

目錄

MNIST 資料集介紹	2
1.1 資料集介紹	2
1.2 資料蒐集流程與格式內容	2
資料分析	3
2.1 數字類別分布	3
2.2 差異性：手寫數字樣本展示（Handwritten Digit Sample Display）	3
2.3 像素強度熱力圖（Pixel Intensity Heatmap）	4
機器學習演算法	5
3.1 HOG 特徵擷取介紹（Introduction to HOG Feature Extraction）	5
3.2 HOG 實際操作流程	6
3.3 支援向量機(Support Vector Machine，簡稱 SVM)	7
3.3.1 SVM 參數設定	8
實作 MNIST 手寫數字辨識	10
4.1 環境設定	10
4.2 MNIST 數字識別過程：	11
4.3 實際訓練流程	11
4.5 檢驗準確度	12
4.6 環境比較	13

MNIST 資料集介紹

1.1 資料集介紹

在 Project 報告中，我們將針對著名的手寫數字資料集 MNIST 進行資料探索與視覺化分析。MNIST 是機器學習與電腦視覺領域中最具代表性的資料集之一，廣泛用於分類模型的訓練與評估。透過本次分析，我們將深入了解資料的結構與分布，並進行特徵視覺化，為後續的機器學習模型（SVM）奠定基礎。

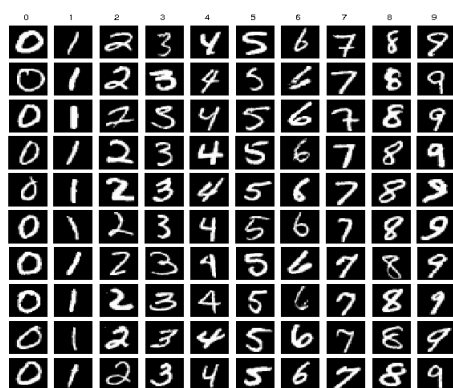


圖 1 MNIST 0~9 數字

1.2 資料蒐集流程與格式內容

本報告所使用的資料集為原始的 MNIST 手寫數字影像資料集，下載來源為 Kaggle：[MNIST Dataset - Hojjatk on Kaggle](#)

此資料集以 影像檔案格式（.bmp）儲存每張 28x28 像素的灰階手寫數字圖像，與某些轉為 CSV 格式的版本不同。本報告直接對這些圖像進行讀取與視覺化處理，保留最原始的圖像結構與像素分布。

表 1 MNIST 資料集

圖像尺寸	28 x 28 像素（共 784 個像素值）
圖像格式	灰階（像素值範圍為 0~255）
標籤類別	共 10 類（數字 0 至 9）
樣本數量	共 70,000 筆
訓練集	60,000 筆
測試集	10,000 筆

