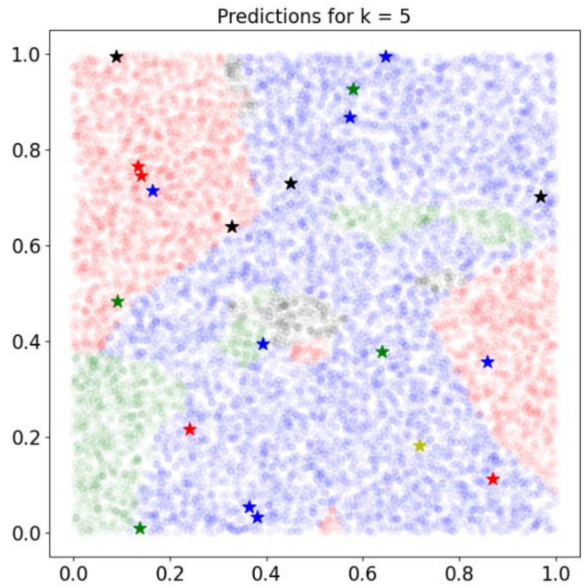
- KnnClassifier :

1. 解題思路 ：這一部分要定義一個 KnnClassifier 的 class。一開始要對 KnnClassifier 做初始化，接著定義 predict 函數。predict 函數只需要先呼 compute_distances_no_loops 計算出與其他樣本的距離，接著用 predict_labels 取得最近的距離即可。

2. 執行結果 ：







由以上三張圖可以看出，當 k 逐漸變大，切割不同類別的邊界會變滑順，且切割出來的區塊也會減少。

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

由上圖可以得知，這個簡單的 Knn model(k = 1)大約有 27.4 %的準確度，比一般人類預測的準確度(10 %)還要高。

```
Got 139 / 500 correct; accuracy is 27.80%
27.8
```

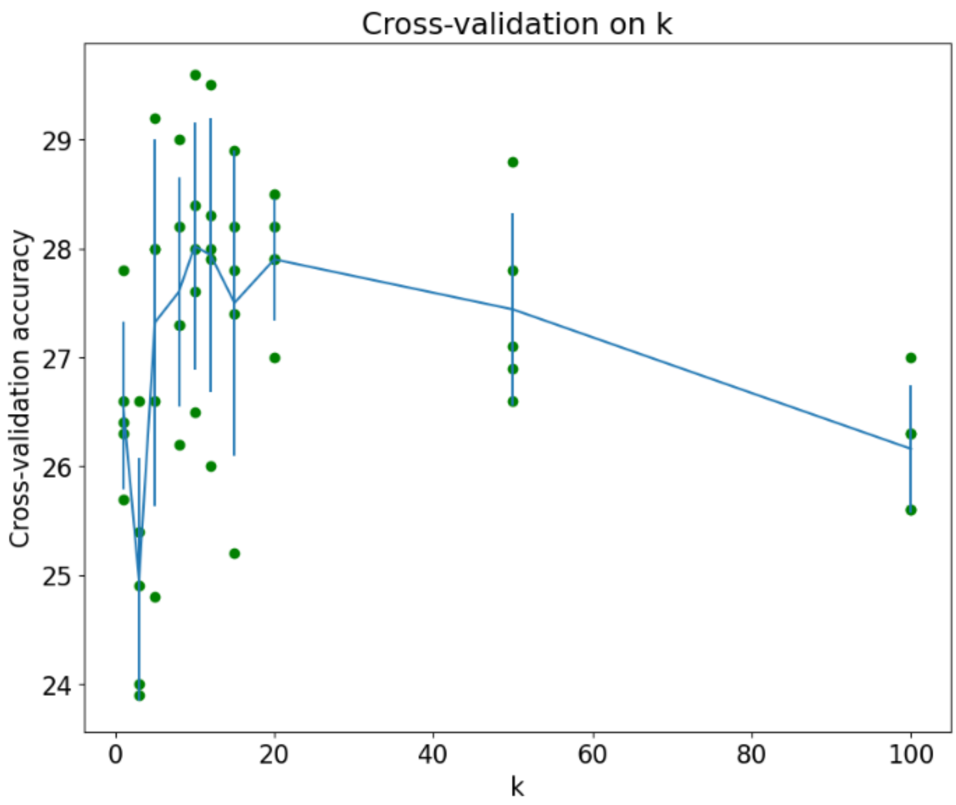
當 k=5 時，準確度稍微提升一點，但是進步很少。

IV. Cross-validation

- knn_cross_validate :

