COMPUTER VISION Lab 03

PCA (Principal Component Analysis)

과목명 : [CSI4116-01] COMPUTER VISION
담당교수 : 연세대학교 컴퓨터공학과 변혜란
학번 : S20131598
이름 : 전상현

REPORT

I. 개발 목표

- PCA(Principal Component Analysis)를 이용하여, 입력받은 이미지들을 고차원에서 저차원의 데이터로 변환한 후 학습한다. 학습된 이미지들을 통하여 입력받은 이미지와 가장 유사한 이미지를 총 3개까지 출력하는 프로그램을 작성한다. Original PCA 이외에도, PCA with Trick, PCA with SVD를 사용한 후 각각 알고리즘의 수행속도와 정확도를 비교하도록 한다.

II. 개발 범위 및 내용

가. 개발 범위

1. PCA Original Algorithm
2. PCA Trick Algorithm
3. PCA with SVD Algorithm

나. 개발 내용

1. PCA Original Algorithm (not implemented)

- 주성분 분석(PCA, Principal Component Analysis)을 이용하여, 입력받은 이미지를 고차원에서 저차원의 데이터로 변환한 후 학습한다.
- 학습된 이미지들을 통하여 입력받은 이미지와 유사한 이미지를 판별한다.

2. PCA Trick Algorithm

- 기존 PCA에서 공분산(Covariance)를 구하는 방식에서 AA^T 대신에 A^TA를 사용함으로서, 연산량을 크게 줄이는 방식으로 PCA를 구현하였다.

3. PCA with SVD Algorithm

- SVD(Singular-Value Decomposition) 방법을 이용하여 PCA를 구현하였다.

III. 결과

1. 제작 내용

__main__

```
if __name__ == "__main__":
    original_imgs = []
    train_imgs = []
    test_imgs = []
    centered_imgs = []
    mean_img = np.zeros([SIZE, ])
    t = ()

    '''...'''

    for i in range(IMG_NUM):
        original_imgs.append(np.array(io.imread('train/s' + str(i+1) + '.1.tif')))
        test_imgs.append(np.array(io.imread('test/s' + str(i+1) + '.2.tif').reshape([SIZE, ])))
        train_imgs.append(np.array(original_imgs[i].reshape([SIZE, ]), dtype='float64'))
        mean_img += train_imgs[i]
    else:
        mean_img /= IMG_NUM

    for i in range(IMG_NUM):
        centered_imgs.append(train_imgs[i] - mean_img)
        t += (centered_imgs[i], )

    train_mat = np.stack(t, axis=1)

    start = timeit.default_timer()
    proj_mat_trick = projection_mat_with_trick(train_mat)
    end = timeit.default_timer()
    print("Time consumtion of PCA trick algorithm is %.6fsec" %(end - start))

    start = timeit.default_timer()
    proj_mat_svd = projection_mat_with_svd(train_mat)
    end = timeit.default_timer()
    print("Time consumtion of PCA SVD algorithm is %.6fsec" % (end - start))

    projected_mat_trick = np.dot(proj_mat_trick, train_mat)
    projected_mat_svd = np.dot(proj_mat_svd, train_mat)

    print("Accuracy of result_train_trick is %.2f%%" % identify_imgs(original_imgs, mean_img, train_imgs, proj_mat_trick
    print("Accuracy of result_test_trick is %.2f%%" % identify_imgs(original_imgs, mean_img, test_imgs, proj_mat_trick,
    print("Accuracy of result_train_svd is %.2f%%" % identify_imgs(original_imgs, mean_img, train_imgs, proj_mat_svd, pr
    print("Accuracy of result_test_svd is %.2f%%" % identify_imgs(original_imgs, mean_img, test_imgs, proj_mat_svd, proj
```

- Main 에서는 PCA에서 공통적으로 수행하는 부분에 대하여 수행한 후, Trick과 SVD Method로 나머지 PCA를 수행하는 함수를 호출한다. 또한 각각의 알고리즘의 수행시간을 측정하고 출력한다.
- 선택된 알고리즘으로 수행한 결과 값을 해당 변수에 저장하고, 각각의 입력에 대해 Recognizing을 수행하는 함수를 호출한다. 또한 Recognition의 결과에 따른 Accuracy를 출력한다.

projection_mat_with_trick

```
def projection_mat_with_trick(eigfaces):
    # With original PCA, Compute the covariance matrix summation(XX^T)
    # But it is too large, Compute X^TX instead, then we easily get eigenvectors and eigenvalue

    cov = np.dot(eigfaces.transpose(), eigfaces)
    eigvals, eigvecs = np.linalg.eig(cov)

    idx = eigvals.argsort()[::-1]
    eigspace = np.dot(eigfaces, eigvecs)[:,idx][:,0:DIM].real

    return normalize_and_transpose_eigspace(eigspace, np.sqrt(eigvals[idx][0:DIM]))
```

- Trick Method 에서는 공분산(Covariance) 계산시에 AA^T 대신에 A^TA를 사용함으로서 계산량을 크게 줄일 수 있다. 실제 수식을 통하여 AA^T의 Eigenvector를 쉽게 구할 수 있다.

projection_mat_with_svd

```
def projection_mat_with_svd(eigfaces):
    # With SVD(Singular Value Decompostion), We do not need to calculate eigenvectors and eigenvalue

    U, S, V = np.linalg.svd(eigfaces, full_matrices=False)

    idx = S.argsort()[::-1]
    eigspace = np.dot(eigfaces, V.transpose())[:,idx][:,0:DIM].real

    return normalize_and_transpose_eigspace(eigspace, S[idx][0:DIM])
```

