محدثه حکیمی – ۴۰۱۲۴۳۰۴۵

گزارش تمرین ۵ کامپیوتری جبر خطی

سؤال یک:

الف) برای این بخش از سؤال باید اول svd رو پیاده‌سازی کنیم. برای لودکردن عکس‌ها از کتابخونه‌ی cv2 استفاده کردم.

```python
import os

import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```python
def svd_decomposition(matrix, precision=np.float64):
    matrix = np.array(matrix, dtype=precision)
    transpose_dot_matrix = np.dot(matrix.T, matrix)
    eigen_values, eigen_vectors = np.linalg.eig(transpose_dot_matrix)
    # sort eigen values and vectors
    idx = eigen_values.argsort()[::-1]
    eigen_values = eigen_values[idx]
    eigen_vectors = eigen_vectors[:, idx]

    # create V, no need to normalize eigen vectors
    v = eigen_vectors
    v_t = v.T
```

یک تابع برای تجزیه‌ی svd می‌نویسیم و اول از همه ماتریس V رو می‌سازیم. برای این کار باید ماتریس A^T A رو حساب کنیم، مقادیر و بردارهای ویژه‌ش رو حساب و سورت کنیم، و در آخر ماتریس V برابر با بردارهای ویژه است. یک precision رو هم در تابع مشخص کردم که برای بالابردن دقت محاسبه‌ی مقادیر ویژه است.

حالا ماتریس سیگما باید ساخته بشه:

```python
# create sigma(E)
singular_values = np.sqrt(eigen_values)
sigma = np.zeros(matrix.shape)
for i in range(len(sigma[0])):
    sigma[i, i] = singular_values[i]
```

این ماتریس درواقع ماتریس قطری مقادیر تکین ماتریس A^TA هست. و تعداد سطرها و ستون‌هاش برابر با ماتریس اصلیه.

و ماتریس U:

برای ماتریس U کافیه کاری که برای رسیدن به ماتریس V انجام دادیم رو بریم با این تفاوت که این بار با ماتریس AA^T جلو می‌ریم.

```
    # create U, AA^T = UΣ^2U^T
    matrix_dot_transpose = np.dot(matrix, matrix.T)
    eigen_values, eigen_vectors = np.linalg.eig(matrix_dot_transpose)
    # sort eigen values and vectors
    idx = eigen_values.argsort()[::-1]
    eigen_vectors = eigen_vectors[:, idx]

    u = eigen_vectors

    return u, sigma, v_t
```

یک تابع هم برای لودکردن تصاویر نوشتم و از کتابخونه‌ی os استفاده کردم:

```
def load_all_images(dir_path):
    images = {}
    for file in os.listdir(dir_path):
        if file.endswith('.jpg'):
            img = cv2.imread(dir_path + file, cv2.IMREAD_GRAYSCALE)
            images[file] = img

    return images
```

برای پیداکردن بهترین match با هر تصویر هم یک تابع نوشتم که عکس با مینیمم فاصله از عکس داده‌شده رو پیدا می‌کنه و برای این کار از مقادیر تکین دو تصویر استفاده می‌کنه:

```
# use singular values to find the image with the best match to the one given
def find_best_match(img_path, images):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    u, sigma, v_t = svd_decomposition(img)

    min_diff = float('inf')
    best_match = ''
    for key, value in images.items():
        u, sigma_i, v_t = svd_decomposition(value)
        diff = np.linalg.norm(sigma - sigma_i)
        if diff < min_diff:
            min_diff = diff
            best_match = key

    return best_match
```

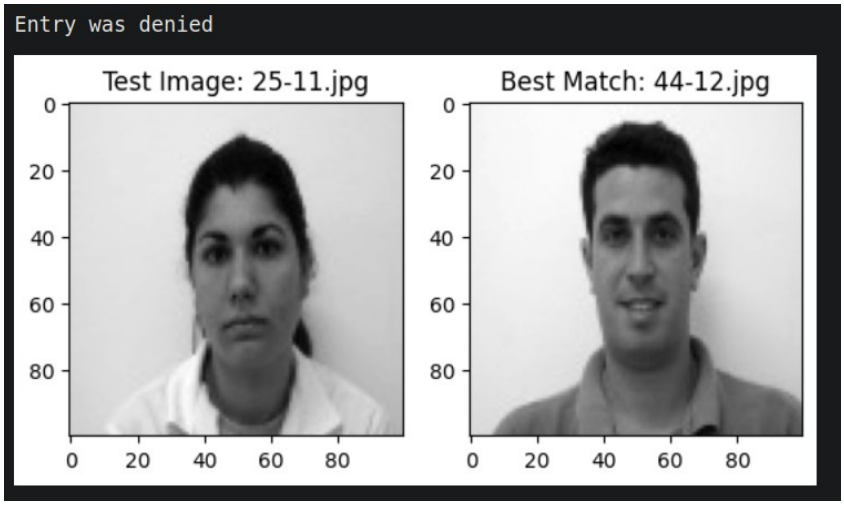
در آخر هم با استفاده از توابع بالا روی testset الگوریتم رو تست می‌کنیم، به این شکل که اگر تصویر match شده، شناسه‌ی برابر با تصویر تست داشت، به عنوان یک پیش‌بینی درست در نظر می‌گیریم:

