수치해석
HW #10

Fabric Texture Recognition

(Discrete Fourier Transform)

2018007956 김채아

[pattern recognition 과정]

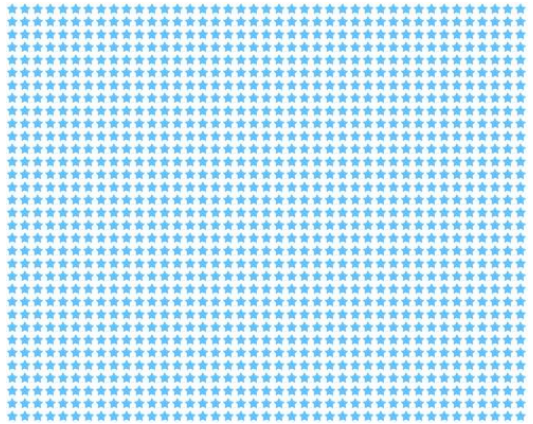
1) 반복되는 패턴 사진을 구한다
2) 패턴 이미지의 −32~+32부분을 DFT 한다 (– 여기서 magnitude 만을 사용)

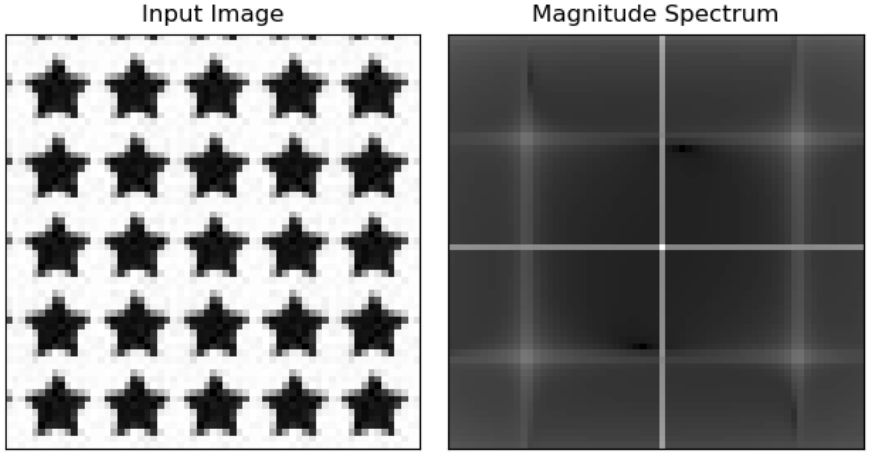
```
h, w = img.shape
dft = cv2.dft(np.float32(img[h//2-32:h//2+32, w//2-32:w//2+32]),flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

save.append(magnitude[33:50,33:50].flatten())
```

3) pattern recognition 한다
   – DC성분을 제외한 계수 값 비교

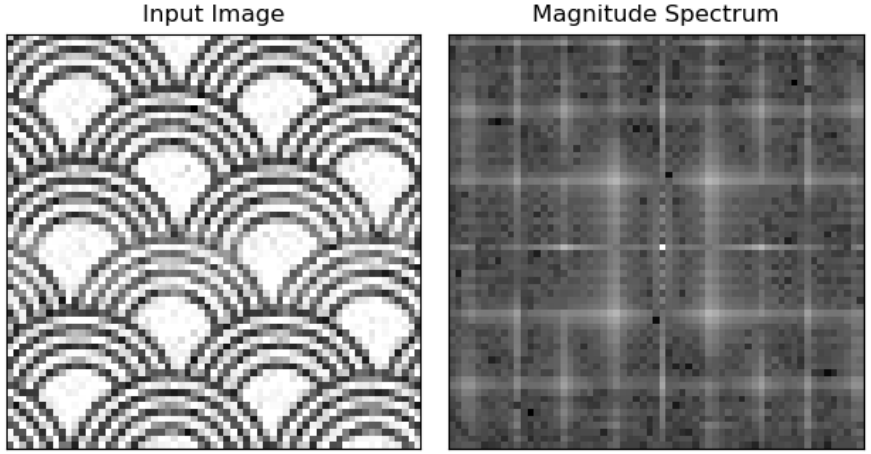
〈pattern recognition〉
랜덤하게 특정 패턴의 block (64x64)를 넣고 전체 20장의 패턴 중에 무엇인지를 맞춘다
–〉 20장의 각 이미지들마다 5번 시행하고,
    그 100번의 시도 중 패턴을 인식한 횟수를 구해서 패턴이 인식되는 비율을 구해본다

[1]