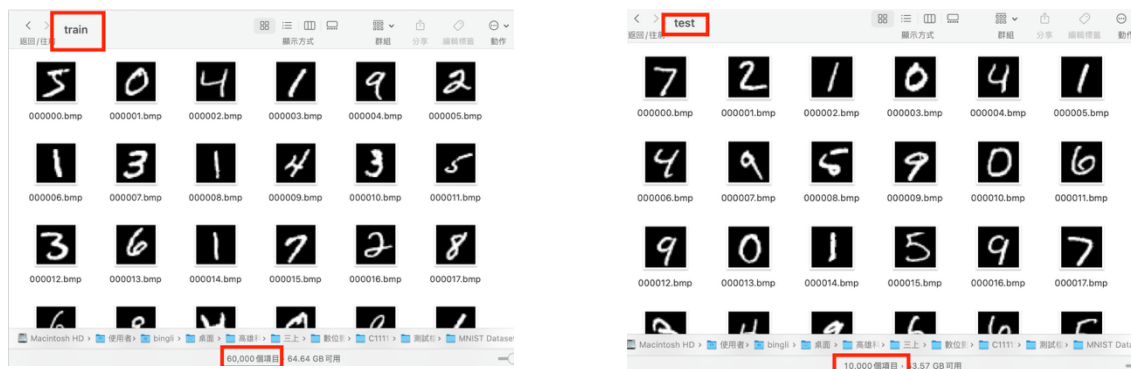


圖 2 訓練與測試集

資料分析

2.1 數字類別分布

首先統計各數字類別出現的次數。根據初步分析，各類別的樣本數分布相對平均，便於模型訓練。

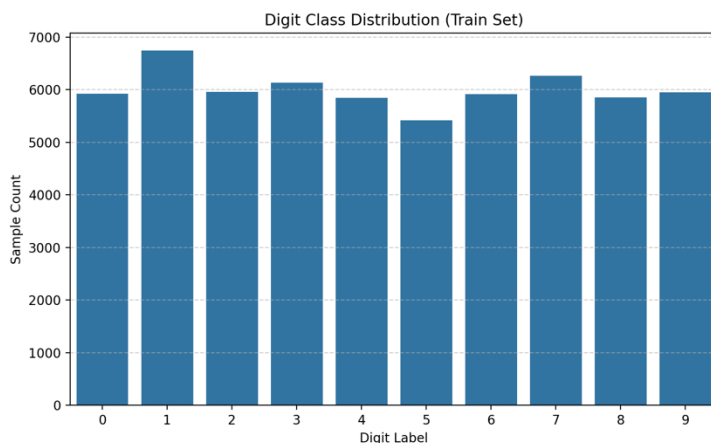


圖 3 統計 train 檔中數字（直方圖）

2.2 差異性：手寫數字樣本展示（Handwritten Digit Sample Display）

為了觀察 MNIST 資料集中各類數字的書寫風格，我們從每個類別（數字 0 到 9）中各隨機挑選三張影像進行展示。可以明顯看出，不同人的手寫筆跡存在筆劃粗細、傾斜角度與形狀上的差異，這些差異會增加分類的困難度，亦同時為模型提供學習的變異特徵。

Handwritten Digit Samples (Each class: 3 samples)

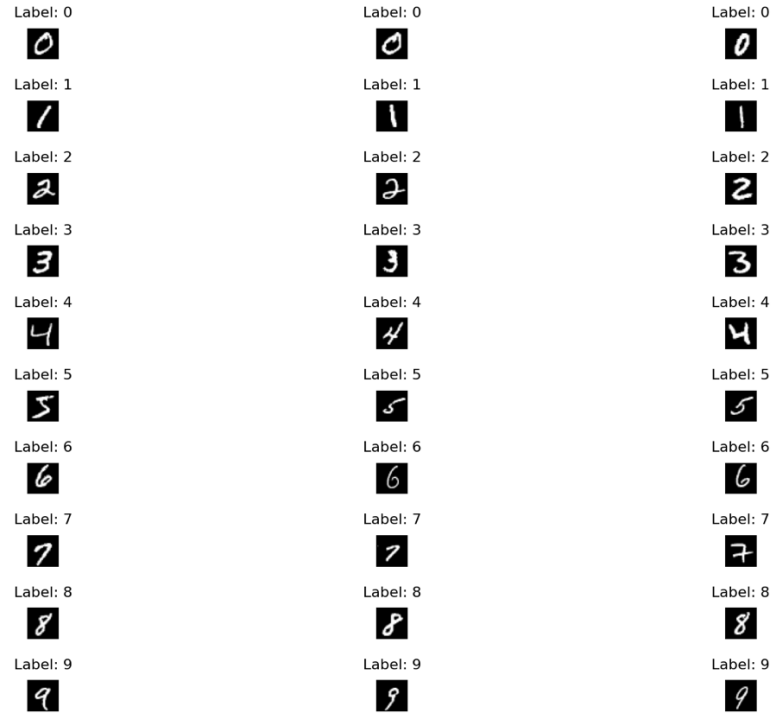


圖 5 手寫樣本展示（手寫差異性）

2.3 像素強度熱力圖（Pixel Intensity Heatmap）

為了更清楚地觀察手寫數字圖像中每個像素的強度分佈，我們將單張 MNIST 影像以兩種形式呈現：左圖為原始 28x28 像素的灰階影像，右圖則為像素強度熱力圖，並在每個像素格中標註其對應的灰階數值（範圍為 0～255）。