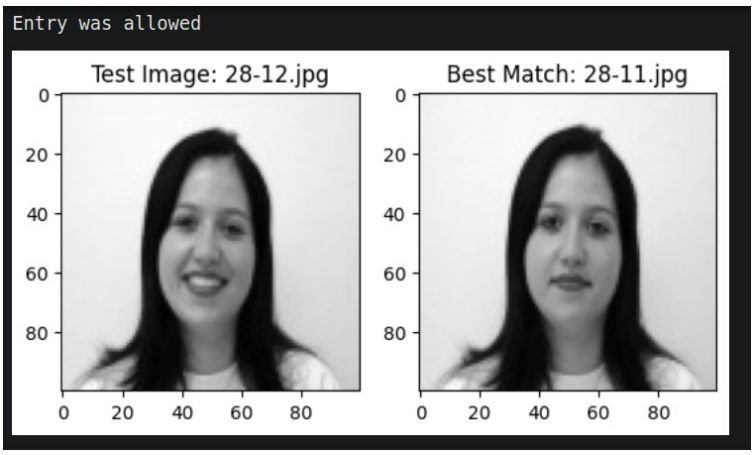
```
# test the function for accuracy
images = load_all_images('CQ1/Testset/')
img_path = ''
correct = 0
total = 0
for key, value in images.items():
    best_match = find_best_match('CQ1/Testset/' + key, load_all_images('CQ1/Dataset/'))
    if best_match.split('-')[0] == key.split('-')[0]:
        correct += 1
        print("Entry was allowed")
    else:
        print("Entry was denied")
    total += 1
```

```
    # plot the images
    plt.subplot(1, 2, 1)
    plt.imshow(value, cmap='gray')
    plt.title(f'Test Image: {key}')
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.imread('CQ1/Dataset/' + best_match, cv2.IMREAD_GRAYSCALE), cmap='gray')
    plt.title(f'Best Match: {best_match}')
    plt.show()

print(f'Accuracy: {correct}/{total}')
```

و خروجی نمونه:






```
Accuracy: 24/33
```

می‌بینیم که دقت ۲۴/۳۳ یا ۷۲.۷۲ درصد بوده.
پ.ن: باقی خروجی‌ها رو طبیعتاً نیاوردم که گزارش شلوغ نشه.

ب) برای این بخش هم ابتدا تابع pca رو می‌نویسیم. یک تابع هم برای flat کردن تصاویر داریم:

```python
# flatten all images
def flatten_images(images):
    image_titles = list(images.keys())
    flattened_images = np.array([image.flatten() for image in images.values()])
    return flattened_images, image_titles
```

حالا تابع pca رو بخش به بخش میارم و توضیح می‌دم، کد هم کامنت داره:

```python
def pca(images, k):
    # 1. calculate the mean for all images
    mean_image = np.mean(images, axis=0)
    zero_mean_images = images - mean_image
```

این قسمت که مشخصه، میانگین تصاویر رو پیدا می‌کنیم و از تصاویر کم می‌کنیم.