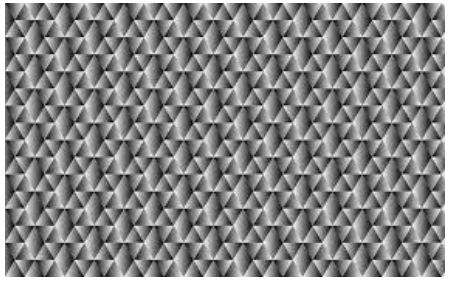


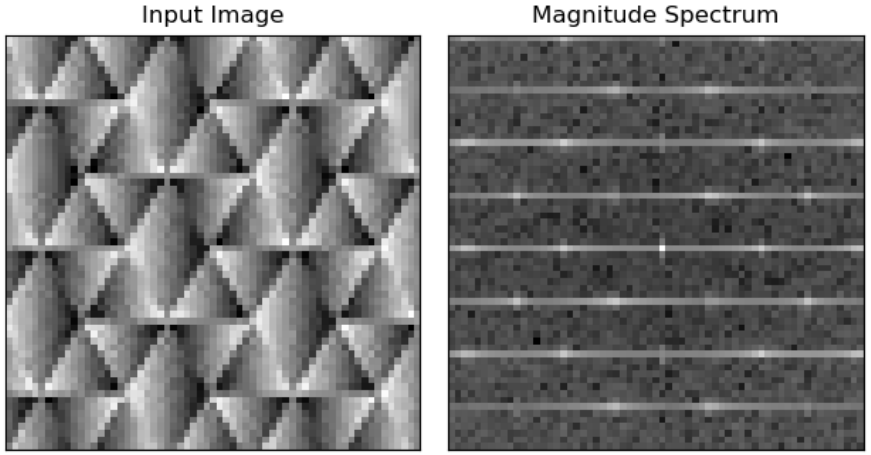

Input Image
Magnitude Spectrum

[2]

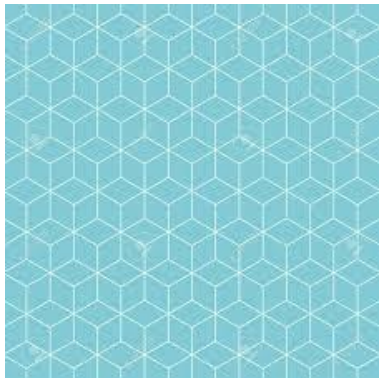

Input Image
Magnitude Spectrum

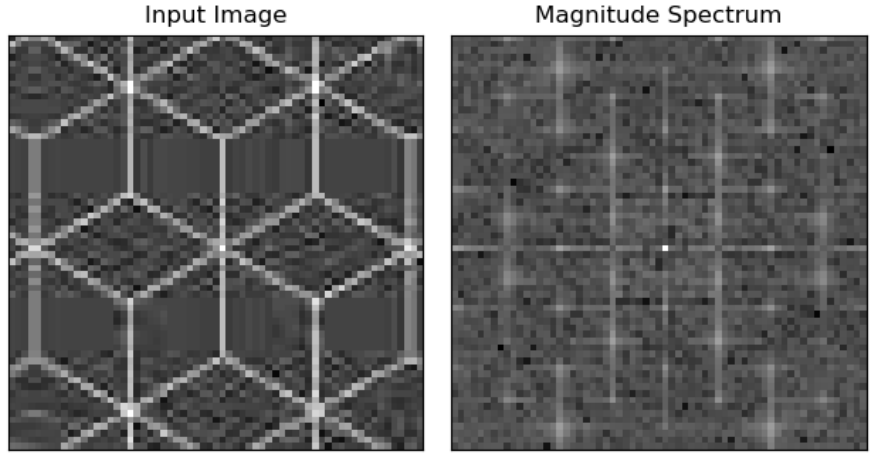
[3]




Input Image
Magnitude Spectrum

[4]




Input Image
Magnitude Spectrum

[5]