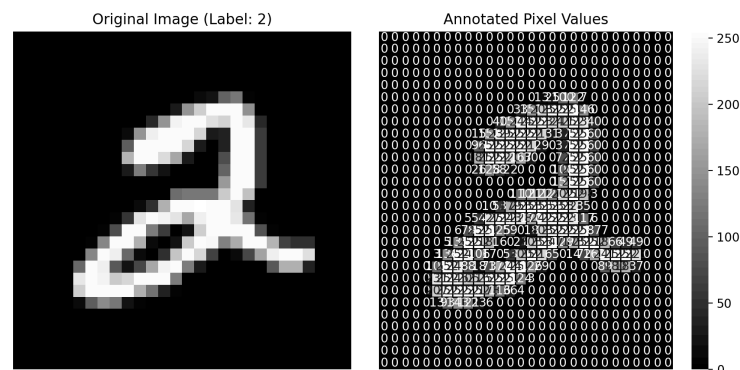


圖 6 原始圖像與像素強度熱力圖之對比

從熱力圖中可以直觀地看出數字筆劃所在位置，這些區域通常具有較高的灰階值（顏色較深或較亮），而背景則趨近於 0。這種視覺化方式不僅有助於我們

理解影像的結構，也方便後續進行特徵提取（例如 HOG）與模型設計。

機器學習演算法

3.1 HOG 特徵擷取介紹（Introduction to HOG Feature Extraction）

在前面的探索與視覺化過程中，我們已經觀察了手寫數字圖像的灰階分布與像素強度。為了讓機器能夠進一步理解圖像中的結構資訊，我們需要將圖像轉換為一組具有判別性的特徵向量。

HOG（Histogram of Oriented Gradients，方向梯度直方圖）是一種常見的圖像特徵擷取技術，能有效保留影像中的邊緣與輪廓特徵，對於辨識手寫數字具有高度的穩定性與辨識能力。

HOG 的基本原理是將影像切分成多個小區塊（cell），並計算每個區塊中像素的梯度方向與大小，再將這些方向資訊以直方圖方式記錄下來，形成特徵向量。這些向量將作為日後分類模型的輸入。

