淺度機器學習作品二：SVD 與影像特徵實驗

學號：411072054 姓名：黃暐宸

作品目標：

Part 1 : PCA 、 SVD 及其在影像處理的應用 :

1. (例題一)選用狗的照片，比較不同的影像切割及重組方式對於圖像品質的影響，使用 SVD 的"Rank q approximation"進行壓縮，以觀察在相同壓縮比下的還原圖像畫質，並嘗試找出還原圖像畫質與切割成幾塊小圖間的關係。
2. (例題二)觀察並隨機顯示 minst_784 手寫數字影像，準備處理大量數據前確保能夠正確理解和處理數據的型態。
3. (例題三)使用手寫數字影像，編寫程式計算 SVD 的"Rank q approximation"壓縮倍數，並根據能量配置或主成分能量佔比選擇適當的 q 值，同時顯示原始圖像和壓縮後的圖像進行比較。

Part 2 : 影像（臉部）特徵的實驗:

(例題四)透過在加密影像圖的前面乘上 $U q$，也就是 : $X_q = U_q U_q^T X = U_q \Sigma_q V_q^T = U_q Z_q$解密 5 張經過加密的影像圖，再選用五張不同類型的照片用由人臉建構的特徵 U 加密再解密，並觀察結果，得出甚麼類型的照片適合用人臉建構的特徵 U 加密與解密。

目標:

1. 透過例題實際使用 SVD 的 “Rank q approximation”壓縮影像，了解其中是透過把圖像矩陣 X 切割，再將每個小圖拉成幾乘幾的向量，最後重組。
2. 透過例題實際操作影像的加密與解密，了解如何透過特徵 U 來加密與解密圖像，並觀察結果。

Part 1 : PCA 、 SVD 及其在影像處理的應用

習題 1：利用 SVD 的'Rank q approximation'對圖像 X 進行壓縮，並在相同的壓縮比下，比較不同重組安排的圖像矩陣 X 的還原圖像品質，找出品質最好的重組安排並解釋其原因。

Background:

SVD 的 "Rank q approximation" 是一種技術，用於對矩陣進行低秩近似，從而實現壓縮的目的，同時盡量保持原始資料的品質。

在圖像處理中，一個圖像可以表示為一個矩陣，其中每個元素代表圖像的像素值。透過對這個像素矩陣進行 SVD，我們可以將其分解為三個矩陣的乘積：$X = U \Sigma V^T$

在 "Rank q approximation" 中，我們僅保留奇異值矩陣 Σ 中的前 q 個奇異值，並相應地修剪U和V^T，以得到近似的原始矩陣。因此，我們得到了近似矩陣$\widetilde{X}$，其表示為： $\widetilde{X} = U \widetilde{\Sigma} V^T$

其中，$\widetilde{\Sigma}$ 是保留了前 q 個奇異值的對角矩陣。

這樣做的好處在於，可以通過選擇較小的 q 值來減少存儲和傳輸圖像所需的資源，同時仍盡可能地保持圖像的品質。當 q 越小，所需的資源就越少，但圖像的近似品質也會相應降低。因此，在設計圖像壓縮算法時，需要在資源使用和圖像品質之間取得平衡。

以下將進行 Rank q approximation，壓縮並保持圖像品質。

(一)定義 montage 函數(將一組影像（存儲在矩陣 A 中）組合成一個大的影像矩陣 M)

程式碼說明:

1. 計算每個影像的大小。這假設每個影像都是正方形，且 A 的行數（即每個影像的像素數）是完全平方數。
2. 創建一個全零矩陣 M，其大小足以容納 m 行和 n 列的影像。
3. 將 A 中的影像重塑為大小為 sz x sz 的正方形，並將其放置在 M 的適當位置。
4. 最後，函數返回組合後的影像矩陣 M。

```
import numpy as np

def montage(A, m, n):
    '''
    Create a montage matrix with mn images
    Inputs:
    A: original pxN image matrix with N images (p pixels), N > mn
    m, n: m rows & n columns, total mn images
    Output:
    M: montage matrix containing mn images
    '''

    sz = np.sqrt(A.shape[0]).astype('int') # image size sz x sz
    M = np.zeros((m*sz, n*sz)) # montage image
    for i in range(m) :
        for j in range(n) :
            M[i*sz: (i+1)*sz, j*sz:(j+1)*sz] = \
            A[:, i*n+j].reshape(sz, sz)
    return M
```

(二)以下將比較四種圖像矩陣 X 的重組安排，並進行 “Rank q approximation”，分別是:

(1) X 不變

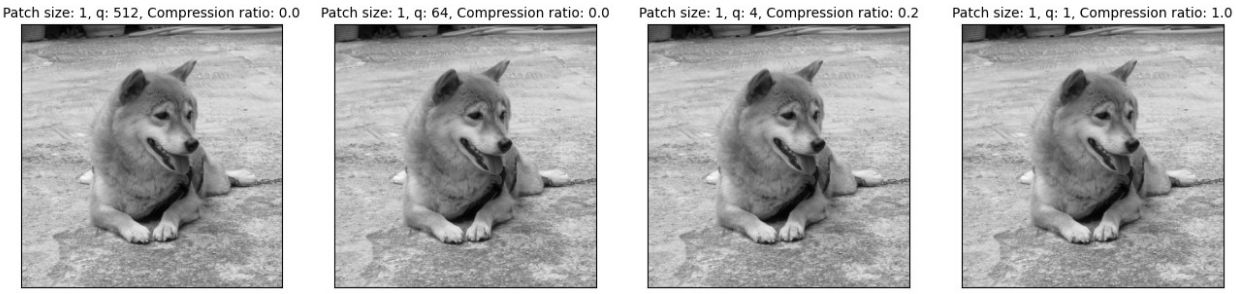
(2) 將 X 以 8×8 小圖（patch）進行切割，再將每個小圖拉成 64×1 的向量，最後重組這些向量並排成新的 $64 \times N$ 矩陣。

(3) 同上，小圖大小為 $16 \times 16 / per$ patch。

(4) 同上，但分割成 $32 \times 32 / per$ patch。

1. 先壓縮沒有進行切割的 X 矩陣，也就是 X 不變

```
import numpy as np
from numpy.linalg import svd
import skimage.util as skutil
from skimage import io
import matplotlib.pyplot as plt

imgfile = "images/dog.JPG" # 512x512x3
X = io.imread(imgfile, as_gray = True)

# 將圖像切割小區域
p, N = X.shape # p by N, p = 512, N = 512
patch_sz = 1
p_patch = patch_sz ** 2
N_patch = int(N * p / p_patch)
patches = skutil.view_as_windows(X, (patch_sz, patch_sz),
step=patch_sz)

M = np.empty((patch_sz*patch_sz, 0))  # initialize M as an empty array
with shape (patch_sz*patch_sz,0)

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
        M = np.append(M, patch, axis=1)  # append the reshaped patch
to M

U, E, VT = svd(M, full_matrices = False)
q_values = np.array([p/1, p/8, p/128, p/512]).astype('int')
fig, ax = plt.subplots(1, 4, figsize=(16, 4))
for i, q in enumerate(q_values):
    Mq = U[:, :q] @ np.diag(E[:q]) @ VT[:q, :]
    ax[i].imshow(Mq.reshape(p, N), cmap = 'gray')
    ratio = N * p / (U.shape[0] + VT.shape[1]) / q
    ax[i].set_title('Patch size: {}, q: {}, Compression ratio:
{:.1f}'.format(patch_sz, q, ratio), fontsize=10)
    ax[i].set_xticks([])
    ax[i].set_yticks([])

plt.show()
```



圖像與結果觀察:

1. q 代表 SVD 中的主成分數量。當 q=512 時，代表在進行 SVD 時保留了 512 個主成分，是全部的主成分，也就是完全沒又進行壓縮，會跟原圖一模一樣。
2. 當 q=64 和 q=4 時，代表在進行 SVD 時分別保留了 64 和 4 個主成分，其中 q=4 的壓縮比例是 0.2。
3. 當 q=1 時，代表在進行 SVD 時保留了 1 個主成分，壓縮比例是 1.0。
4. 從以上的結果可知，如果 X 沒有進行切割，那麼即使 q 只保留一個主成分，壓縮比例還是只有 1.0，這是因為 SVD 的壓縮效果來自於將原始矩陣分解為較小的矩陣。如果原始矩陣 X 沒有進行切割，那麼即使只保留一個主成分，仍然需要存儲整個矩陣，因此壓縮比例還是 1.0。
1. 因為對 X 不變進行壓縮重組代表根本沒有進行壓縮，仍然會比較這四種圖像矩陣 X 的重組安排，但是主要還是看切成 8X8、16X16、32X32 的圖像小圖的壓縮重組結果，以下定義了一個名為 process_image 的副程式進行比較:

程式碼說明:

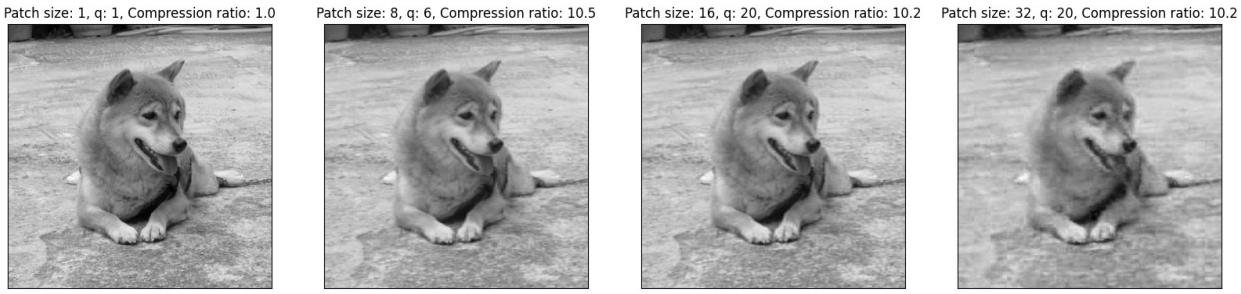
1. 定義了一個名為 process_image 的副程式，只要套用並改參數就能執行多次結果。
2. 將圖像切割成多個小區域，並將每個區域的像素值存儲在一個矩陣中。
3. 使用奇異值分解（SVD）對該矩陣進行壓縮，並保留前 q 個主成分來重建圖像。
4. 套用 montage 函數將壓縮後的小圖像區域重新組合成一個完整的圖像。

```
import numpy as np
from numpy.linalg import svd
import skimage.util as skutil
from skimage import io
import matplotlib.pyplot as plt


def process_image(imgfile, patch_sz, q_value, ax):
    X = io.imread(imgfile, as_gray = True)

    # 將圖像切割小區域
    p, N = X.shape # p by N, p = 512, N = 512
    p_patch = patch_sz ** 2
    N_patch = int(N * p / p_patch)
    patches = skutil.view_as_windows(X, (patch_sz, patch_sz),
step=patch_sz)

    M = np.empty((patch_sz*patch_sz, 0))  # initialize M as an empty
```

```
array with shape (patch_sz*patch_sz,0)

    for i in range(patches.shape[0]):
        for j in range(patches.shape[1]):
            patch = patches[i, j].reshape(-1, 1)
            M = np.append(M, patch, axis=1)  # append the reshaped
patch to M

    U, E, VT = svd(M, full_matrices = False)
    Mq = U[:, :q_value] @ np.diag(E[:q_value]) @ VT[:q_value, :]
    ax.imshow(montage(Mq, int(p/patch_sz), int(p/patch_sz)), cmap =
'gray')
    ratio = N * p / (U.shape[0] + VT.shape[1]) / q_value
    ax.set_title('Patch size: {}, q: {}, Compression ratio:
{:.1f}'.format(patch_sz, q_value, ratio))
    ax.set_xticks([])
    ax.set_yticks([])
```

1. 產生四種圖擺在一起比較還原品質，分別是: (1)Patch size : 1, q = 1, 壓縮比例 : 1.0 (2)Patch size : 8, q = 6, 壓縮比例 : 10.5 (3)Patch size : 16, q = 20, 壓縮比例 : 10.2 (4)Patch size : 32, q = 20, 壓縮比例 : 10.2

```
imgfile = "images/dog.JPG"
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

# Process and display the images
process_image(imgfile, 1, 1, axs[0])
process_image(imgfile, 8, 6, axs[1])
process_image(imgfile, 16, 20, axs[2])
process_image(imgfile, 32, 20, axs[3])

plt.show()
```

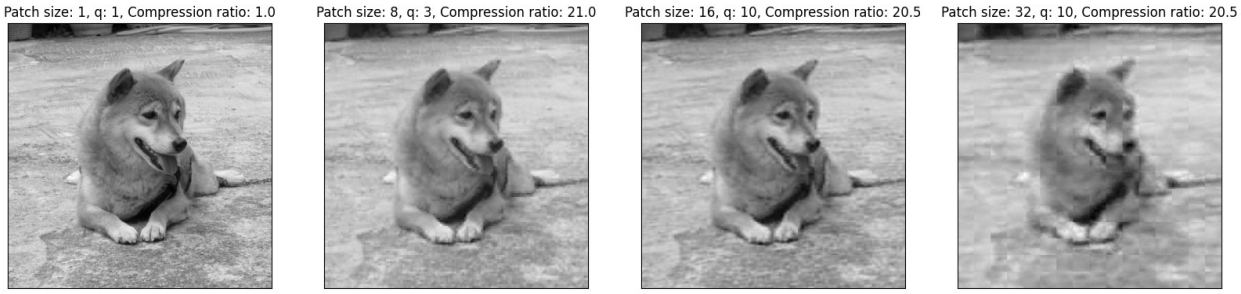


圖像觀察:

1. 主要比較右邊三張圖，這三張圖切成不同大小的小圖再壓縮重組，壓縮比例大約都是 10. 多。
2. 觀察圖像，發現切成 32X32 的小圖再重組的圖畫質最差，而切成 8X8 和 16X16 的的小圖再重組的圖畫質差不多。
3. 由上面的結果，大致能得出一個結論是，切成越小的小圖再壓縮重組的畫質會越差。

1. 產生四種圖擺在一起比較還原品質，分別是: (1)Patch size : 1, q = 1, 壓縮比例 : 1.0 (2)Patch size : 8, q = 3, 壓縮比例 : 21 (3)Patch size : 16, q = 10, 壓縮比例 : 20.5 (4)Patch size : 32, q = 10, 壓縮比例 : 20.5

```
imgfile = "images/dog.JPG"
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

# Process and display the images
process_image(imgfile, 1, 1, axs[0])
process_image(imgfile, 8, 3, axs[1])
process_image(imgfile, 16, 10, axs[2])
process_image(imgfile, 32, 10, axs[3])

plt.show()
```



1. 8X8 & 16X16 的比較(壓縮倍數約等於 65)

```
imgfile = "images/dog.JPG"
fig, axs = plt.subplots(1, 2, figsize=(20, 5))
process_image(imgfile, 8, 1, axs[0])
process_image(imgfile, 16, 3, axs[1])
plt.show()
```

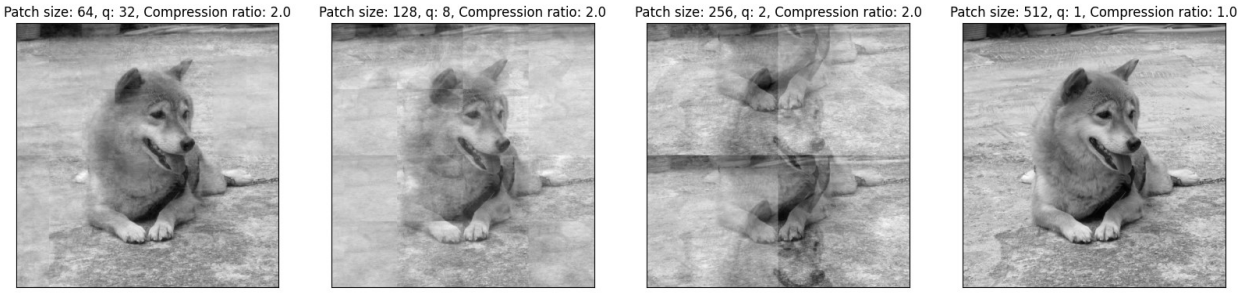
圖像觀察:

1. 主要比較右邊三張圖，這三張圖切成不同大小的小圖再壓縮重組，壓縮比例大約都是 20.5-21。

2. 觀察圖像，發現切成 32X32 的小圖再重組的圖畫質最差，而切成 8X8 小圖再重組的圖畫質比 16X16 的畫質還差一點，從下圖也可以發現當切成 8X8 和 16X16 小圖，且壓縮倍數約等於 65 時，16X16 的還原畫質比 8X8 好很多。
3. 由上面的結果可知，切成越小的小圖再壓縮重組的畫質不一定會越差，有很多的影響因素，用不同張圖也可能產生不同的結果。
1. 特別想法

產生四種圖擺在一起比較還原品質，分別是: (1)Patch size : 64, q = 32, 壓縮比例 : 2 (2)Patch size : 128, q = 8, 壓縮比例 : 2 (3)Patch size : 256, q = 2, 壓縮比例 : 2 (4)Patch size : 512, q = 1, 壓縮比例 : 1

```
imgfile = "images/dog.JPG"
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

# Process and display the images
process_image(imgfile, 64, 32, axs[0])
process_image(imgfile, 128, 8, axs[1])
process_image(imgfile, 256, 2, axs[2])
process_image(imgfile, 512, 1, axs[3])
plt.show()
```