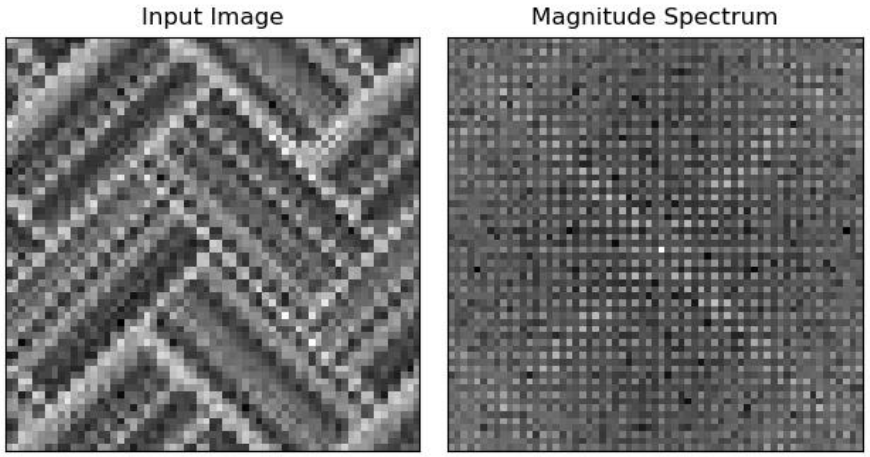
image rescaling

img size = 999x999

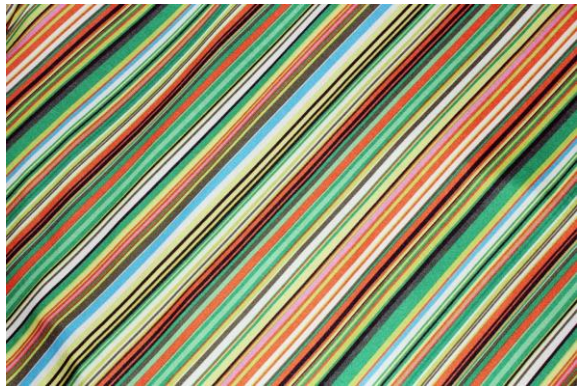
↓

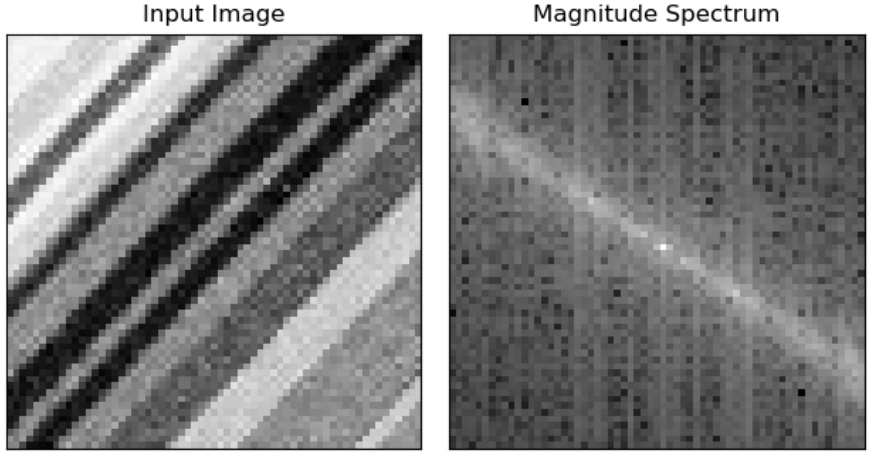
img size = 100x100

DFT

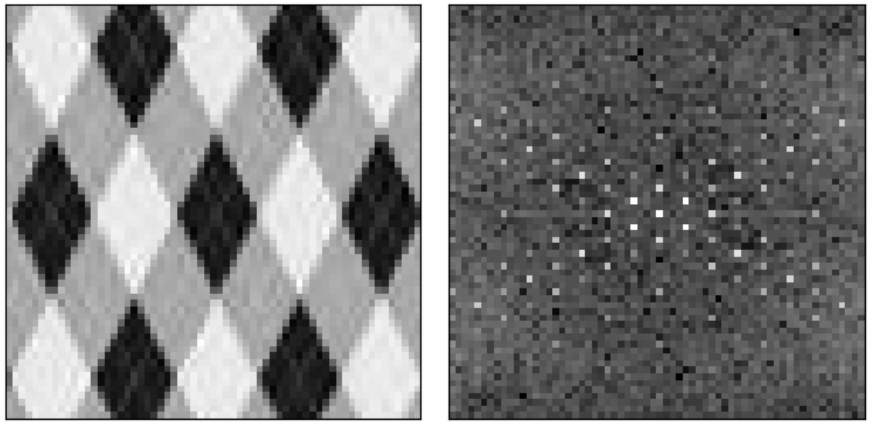


[6]

