第二次作品：SVD 與影像特徵實驗

學號：410978040

姓名：黃冠翔

作品目標：

- 學習 PCA 和 SVD 在影像處理的應用。
- 學習利用 Rank q approximation 將影像壓縮和重組。
- 學習利用特徵臉(Eigenfaces) 進行影像辨識、影像加密或影像壓縮。

Lesson 6

第 1 題：

將一張圖像 X 利用 SVD 的 “Rank q approximation”，能達到壓縮的目的並保持圖像的品質。比較下列幾種對於圖像矩陣 X 的重組安排，並進行 “Rank q approximation”，在同樣的壓縮比之下，觀察還原後的圖像品質哪個最好？能說出理由嗎？

(1) X 不變。

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


imgfile = mpimg.imread("Lenna.png")
plt.axis('off')
plt.title('Original')
plt.imshow(imgfile)
```

```
<matplotlib.image.AxesImage at 0x2a1ccd27fa0>
```

Original

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

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

# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 8x8 的小區域
patch_size = 8
patches = skutil.view_as_windows(X, (patch_size, patch_size),
step=patch_size)

# 將每個小區域拉成 64x1 的向量
p, N = X.shape
patch_pixels = patch_size ** 2
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
```

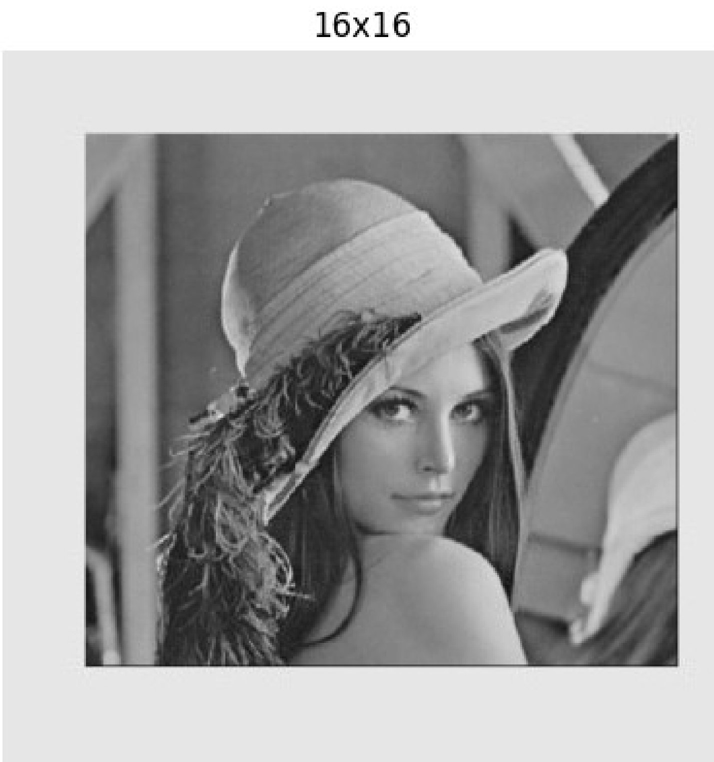
```python
        patch = patches[i, j].reshape(-1, 1)
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q 個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:, 
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] = 
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('8x8')
plt.show()
```

8x8

(3)同上，小圖大小為 16 x 16/per patch。

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

# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 16x16 的小區域
patch_size = 16
patches = skutil.view_as_windows(X, (patch_size, patch_size),
step=patch_size)

# 將每個小區域拉成 64x1 的向量
p, N = X.shape
patch_pixels = patch_size ** 2
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
```

```python
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q  個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:,
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] =
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('16x16')
plt.show()
```



(4)同上，但分割成 32 x 32/per patch。

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

# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 32x32 的小區域
patch_size = 32
patches = skutil.view_as_windows(X, (patch_size, patch_size),
step=patch_size)

# 將每個小區域拉成 64x1 的向量
p, N = X.shape
patch_pixels = patch_size ** 2
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
```

```python
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q 個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:,
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] =
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('32x32')
plt.show()
```

32x32

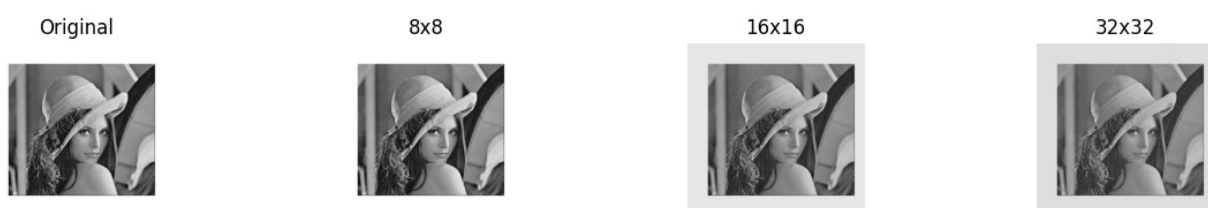
(5)將圖像並列比較。

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

fig, ax = plt.subplots(1, 4, figsize=(15, 2))
plt.subplot(141)
imgfile = mpimg.imread("Lenna.png")
plt.axis('off')
plt.title('Original')
plt.imshow(imgfile, cmap='gray')


plt.subplot(142)
# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 8x8 的小區域
patch_size = 8
patches = skutil.view_as_windows(X, (patch_size, patch_size),
step=patch_size)
```

