HW10 REPORT

2016024893 오성준

과제의 구현1

```
def fourier(name,a,b,i):
    global mean_mat
    img_ = cv2.imread(name, cv2.IMREAD_GRAYSCALE)
    img__=cv2.resize(img_, dsize=(128,128))
    img=img__[a:a+64,b:b+64]
    f=np.fft.fft2(img)
    fshift=np.fft.fftshift(f)
    magnitude=32*np.log(np.abs(fshift)+1)
    sum=0
    for k in range(64):
        for j in range(64): sum+=magnitude[k][j]**2
    sum=np.sqrt(sum)
    magnitude=magnitude/sum

    matrix=magnitude[32:64,32:64]
    vector=matrix.flatten()
    tmp=np.dot(vector,vector.T)
    vector=vector*10/np.sqrt(tmp) #값들을 normalization 시킴
    mean_mat[i]+=vector
```

- 파일을 읽고나서 FFT2와 FFTSHIFT라이브러리를 통해서 계수들을 구했습니다.
- 128X128로 이미지를 줄여서 최대한 많이 PATTERN을 읽을 수 있도록 했습니다.
- MAGNITUDE를 구할 때, ABS한 값에 1을 더한 것은 FSHIFT값에 0이 있는게 생겨서 프로그램에 에러가 나서 그렇게 했습니다.
- MAGNITUDE 값의 범위가 64X64를 잡을 때마다 약간씩 달라졌기 때문에 이에 대한 편차를 줄이고자 NORMALIZATION을 취했습니다. (밝기에 의한 차이를 최소화하려 했습니다.
- VECTOR는 오른쪽 아래부분을 사용했습니다.

과제의 구현2

```
def decision(name,a,b):
    global mean_mat
    #print(str)
    img_ = cv2.imread(name, cv2.IMREAD_GRAYSCALE)
    img__=cv2.resize(img_, dsize=(128,128))
    img=img__[a:a+64,b:b+64]
    f=np.fft.fft2(img)
    fshift=np.fft.fftshift(f)
    magnitude=32*np.log(np.abs(fshift)+1)
    sum=0
    for k in range(64):
        for j in range(64): sum+=magnitude[k][j]**2
    sum=np.sqrt(sum)
    magnitude=magnitude/sum

    matrix=magnitude[32:64,32:64]
    vector=matrix.flatten()
    tmp=np.dot(vector,vector.T)
    vector=vector*10/np.sqrt(tmp) #값들을 normalization 시킴

    mini=1.0
    num=25
    min_n=0
    for i in range(20):
        v=vector-mean_mat[i]
        dist=np.sqrt(np.dot(v,v.T))
        if mini>dist:
            num=i
            min_n=mini
            mini=dist

    return num,mini,min_n
```

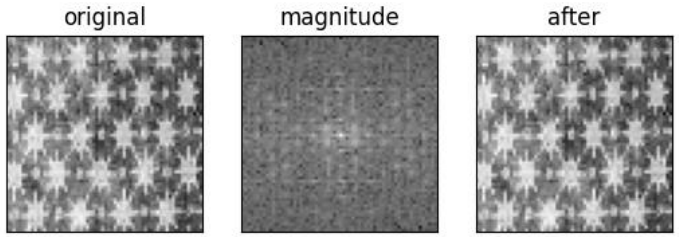
- DECISION은 입력 받는 사진의 값들을 구하는 것으로 미리 저장해둔 값과 비교해서 가장 가까운 PATTERN을 뽑아내도록 했습니다.
- MINI값은 DIST에 관한 것인데 DIST가 가장 작은 것을 고르도록 하면서, 모든 DIST값이 1보다 크다면 PATTERN이 없는 것으로 인식하게 했습니다. (결과에서 자세히 서술)
- MEAN_MAT행렬은 20개의 사진의 5개의 부분의 값들의 평균을 구해둔 것입니다.
- MIN_N은 MINI갱신 후 그 다음 작은 값입니다. 아예 새로운 패턴에 대해서 반응하기 위해서 받아봤습니다.