圖像觀察:

1. 從前三張圖可以觀察到，在壓縮比例都等於 2 時，64X64 的畫質最好，再來是 128X128，最差的是 256X256，可推論出切成越小塊還原畫質越差。
2. 至於切成 512X512 為什麼還原畫質這麼好，原因是原始圖像的大小也是 512X512 像素，所以實際上圖像並沒有被切割，而是被視為一個整體來處理，所以畫質才這麼好。
1. 結論: 1.由上面的幾次試驗得出，把圖切成越小塊還原畫質通常會越差，但是在切成 8X8 和 16X16 的小圖且壓縮比例一樣的狀況下，反而是切成 16X16 的還原畫質比較好，所以其實把「圖切成越小塊還原畫質會越差」不是一定的，可能會會有各種狀況產生，使用的圖片不同也可能產生影響。 2.造成以上狀況的原因可能是: (1)資訊損失:當將圖像切成更小的塊時，每個塊包含的資訊就更少。如果在這些小塊上進行壓縮，可能會導致更多的資訊損失，從而降低還原的畫質。 (2)壓縮效率:不同大小的塊可能有不同的壓縮效率。在某些情況下，較大的塊可能能更有效地壓縮，從而在還原時保留更多的資訊。 (3)圖像內容:不同的圖像可能有不同的細節和結構。某些圖像可能在切成較大的塊時能保留更多的細節，而其他圖像可能在切成較小的塊時效果更好。

習題 2：觀看影像圖，以確掌握將要處理的 70000 張手寫圖像影像及其資料型態，並寫一段程式碼來觀察這些手寫數字的影像與品質，且每次執行都能隨機觀看到不同的影像。

(一)讀取 MNIST 手寫數字資料集

```
from scipy.io import loadmat

mnist = loadmat("mnist-original.mat")
X = mnist["data"]
y = mnist["label"][0]
```

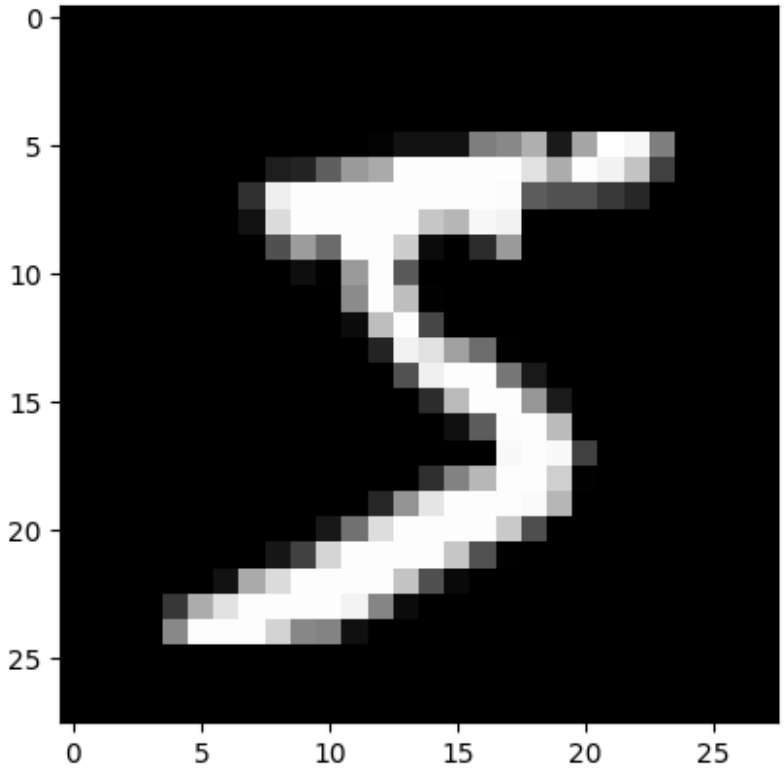
(二)從檔案或網路上獲取 MNIST 手寫數字資料集

程式碼說明:

1. 定義了一個資料檔案名稱 mnist_digits_784.pkl，並檢查該檔案是否存在。如果存在，則從檔案中讀取資料；如果不存在，則從 OpenML 網站獲取 MNIST 資料集，並將其保存到檔案中。(因為檔案不小，以此設計節省之後的檔案讀取時間)
2. 將資料和標籤分別存儲在 X 和 y 變數中。
3. 選擇了第一個影像，並將其顯示出來。

```
from sklearn.datasets import fetch_openml
import numpy as np
import matplotlib.pyplot as plt
import pickle
import os
data_file = "mnist_digits_784.pkl"
# Check if data file exists
if os.path.isfile(data_file):
    # Load data from file
    with open(data_file, "rb") as f:
        data = pickle.load(f)
else:
    # Fetch data from internet
    data = fetch_openml("mnist_784", version=1, parser="auto")
    # Save data to file
    with open(data_file, "wb") as f:
        pickle.dump(data, f)

X, y = np.array(data.data).T, np.array(data.target).astype("int")

# import numpy as np
i = 0
img = X[:, i]
sz = np.sqrt(len(img)).astype("int")
plt.imshow(np.array(img).reshape(sz, sz), cmap="gray")
plt.show()
```

```
# pickle.dump(data, f)
```



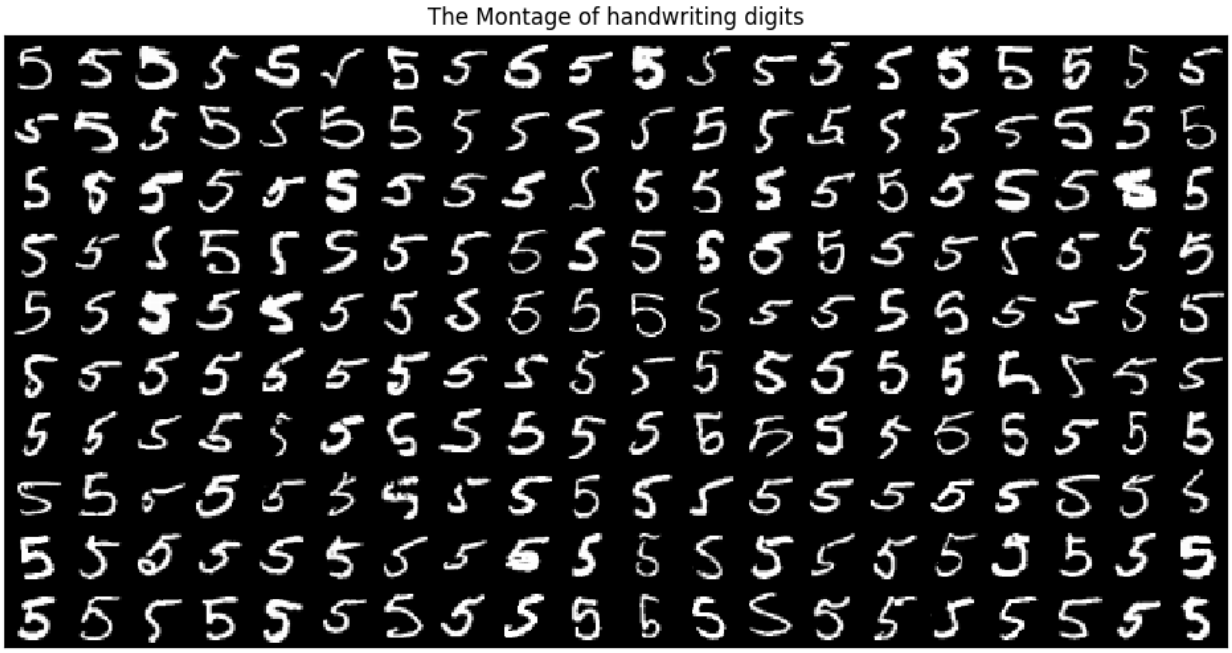
(三)篩選出 200 個數字 5 影像

程式碼說明:

1. 選擇所有標籤為 5 的影像。
2. 從這些影像中隨機選擇 10*20 個影像。
3. 創建一個新的陣列來存儲這些選擇的影像
4. 調用了 montage 函數來創建一個由這些影像組成的拼貼畫。

```
import numpy as np

digit_to_show = 5
# 從數字 "5" 的影像中選擇 m*n 個影像
Digit = X[:, y == digit_to_show]
selected_images = np.random.choice(np.arange(Digit.shape[1]),
replace=False, size=10*20)

# 創建一個新的陣列來存儲選擇的影像
Selected_Digit = Digit[:, selected_images]

plt.figure(figsize = (12, 6))
M = montage(Selected_Digit, 10, 20)
```

```
plt.imshow(M, cmap ="gray", interpolation = 'nearest')
plt.xticks([])
plt.yticks([])
plt.title('The Montage of handwriting digits')
plt.show()
```



