110-1 Deep Learning for Computer Vision

NTU GICE R10942051 Alex Chen 陳柏銍

Problem 1: Principal Component Analysis (100%)

Principal component analysis (PCA) is a technique of dimensionality reduction, which linearly maps data onto a lower-dimensional space, so that the variance of the projected data in the associated dimensions would be maximized. In this problem, you will perform PCA on a dataset of face images.

The folder `p1_data` contains face images of 40 different subjects (classes) and 10 grayscale images for each subject, all of size $(56, 46)$ pixels. Note that `i_j.png` is the j-th image of the i-th person, which is denoted as $\textbf{person}_i\textbf{image}_j$ for simplicity.

First, split the dataset into two subsets (i.e., training and testing sets). The first subset contains the first 9 images of each subject, while the second subset contains the remaining images. Thus, a total of $9 \times 40 = 360$ images are in the training set, and $1 \times 40 = 40$ images in the testing set.

In this problem, you will compute the eigenfaces of the training set, and project face images from both the training and testing sets onto the same feature space with reduced dimension.

【Data Preprocessing】

- Import Necessary Library

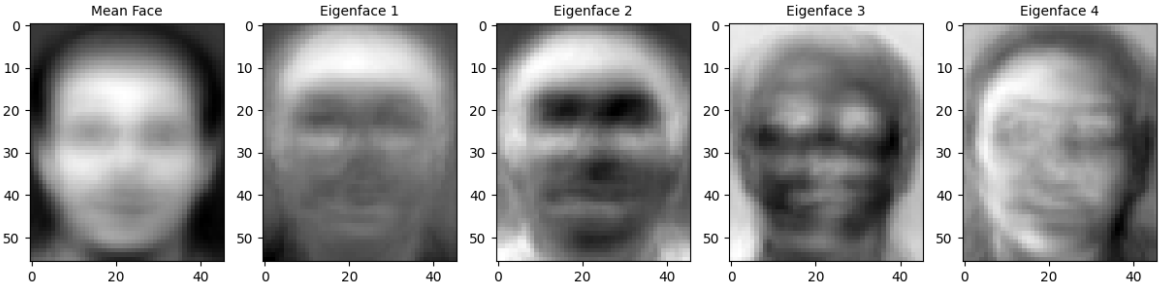
```
import os
import numpy as np
import pandas as pd
from pandas.plotting import table
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

- Read/Preprocessing Image File

```
path = "C:/Users/alex5/碩一(上)/Course/深度學習於電腦視覺/HW/HW0/p1_data"

x_train = []
y_train = []

x_test = []
y_test = []

for filename in os.listdir(path):
    # print(filename)
    # i location
    filename_underscore = filename.find("_")
    # print(filename_underscore)
    # j location
    filename_dot = filename.find(".")

    # i
    image_label = filename[0:filename_underscore]
    # print(image_label)
    # j
    image_number = filename[filename_underscore+1:filename_dot]
    # print(image_number)

    # 讀取每個image的絕對位置
    filepath = os.path.join(path, filename)
    # print(filepath)
    img = plt.imread(filepath, "RGB")
    # 拉成一個row vector
    img = img.reshape(-1)

    image_shape = plt.imread(os.path.join(path, "1_1.png")).shape
    # print(image_shape)
    if int(image_number) <= 9:
        x_train.append(img)
        y_train.append(image_label)
    else:
        x_test.append(img)
        y_test.append(image_label)
```

1. (20%) Perform PCA on the training set. Plot the mean face and the first four eigenfaces.

【Image Output for 1】



【Code】

```
# Problem 1
# 轉成np array
x_train = np.array(x_train)
y_train = np.array(y_train)

x_test = np.array(x_test)
y_test = np.array(y_test)

x_mean_face = np.mean(x_train, axis=0)

# image number of training set
N = x_train.shape[0]

pca = PCA(n_components = N-1)
pca_result = pca.fit(np.subtract(x_train, x_mean_face))

# 獲得前4個eigen值
eigenface1 = pca_result.components_[0].reshape(image_shape)
eigenface2 = pca_result.components_[1].reshape(image_shape)
eigenface3 = pca_result.components_[2].reshape(image_shape)
eigenface4 = pca_result.components_[3].reshape(image_shape)
x_mean_face = np.reshape(x_mean_face, image_shape)

plt.figure(figsize=(15, 10))
first = plt.subplot(1, 5, 1)
plt.imshow(x_mean_face, cmap = "gray")
plt.title("Mean Face", fontsize = 10)

second = plt.subplot(1, 5, 2)
plt.imshow(eigenface1, cmap = "gray")
plt.title("Eigenface 1", fontsize = 10)

third = plt.subplot(1, 5, 3)
plt.imshow(eigenface2, cmap = "gray")
plt.title("Eigenface 2", fontsize = 10)

fourth = plt.subplot(1, 5, 4)
plt.imshow(eigenface3, cmap = "gray")
plt.title("Eigenface 3", fontsize = 10)

fifth = plt.subplot(1, 5, 5)
plt.imshow(eigenface4, cmap = "gray")
plt.title("Eigenface 4", fontsize = 10)