  1. 解題思路 ： 先利用 chunk 將訓練資料與訓練標籤分割成指定 fold 數目，並將結果存成 list。接著利用 for loop 走過 k_choices 中每一個 k 值，對於每一個 k 值，利用 for loop 讓分割後的訓練資料與訓練標籤中的每一個 fold 皆有機會當 validation set，可以透過每一次都選擇 list 的最後一個元素，結束驗證後將被選中的元素放到 list 的第一個位置。
  2. 執行結果 ：

```
Got 266 / 1000 correct; accuracy is 26.60%
Got 278 / 1000 correct; accuracy is 27.80%
Got 264 / 1000 correct; accuracy is 26.40%
Got 257 / 1000 correct; accuracy is 25.70%
Got 263 / 1000 correct; accuracy is 26.30%
Got 254 / 1000 correct; accuracy is 25.40%
Got 266 / 1000 correct; accuracy is 26.60%
Got 240 / 1000 correct; accuracy is 24.00%
Got 249 / 1000 correct; accuracy is 24.90%
Got 239 / 1000 correct; accuracy is 23.90%
Got 280 / 1000 correct; accuracy is 28.00%
Got 292 / 1000 correct; accuracy is 29.20%
Got 280 / 1000 correct; accuracy is 28.00%
Got 266 / 1000 correct; accuracy is 26.60%
Got 248 / 1000 correct; accuracy is 24.80%
Got 273 / 1000 correct; accuracy is 27.30%
Got 290 / 1000 correct; accuracy is 29.00%
Got 273 / 1000 correct; accuracy is 27.30%
Got 282 / 1000 correct; accuracy is 28.20%
Got 262 / 1000 correct; accuracy is 26.20%
Got 280 / 1000 correct; accuracy is 28.00%
Got 284 / 1000 correct; accuracy is 28.40%
Got 276 / 1000 correct; accuracy is 27.60%
Got 296 / 1000 correct; accuracy is 29.60%
Got 265 / 1000 correct; accuracy is 26.50%
Got 280 / 1000 correct; accuracy is 28.00%
Got 283 / 1000 correct; accuracy is 28.30%
Got 279 / 1000 correct; accuracy is 27.90%
Got 295 / 1000 correct; accuracy is 29.50%
Got 260 / 1000 correct; accuracy is 26.00%
Got 274 / 1000 correct; accuracy is 27.40%
Got 282 / 1000 correct; accuracy is 28.20%
Got 278 / 1000 correct; accuracy is 27.80%
Got 289 / 1000 correct; accuracy is 28.90%
Got 252 / 1000 correct; accuracy is 25.20%
Got 285 / 1000 correct; accuracy is 28.50%
Got 282 / 1000 correct; accuracy is 28.20%
Got 279 / 1000 correct; accuracy is 27.90%
Got 279 / 1000 correct; accuracy is 27.90%
Got 270 / 1000 correct; accuracy is 27.00%
Got 266 / 1000 correct; accuracy is 26.60%
Got 269 / 1000 correct; accuracy is 26.90%
Got 278 / 1000 correct; accuracy is 27.80%
Got 288 / 1000 correct; accuracy is 28.80%
Got 271 / 1000 correct; accuracy is 27.10%
Got 263 / 1000 correct; accuracy is 26.30%
Got 256 / 1000 correct; accuracy is 25.60%
Got 263 / 1000 correct; accuracy is 26.30%
Got 270 / 1000 correct; accuracy is 27.00%
Got 256 / 1000 correct; accuracy is 25.60%
k = 1 got accuracies: [26.6, 27.8, 26.4, 25.7, 26.3]
k = 3 got accuracies: [25.4, 26.6, 24.0, 24.9, 23.9]
k = 5 got accuracies: [28.0, 29.2, 28.0, 26.6, 24.8]
k = 8 got accuracies: [27.3, 29.0, 27.3, 28.2, 26.2]
k = 10 got accuracies: [28.0, 28.4, 27.6, 29.6, 26.5]
k = 12 got accuracies: [28.0, 28.3, 27.9, 29.5, 26.0]
k = 15 got accuracies: [27.4, 28.2, 27.8, 28.9, 25.2]
k = 20 got accuracies: [28.5, 28.2, 27.9, 27.9, 27.0]
k = 50 got accuracies: [26.6, 26.9, 27.8, 28.8, 27.1]
k = 100 got accuracies: [26.3, 25.6, 26.3, 27.0, 25.6]
```



由上圖可以看出，最佳的 k 大約落在 10，左右。

- knn_get_best_k :

  1. 解題思路 ： 先創建一個 dictionary，透過 for loop 計算上一題 k_to_accuracies 的對於每一個 k 而言的平均，並將此平均與 k 值儲存到剛才創建的 dictionary。最後透過 max( )取得擁有最大平均值的 k 值。

  2. 執行結果 ：

```
Best k is  10
Got 141 / 500 correct; accuracy is 28.20%

28.2
```

  最佳 k 值為 10，與上一題結果相同。

- 利用訓練好的模型預測整的資料集 ：

```
Got 3386 / 10000 correct; accuracy is 33.86%

33.86
```

  預測後的準確度為 33.86 %。

V. 額外嘗試

- 使用更多樣本訓練 knn :

  我嘗試使用 50000 個樣本來做訓練。

  1. k = 1 :

  ```
  Got 3539 / 10000 correct; accuracy is 35.39%
  35.39
  ```

  k = 1 時，準確度為 35.39 %。

  2. k = 5 :

  ```
  Got 3398 / 10000 correct; accuracy is 33.98%
  33.98
  ```

  k = 5 時，準確度為 33.98 %。不管 k = 1 or 5，準確度都有些微提升。

  3. cross validation :

  ```
  Best k is  1
  Got 3539 / 10000 correct; accuracy is 35.39%
  35.39
  ```