(四)觀察手寫數字的影像與品質

程式碼說明:

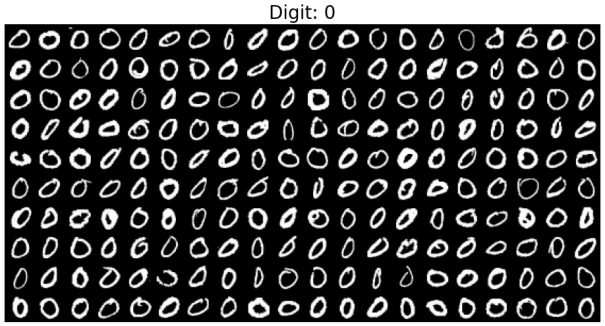
1. 透過迴圈對於每個從 0 到 9 的數字，隨機選擇 50 個影像，創建一個新的陣列來存儲選擇的影像，再將選擇的影像放在網格的一個子圖中。
2. 用 np.random.choice 函數從提供的陣列中隨機選擇元素。replace=False 表示選擇是無放回的，也就是說，同一個元素不會被選擇兩次。size=m*n 指定選擇的元素數量。

```
import numpy as np
import matplotlib.pyplot as plt

fig, axs = plt.subplots(5, 2, figsize=(20, 25))  # Increase the figure size

# 從數字"0" 到"9" 的影像中選擇 m*n 個影像
for digit_to_show in range(10):
    Digit = X[:, y == digit_to_show]
    selected_images = np.random.choice(np.arange(Digit.shape[1]), replace=False, size=m*n)

    # 創建一個新的陣列來存儲選擇的影像
    Selected_Digit = Digit[:, selected_images]
```

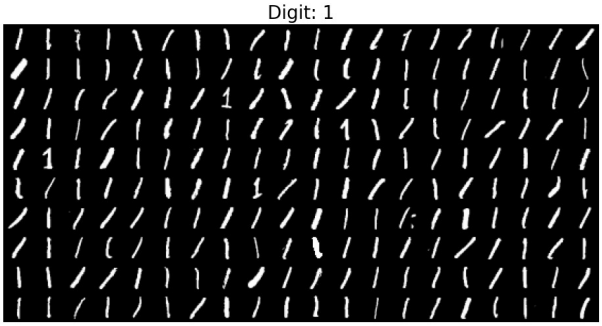
```
    # 將選擇的影像放在網格的一個子圖中
    M = montage(Selected_Digit, m, n)
    axs[digit_to_show // 2, digit_to_show % 2].imshow(M, cmap='gray')
    axs[digit_to_show // 2, digit_to_show % 2].axis('off')
    axs[digit_to_show // 2, digit_to_show % 2].set_title('Digit:
{}'.format(digit_to_show), fontsize=20)  # Increase the title font
size

plt.tight_layout()
plt.suptitle('The Montage of handwriting digits', y=1.02, fontsize=25)
# Increase the suptitle font size
plt.show()
```

The Montage of handwriting digits

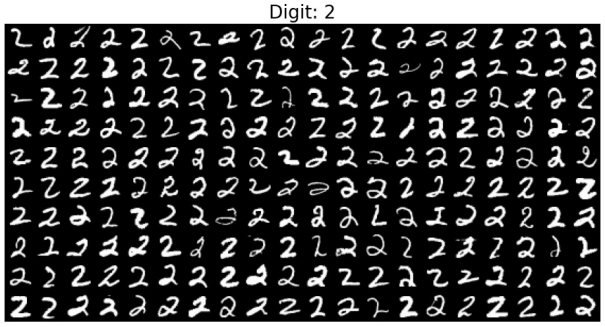
Digit: 0



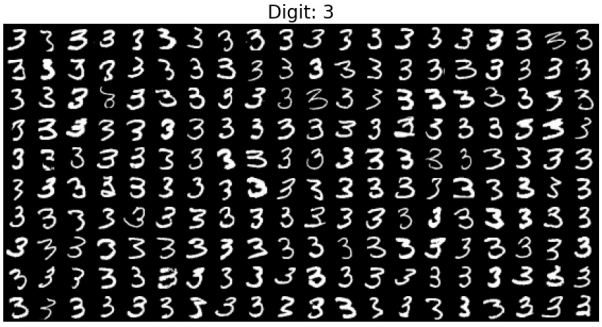
Digit: 1



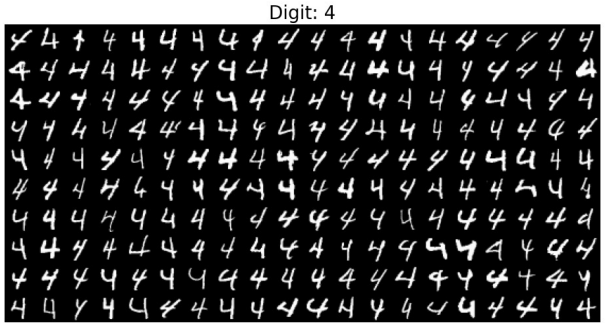
Digit: 2



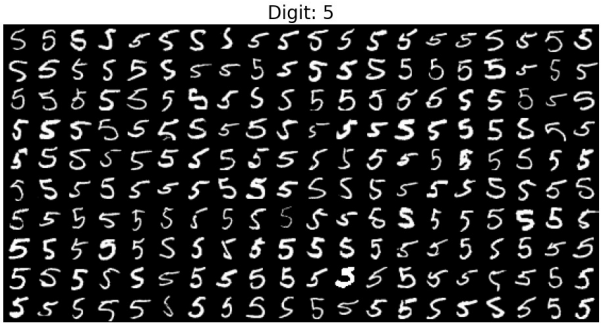
Digit: 3



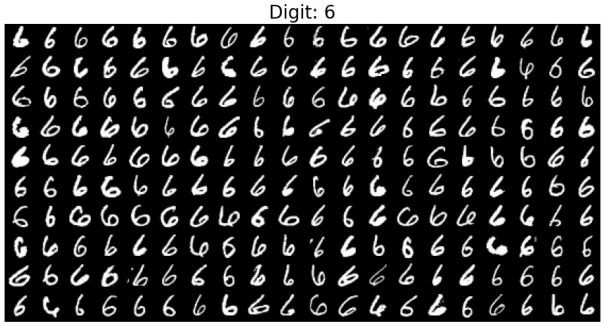
Digit: 4



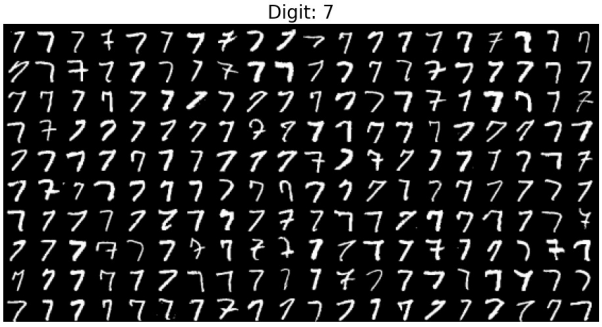
Digit: 5



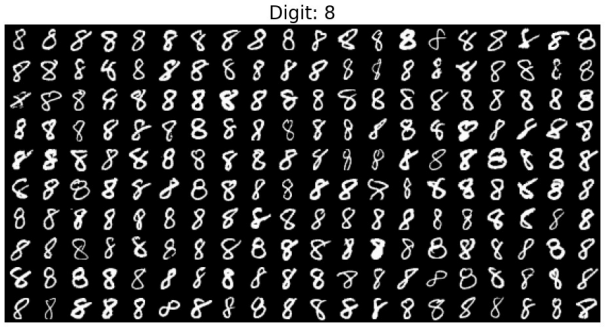
Digit: 6



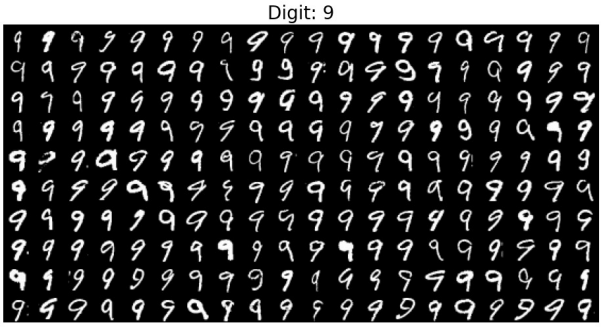
Digit: 7



Digit: 8



Digit: 9



圖像觀察: 重複跑了這支程式，並在這些影像圖裡，發現這些手寫數字都十分有特色，各種樣貌都有，但也發現某些手寫數字 1 和 7 十分的相似，還有 0 和 6，如果要做影像辨識，這幾組可能是容易搞混的。