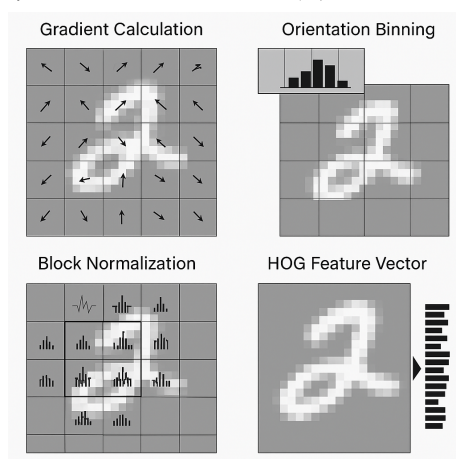


圖 7 HOG 特徵截取方式

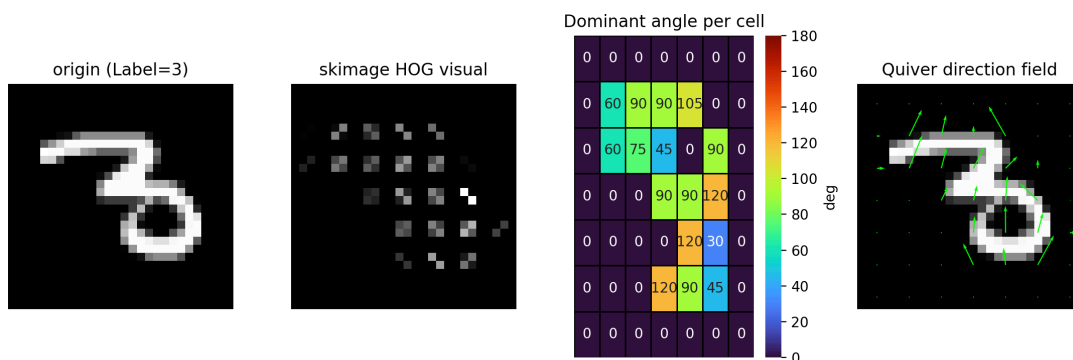


圖 8 HOG 完整流程視覺化

3.2 HOG 實際操作流程

HOG 是一種用來提取影像中邊緣和形狀特徵的技術，這些特徵能幫助分類器像 SVM 進行圖像分類。下面是本報告中提取 HOG 特徵的基本步驟：

1. 將影像轉為灰階

這是處理圖像的第一步，因為灰階圖像可以去除顏色干擾，讓我們專注於影像的結構特徵。

2. 計算每個像素的梯度（Gradient）

在影像的每個像素點，計算它的梯度（變化率）。梯度表示像素顏色變化的程度和方向，這些信息有助於描述影像邊緣和紋理。

3. 將影像切成小格子（Cells）

將影像分割成若干小的區域，這些區域稱為「cell」。每個 cell 會計算出一個方向梯度直方圖，表示這個區域內的邊緣方向分佈。

4. 建立方向直方圖（Histogram）

每個 cell 內部的像素梯度會按照方向分成不同的「bin」，這樣就得到了該 cell 內的方向直方圖。每個 bin 表示某一方向上的梯度強度。

5. 組成 Block 並進行正規

將相鄰的 cells 組合成一個 block。這些 blocks 會經過正規化處理，以避免因為光照變化而導致的影像特徵失真。

6. 將所有 block 向量串接成完整 HOG 特徵向量

最後，將所有 block 的特徵向量串接在一起，形成一個完整的 HOG 特徵向量，這個向量將會用來進行後續的分類訓練（如 SVM）。

3.3 支援向量機(Support Vector Machine, 簡稱 SVM)