plt.savefig("C:/Users/alex5/碩一(上)/Course/深度學習於電腦視覺/HW/HW0/output/problem1.png")
```

First, calculate the mean of the features which can get the mean face. As for the first four eigenfaces, using the operation taught in class to realize PCA and fit the model onto the data.

2. (20%) If the last digit of your student ID number is odd, take **person$_2$image$_1$**. If the last digit of your student ID number is even, take **person$_8$image$_1$**. Project it onto the PCA eigenspace you obtained above. Reconstruct this image using the first $n = 3, 50, 170, 240, 345$ eigenfaces. Plot the five reconstructed images.

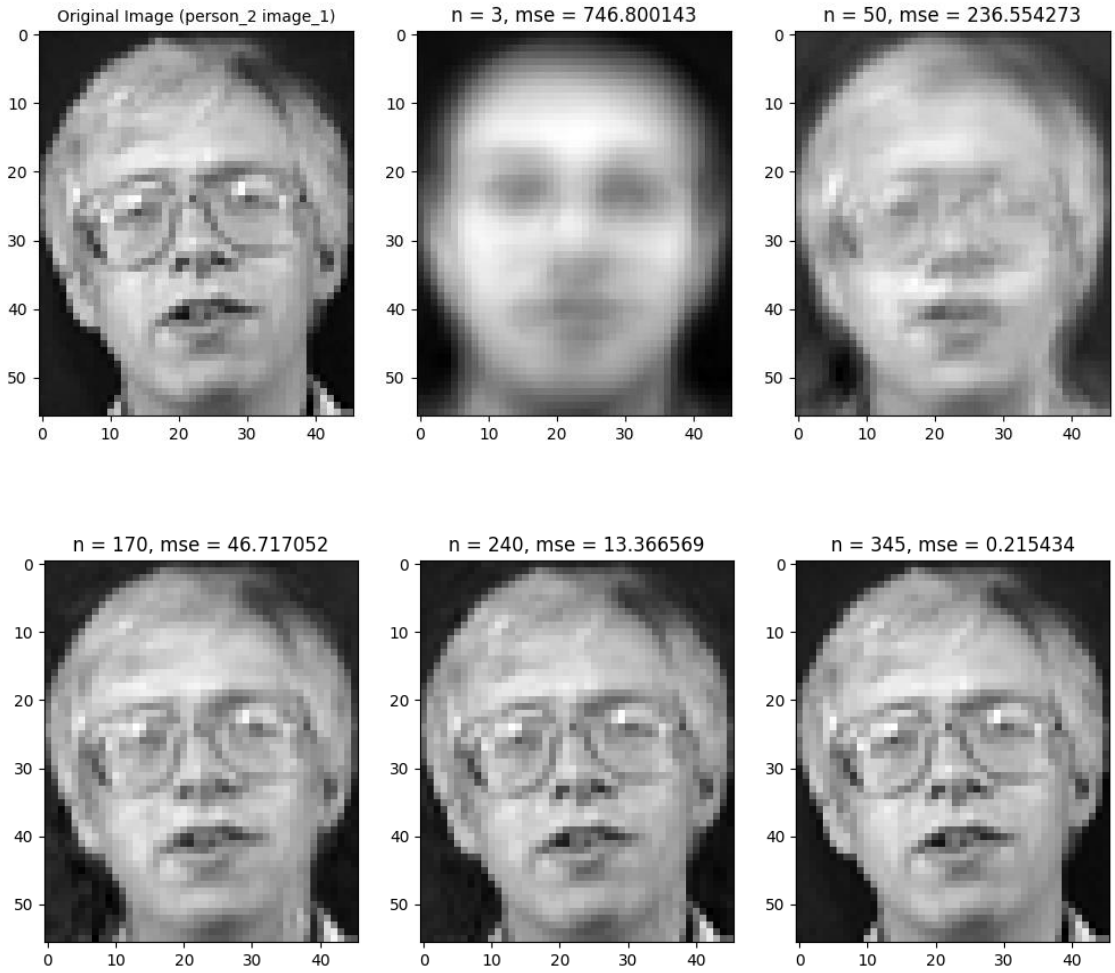
The last digit of my student ID is 1, so I am taking **person$_2$image$_1$**.

3. (20%) For each of the five images you obtained in 2., compute the mean squared error (MSE) between the reconstructed image and the original image. Record the corresponding MSE values in your report.

【Code】