```
# 將每個小區域拉成 64x1 的向量
p, N = X.shape
patch_pixels = patch_size ** 2
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q  個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:, 
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] = 
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('8x8')


plt.subplot(143)
# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 16x16 的小區域
patch_size = 16
patches = skutil.view_as_windows(X, (patch_size, patch_size), 
step=patch_size)

# 將每個小區域拉成 64x1 的向量
```

```
p, N = X.shape
patch_pixels = patch_size ** 2
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q 個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:,
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] =
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('16x16')

plt.subplot(144)
# 讀取圖像
imgfile = "Lenna.png"
X = io.imread(imgfile, as_gray=True)

# 將圖像切割成 32x32 的小區域
patch_size = 32
patches = skutil.view_as_windows(X, (patch_size, patch_size),
step=patch_size)

# 將每個小區域拉成 64x1 的向量
p, N = X.shape
patch_pixels = patch_size ** 2
```

```python
N_patches = int(N * p / patch_pixels)
patch_vectors = np.empty((patch_pixels, 0))

for i in range(patches.shape[0]):
    for j in range(patches.shape[1]):
        patch = patches[i, j].reshape(-1, 1)
        patch_vectors = np.append(patch_vectors, patch, axis=1)

# 進行 SVD 分解
U, Sigma, VT = svd(patch_vectors, full_matrices=False)

# 保留前 q 個奇異值，這裡先設定 q = 64
q = 64
U_q = U[:, :q]
Sigma_q = np.diag(Sigma[:q])
VT_q = VT[:q, :]

# 重組向量成新的 64xN 矩陣
reconstructed_patches = U_q @ Sigma_q @ VT_q
reconstructed_image = np.zeros(X.shape)

idx = 0
for i in range(0, X.shape[0], patch_size):
    for j in range(0, X.shape[1], patch_size):
        reconstructed_patch = reconstructed_patches[:,
idx].reshape(patch_size, patch_size)
        reconstructed_image[i:i+patch_size, j:j+patch_size] =
reconstructed_patch
        idx += 1