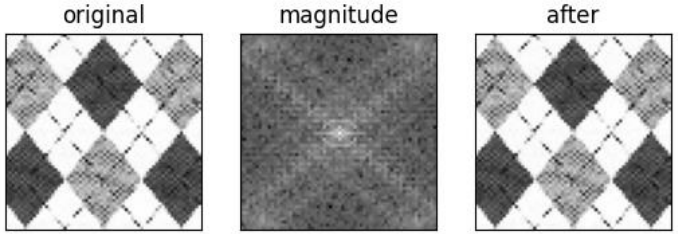
과제의 구현3

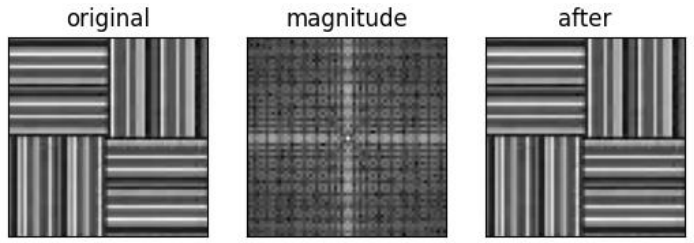
```
path_dir='data'
file_list1=os.listdir(path_dir)

for i in range(20):
    str='data/'+file_list1[i]
    fourier(str,32,32,i)
    fourier(str,48,48,i)
    fourier(str,16,32,i)
    fourier(str,48,16,i)
    fourier(str,16,16,i)
    mean_mat[i]/=5

path_dir='input'
file_list2=os.listdir(path_dir)

for i in range(23):
    str='input/'+file_list2[i]
    recog, one, two=decision(str,64,64)
    if recog<21:print(file_list2[i],file_list1[recog])
    else:
        print(file_list2[i], "no")
    #print(one,two)
    #print("\n")
```

- DATA인 20개의 사진에 대해서 5가지의 위치에 대한 평균값을 구했습니다.
- 기존에 있던 20장에 더해서, DATA에 속하지 않은 3장의 사진을 포함해서 과제를 진행했습니다.
- RECOG는 PATTERN이 같다고 인식한 사진의 번호인데, 이 번호가 21이라면 같은 사진이 없다고 인식하도록 했습니다.
- INPUT에 해당한 사진의 값을 구할 때는 (64,64)나 (64,50) 등으로 설정하여서 겹치지 않도록 했습니다.