```python
# Problem 2 & 3
# my student id: R10942051(odd), use person_2 image_1
original_face = plt.imread(os.path.join(path, str(2) + "_" + str(1) + ".png"), "RGB")
plt.figure(figsize = (25, 15))
original_image = plt.subplot(1, 6, 1)
plt.title("Original Image (person_2 image_1)", fontsize=10)
plt.imshow(original_face, cmap = "gray")

pca_original_image = pca_result.transform(np.subtract(original_face.reshape(1, -1), x_mean_face.reshape(-1)))

j = 2
for i in (3, 50, 170, 240, 345):
    image_pca = np.dot(pca_original_image[0,:i], pca_result.components_[:i]) + x_mean_face.reshape(-1)
    mse = mean_squared_error(image_pca.reshape((1, image_pca.shape[0])), original_face.reshape(1, -1)) * 255 * 255
    image_pca = image_pca.reshape(original_face.shape)
    plt.subplot(1, 6, j)
    plt.title("n = %s, mse = %.6f" % (i, mse))
    plt.imshow(image_pca, cmap = "gray")
    j = j + 1
plt.savefig("C:/Users/alex5/碩一(上)/Course/深度學習於電腦視覺/HW/HW0/output/problem2&3.png")
```

【Image Output for 2 & 3】



N	MSE
3	746.800143
50	236.554273
170	46.717052
240	13.366569
345	0.215434

First, I normalized the image by subtracting the mean face from the original face, and then project the data onto the PCA space. Second, I use the PCA space dimensions 3, 50, 170, 240, 345 to reconstruct the image. Last, calculate the mean squared error between the original image and the reconstructed image but because of I've normalized in the first step, I need to times 255^2 to obtain the desire MSE (*Hint*—When calculating MSE, your pixel values should be in the range of $[0,255]$).

4. (20%) Now, apply the *k*-nearest neighbors algorithm to classify the testing set images. First, you will need to determine the best k and n values by 3-fold cross-validation. For simplicity, the choices for such hyperparameters are $k = \{1, 3, 5\}$ and $n = \{3, 50, 170\}$. Show the cross-validation results and explain your choice for (k, n)

【Code】

```
# Problem 4
x_train_normalized = pca_result.transform(np.subtract(x_train, x_mean_face.reshape(-1)))
# y_train same as above

parameters = {"n_neighbors": [1, 3, 5]}
KNN = KNeighborsClassifier()
best_result = GridSearchCV(KNN, parameters, cv = 3)

df = dict()

for i in (3, 50, 170):
    best_result.fit(x_train_normalized[:, :i], y_train)
    df["n = " + str(i)] = np.array(best_result.cv_results_["mean_test_score"])

df = pd.DataFrame.from_dict(df, orient = "index")
df.columns = ["k = 1", "k = 3", "k = 5"]
df.index = ["n = 3", "n = 50", "n = 170"]

fig = plt.figure()
ax = fig.add_subplot(111, frame_on = False)
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)

table(ax, df, loc = "center")
plt.savefig("C:/Users/alex5/碩一(上)/Course/深度學習於電腦視覺/HW/HW0/output/problem4.png")
```

【Image Output for 4】

	k = 1	k = 3	k = 5
n = 3	0.65	0.6055555555555555	0.5472222222222222
n = 50	0.9611111111111111	0.9055555555555556	0.8555555555555556
n = 170	0.9555555555555556	0.8916666666666666	0.8388888888888889

First, I normalize and project the training data onto the PCA subspace and then configure $k - Nearest\ Neighbors\ Model$ with k equals to 1, 3 and 5. Second, perform 3fold cross-validation and last, show the cross-validation results in terms of mean test score. My choice for (k, n) will explain on 5.

5. (20%) Use your hyperparameter choice in 4., and report the recognition rate of the testing set.

【Code】

```
# Problem 5
# my_best_parameters
k = 1
n = 50

pca_test = pca_result.transform(np.subtract(x_test, x_mean_face.reshape(-1)))

# x_train_normalized same as above
# y_train same as above

KNN_optional = KNeighborsClassifier(n_neighbors = k)
KNN_optional.fit(x_train_normalized[:, :n], y_train)
prediction = KNN_optional.predict(pca_test[:, :n])
print("k = ", k)
print("n = ", n)
print("Accuracy on the testing set: ", accuracy_score(y_true = y_test, y_pred = prediction))
```

【Image Output for 5】

```
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  1
n =  3
Accuracy on the testing set:  0.725
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  1
n =  50
Accuracy on the testing set:  0.925
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  1
n =  170
Accuracy on the testing set:  0.95
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  3
n =  3
Accuracy on the testing set:  0.65
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  3
n =  50
Accuracy on the testing set:  0.9
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  3
n =  170
Accuracy on the testing set:  0.875
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  5
n =  3
Accuracy on the testing set:  0.575
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  5
n =  50
Accuracy on the testing set:  0.875
PS C:\Users\alex5\碩一(上)\Course\深度學習於電腦視覺\HW\HW0> python ./pca.py
k =  5
n =  170
Accuracy on the testing set:  0.9
```

<Accuracy on the testing set>

n \ k	1	3	5
3	0.725	0.65	0.575
50	0.925	0.9	0.875
170	0.95	0.875	0.9

My choice for the hyperparameters (k, n) will be $(\mathbf{1}, \mathbf{50})$. The reason is that if we take a look at the accuracy on the testing set, we can find out that using (k, n) being $(1, 170)$ get a higher accuracy than (k, n) being $(1, 50)$. However, we only get a 0.025 increase in accuracy, but we need to increase the dimension by 240% which I think it may be not cost-effective.

The accuracy on the testing set is **0.925** using the $(k, n) = (1, 50)$.