# 顯示重建的圖像
plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.title('32x32')
plt.show()
```



討論：

- 先讀取原始圖，接著再分別列出以 8x8、16x16 和 32x32 進行切割後重組的影像，最後將所有影像合併進行比較。
- 跟原始圖相比以 8x8 進行切割後重組，大致上差異不大。
- 但以 16x16 切割後重組，與原始圖相比有明顯模糊的趨勢。
- 最後以 32x32 切割後重組，則是最為模糊的一組。

第 2 題：

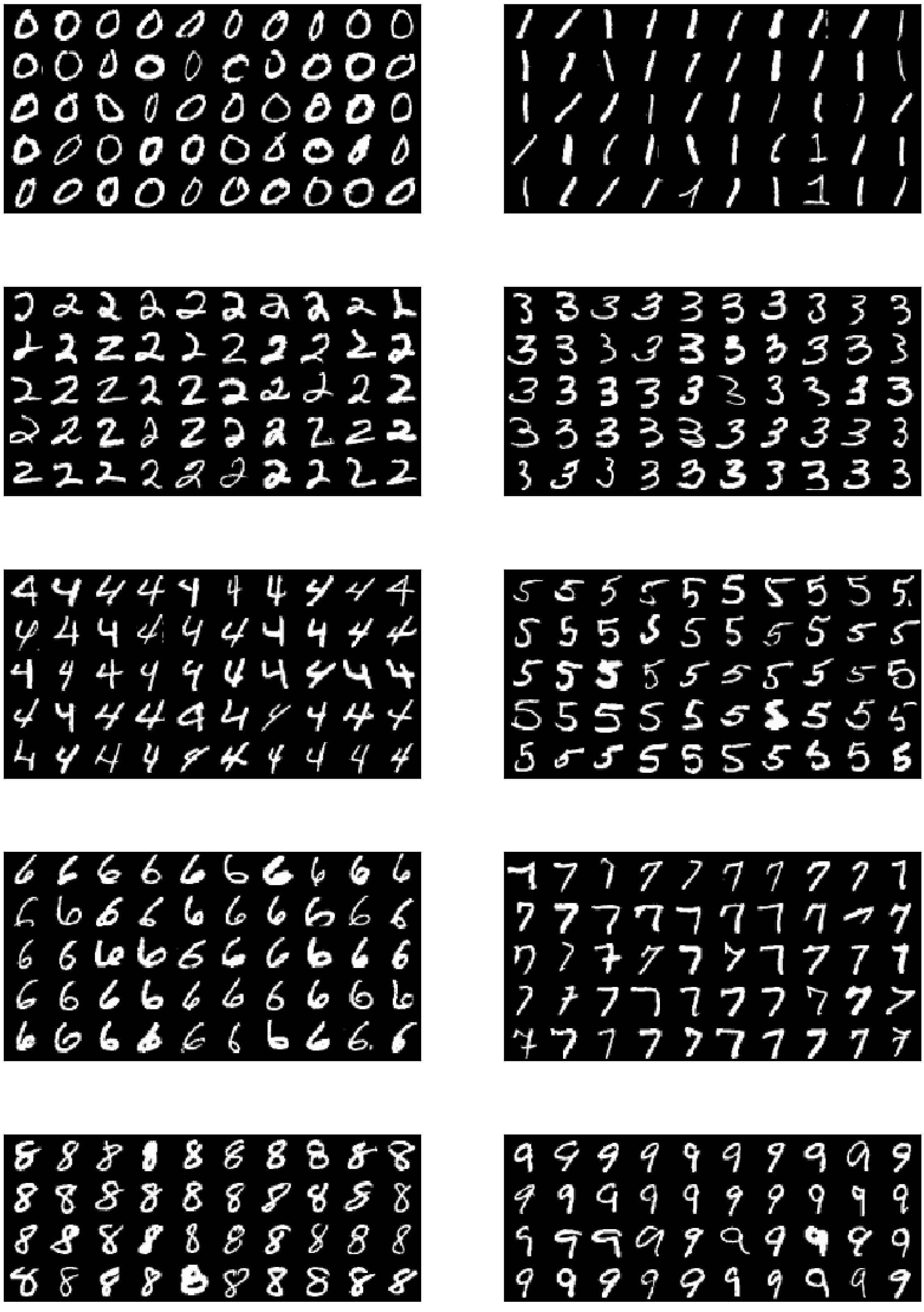
處理大量影像前，有必要觀看影像圖，以確定能掌握將要處理的影像及其資料型態。以 70000 張手寫圖像為例，每個數字約 7000 字，需要寫一段程式碼來觀察這些手寫數字的影像與品質，且每次執行都能隨機觀看到不同的影像，如下圖左（共兩排含 0~9 的數字各 50 個）與圖右的影像是兩次執行的結果。請靜下心來仔細寫這段程式碼，可以按下圖的方式呈現，或用自己的方式都歡迎。類似像這樣的程式基本功事非常重要且必要的。

```
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

from scipy.io import loadmat
import matplotlib.pyplot as plt

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

fig, ax = plt.subplots(5, 2, figsize=(10, 15))

plt.subplot(521)
digit_to_show = 0
Digit = X[:, y==digit_to_show]
m, n = 5, 10 # A m x n montage (total mn images)
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(522)
digit_to_show = 1
```

```
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(523)
digit_to_show = 2
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(524)
digit_to_show = 3
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(525)
digit_to_show = 4
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(526)
digit_to_show = 5
Digit = X[:, y==digit_to_show]
m, n = 5, 10
```

```
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(527)
digit_to_show = 6
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(528)
digit_to_show = 7
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(529)
digit_to_show = 8
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])

plt.subplot(5, 2, 10)
digit_to_show = 9
Digit = X[:, y==digit_to_show]
m, n = 5, 10
A = np.random.choice(np.arange(Digit.shape[1]), replace=False,
size=50)
```