실행 결과0 – MAGNITUDE 예시들



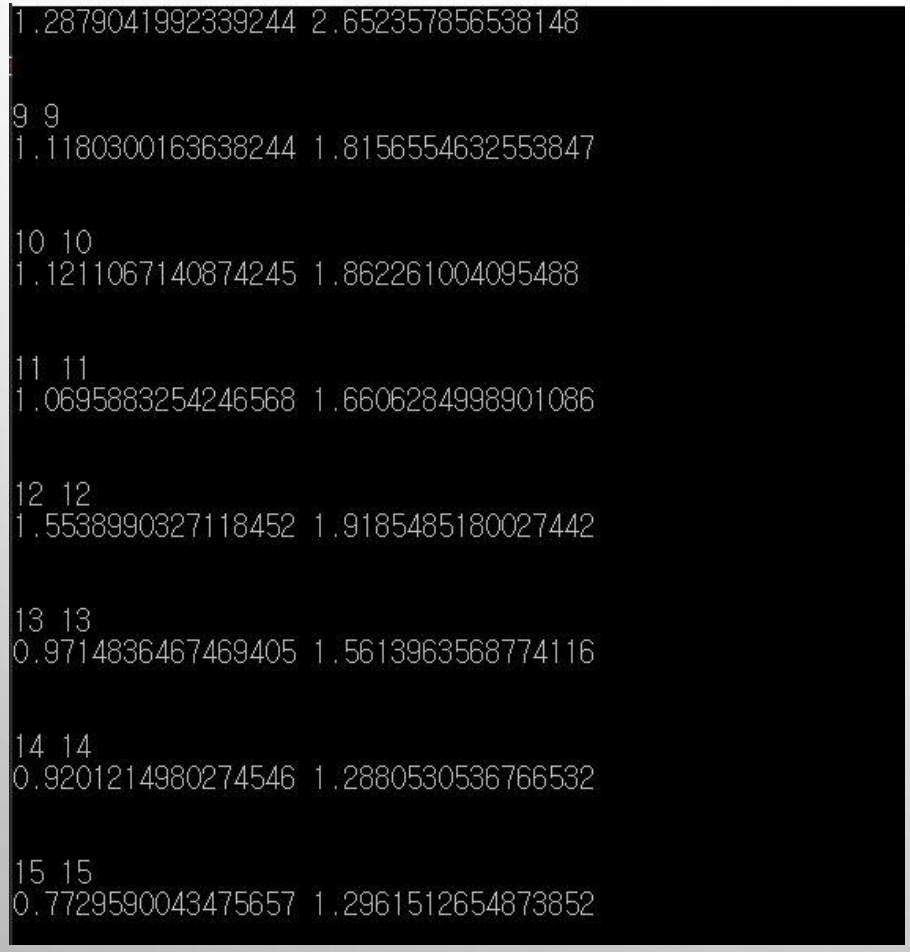




실행 결과1

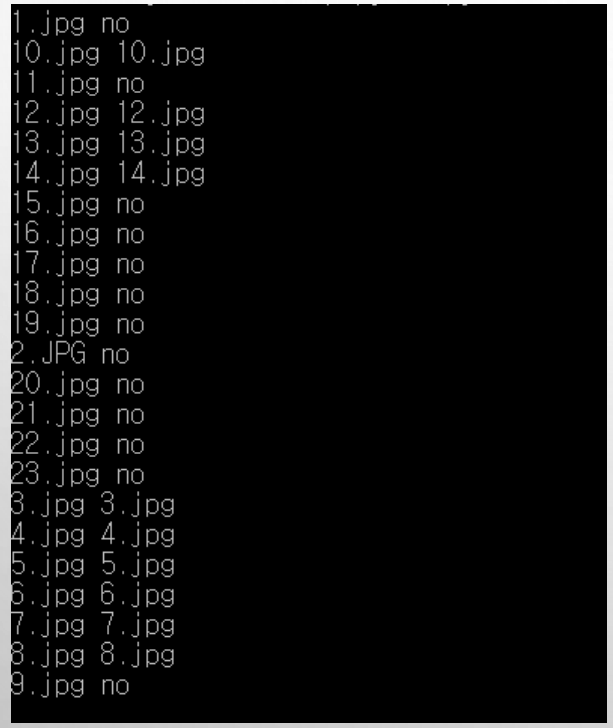
- 이 때 초기 MINI값은 100으로 했었고, MINI 값이 충분히 크다면, DATA에 있는 사진들은 모두 똑바로 인식하는 모습입니다.

실행 결과2



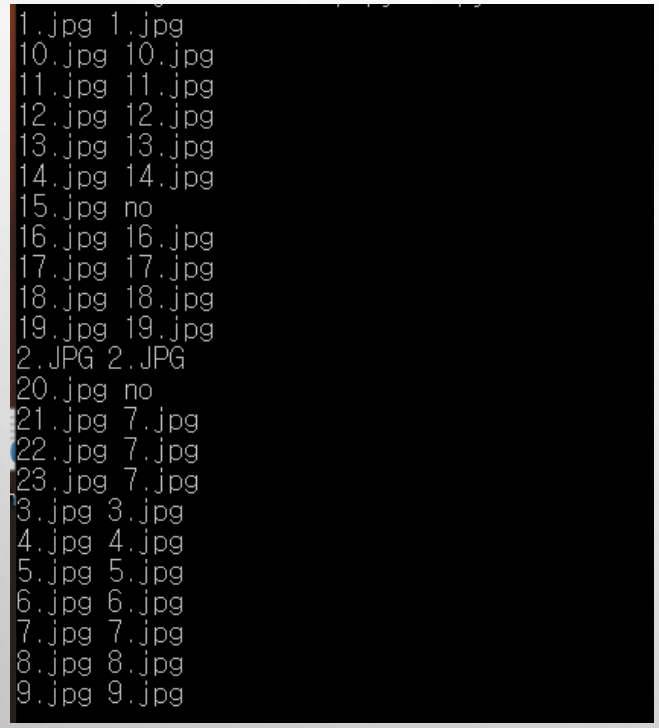
- 이 때 관측할 수 있었던 것은 해당 PATTERN이 PATTERN이라고 인식할 수 있을 때, 최솟값들이 상당이 다양한 분포를 가지는 것이었습니다.
- 14장 정도는 1보다 작았지만, 제일 클 때는 1.55까지 가기도 했습니다. 하지만 다른 사진들은 그 다음으로 작은 값이 1.55보다 작아져서 MINI값을 정확하게 설정할 수 없었습니다.