在開始使用 SVM 前我先演示一般的分類方式如圖 9，這張圖展示了傳統分類方法的過程，但在資料過於混亂或無法清晰分割時，會導致分類錯誤。

一般分類方法的挑戰：

原始資料：圖片中的藍色和粉紅色點代表兩個不同的類別，這些點在原始空間中已經有明顯的區別。

進行分類：透過一條直線（分隔線）將資料分成兩個區域，藍色點在一邊，粉紅色點在另一邊。這是一個理想的情境，當資料能夠被簡單的直線分割時，分類結果清晰且準確。

資料混亂後的分類困難：當資料變得更加複雜且無法清晰分割時，原本的分類方法會產生錯誤。這是因為資料點分佈變得更加混亂，傳統方法無法使用簡單的直線分割所有類別。

過於混亂難以分類：當資料中出現過於混亂的情況時，即便是進行多次調整，分類仍然無法達到預期的效果，這時候需要使用更先進的方法（如 SVM）來處理這些困難的資料集。

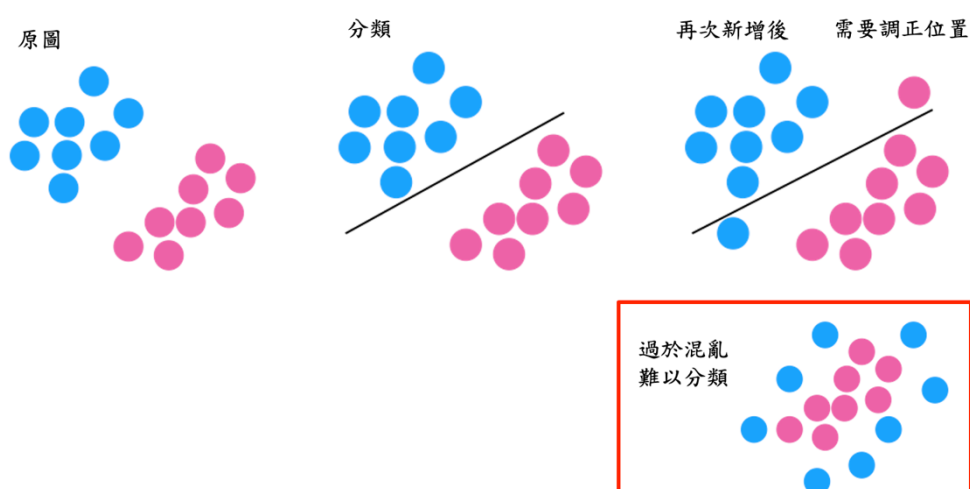
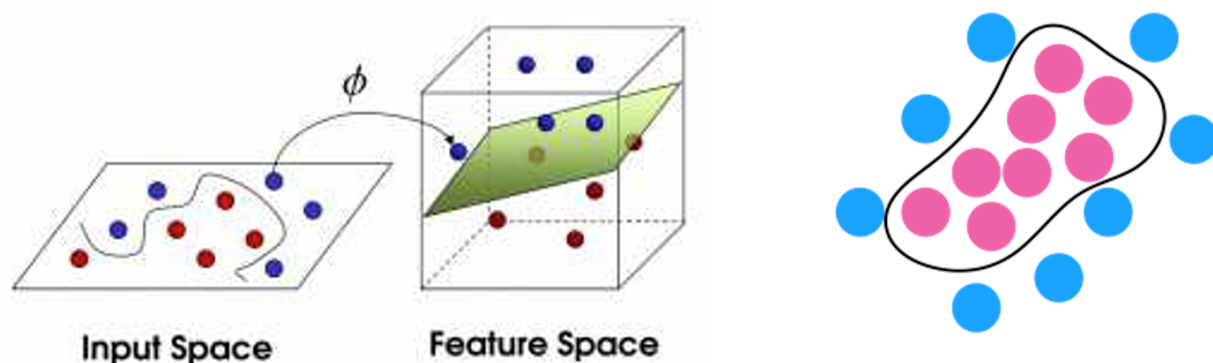


圖 9 非 SVM 分類方式（難以分類）

SVM 是一種監督式的學習方法，用統計風險最小化的原則來估計一個分類的超平面（hyperplane），其基礎的概念非常簡單，就是找到一個決策邊界（decision boundary）讓兩類之間的邊界（margins）最大化，使其可以完美區隔開來。SVM 參數設定：經過 HOG 處理後，圖像會被分割成多個 cell 並建立方向直方圖，再將這些資訊以 block 為單位進行正規化與合併，最終形成一組高維度的特徵向量，用於機器學習分類模型的輸入資料。

當資料在原始空間中無法被清楚區分時，SVM 會將資料映射到一個更高維的空間，在這個新空間中尋找一個最佳的超平面來分割資料。這種方法能夠有效地處理那些在低維空間中難以分割的資料，並且確保分類邊界與資料點之間的最大間隔。SVM 的核心是找出那些離超平面最近的支撐向量，並通過最大化這些支撐向量與超平面之間的距離（即邊界），來提高分類準確率並增強模型的泛化能力，這使得它特別適用於處理高維度的資料。



找出超平面 (hyperplane) 正確區分不同類別的資料

圖 10 SVM 分類方式

3.3.1 SVM 參數設定

經過 HOG 處理後，圖像會被分割成多個 cell 並建立方向直方圖，再將這些資訊以 block 為單位進行正規化與合併，最終形成一組高維度的特徵向量，用於機器學習分類模型的輸入資料。