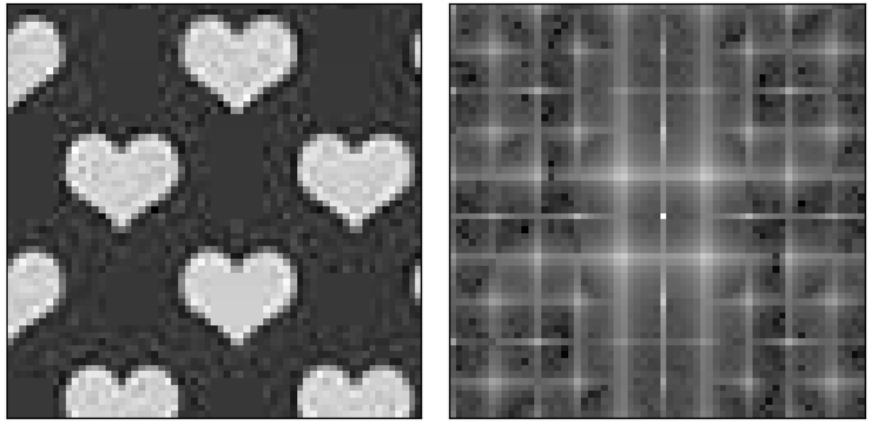




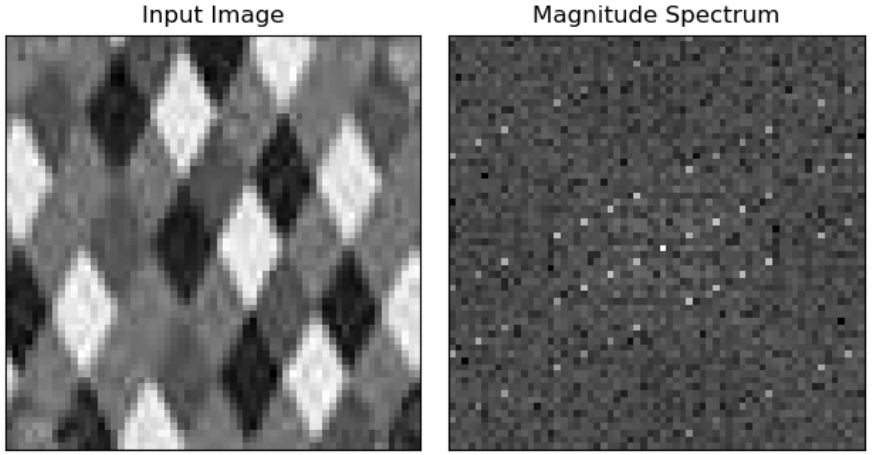
[7]


Input Image
Magnitude Spectrum

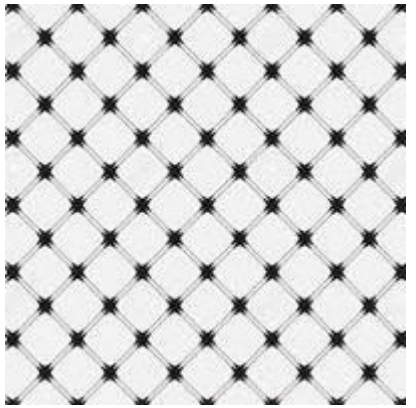
[8]

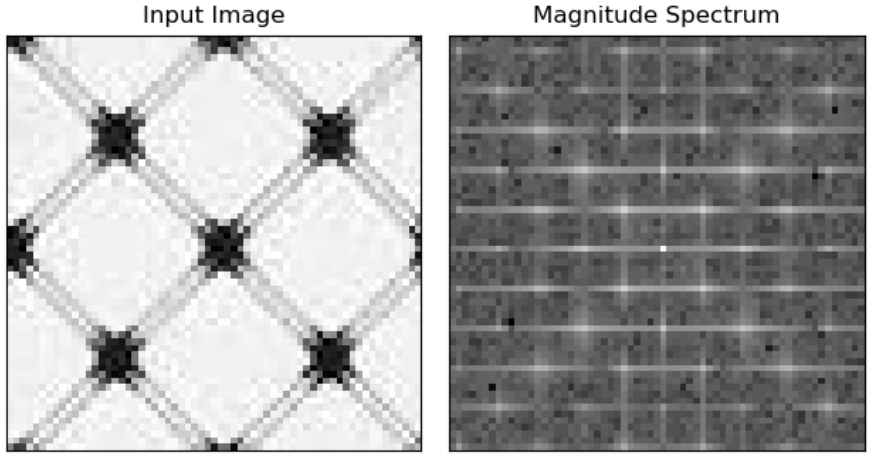

Input Image
Magnitude Spectrum

[9]


Input Image
Magnitude Spectrum

[10]




Input Image
Magnitude Spectrum

[11]

