수치해석 HW #10

Numerical analysis

JaeHoon KANG
강 재 훈

HW #10

CONTENTS

HW #10

01 준비 과정

사진 구하기

우리는 64x64 크기의 block에 패턴 1개를 넣고 fft를 진행할 것이므로 64x64로 나누기 편한 크기의 사진들을 구했습니다.

기준으로는 일단 64의 배수의 길이를 가지며 최대한 정사각형 모양의 이미지를 구하였습니다.

또한 256x256 크기로 resize해서 사용할 것이기 때문에 256x256 크기로 resize 했을때 64x64 block에 패턴이 다 들어오는지도 중요한 요소였습니다.

HW #10

01 준비 과정

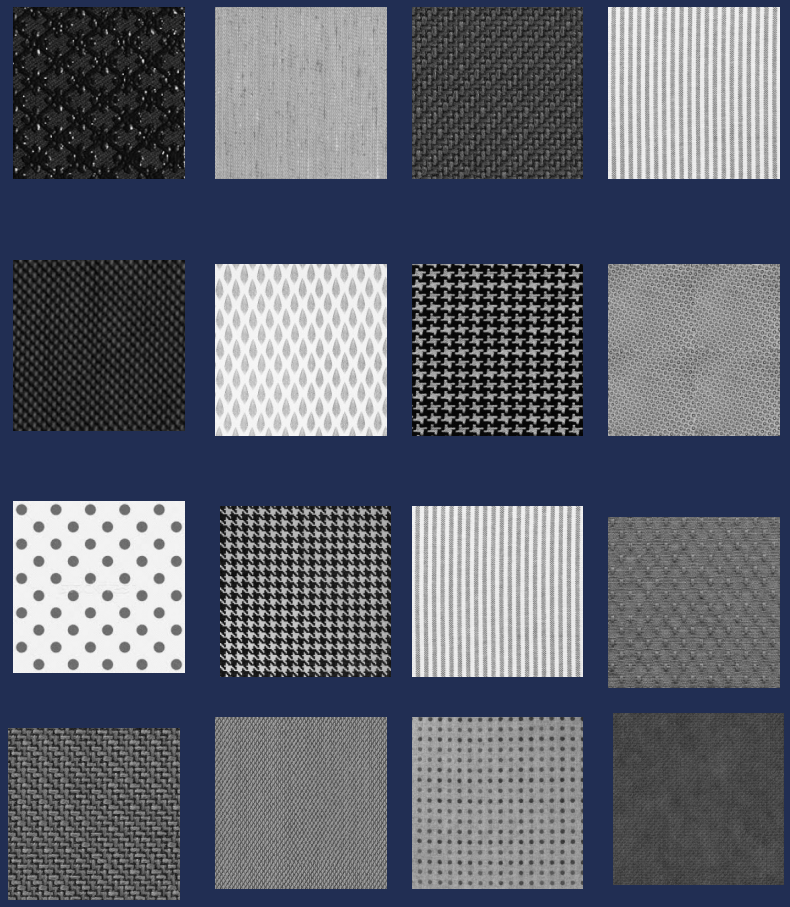


사진 가공하기

구한 이미지들을 256x256 사이즈로 resize하고 greyscale로 바꿔주었습니다.

이후 opencv의 함수인 cv2.imshow로 한장 한장씩 보며 64x64 block에 적절한 패턴 한번이 다 들어가는지 직접 확인하였습니다.

resize와 greyscale 후 이미지가 많이 훼손되거나 64x64 block에 적절한 패턴 한번이 들어오지 않는 경우 새로운 이미지를 구하였습니다.

HW #10

02 코드 설명

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
```

사용된 library

Numpy = 행렬의 계산을 위해 사용

Opencv = 이미지 처리를 위해 사용

Os = 이미지 파일들을 효과적으로 다루기 위해 사용

Matplotlib = 이미지를 출력하기 위해 사용

HW #9

02 코드 설명

```
def fftBlock( filename, start_x, end_x, start_y, end_y ):
def calMean(filename):
def meanifft(filename):
def cmpself(filename):
def cmpother(filename, filename2):
def dominantC(filename):
```

사용된 함수 설명

fftBlock = filename으로 들어온 사진에 대해 start_x, end_x, start_y, end_y 크기 만큼의 block에 대해 fft를 실행합니다.

calMean = 256x256사진에 대해 64x64 크기 만큼 총 16번 fft를 진행하고 그 값의 평균을 구합니다.

meanifft = 한 사진의 mean값을 구하고 그 mean 값을 ifft 해서 결과를 봅니다.

HW #9

02 코드 설명