最終形成一組高維度的特徵向量，用於機器學習分類模型的輸入資料。

步驟	操作說明	詳細解釋
1. 創建 SVM 模型	<code>svm = cv2.ml.SVM_create()</code>	創建一個新的 SVM 模型。
2. 設定 SVM 類型	<code>svm.setType(cv2.ml.SVM_C_SVC)</code>	設定 SVM 類型為 C-SVC，適用於多類別分類問題。
3. 設定核函數	<code>svm.setKernel(cv2.ml.SVM_LINEAR)</code>	設定 SVM 使用線性核函數，適用於線性可分的情況。
4. 設定停止條件	<code>svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 10000, 1e-6))</code>	設定停止條件，最多訓練 10,000 次，或當誤差小於 $1e-6$ 時停止。
5. 訓練模型	<code>svm.train(train_image_mat, cv2.ml.ROW_SAMPLE, train_labels_mat)</code>	用訓練資料進行模型訓練，這裡的資料是 HOG 特徵向量和標籤。

```
# === Step 4: 訓練 SVM 模型 ===
svm = cv2.ml.SVM_create()
svm.setType(cv2.ml.SVM_C_SVC)
svm.setKernel(cv2.ml.SVM_LINEAR) # 線性核函數 (Linear Kernel)
svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 10000, 1e-6))

svm.train(train_image_mat, cv2.ml.ROW_SAMPLE, train_labels_mat)
```

圖 10 SVM 參數設定

選擇使用線性核函數 (Linear Kernel)，因為經過 HOG 特徵轉換後，數據已在高維空間展現出明顯可分性，此時使用線性分類器不僅能保留分類能力，同時也具備更高的運算效率與解釋性

實作 MNIST 手寫數字辨識

4.1 環境設定

在這個步驟中，我們需要安裝並配置適當的環境來進行手寫數字辨識的實驗。以下是步驟：

1. 安裝 opencv-python、scikit-learn、matplotlib 和 numpy 等常用工具來處理數字辨識的流程。例如：`pip install opencv-python scikit-learn matplotlib numpy`
2. 設定 Python 虛擬環境：
 - 使用虛擬環境來隔離依賴，確保實驗的穩定性。
 - 創建虛擬環境：`python -m venv venv`
 - 啟動虛擬環境：`source venv/bin/activate`
3. 資料集準備：

下載 MNIST 資料集並解壓縮，設置圖像路徑與標籤文件。確保資料夾結構與路徑正確，方便後續的圖像加載與標籤處理。

在本報告中使用兩種不同的編程環境和 IDE 來進行實作：
在這個步驟中，我們將介紹 Google Colab 和 MacOS 本機環境的設置與差異，並討論它們對於程式執行的影響。

- 程式執行環境
 - Google colab
 - 上傳資料集到雲端
 - 載入需要的函式庫
 - 掛載Google Drive雲端
 - 寫好程式碼
 - 跑每一張圖片都需要到雲端索取（速度慢）
 - MacOS（額外測試）
 - 下載資料集到本機
 - 設定虛擬環境（並且啟用）
 - 下載對應版本的函式庫
 - 寫好程式碼
 - 在本機執行程式

colab



圖 11 程式執行環境說明

4.2 MNIST 數字識別過程：

MNIST 原圖：首先，我們讀取手寫數字資料集（MNIST），這些資料包含了許多數字圖像，每個圖像都是 28x28 像素的灰階圖片，並有對應的標籤（例如，數字 0、1、2 等）。

HOG 特徵提取：接下來，使用 HOG（Histogram of Oriented Gradients）來提取每張圖像的特徵。HOG 是一種將圖像轉換為有意義的向量的方法，能夠捕捉到圖像中的邊緣和形狀特徵。