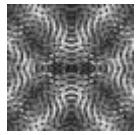
DFT

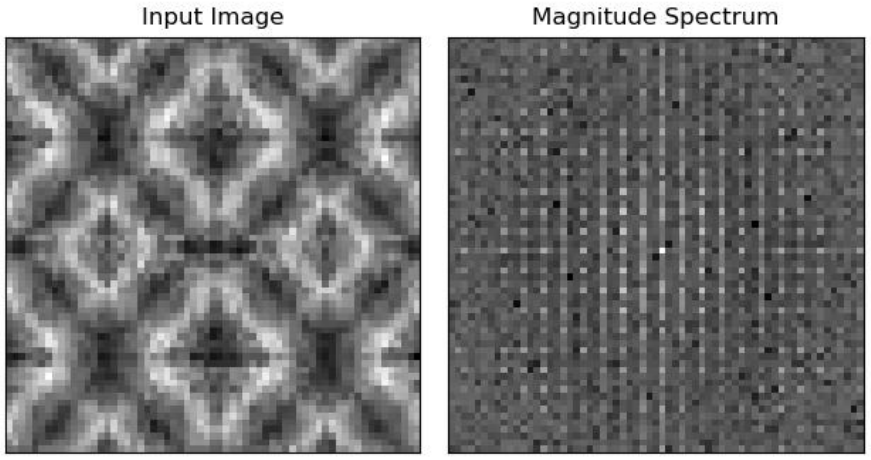
image rescaling

img size = 200x200

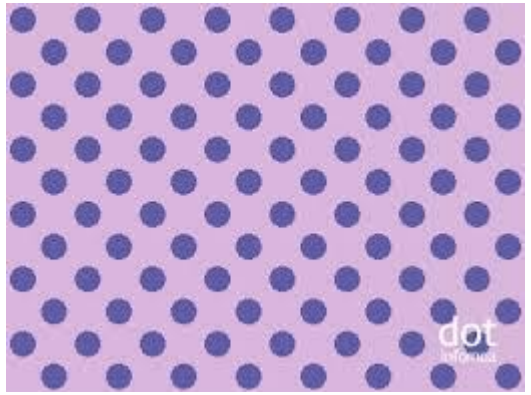
↓

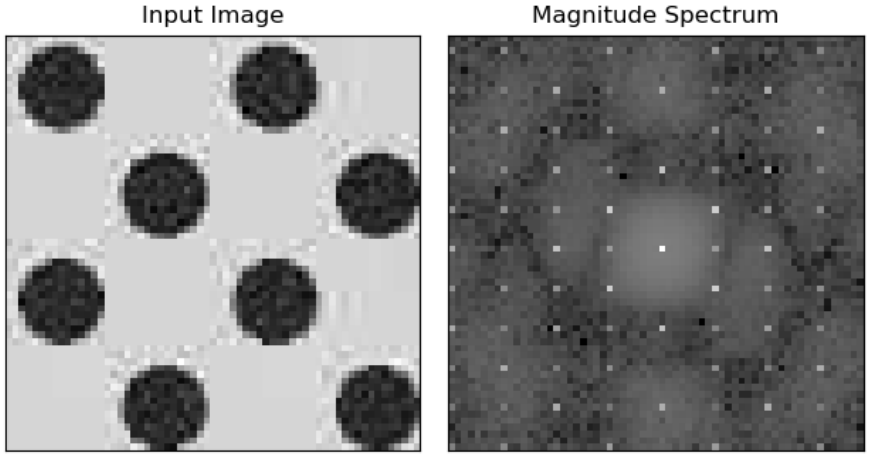
img size = 100x100





[12]





[13]

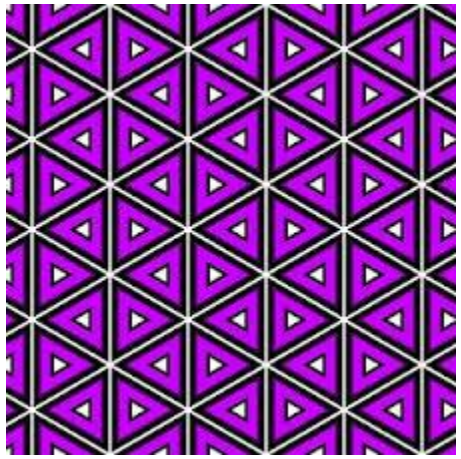


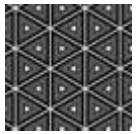
image rescaling

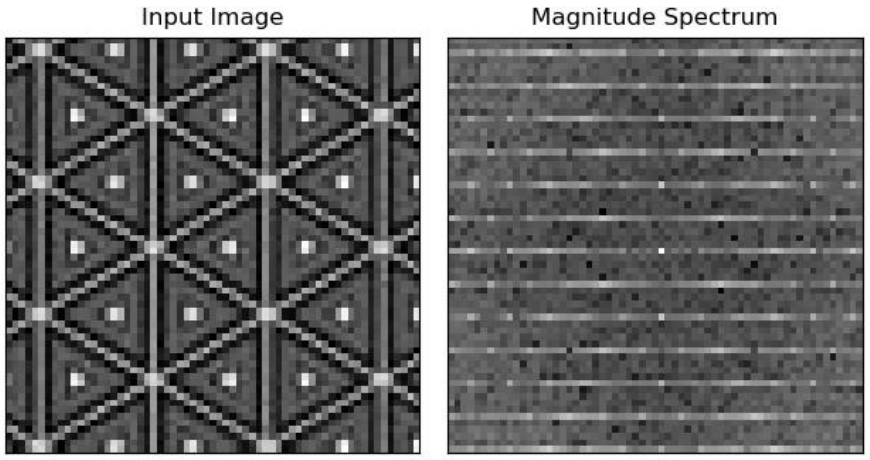
img size = 225x225

↓

img size = 100x100

DFT





[14]

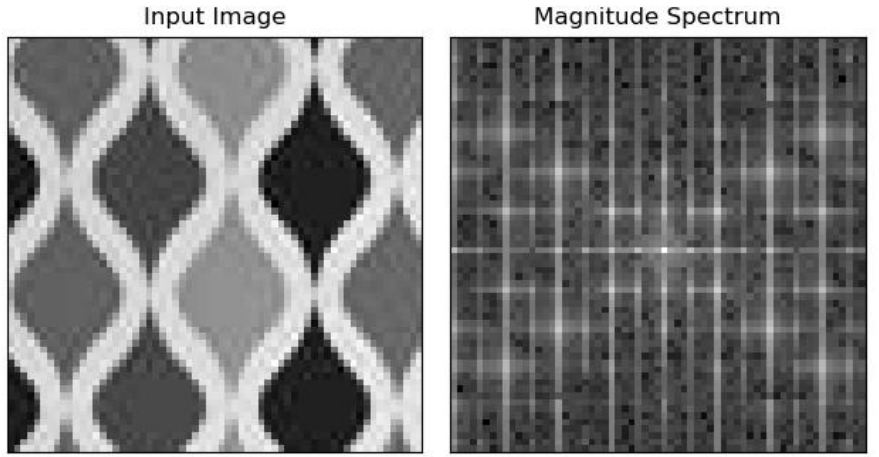
image rescaling

img size = 1500x1500

↓

img size = 200x200

DFT



[15]