- SVD Method에서는 공분산을 구하는 대신, SVD(Singular-Value Decomposition)을 수행함으로서, Eigenvector와 Eigenvalue를 구할 수 있다.

normalize_and_transpose_eigspace

```
def normalize_and_transpose_eigspace(eigspace, size):
    # If we use trick or svd method, then calculated eigenvector is not unit vector, so we need to normalize.
    # Returning transform of projection matrix, we can reduce computation

    for i in range(SIZE):
        eigspace[i,:] /= size
    return eigspace.transpose()
```

- 입력 받은 Eigenspace의 열벡터를 크기가 1인 Unit Vector로 변경한 후, transpose한 값을 Return 한다. Transpose한 Matrix를 Return 함으로써 이미지를 identify 할 때 계산량을 줄여줄 수 있다.

identify_imgs

```
def identify_imgs(original_imgs, mean_img, test_imgs, proj_mat, projected_mat, path):
    cnt = 0
    for i in range(IMG_NUM):
        diff = projected_mat - np.expand_dims(np.dot(proj_mat, (test_imgs[i] - mean_img)), 1)
        dist = np.linalg.norm(diff, axis=0)
        idx = dist.argsort()

        if idx[0] == i:
            cnt += 1

        io.imsave(path + '/s' + str(i+1) + '_3.tif', np.concatenate((original_imgs[idx[0]], original_imgs[idx[1]], origi
    return cnt / IMG_NUM * 100.0
```

- 입력 받은 이미지들을 Train 되어진 이미지와 비교하여, 가장 비슷한 순서대로 3개의 이미지를 저장한다. 전체 이미지에 대한 비교를 끝낸 후 Accuarcy를 Return 한다.

2. 시험 및 평가 내용

A. Time Comsumption

```
        start = timeit.default_timer()
        proj_mat_trick = projection_mat_with_trick(train_mat)
        end = timeit.default_timer()
        print("Time consumtion of PCA trick algorithm is %.6fsec" %(end - start))
```

그림 1. 수행 시간 측정 코드

	1회	2회	3회	4회	5회	평균(sec)
Trick	0.115117	0.078738	0.094693	0.073170	0.088189	0.089981
SVD	0.139133	0.140820	0.158267	0.129720	0.140000	0.141588

표 1. 수행 시간 측정 결과표

프로그램에서 공통적으로 수행하는 부분을 제외하고, 각 알고리즘에 따른 걸리는 시간만을 측정하였다. 이 경우 Trick 알고리즘이 SVD 보다 조금 더 빠름을 확인하였다. 실제 연산 시에 Trick 알고리즘에서 (40, 40) 크기의 Matrix에 대하여 Eigenpair 쌍을 구하는 시간보다 SVD 알고리즘에서 (10304, 10304) 크기의 Matrix를 Singular-Value Decomposition을 수행하는 시간이 더 오래 걸림을 확인하였다.

B. Accuracy

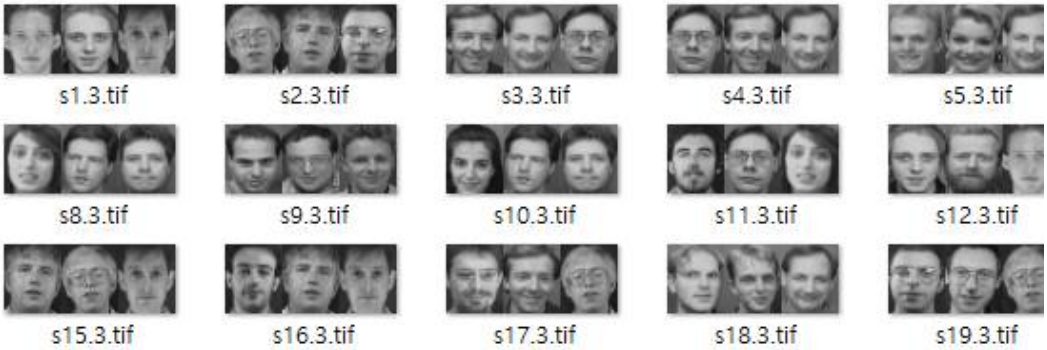

그림 2. 출력 이미지 예시

	Train	Test
Trick	40/40 (100.00%)	35/40 (87.50%)
SVD	40/40 (100.00%)	35/40 (87.50%)

표 2. 정확도 측정 결과표

프로그램에서 학습한 이미지를 통하여, 실제 입력된 이미지와 같은 학습된 이미지를 출력하는지 여부를 통하여 정확도를 계산하였다. 실제 학습된 이미지를 다시 입력으로 주었을 때 정확도가 100% 임을 확인하였으며, 테스트 이미지를 입력으로 주었을 때에도 비교적 높은 정확도를 얻을 수 있었다. 또한 DIM이 증가함에 따라서 정확도가 조금씩 증가함을 확인하였다.

Ⅳ. 기타

1. 느낀 점

수업시간에 다루었던 주성분 분석(PCA, Principal Component Analysis)을 이용하여 이미지를 처리해보았다. 실제 프로그램 작성 시에는 Eigenvector의 크기가 unit vector가 아님으로서 인해 Accuracy가 감소하는 현상이 발생하였다. 따라서 직접 Eigenvector를 Eigenvalue의 제곱근 값으로 나눔을 통해 해당 문제를 해결하였다. 또한 실제 numpy 모듈안의 여러 가지 기능을 사용함으로서, numpy 모듈을 익히는데 큰 도움이 되었다.