```
def fftBlock( filename, start_x, end_x, start_y, end_y ):
def calMean(filename):
def meanifft(filename):
def cmpself(filename):
def cmpother(filename, filename2):
def dominantC(filename):
```

사용된 함수 설명

cmpself = 1개의 사진에 대해 mean 값을 구하고 16개의 block에 대해 mean 값을 뺀 뒤 제곱 후 다 더해서 16으로 나눈 행렬을 return 합니다.

cmpother = filename2로 들어온 사진의 16개의 block들에 대해 filename1로 들어온 사진의 mean 값을 뺀 뒤 제곱을 하고 16으로 나눕니다. 이 나눈값의 원소를 모두 더해서 결과 파일에 저장합니다.

HW #9

02 코드 설명

```
def fftBlock( filename, start_x, end_x, start_y, end_y ):
def calMean(filename):
def meanifft(filename):
def cmpself(filename):
def cmpother(filename, filename2):
def dominantC(filename):
```

사용된 함수 설명

dominantC = 64x64 block에서 DC 값을 제외하고 가장 큰 계수들을 저장합니다.

dominantC2.txt에는 16개의 block에서 DC 성분을 제외하고 가장 큰 계수 15개를 인덱스와 상관없이 저장했습니다.

dominantC.txt에는 16개의 block에서 DC 성분을 제외하고 가장 큰 계수 15개를 인덱스와 함께 저장했습니다.

HW #10

03 실행 결과1

```
0.jpg

row : 0 col : 0
dominant1 : [8][12] 45817.692
dominant2 : [56][52] 45817.69
dominant3 : [8][52] 39669.581
dominant4 : [56][12] 39669.58
dominant5 : [9][13] 23626.596
dominant6 : [55][51] 23626.59
dominant7 : [9][51] 21630.661
dominant8 : [55][13] 21630.66
dominant9 : [57][13] 15823.43
dominant10 : [7][51] 15823.43
dominant11 : [9][11] 15054.51
dominant12 : [55][53] 15054.5
dominant13 : [7][13] 15049.50
dominant14 : [57][51] 15049.5
dominant15 : [11][51] 13010.9

row : 0 col : 1
dominant1 : [8][12] 46141.152
dominant2 : [56][52] 46141.15
dominant3 : [8][52] 32109.375
dominant4 : [56][12] 32109.37
dominant5 : [9][13] 23040.390
dominant6 : [55][51] 23040.39
```

```
2.jpg

row : 0 col : 0
dominant1 : [7][7] 63033.55185642049
dominant2 : [57][57] 63033.55185642048
dominant3 : [57][7] 63023.04954950177
dominant4 : [7][57] 63023.04954950176
dominant5 : [8][8] 56430.95994932463
dominant6 : [56][56] 56430.95994932463
dominant7 : [8][56] 56374.439856109835
dominant8 : [56][8] 56374.439856109835
dominant9 : [6][58] 46383.693798451386
dominant10 : [58][6] 46383.69379845138
dominant11 : [6][6] 46297.476078133215
dominant12 : [58][58] 46297.476078133215
dominant13 : [3][61] 31670.692571172287
dominant14 : [61][3] 31670.692571172287
dominant15 : [3][3] 31638.496773823783

row : 0 col : 1
dominant1 : [7][57] 63096.934153918555
dominant2 : [57][7] 63096.934153918555
dominant3 : [57][57] 63035.894734674584
dominant4 : [7][7] 63035.89473467458
dominant5 : [8][56] 56462.22466019461
dominant6 : [56][8] 56462.22466019461
dominant7 : [8][8] 56436.08707876418
```

사진 인식하기

사진 0에서는 [8][12]번째 계수, [56][52]번째 계수, [56][12]번째 계수 … 등이 가장 큰 값을 가지고 그 값들이 비슷한 값을 가지는 것을 확인할 수 있습니다.

사진 2에서는 [7][7]번째 계수, [57][57]번째 계수, [57][7]번째 계수 … 등이 가장 큰 값을 가지고 그 값들이 비슷한 값을 가지는 것을 확인할 수 있습니다.

그런데 자세히 보면 사진 0에서 [8][12]와 [56][52]는 대칭 위치에 있는 것을 알 수 있습니다. 따라서 우리는 64x64 block 전체를 사용 하는 것이 아닌 대칭되는 성질을 이용해서 일부분만 사용 할 수 있습니다.

HW #10

03 실행 결과1