  最後得到的準確度為 35.39 %，雖然有少量資料訓練出來的模型好，但是提升幅度有限。

- 使用 ResNet50 :

  由於 KNN 做出來的準確度大約只有 30 %，因此我嘗試使用 tensorflow.keras.applications 中的 ResNet50 來訓練。

  1. ResNet50 : 採用深度殘差學習的思想，透過引入跳要連接來建構模型，使訓練更加容易。由於 ResNet50 使用卷積層來提取圖像特徵，因此我認為將它用在本次作業是不錯的選擇。

  2. 訓練結果 :

  ```
  Epoch 17/20
  391/391 [==============================] - 35s 88ms/step - loss: 0.4301 - accuracy: 0.8455 - val_loss: 1.2071 - val_accuracy: 0.6802
  Epoch 18/20
  391/391 [==============================] - 35s 90ms/step - loss: 0.4186 - accuracy: 0.8513 - val_loss: 1.2024 - val_accuracy: 0.6852
  Epoch 19/20
  391/391 [==============================] - 36s 91ms/step - loss: 0.4012 - accuracy: 0.8558 - val_loss: 1.2452 - val_accuracy: 0.6816
  Epoch 20/20
  391/391 [==============================] - 35s 90ms/step - loss: 0.3835 - accuracy: 0.8621 - val_loss: 1.2916 - val_accuracy: 0.6779
  ```

  最終 validation accuracy 大約停留在 68 %左右便無法再上升，此準確度比 KNN 高出許多。

- 使用 CNN：

CNN 為影像辨識常用的深度學習架構，因此本次作業我也嘗試建立一個簡單的 CNN 來做比較。

CNN 架構如下 ：

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 29, 29, 64)        3136

 conv2d_17 (Conv2D)          (None, 26, 26, 64)        65600

 max_pooling2d_8 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 dropout_8 (Dropout)         (None, 13, 13, 64)        0

 conv2d_18 (Conv2D)          (None, 10, 10, 128)       131200

 conv2d_19 (Conv2D)          (None, 7, 7, 128)         262272

 max_pooling2d_9 (MaxPoolin  (None, 3, 3, 128)         0
 g2D)

 dropout_9 (Dropout)         (None, 3, 3, 128)         0

 flatten_4 (Flatten)         (None, 1152)              0

 dense_12 (Dense)            (None, 1024)              1180672

 dense_13 (Dense)            (None, 1024)              1049600

 dense_14 (Dense)            (None, 10)                10250

=================================================================
Total params: 2702730 (10.31 MB)
Trainable params: 2702730 (10.31 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

訓練結果如下 ：

```
Epoch 32/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6998 - accuracy: 0.7550 - val_loss: 0.7712 - val_accuracy:
0.7403
Epoch 33/40
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6899 - accuracy: 0.7597 - val_loss: 0.8295 - val_accuracy:
0.7258
Epoch 34/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6802 - accuracy: 0.7613 - val_loss: 0.8505 - val_accuracy:
0.7211
Epoch 35/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6717 - accuracy: 0.7648 - val_loss: 0.8003 - val_accuracy:
0.7291
Epoch 36/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6740 - accuracy: 0.7651 - val_loss: 0.8117 - val_accuracy:
0.7326
Epoch 37/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6624 - accuracy: 0.7687 - val_loss: 0.8366 - val_accuracy:
0.7214
Epoch 38/40
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6617 - accuracy: 0.7697 - val_loss: 0.8148 - val_accuracy:
0.7266
Epoch 39/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6574 - accuracy: 0.7742 - val_loss: 0.8149 - val_accuracy:
0.7290
Epoch 40/40
1563/1563 [==============================] - 43s 27ms/step - loss: 0.6562 - accuracy: 0.7707 - val_loss: 0.7833 - val_accuracy:
0.7386
```

最高的準確度為 74.03 %，為目前最高的模型。

- 心得：

由這次的簡單嘗試，我得到一下幾點結論：

1. 增加訓練資料可以增加模型的準確度，但是增加幅度有限，如果要求更高的準確度，需要對模型做進一步的優化或是使用其他更強大的模型。

2. 比較大的模型雖然可以有效提升準確度，但訓練時間會比較長，以這次的嘗試來說，KNN 只需要花幾秒鐘便可以訓練完成，而 CNN 卻需要花上 30 分鐘才能夠訓練完成，因此我認為選擇模型不只是要考慮到準確度，還要針對自己的需求在各個方面做取捨。

VI. Reference :

[1] Carlos Andres Polania “Transfer learning with ResNet50 from keras and CIFAR-10” https://www.linkedin.com/pulse/transfer-learning-resnet50-from-keras-cifar-10-carlos-andres-polania/?trk=article-ssr-frontend-pulse_more-articles_related-content-card

[2] Devashree Madhugiri “Using CNN for Image Classification on CIFAR-10 Dataset” https://devashree-madhugiri.medium.com/using-cnn-for-image-classification-on-cifar-10-dataset-7803d9f3b983

[3] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/