실행 결과3 MINI=1



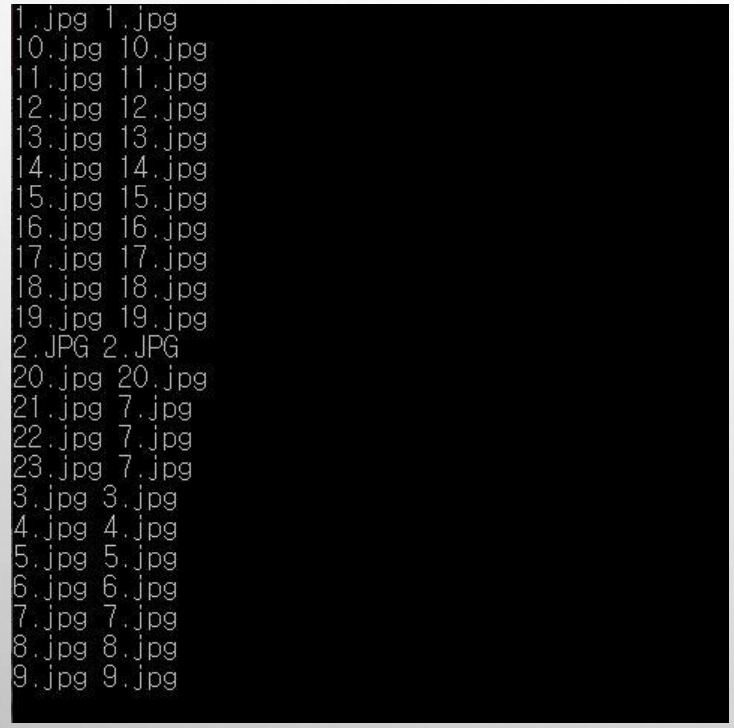
- 새로 들어온 사진들을 걸러내는데 성공했습니다.
- 하지만 기존의 사진들의 RECOGNITION 또한 상당수 실패한 모습입니다.

실행 결과3 MINI=1.2 OR 1.4



- 기존에 있던 사진 2개의 RECOGNITION을 실패했습니다.
- 더욱이 새로 들어온 사진을 걸러내는데 실패했습니다.
- 값이 애매하면 성능 또한 애매해진 것 같습니다.

실행 결과3 MINI=1.6



- 기존의 사진들을 인식하는데 성공했습니다.
- 그와 동시에, 새로 들어온 사진들을 걸러내는데 실패했습니다.

결과 분석

<부족한 점>

- MINI값을 조정해서 적어도 DATA 내에 있는 사진들의 PATTERN을 거의 100프로 인식하는데 성공했지만, DISTANCE 값이 다양해지는 바람에 새로 들어온 사진을 걸러내는 작업에는 정확하게 성공하지 못했다.

<생각해본 보안 방법>

- 64X64에 PATTERN이 몇 개 들어가느냐에 따라서 DIST값을 인위적으로 조정하는 방안이 가능할 수 있을 것 같다.

- 다른 OPTIMIZATION방법을 사용해서 결과를 구할 수 도 있었을 것이다.