```
0.jpg

row : 0 col : 0
dominant1 : [8][12] 45817.692
dominant2 : [56][52] 45817.69
dominant3 : [8][52] 39669.581
dominant4 : [56][12] 39669.58
dominant5 : [9][13] 23626.596
dominant6 : [55][51] 23626.59
dominant7 : [9][51] 21630.661
dominant8 : [55][13] 21630.66
dominant9 : [57][13] 15823.43
dominant10 : [7][51] 15823.43
dominant11 : [9][11] 15054.51
dominant12 : [55][53] 15054.5
dominant13 : [7][13] 15049.50
dominant14 : [57][51] 15049.5
dominant15 : [11][51] 13010.9

row : 0 col : 1
dominant1 : [8][12] 46141.152
dominant2 : [56][52] 46141.15
dominant3 : [8][52] 32109.375
dominant4 : [56][12] 32109.37
dominant5 : [9][13] 23040.390
dominant6 : [55][51] 23040.39
```

```
2.jpg

row : 0 col : 0
dominant1 : [7][7] 63033.55185642049
dominant2 : [57][57] 63033.55185642048
dominant3 : [57][7] 63023.04954950177
dominant4 : [7][57] 63023.04954950176
dominant5 : [8][8] 56430.95994932463
dominant6 : [56][56] 56430.95994932463
dominant7 : [8][56] 56374.439856109835
dominant8 : [56][8] 56374.439856109835
dominant9 : [6][58] 46383.693798451386
dominant10 : [58][6] 46383.69379845138
dominant11 : [6][6] 46297.476078133215
dominant12 : [58][58] 46297.476078133215
dominant13 : [3][61] 31670.692571172287
dominant14 : [61][3] 31670.692571172287
dominant15 : [3][3] 31638.496773823783

row : 0 col : 1
dominant1 : [7][57] 63096.934153918555
dominant2 : [57][7] 63096.934153918555
dominant3 : [57][57] 63035.894734674584
dominant4 : [7][7] 63035.89473467458
dominant5 : [8][56] 56462.22466019461
dominant6 : [56][8] 56462.22466019461
dominant7 : [8][8] 56436.08707876418
```

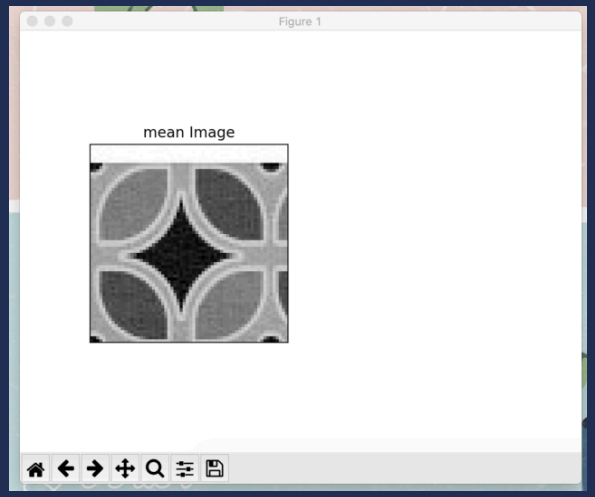
사진 인식하기

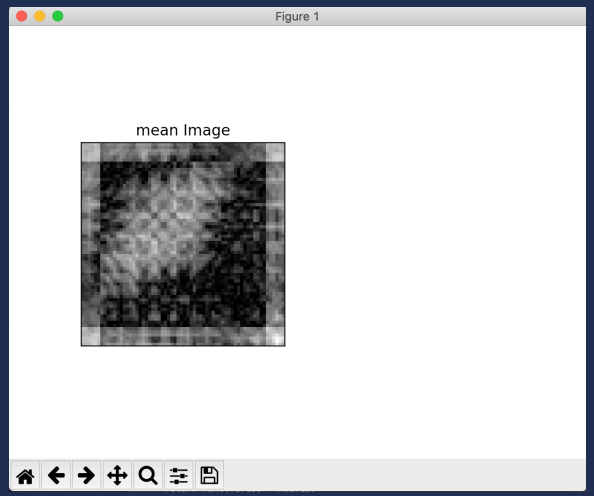
사진마다 각각의 블록들은 같은 위치에서 가장 큰 계수를 가지고
다른 사진들 사이에서는 가장 큰 계수를 가지는 위치가 다른 것을 확인 할 수 있습니다.

따라서 어떤 사진의 블록들에서 가장 큰 값을 가지는 위치를 저장해두고 비교한다면 서로 다른 사진인지 같은 사진인지 비교 할 수 있게 됩니다.

HW #10

03 실행 결과2



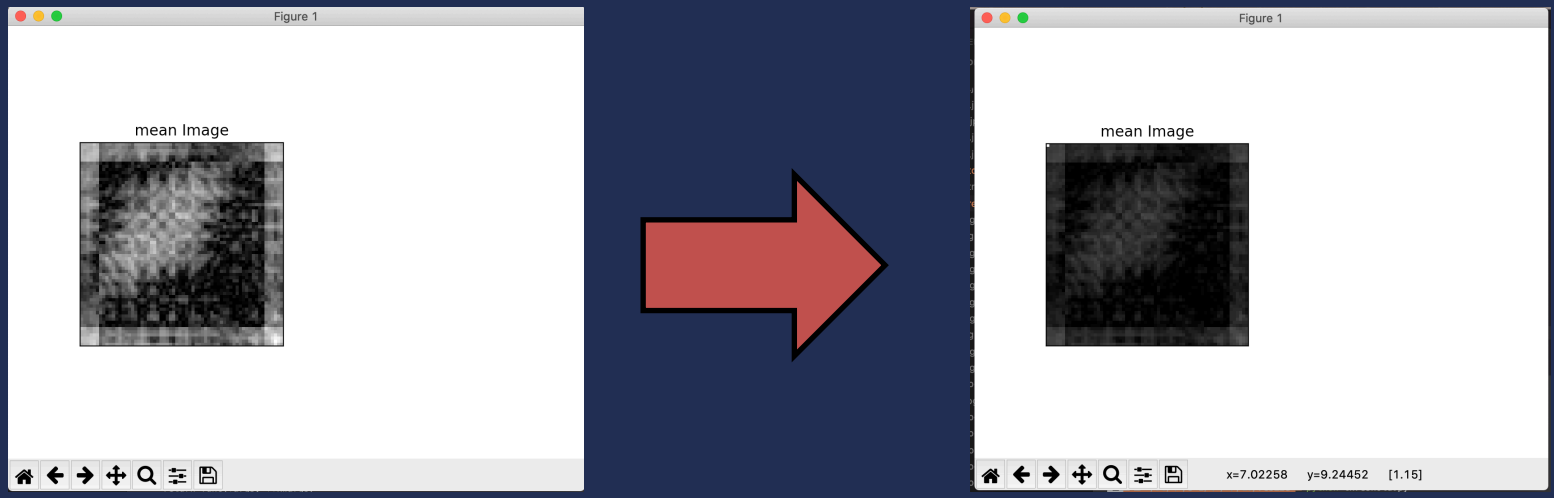


또다른 인식방법을 생각해보았습니다.

왼쪽은 64x64 block의 원본 이미지를 출력한 것이고
오른쪽은 64x64 block 16개의 평균값을 ifft하여 출력한 것입니다.
이 평균값과 많이 다른값의 기준은 평균값에 직접 숫자를 더해가며
패턴을 알아볼수 없을 정도로 하는 그 값을 임계값으로 설정하였습니다.

HW #10

03 실행 결과2



mean 값으로 구한 행렬의 모든 원소에
100씩을 더한 후 ifft한 경우 입니다.
이제 패턴을 거의 알아볼 수 없게 되었으므로
모든 원소에 100을 더한 후 제곱을 하고 16으로 나눈 뒤
그 값을 모두 더한 값인 40960000.0을 임계값으로 정하고
cmpother을 수행한 결과 절대값이 이 임계값보다 크다면
다른 패턴으로 인식하겠습니다.

HW #10

03 실행 결과2

사진 인식하기