習題 3：寫一支程式，當調整 q 值時，可以算出壓縮的倍數，並同時顯示原圖與壓縮後還原的圖各 100 張做為比較（任選 100 張）。根據 $\sigma_1, \sigma_2, \cdots, \sigma_r$ 的「能量配置」來決定 q 的選擇，並計算所採用的主成分的能量佔比，最後列印出這個佔比。

(一)從數據集 X 中選擇所有與特定數字相對應的影像

```
digit_to_show = 0
Digit = X[:, y == digit_to_show]
```

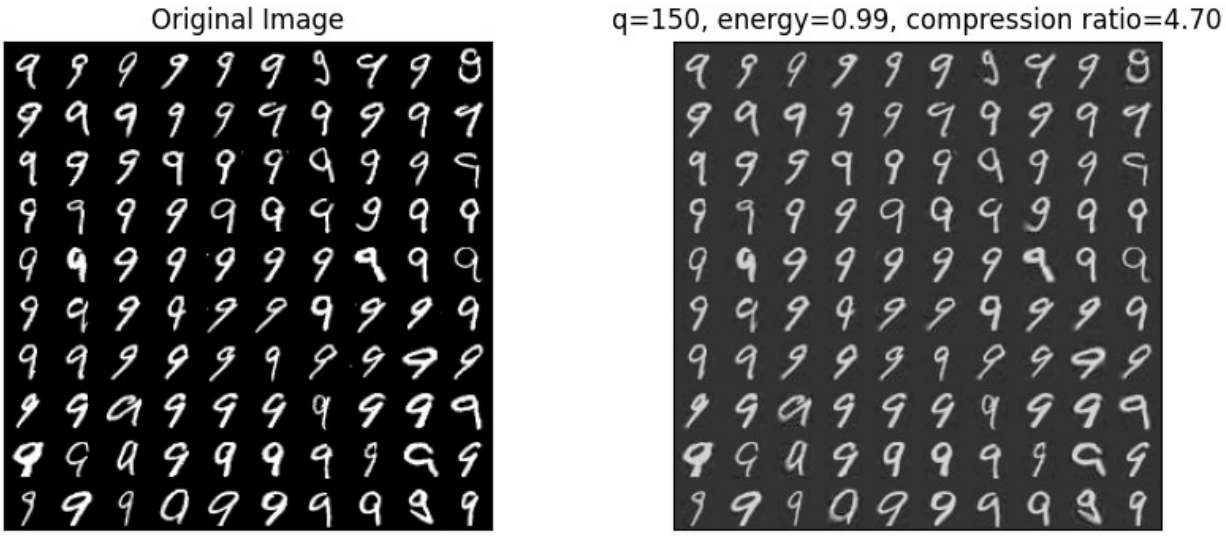
(二)這支程式調整 q 值時，可以算出壓縮的倍數，並同時顯示原圖與壓縮後還原的圖各 100 張做為比較（任選 100 張）

程式碼說明:

1. 對 Digit 進行奇異值分解，並返回 U, E, VT 三個矩陣。其中，U 和 VT 是正交矩陣，E 是一個對角矩陣，對角線上的元素是奇異值。
2. 計算奇異值的平方的累積比例作為能量佔比。
3. 設定壓縮倍數 q。
4. 根據壓縮倍數 q 來計算壓縮後的圖像。
5. ratio = N * p / (U.shape[0] + VT.shape[1]) / q: 這行程式碼計算壓縮比例(比例表示原始數據大小與壓縮後數據大小的比例)。

```
U, E, VT = svd(Digit, full_matrices = False)

# 計算奇異值的平方的累積比例作為能量佔比
energy = np.cumsum(E**2) / np.sum(E**2)

q = 150

fig, ax = plt.subplots(1, 2, figsize=(10, 4))
ax[0].imshow(montage(Digit, m, n), cmap = 'gray')
ax[0].set_title('Original Image')
ax[0].set_xticks([])
ax[0].set_yticks([])

m,n = 10, 10
Mq = U[:, :q] @ np.diag(E[:q]) @ VT[:q, :]
ax[1].imshow(montage(Mq, m, n), cmap = 'gray')

N, p = Digit.shape  # N 和 p 分別為 Digit 的行數和列數
```

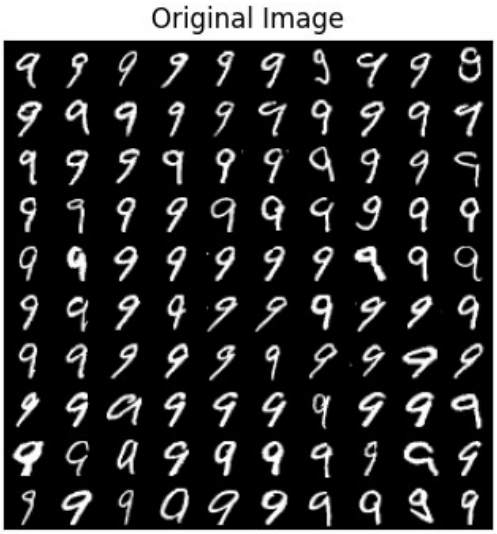
```
ratio = N * p / (U.shape[0] + VT.shape[1]) / q  # 將壓縮比例的計算方式改為
N * p / (U.shape[0] + VT.shape[1]) / q

ax[1].set_title('q={}, energy={:.2f}, compression
ratio={:.2f}'.format(q, energy[q-1], ratio))
ax[1].set_xticks([])
ax[1].set_yticks([])

plt.show()
```

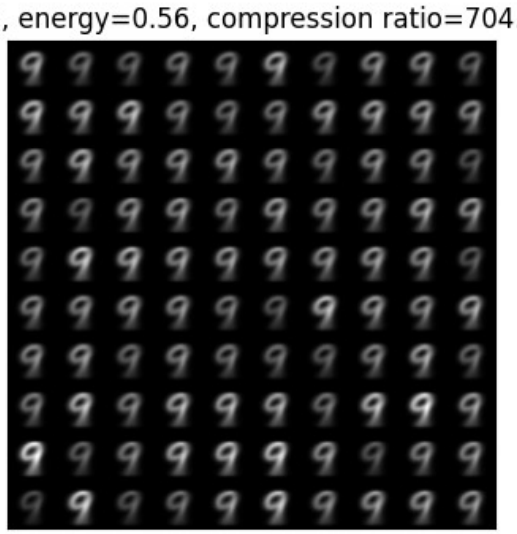


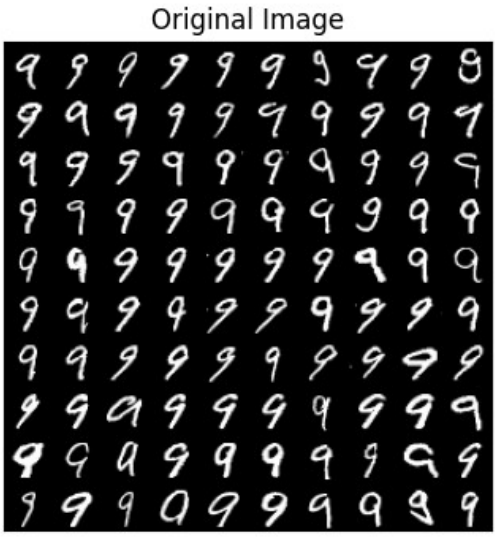
(三)寫成副程式 plot_svd_compression ，只需調用這個函數並更改 q 的值就能執行多次結果。

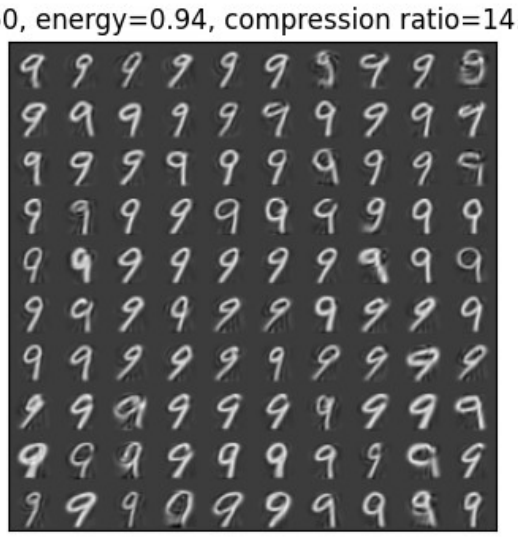
```
def plot_svd_compression(Digit, q, m=10, n=10):
    U, E, VT = svd(Digit, full_matrices = False)

    # 計算奇異值的平方的累積比例
    energy = np.cumsum(E**2) / np.sum(E**2)

    fig, ax = plt.subplots(1, 2, figsize=(10, 4))
    ax[0].imshow(montage(Digit, m, n), cmap = 'gray')
    ax[0].set_title('Original Image')
    ax[0].set_xticks([])
    ax[0].set_yticks([])

    Mq = U[:, :q] @ np.diag(E[:q]) @ VT[:q, :]
    ax[1].imshow(montage(Mq, m, n), cmap = 'gray')

    N, p = Digit.shape  # N 和 p 分別為 Digit 的行數和列數
    ratio = N * p / (U.shape[0] + VT.shape[1]) / q  # 將壓縮比例的計算方式
改為 N * p / (U.shape[0] + VT.shape[1]) / q
```