حالا باید ماتریس کوواریانس رو بسازیم. اول با استفاده از فرمول سعی کردم بسازمش اما انقدر بزرگ بود که عملاً ران‌کردنش ساعتها طول می‌کشید. در آخر تصمیم گرفتم کد رو کامنت کنم و از np.cov استفاده کنم:

```python
    # 2. find the S matrix, that is the covariance matrix of the zero mean images
    # S = 1/(N-1) * UE^2U^T
    # U, sigma, _ = svd_decomposition(zero_mean_images)
    # sigma = np.square(sigma)
    # S = np.dot(np.dot(U, sigma), U.T) / (len(images) - 1)
    S = np.cov(zero_mean_images, rowvar=False)
```

در قدم بعدی هم بردارها و مقادیر ویژه‌ش رو سورت می‌کنیم و kتای اول رو همراه تصویر میانگین برمی‌گردونیم:

```
    # 3. find the eigen values and eigen vectors of the S matrix
    eigen_values, eigen_vectors = np.linalg.eigh(S)
    # sort eigen values and vectors
    idx = eigen_values.argsort()[::-1]
    eigen_vectors = eigen_vectors[:, idx]

    # 4. use eigen vectors that have the highest eigenvalue/trace
    p = eigen_vectors[:, :k]

    return mean_image, p
```

یک تابع هم برای ساختن دوباره‌ی عکس‌ها به کمک بردارهای ویژه‌ی انتخاب‌شده داریم:

```
def reconstruct_image(image, mean_image, p):
    zero_mean_image = image - mean_image
    # 5. project the zero mean images onto the eigen vectors
    projected_image = np.dot(zero_mean_image, p)
    # 6. reconstruct images using the selected eigen vectors
    reconstructed_image = np.dot(projected_image, p.T) + mean_image
    return reconstructed_image
```

در آخر هم تست توابع بالا که خیلی توضیح خاصی نداره، از k برابر ۱ تا ۲۵ جلو می‌ریم و دقت رو محاسبه می‌کنیم، تا حد زیادی هم مشابه بخش الف هست:

```
# test the function for accuracy
def euclidean_distance(x, y):
    return np.linalg.norm(x - y)

def find_best_match_pca(img_path, images, k):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    flattened_images, image_titles = flatten_images(images)
    mean_image, p = pca(flattened_images, k)

    img = img.flatten()
    reconstructed_img = reconstruct_image(img, mean_image, p)
    min_diff = float('inf')
    best_match = ''
    for i in range(len(flattened_images)):
        diff = euclidean_distance(reconstructed_img, flattened_images[i])
        if diff < min_diff:
            min_diff = diff
```

```
            min_diff = diff
            best_match = image_titles[i]

    return best_match

images = load_all_images('CQ1/Dataset/')
flattened_images, image_titles = flatten_images(images)

correct = np.zeros(25)
total = 33
```

```
# perform pca for k in 1 to 25
for k in range(1, 26):
    mean_image, p = pca(flattened_images, k)

    reconstructed_images = np.array([reconstruct_image(image, mean_image, p) for image in flattened_images])

    # reshape the images
    reconstructed_images = np.array([image.reshape(100, 100) for image in reconstructed_images])

    correct[k-1] = 0
    for i in range(len(flattened_images)):
        best_match = find_best_match_pca('CQ1/Dataset/' + image_titles[i], images, k)
        if best_match.split('-')[0] == image_titles[i].split('-')[0]:
            correct[k-1] += 1
            print("Entry was allowed")
        else:
            print("Entry was denied")
```

```
            # plot the images
            plt.subplot(1, 2, 1)
            plt.imshow(cv2.imread('CQ1/Dataset/' + image_titles[i], cv2.IMREAD_GRAYSCALE), cmap='gray')
            plt.title(f'Test Image: {image_titles[i]}')
            plt.subplot(1, 2, 2)
            plt.imshow(reconstructed_images[i], cmap='gray')
            plt.title(f'Best Match: {best_match}')
            plt.show()

        print(f'Accuracy for k={k}: {correct[k-1]}/{total}')

# plot the accuracy over k
plt.plot(np.arange(1, 26), correct/total)
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k')
plt.show()
```

و اما اتفاقی که در نهایت موقع ران‌کردن کد افتاد این بود که حتی برای فقط $k = 1$ بیشتر از یک ساعت طول کشید و همون هم کامل انجام نشد. هم داخل کولب و هم لپ‌تاپ خودم هیچ نتیجه‌ی درستی نتونستم بگیرم. کد رو هم هرچی بالا پایین کردم به جایی نرسید.
خروجی رو تا جایی که اجرا شده اینجا می‌تونید ببینید.

ج) مدت زمان اجرای pca به وضوح خیلی بیشتر از svdئه. اما برای kهای مناسب دقتش می‌تونه بیشتر از svd باشه. ماتریس کوواریانس ایجادشده و محاسبه‌ی مقادیر و بردارهای ویژه‌ش زیادی وقت‌گیره. در حالت کلی بهتره برای دیتای ساده‌تر و کم‌حجم‌تر از pca و از svd برای حالت‌های دیگه استفاده کنیم.