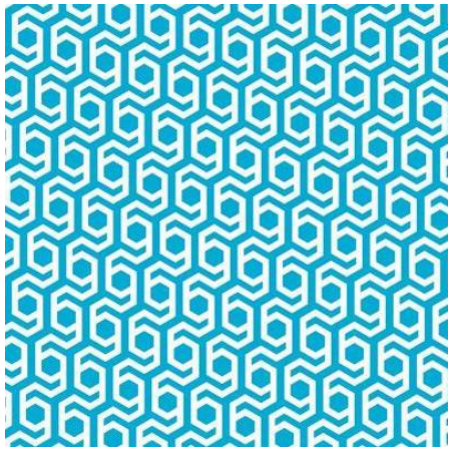


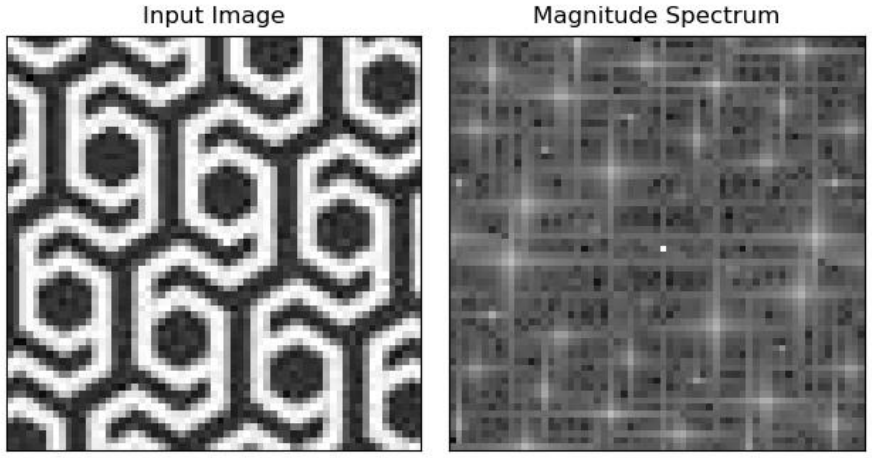
image rescaling

img size = 400x400

↓

img size = 100x100

DFT



[16]

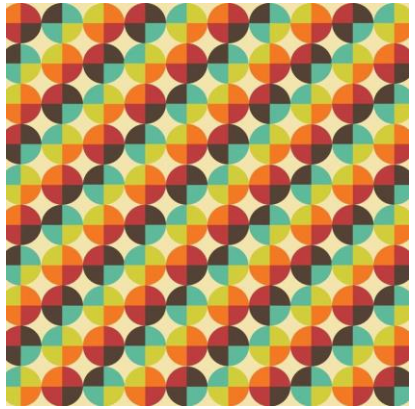


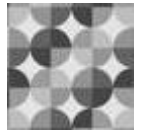
image rescaling

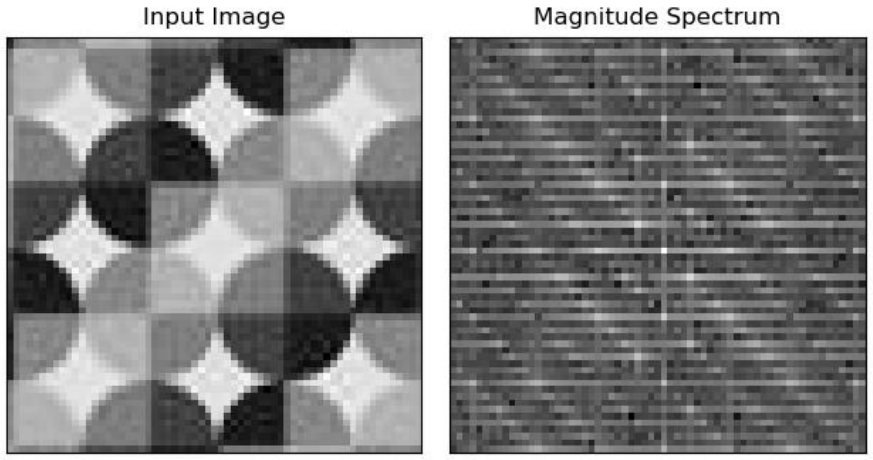
img size = 800x800

↓

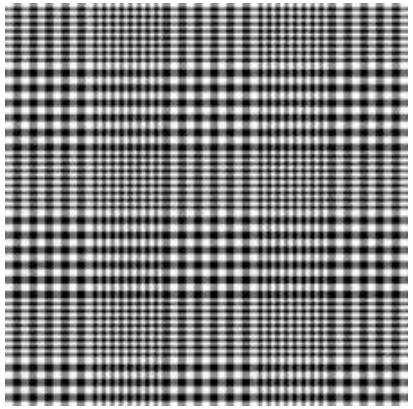
img size = 400x400

DFT





[17]