```
Digit1 = Digit[:, A]
M = montage(Digit1, m, n)
plt.imshow(M, cmap = 'gray', interpolation = 'nearest')
plt.xticks([])
plt.yticks([])
plt.show()
```



討論：

- 先利用第一段程式碼定義 montage，再利用第二段程式碼讀取影像，第三段程式碼則是隨機從每個數字(0~9)抽取 50 個影像。
- 將抽取的影像一起排列便可觀察這些手寫數字的影像與品質。
- 數字的影像大致能辨認，但還是有少數影像品質不佳，較難辨認。

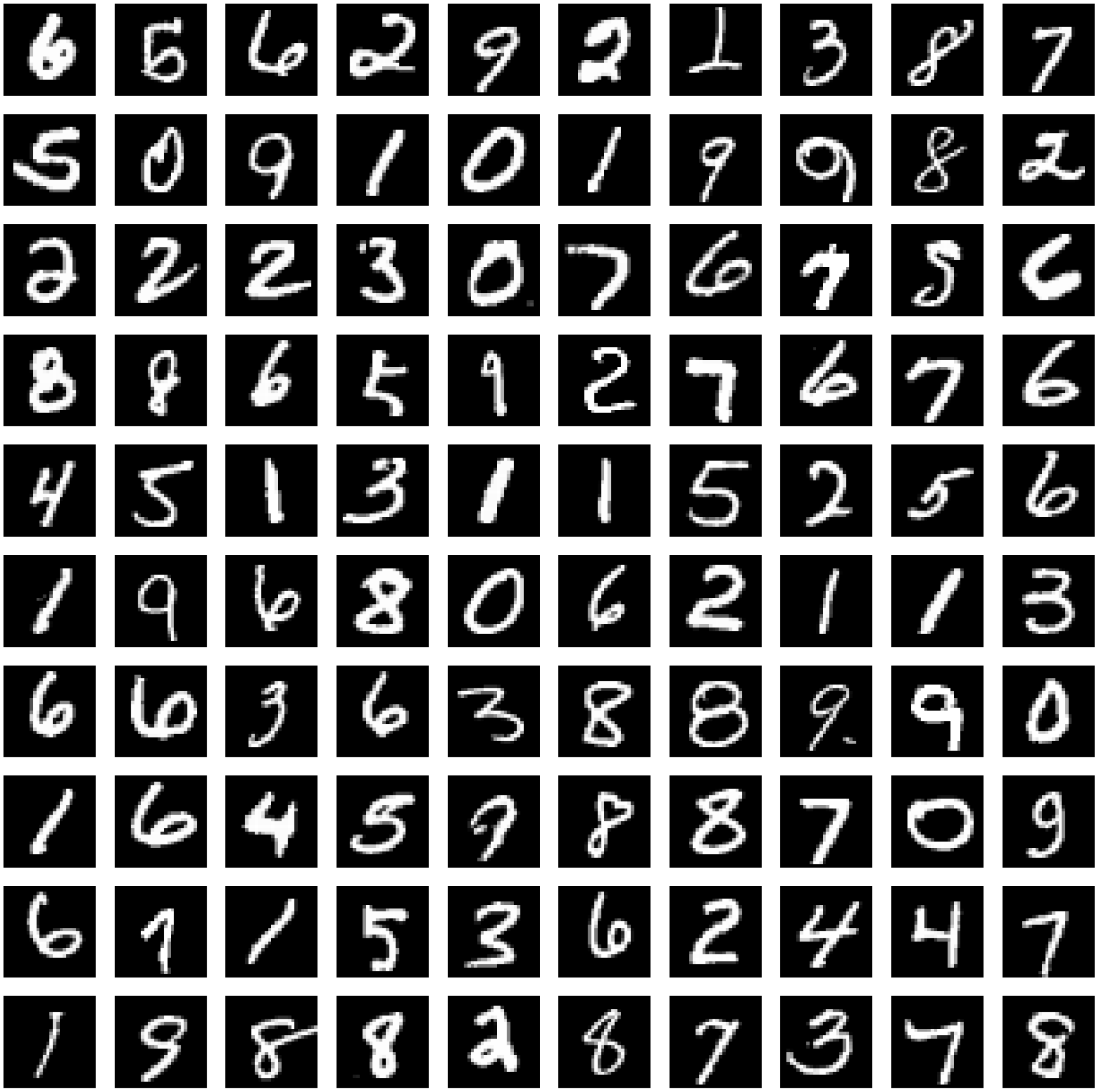
第 3 題：

每張大小 28×28 的手寫數字圖像 70000 張，不經壓縮前的儲存空間為 54.88 M Bytes。若進行 SVD 的 “Rank q approximation”，則壓縮倍數由 q 決定。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）。另外 q 的選擇可以根據 $\sigma_1, \sigma_2, \cdots, \sigma_r$ 的「能量配置」來決定，或說決定 q 之後，可以計算所採用的主成分的能量佔比，本題也可以順便列印出這個佔比。

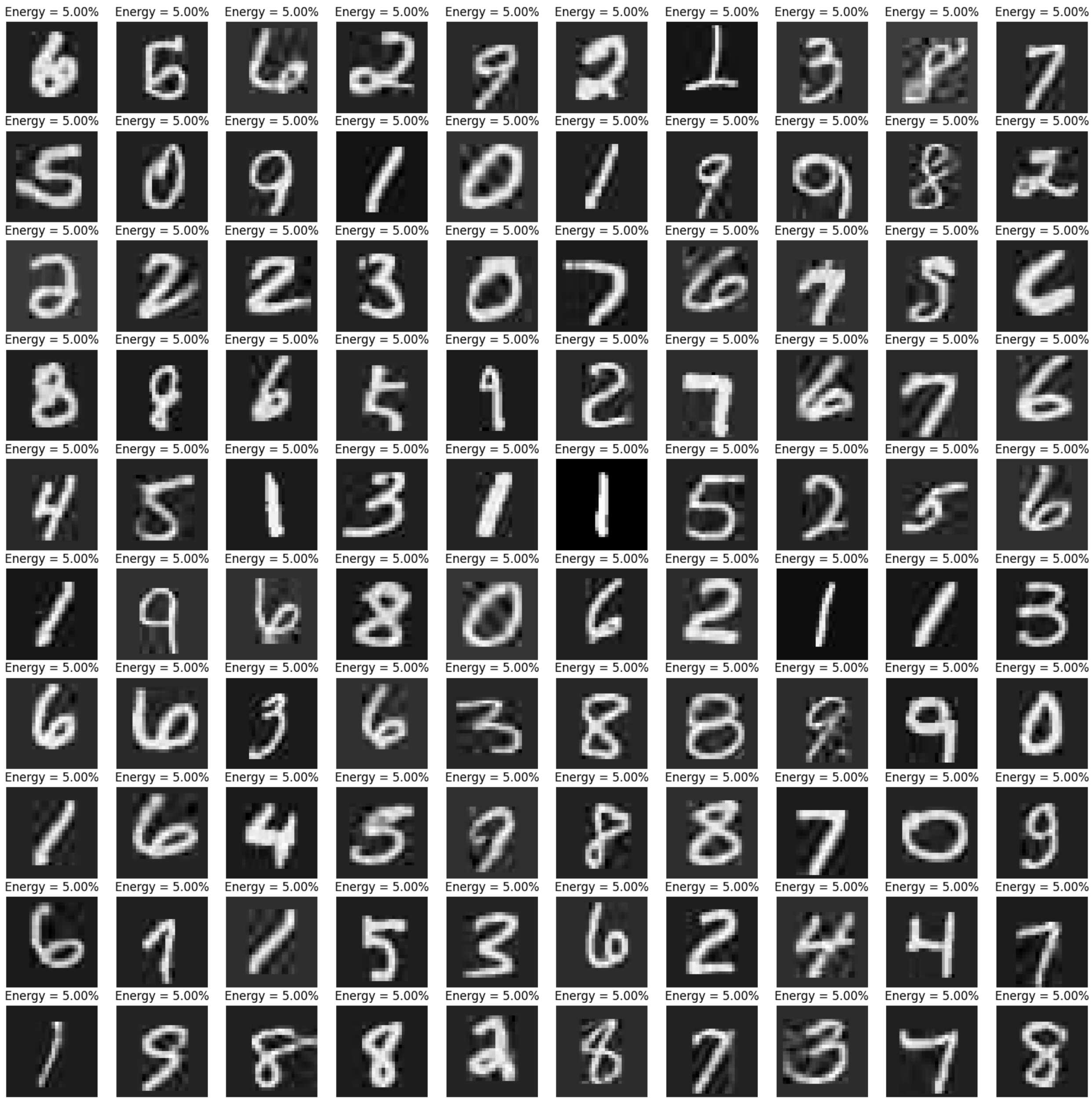
```
import numpy as np
import matplotlib.pyplot as plt