特徵向量：經過 HOG 處理後，每張圖像會被轉換為一組特徵向量，這些向量描述了圖像的局部邊緣方向分佈，並且會送入支持向量機（SVM）進行訓練。

SVM 訓練模型：使用 SVM（支持向量機）來進行分類。SVM 透過將數據映射到更高維度的空間來尋找一條最佳的超平面，將不同類別的資料分開。SVM 的優勢在於能夠處理高維數據並最大化分類邊界。

模型預測結果：訓練完成後，模型會用來預測新的數字圖像，並將結果輸出。這個過程會利用在訓練過程中學到的特徵與邊界來進行分類。

MNIST 原圖 → HOG 特徵擷取 → 特徵向量 → SVM 訓練 → 模型預測結果

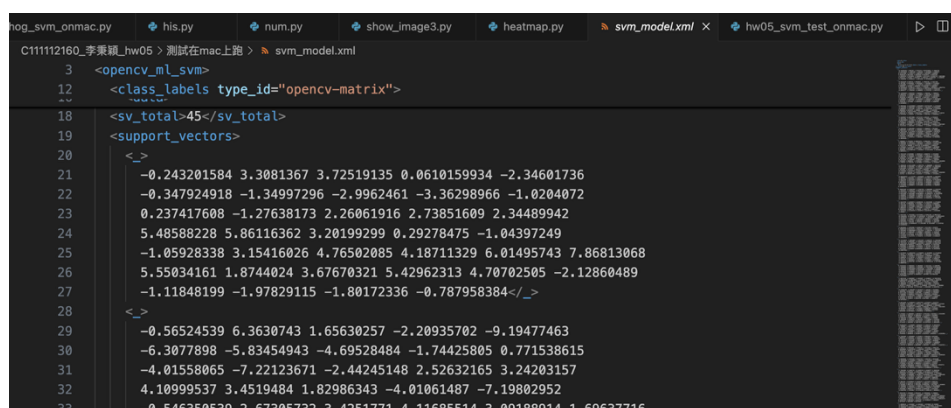
圖 12 MNIST 數字識別流程

4.3 實際訓練流程

首先，執行程式並讀取標籤文件及資料夾中的圖像檔案。接著檢查測試集中的標籤和圖像是否對應，確保資料的正確性。在顯示圖像後，將其關閉，並讓程式繼續進行後續處理。接下來進行 HOG 特徵提取，將每張圖像轉換為可用來訓練模型的特徵向量。完成特徵提取後，開始使用 SVM 進行模型訓練，根據圖像特徵來學習數字分類。最後，將訓練完成的模型儲存，以便未來可以用來對新圖像進行分類預測。

步驟流程如下：

1. 執行程式
2. 讀取標籤和資料夾圖檔
3. 檢查測試標籤和圖像是否對應
4. 關閉圖像，讓程式繼續跑
5. HOG 特徵提取
6. 開始訓練 SVM 模型
7. 將訓練好的模型儲存



```
3 <opencv_ml_svm>
12 <class_labels type_id="opencv-matrix">
18 <sv_total>45</sv_total>
19 <support_vectors>
20 <_>
21 -0.243201584 3.3081367 3.72519135 0.0610159934 -2.34601736
22 -0.347924918 -1.34997296 -2.9962461 -3.36298966 -1.0204072
23 0.237417608 -1.27638173 2.26061916 2.73851609 2.34489942
24 5.48588228 5.86116362 3.20199299 0.29278475 -1.04397249
25 -1.05928338 3.15416026 4.76502085 4.18711329 6.01495743 7.86813068
26 5.55034161 1.8744024 3.67670321 5.42962313 4.70702505 -2.12860489
27 -1.11848199 -1.97829115 -1.80172336 -0.787958384</_>
28 <_>
29 -0.56524539 6.3630743 1.65630257 -2.20935702 -9.19477463
30 -6.3077898 -5.83454943 -4.69528484 -1.74425805 0.771538615
31 -4.01558065 -7.22123671 -2.44245148 2.52632165 3.24203157
32 4.10999537 3.4519484 1.82986343 -4.01061487 -7.19802952
33 -0.546350530 2.67305732 3.4751771 4.11695514 3.00188014 1.60637716
```