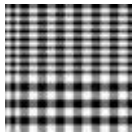
DFT

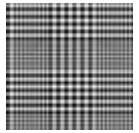
image rescaling

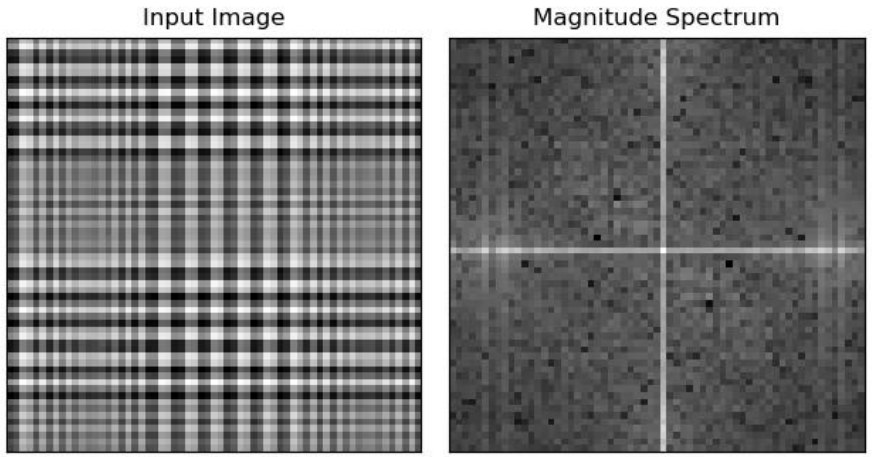
img size = 224x224

↓

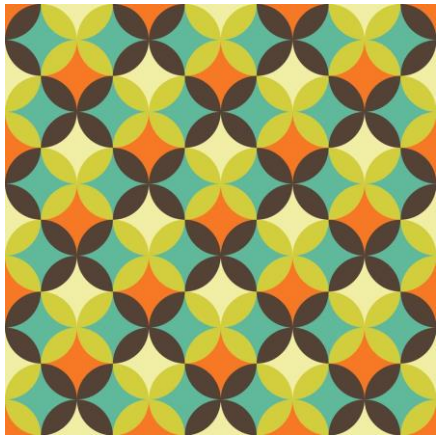
img size = 100x100







[18]