# 導入 MNIST 數據集
from scipy.io import loadmat
mnist = loadmat("mnist-original.mat")
X = mnist["data"].T  # 轉置以符合 SVD 的要求
y = mnist["label"][0]

# 選取 100 張手寫數字圖像
num_images = 100
sample_indices = np.random.choice(X.shape[0], num_images,
replace=False)
sample_images = X[sample_indices]

def svd_rank_q_approximation(image, q):
    """
    對圖像進行 SVD 的 Rank q approximation，返回壓縮後的圖像和相應的能量佔比
    """
    U, sigma, Vt = np.linalg.svd(image, full_matrices=False)

    # 保留前 q 個奇異值，並形成對角矩陣
    sigma_q = np.diag(sigma[:q])

    # 重構壓縮後的圖像
    compressed_image = U[:, :q] @ sigma_q @ Vt[:q, :]

    # 計算能量佔比
    energy_percentage = np.sum(sigma[:q]) / np.sum(sigma)

    return compressed_image, energy_percentage

# 設置不同的 q 值
q_values = [5, 10, 20, 50]
```

```
# 顯示原始圖像和壓縮後的圖像進行比較
plt.figure(figsize=(20, 20))
for i, index in enumerate(sample_indices):
    original_image = sample_images[i].reshape(28, 28)
    plt.subplot(10, num_images // 10, i + 1)
    plt.imshow(original_image, cmap='gray')
    plt.axis('off')
plt.suptitle('Original Images')
plt.show()