```
0.jpg vs 0.jpg
0.0

0.jpg vs 1.jpg
1401969389.875

0.jpg vs 2.jpg
35193463161.03125

0.jpg vs 3.jpg
8157298411.031251

0.jpg vs 4.jpg
1570134458.9062498

0.jpg vs 5.jpg
1286641527.9999995

0.jpg vs 6.jpg
8369778950.687502

0.jpg vs 7.jpg
```

```
0.jpg vs 0.jpg
-972014459.453125

0.jpg vs 1.jpg
429954930.4218749

0.jpg vs 2.jpg
34221448701.578125

0.jpg vs 3.jpg
7185283951.578126

0.jpg vs 4.jpg
598119999.4531248

0.jpg vs 5.jpg
314627068.5468745

0.jpg vs 6.jpg
7397764491.234377

0.jpg vs 7.jpg
```

cmpother 함수의 결과를 텍스트 파일에 저장하였습니다.

수를 편하게 보기 위해 자기 자신과 비교하였을때는 0으로 만들어주었습니다. 텍스트 파일에 저장하기 전에 filename에 해당하는 mean 값을 구하고 모든 원소의 합을 구해 빼주는 방식으로 구현하였습니다.

우리가 앞서 설정한 임계값 40960000.0 보다 절대값이 모두 크니 다른 패턴으로 인식할 수 있습니다.

위의 result.txt는 자기 자신과 비교하였을때 값을 0으로 안만들어준 결과이고 resultN.txt는 자기 자신과의 비교 결과를 0으로 만들어주지 않은 결과입니다.

감사합니다

THANK YOU

JaeHoon KANG
강 재 훈