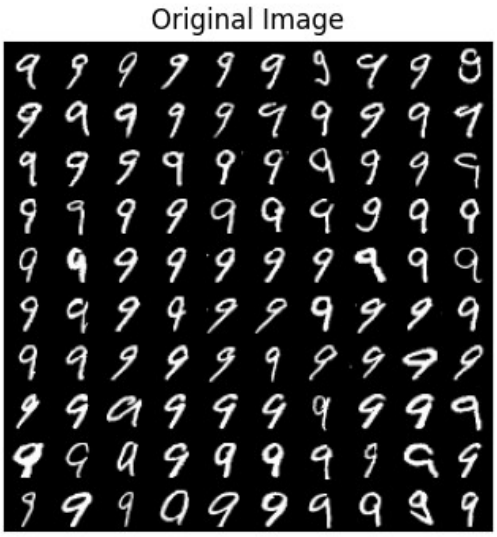
```
    ax[1].set_title('q={}, energy={:.2f}, compression
ratio={:.2f}'.format(q, energy[q-1], ratio))
    ax[1].set_xticks([])
    ax[1].set_yticks([])

    plt.show()

plot_svd_compression(Digit, q=1)
plot_svd_compression(Digit, q=50)
plot_svd_compression(Digit, q=150)
```

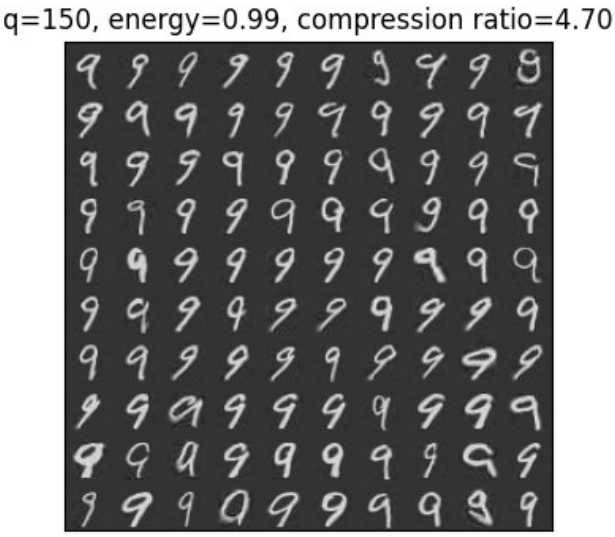












圖像觀察:

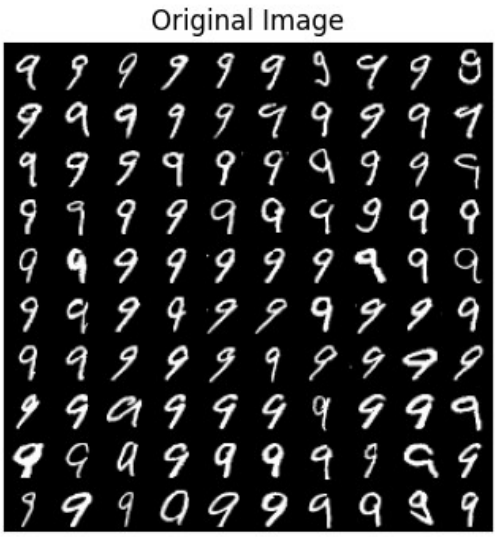
1. q = 1 時，energy=0.56，壓縮的倍數是 704.61 倍，這時的 9 都是整個手寫圖最重要的主成分，也就是構成 9 的主要骨幹，不同的手寫 9 幾乎都含有這個成分。
2. q = 50 時，energy=0.94，壓縮的倍數是 14.09 倍，這時的 9 會保留原始圖像的大部分信息。由於能量為 0.94，這表示前 50 個主成分已經包含了原始圖像 94% 的信息。因此，壓縮後的 9 與原始圖像非常相似，並且能觀察到不同的手寫 9 各自的手寫特色。
3. q = 150 時，energy=0.99，壓縮的倍數是 4.70 倍，這時的 9 會保留原始圖像的絕大部分信息。由於能量為 0.99，這表示前 150 個主成分已經包含了原始圖像 99% 的信息。因此，壓縮後的 9 應該與原始圖像幾乎一模一樣。

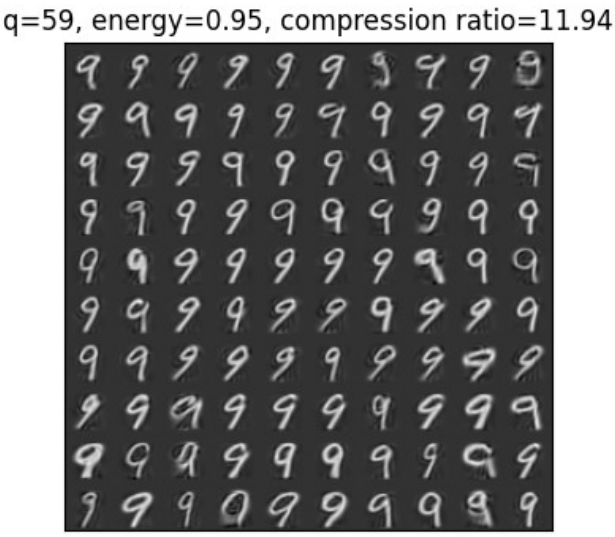
(四)q 的選擇根據 $\sigma_1,\sigma_2,\cdots,\sigma_r$ 的「能量配置」來決定

程式碼說明: 可以選擇壓縮後的圖像保留多少原始圖像的能量，再根據能量配置算出 q。

```
U, E, VT = svd(Digit, full_matrices = False)

# 計算奇異值的平方的累積比例作為能量佔比
energy = np.cumsum(E**2) / np.sum(E**2)

# 設定能量閾值
energy_threshold = 0.95

# 選擇一個 q，使得前 q 個奇異值的平方的累積比例超過能量閾值
q = np.where(energy > energy_threshold)[0][0] + 1

fig, ax = plt.subplots(1, 2, figsize=(10, 4))
ax[0].imshow(montage(Digit, m, n), cmap = 'gray')
ax[0].set_title('Original Image')
ax[0].set_xticks([])
ax[0].set_yticks([])

m,n = 10, 10
```

```
Mq = U[:, :q] @ np.diag(E[:q]) @ VT[:q, :]
ax[1].imshow(montage(Mq, m, n), cmap = 'gray')

N, p = Digit.shape  # N 和 p 分別為 Digit 的行數和列數
ratio = N * p / (U.shape[0] + VT.shape[1]) / q  # 將壓縮比例的計算方式改為
N * p / (U.shape[0] + VT.shape[1]) / q

ax[1].set_title('q={}, energy={:.2f}, compression
ratio={:.2f}'.format(q, energy[q-1], ratio))
ax[1].set_xticks([])
ax[1].set_yticks([])

plt.show()
```





圖像觀察: 我選擇保留 0.95%的能量，算出 q=59，壓縮倍數=11.94，這時的 9 與原始圖像非常相似，且能觀察到不同的手寫 9 各自的手寫特色。

Part 2 : 影像（臉部）特徵的實驗

習題 4：有 5 張經過 Yale Faces 38 人 2410 張人臉圖像矩陣 X 的 SVD 的特徵 U 加密的影像圖，其加密方式: $X = U \Sigma V^T$ ，取 U 作為影像加密的工具，即假設向量 x 代表一張原圖影像，則 $U[:, 0:q)^T$ x 代表該影像的前 q 個主成分，以此作為加密影像。

1. 對以上的五張影像進行解密

2. 自行找 5 張照片加密並解密

(一)先讀取和處理 "allFaces.mat" 的 MATLAB 檔案，檔案裡面是 Yale Faces 38 人 2410 張人臉圖像矩陣。

```
import numpy as np
import scipy.io
D = scipy.io.loadmat("allFaces.mat")
X = D["faces"] # 32256 x 2410, each column represents an image
y = np.ndarray.flatten(D["nfaces"])
m = int(D["m"]) # 168
n = int(D["n"]) # 192
n_persons = int(D["person"]) # 38
```

```
C:\Users\wesley\AppData\Local\Temp\ipykernel_24868\1351594050.py:6:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  m = int(D["m"]) # 168
C:\Users\wesley\AppData\Local\Temp\ipykernel_24868\1351594050.py:7:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  n = int(D["n"]) # 192
C:\Users\wesley\AppData\Local\Temp\ipykernel_24868\1351594050.py:8:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  n_persons = int(D["person"]) # 38
```