for q in q_values:
    plt.figure(figsize=(20, 20))
    for i, index in enumerate(sample_indices):
        original_image = sample_images[i].reshape(28, 28)
        compressed_image, energy_percentage =
svd_rank_q_approximation(original_image, q)
        plt.subplot(10, num_images // 10, i + 1)
        plt.imshow(compressed_image, cmap='gray')
        plt.axis('off')
        plt.title('Energy = {:.2f}%'.format(q, energy_percentage *
100))
    plt.suptitle('Compressed Images (q = {})'.format(q))
    plt.show()
```

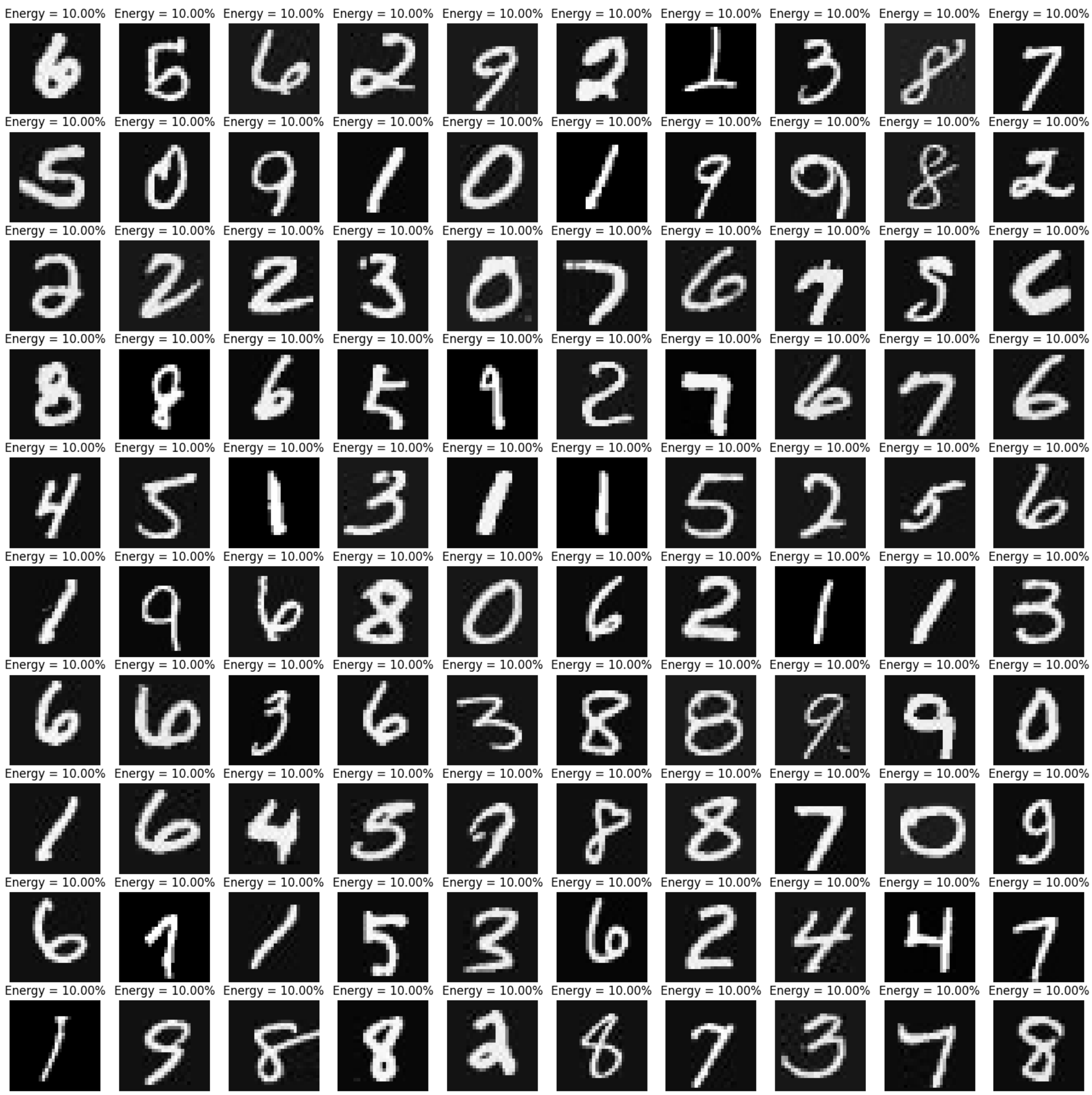
Original Images