圖 13 訓練好的模型（svm_model.xml）

4.5 檢驗準確度

步驟流程如下：

1. 執行程式
2. 呼叫訓練好的模型
3. 讀取標籤和資料夾圖檔
4. 檢查測試標籤和圖像是否對應
5. 關閉圖像，讓程式繼續跑
6. HOG 特徵提取、轉換格式
7. 使用模型來預測
8. 計算準確度

經測驗結果，準確度高達 92.61%。

```

第9991筆：預測：7，標籤：7，累積正確數：9252，累積錯誤數：739
第9992筆：預測：8，標籤：8，累積正確數：9253，累積錯誤數：739
第9993筆：預測：9，標籤：9，累積正確數：9254，累積錯誤數：739
第9994筆：預測：0，標籤：0，累積正確數：9255，累積錯誤數：739
第9995筆：預測：1，標籤：1，累積正確數：9256，累積錯誤數：739
第9996筆：預測：2，標籤：2，累積正確數：9257，累積錯誤數：739
第9997筆：預測：3，標籤：3，累積正確數：9258，累積錯誤數：739
第9998筆：預測：4，標籤：4，累積正確數：9259，累積錯誤數：739
第9999筆：預測：5，標籤：5，累積正確數：9260，累積錯誤數：739
第10000筆：預測：6，標籤：6，累積正確數：9261，累積錯誤數：739
準確度是：92.61%

```

準確率為92.61%

圖 14 準確度計算

4.6 環境比較

經由兩種不同環境驗證，使用本機執行訓練模型，速度更快。

使用 Google Colab，大約會需要六小時的時間完成訓練，而使用 MacOS 進行訓練，只需要花 20 秒的時間。

比較項目	Google Colab	MacOS 本機	說明與影響因素
硬體資源	雲端 GPU/TPU (免費版有限制)	依你的筆電/桌機規格而定	Colab 虛擬機器可能共用資源，性能不穩定；Mac 硬體穩定且直接控制
CPU 性能	可能是虛擬化環境，CPU 不一定強	依實體處理器等級，通常效能好	Mac CPU 可充分使用，虛擬環境可能受限於分配資源
硬碟速度	網路檔案系統、雲端存取速度受限	SSD/高速本地硬碟	資料讀取/寫入速度差異大，影響整體執行時間
GPU 使用	有 GPU 可用 (依計畫與時間限制)	可能無 GPU，純 CPU 運算	取決於專案是否能用 GPU 加速 (你專案可能是 CPU 密集型)
網路延遲	載入資料及依賴需網路	本地硬碟即時讀寫	Colab 載入大檔案、頻繁存取網路資源會拖慢執行
執行環境	Docker 容器，限定軟體版本	可自由安裝與優化軟體環境	MacOS 可以安裝優化的 Python 版本與套件，提升效率
使用便利性	無須配置硬體，自動有 GPU 支援	需自行設定環境	Colab 適合快速試驗、演示，MacOS 適合長期穩定開發
執行時間	專案約 6 小時多	專案約 20 秒	反差極大，說明你的 Mac 本機硬體及軟體優化效果非常好

圖 15 環境差異比較

```

處理第 59998 張圖
處理第 59999 張圖
處理第 60000 張圖
當前路徑：/Users/bingli/Desktop/高雄科技大學/三上/數位影像處理/C111112160_李秉穎
秉穎_hw05/測試在mac上跑
SVM 模型已儲存至：測試在mac上跑資料夾中
訓練 SVM 花費時間：20.536 秒

```

圖 16 MacOS 跑訓練模型花費時間