(二)定義 show_montage 函數

程式碼說明:

1. X 是一個矩陣，其中每行代表一個圖像。n 和 m 是每個圖像的維度。h 和 w 是要創建的拼貼的維度。
2. 使用 plt.subplots(h, w, figsize=(w, h)) 創建一個具有 h 行 w 列子圖的圖形。
3. 如果 X 中的圖像數量少於 w * h，則將 X 用零向量填充，直到它有 w * h 個圖像。
4. 歷圖形中的每個子圖。對於每個子圖，將對應的圖像從 X 中重塑為 m 行 n 列的形狀，並在子圖上顯示。

```
def show_montage(X, n, m, h, w):
    '''
    X: 影像資料矩陣，每行代表一張影像
    n, m: 每張影像的大小 n x m
    h, w : 建立一個蒙太奇圖陣，大小 figsize = (w,h)
    '''
    fig, axes = plt.subplots(h, w, figsize=(w, h))
    if X.shape[1] < w * h: # 影像張數不到 w x h 張，用 0 向量補齊
        X = np.c_[X, np.zeros((X.shape[0], w*h-X.shape[1]))]
    for i, ax in enumerate(axes.flat):
        ax.imshow(X[:,i].reshape(m, n).T, cmap="gray")
        ax.set_xticks([])
        ax.set_yticks([])
    plt.show()
```

(三)從 X 中選取數據集中不同人(38 人)的圖像，並使用 show_montage 函數將它們顯示出來。

```
all_diff_persons = np.zeros((m*n, n_persons))
cnt = 0
for i in range(n_persons):
    all_diff_persons[:,i] = X[:,cnt]
    cnt = cnt + y[i]
show_montage(all_diff_persons , n, m, 4,10)
```

(四)透過 Yale Faces 38 人 2410 張人臉圖像矩陣 X 的 SVD 中的特徵 U 解密這 5 張影像

(1)讀取 5 張經過加密的影像圖的 excel 檔

```
import csv
def read_csv_with_pandas(file_name):
    data = pd.read_csv(file_name, header=None, skiprows=1)
    return data.values
read_csv_with_pandas('五張加密的影像_2024.csv')
```

```
array([[-1.11389301e+02, -1.04129980e+02, -1.08180651e+02,
        -9.65371612e+01, -8.66466684e+01],
       [-1.66082033e+01, -3.00829165e+01, -2.54185353e+01,
        -9.15095952e+00, -2.92607486e+01],
       [ 2.32625886e+01,  1.35802598e+01,  2.40313050e+01,
         1.66706924e+01,  2.56003355e+01],
       ...,
       [-3.28841710e-02,  1.01011753e-01, -5.11599118e-02,
         2.89639879e-01,  1.95135224e-01],
       [ 9.42688962e-02, -8.45146890e-02,  9.94247733e-02,
         5.86329584e-02,  3.11638176e-01],
       [-2.92293533e-02, -3.99655773e-02, -3.81157792e-02,
        -6.35402099e-02, -3.70059603e-02]])
```

(2)解密原理: 加密影像圖(eigen face) 是 : $U_q^T X = \Sigma_q V_q^T = Z_q$ 現在我要在加密影像圖的前面乘上 $U q$ ，也就是 : $X_q = U_q U_q^T X = U_q \Sigma_q V_q^T = U_q Z_q$ 這樣就能解密影像圖。

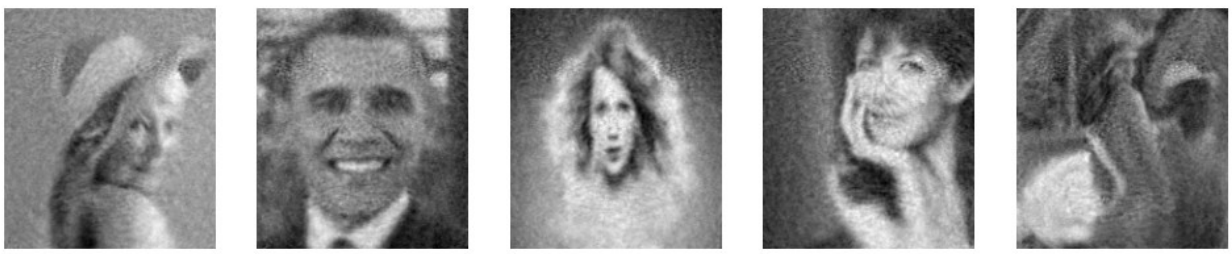
(3)解密第一張圖

程式碼說明:

1. 取得第一欄的數據

2. 計算了 X 中每一列的平均值，並將 X 中的每一列都減去其對應的平均值，這樣做可以將數據中心化。
3. 對中心化後的數據 X 進行奇異值分解（SVD）
4. 取前 q=2000 個主成分，將 U_q 和 first_column 進行矩陣乘法，得到新的數據 x。

```
from scipy.io import loadmat
from numpy.linalg import svd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def load_data(file_name):
    D = loadmat(file_name)
    X = D['faces']
    avgFace = X.mean(axis=1)
    X_avg = X - avgFace.reshape(-1, 1)
    return X_avg

def perform_svd(X_avg):
    U, E, VT = svd(X_avg, full_matrices=False)
    return U

def display_images(X, n, m, h, w):
    fig, axes = plt.subplots(h, w, figsize=(12, 12))
    if X.shape[1] < w * h:
        X = np.c_[X, np.zeros((X.shape[0], w*h-X.shape[1]))]
    for i, ax in enumerate(axes.flat):
        ax.imshow(X[:,i].reshape(m, n).T, cmap='gray')
        ax.axis('off')
    plt.show()

def main():
    X_avg = load_data('allFaces.mat')
    U = perform_svd(X_avg)
    Uq = U[:, :2000]
    data_array = read_csv_with_pandas('五張加密的影像_2024.csv')
    F = Uq @ data_array
    display_images(F, 192, 168, 1, 5)

main()
```



圖像觀察:

1. 五張圖還原出來可以明顯看到五個人的樣子，但是畫質沒有很好，原因應該是只選擇了前 q 個主成分來重建圖，並且經過壓縮還原，圖片的畫質自然會下降。
2. 由於使用的是 Yale Faces 數據集的 U 矩陣來重建其他圖像，如果這些圖像與 Yale Faces 數據集的圖像在某些關鍵特徵上有顯著差異（例如，光線條件、面部表情、頭部姿勢等），那麼重建的畫質可能會受到影響。

(五)找 5 張照片（大小必須同 Yale Faces 的 192×168 或自行 Resize），含人臉、水果、風景 … 等進行加密後（q 自選），再解密，觀察這些解密後的影像的效果，是否人臉的表現比較好？其他非人臉影像，如風景影像，能透過由人臉建構的特徵 U 加密嗎？（即解密後能否看到原圖模樣？）

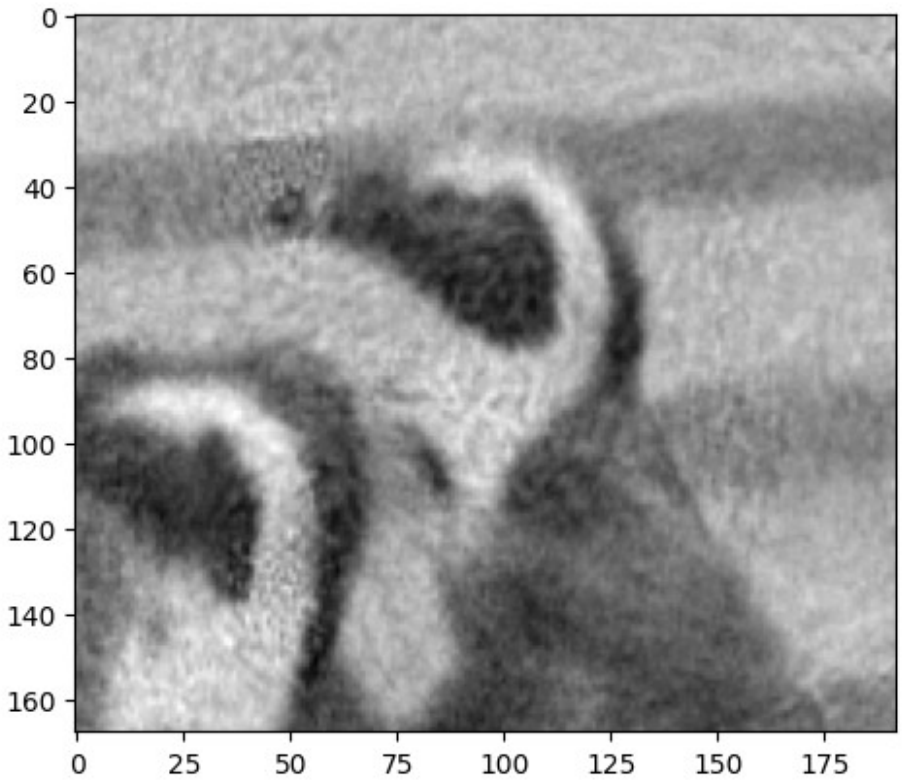
(1)加密企鵝圖片再解密

程式碼說明:

1. 讀取並處理圖像：程式碼讀取一張圖像，將其轉換為 NumPy 陣列。
2. 讀取並處理數據集：程式碼讀取一個包含多張臉部圖像的數據集，並對其進行平均臉部處理。
3. 進行奇異值分解（SVD）：程式碼對處理後的數據集進行 SVD，並選擇前 q(2000) 個奇異值。
4. 加密圖像：程式碼將原始圖像投影到由選擇的奇異值構成的空間，進行加密。
5. 解密並顯示圖像：程式碼將加密的圖像重新投影回原始空間進行解密，並將解密後的圖像顯示出來。

```
import numpy as np
from PIL import Image
from scipy.io import loadmat
from numpy.linalg import svd
import matplotlib.pyplot as plt

# 讀取影像
img = Image.open('images/penguin_1-modified.JPG')
img_array = np.array(img)

# 確保影像是灰度的
if len(img_array.shape) > 2:
    img_array = img_array.mean(axis=2)

# 讀取 allFaces.mat
D = loadmat('allFaces.mat')
X = D['faces']
avgFace = X.mean(axis=1)
X_avg = X - avgFace.reshape(-1, 1)

# 進行 SVD
U, E, VT = svd(X_avg, full_matrices=False)

# 選擇前 q 個奇異值
```

```
q = min(2000, U.shape[1])  # 確保 q 不大於 n
Uq = U[:, :q]

# 加密
img_array_reshaped = img_array.reshape(-1)
x_new = Uq.T @ img_array_reshaped

# 解密
x_decrypted = Uq @ x_new

# 將解密後的影像改回原始形狀
x_decrypted = x_decrypted.reshape(img_array.shape)

# 顯示解密後的影像
plt.imshow(x_decrypted, cmap='gray')
plt.show()
```



(2)把前述程式碼改寫成 Encrypt_Decrypt 副程式加密並解密五張不同類型的圖:

```
import numpy as np
from PIL import Image
from scipy.io import loadmat
from numpy.linalg import svd
import matplotlib.pyplot as plt
```

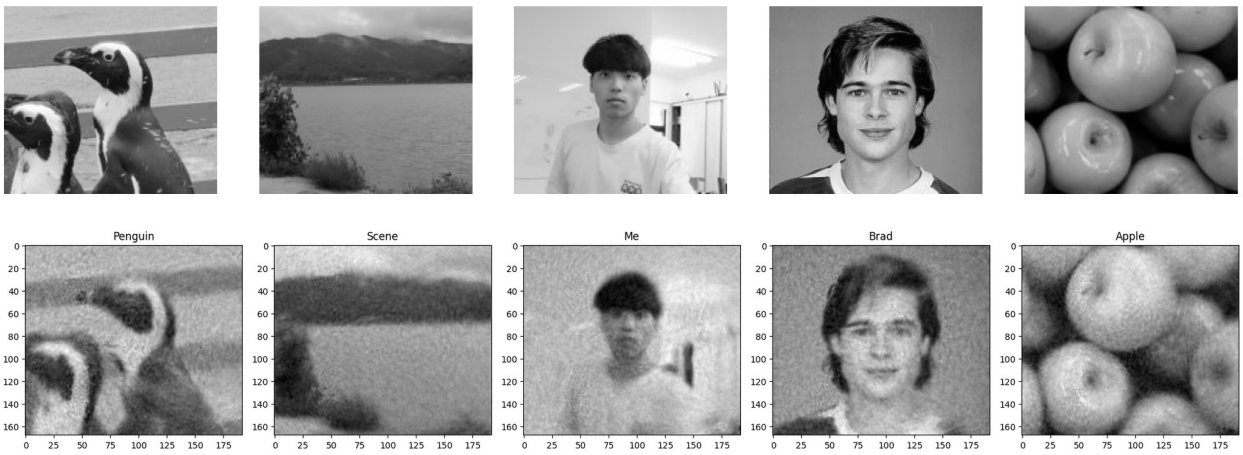
```
def load_and_convert_image(image_path):
    img = Image.open(image_path)
    img_array = np.array(img)
    if len(img_array.shape) > 2:
        img_array = img_array.mean(axis=2)
    return img_array

def load_faces_data():
    D = loadmat('allFaces.mat')
    X = D['faces']
    avgFace = X.mean(axis=1)
    X_avg = X - avgFace.reshape(-1, 1)
    return X_avg

def perform_svd_and_select_q(X_avg, q):
    U, E, VT = svd(X_avg, full_matrices=False)
    q = min(q, U.shape[1])  # 確保 q 不大於 n
    Uq = U[:, :q]
    return Uq

def encrypt_and_decrypt_image(img_array, Uq):
    img_array_reshaped = img_array.reshape(-1)
    x_new = Uq.T @ img_array_reshaped
    x_decrypted = Uq @ x_new
    x_decrypted = x_decrypted.reshape(img_array.shape)
    return x_decrypted

def display_image(img_array, title, subplot):
    plt.subplot(subplot)
    plt.title(title)
    plt.imshow(img_array, cmap='gray')

def Encrypt_Decrypt(image_path, q, title, subplot):
    img_array = load_and_convert_image(image_path)
    X_avg = load_faces_data()
    Uq = perform_svd_and_select_q(X_avg, q)
    x_decrypted = encrypt_and_decrypt_image(img_array, Uq)
    display_image(x_decrypted, title, subplot)
```

執行 Encrypt_Decrypt 並解密下述五張圖(q=2400):

1. 第一張:企鵝
2. 第二張:風景
3. 第三張:我自己
4. 第四張:外國男星
5. 第五張:蘋果

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
# 讀取圖片
images = ['images/penguin_1-modified.JPG', 'images/scene-
modified.JPG',
'images/me-modified.JPG', 'images/brad-modified.JPG', 'images/apple-
modified.JPG']

plt.figure(figsize=(20, 4))

# 顯示圖片
for i, image in enumerate(images):
    img = mpimg.imread(image)
    plt.subplot(1, len(images), i+1)
    plt.imshow(img)
    plt.axis('off')  # 隱藏座標軸

plt.show()

plt.figure(figsize=(20, 4))  # 設定圖片大小
Encrypt_Decrypt('images/penguin_1-modified.JPG', 2400, 'Penguin', 151)
Encrypt_Decrypt('images/scene-modified.JPG', 2400, 'Scene', 152)
Encrypt_Decrypt('images/me-modified.JPG', 2400, 'Me', 153)
Encrypt_Decrypt('images/brad-modified.JPG', 2400, 'Brad', 154)
Encrypt_Decrypt('images/apple-modified.JPG', 2400, 'Apple', 155)
plt.tight_layout()  # 自動調整子圖間的間距
plt.show()
```



圖像觀察:

1. 企鵝:圖片經過 Yale Faces 的人臉特徵 U 加密解密後，變得十分模糊，直接看圖無法判斷是企鵝。
2. 風景:圖片經過 Yale Faces 的人臉特徵 U 加密解密後，也變得十分模糊，完全看不出來是風景圖。
3. 我自己:圖片經過 Yale Faces 的人臉特徵 U 加密解密後，還算清楚，看的出來是我本人。
4. 外國男星:圖片經過 Yale Faces 的人臉特徵 U 加密解密後，是五張裡最清楚的。
5. 蘋果:圖片經過 Yale Faces 的人臉特徵 U 加密解密後，還算清楚，看的出來是蘋果。

結論：

1. 這五張圖片經過 Yale Faces 的人臉特徵 U 進行加密和解密後，其還原畫質有顯著差異。特別是非人臉圖像（如企鵝和風景）變得非常模糊，幾乎無法識別。而人臉圖像（如自己和外國男星）。這可能是因為 Yale Faces 的人臉特徵 U 主要捕捉到的是人臉的特徵，對於非人臉的圖像，其特徵可能無法被有效捕捉，因此在還原時會失去很多細節。這個結果顯示，使用特定類型（如人臉）的特徵進行圖像加密和解密，可能並不適用於所有類型的圖像。
2. 比較特別的是，蘋果能夠相對清晰地被識別出來，原因可能是其具有獨特形狀，但不同的蘋果圖產生的結果能也不同，還有待未來繼續研究。