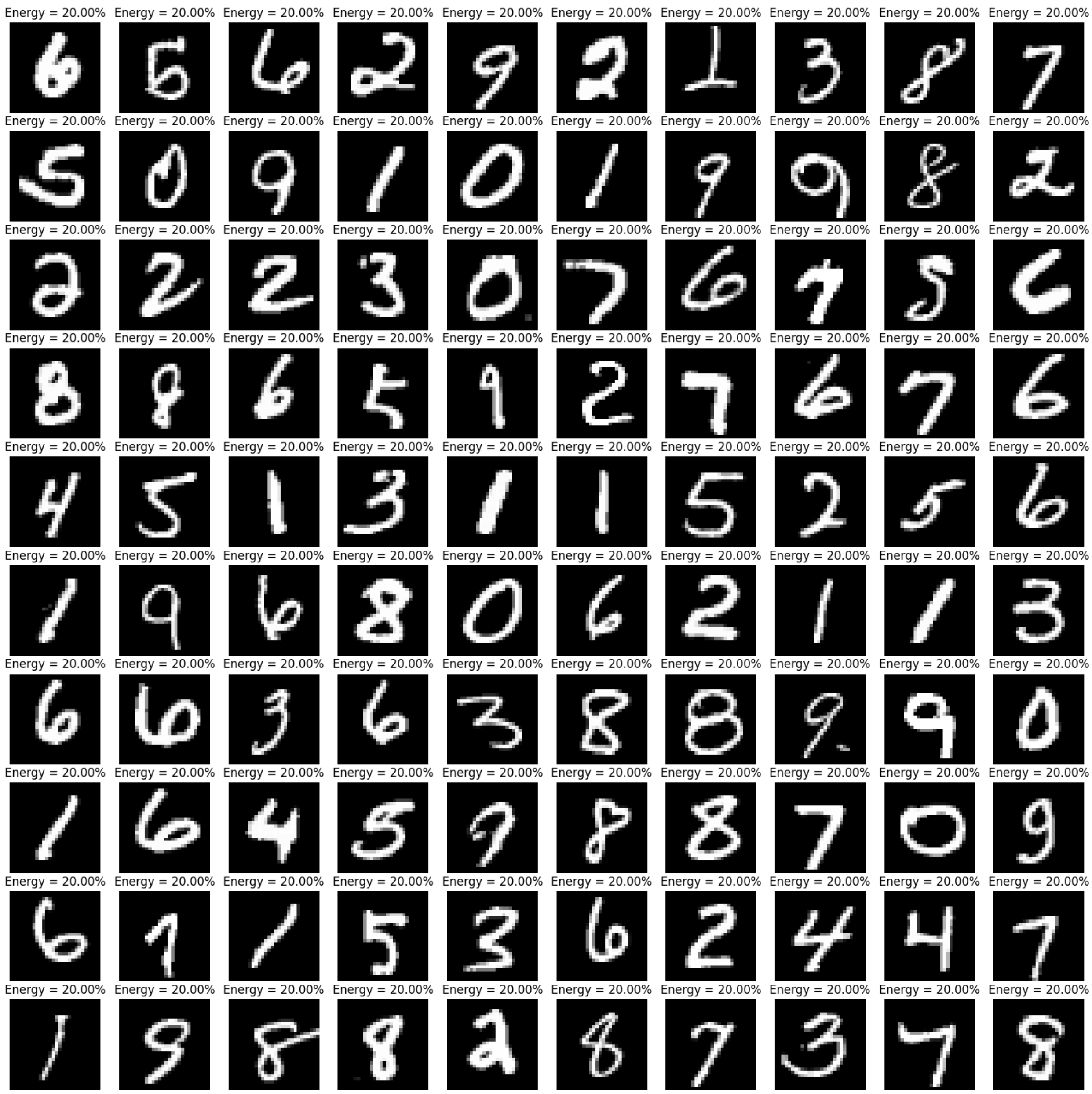
Compressed Images (q = 5)

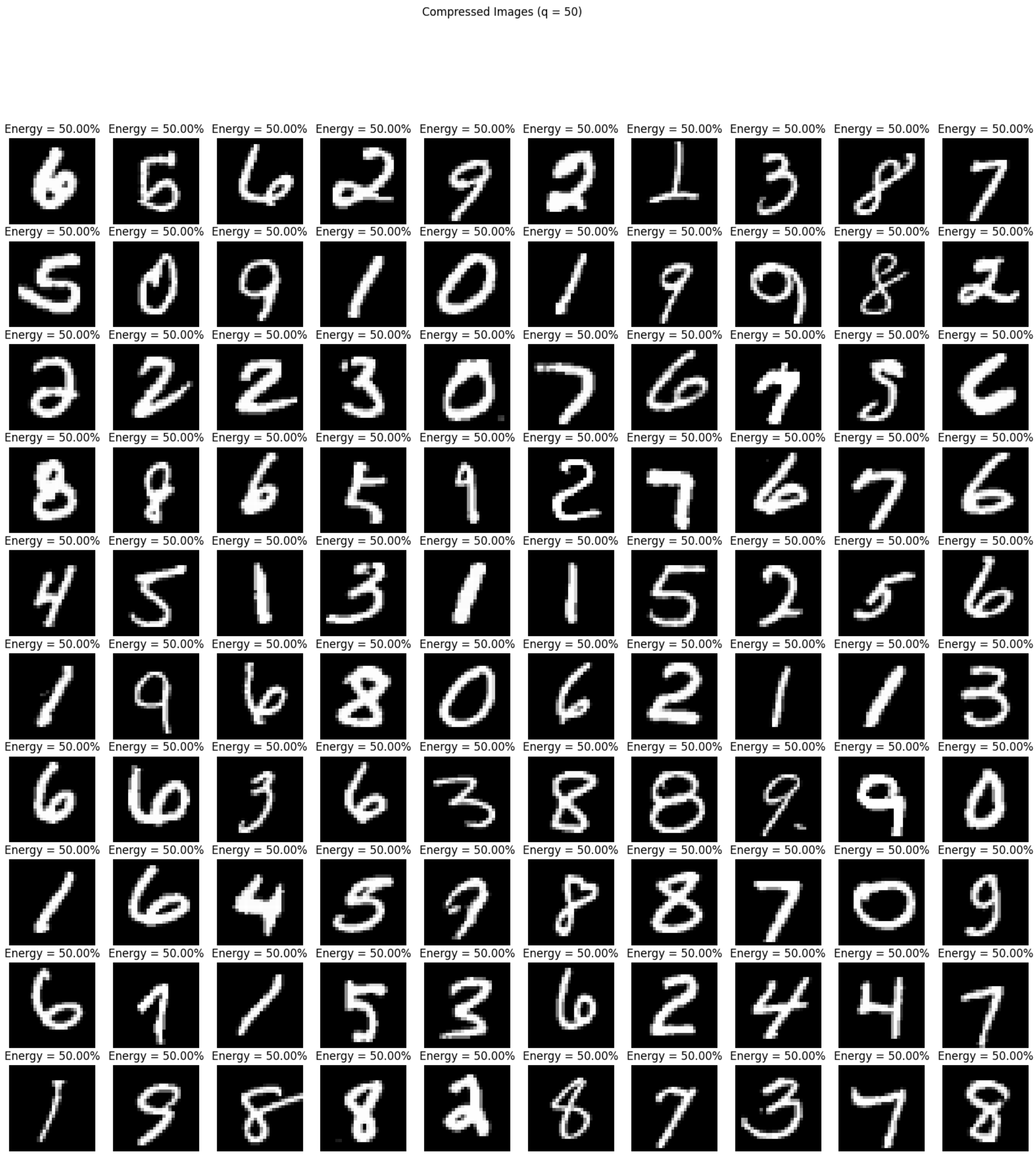


Compressed Images (q = 10)



Compressed Images (q = 20)





討論：

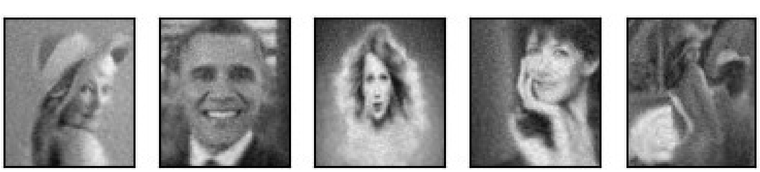
- 定義函數 svd_rank_q_approximation 來執行 SVD 和 Rank q approximation，並計算壓縮倍數和主成分的能量佔比。
- 當 q=5 時，主成分的能量佔比為 5%，影像非常模糊。
- 當 q=10 時，主成分的能量佔比為 10%，影像稍微清晰一些。
- 當 q=20 時，主成分的能量佔比為 20%，影像更為清晰。
- 當 q=50 時，主成分的能量佔比為 50%，影像與原始圖畫質接近。
- 由上可知，當 q 愈大，則影像愈清晰，愈接近原始圖。

Lesson 7

第 1 題：

有 5 張經過加密的影像圖，其加密的方式採 Yale Faces 38 人 2410 張人臉圖像矩陣 X 的 SVD，即 $X = U\Sigma V^T$，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。請注意：這 5 張影像圖的主成分採 q=2000，矩陣 X 先減去平均值，再執行 SVD 得到 U。

```python
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()

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

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

B = pd.read_csv('五張加密的影像_2024.csv')
avgFace = X.mean(axis = 1).reshape(-1, 1)
X_avg = X - np.tile(avgFace, (1, X.shape[1]))
U, E, VT = svd(X_avg, full_matrices= False)
q = 2000 # df=(2000, 5)
Uq = np.dot(U[:, :q], B)
show_montage(Uq, n, m, 1, 5)
```



討論：

- 先利用第一段程式碼定義 montage，再利用第二段程式碼解密影像。
- 若影像只有人臉時，則畫面品質較清晰；若影像含有人臉以外的部分，則畫面會較為模糊。