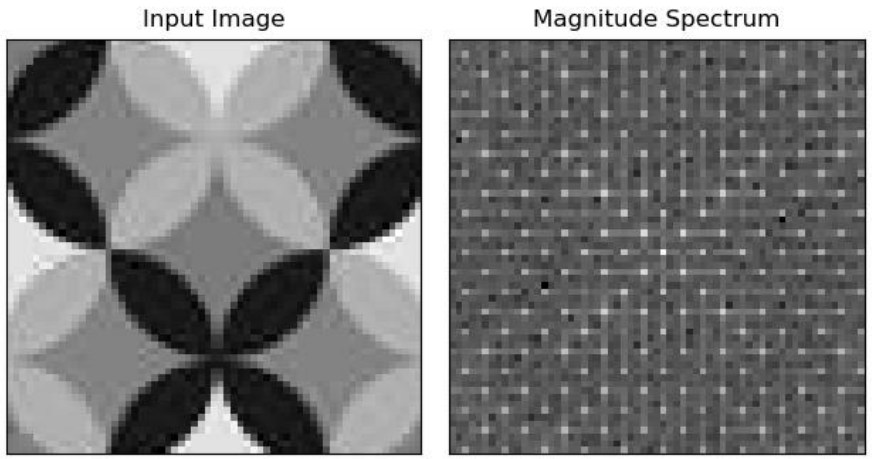
DFT

image rescaling

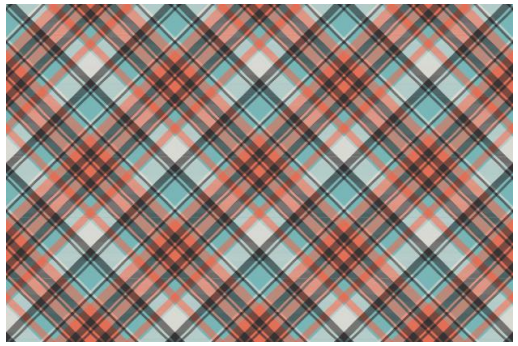
img size = 1000x1000

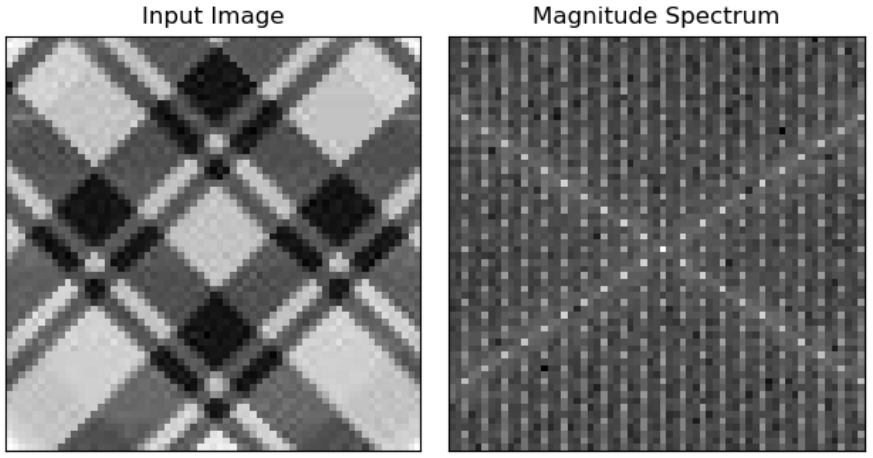
↓

img size = 200x200

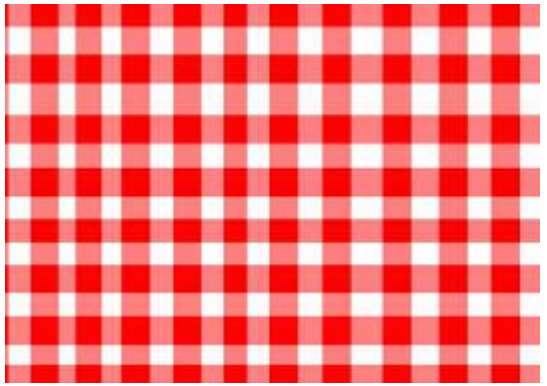


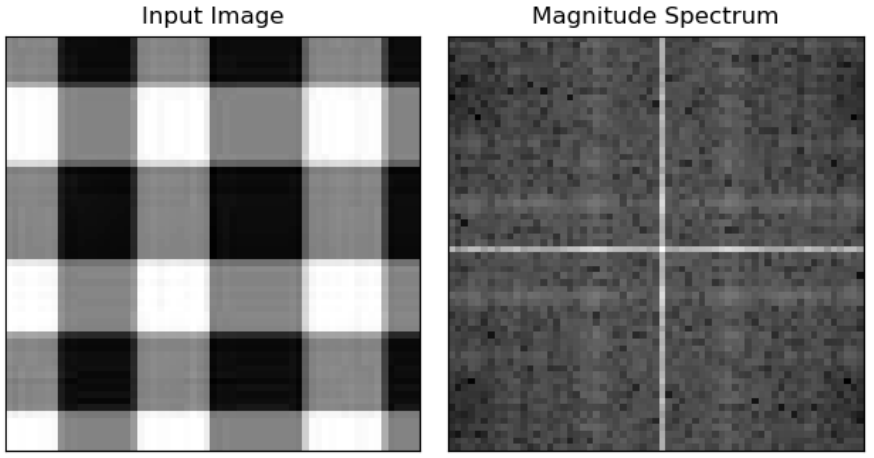
[19]



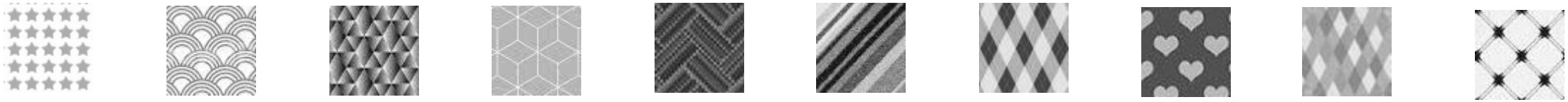

Input Image
Magnitude Spectrum

[20]

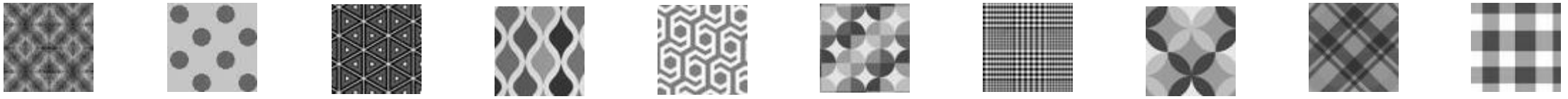



Input Image
Magnitude Spectrum

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)



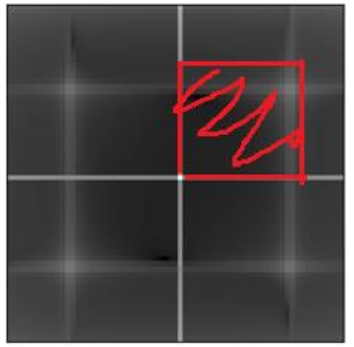
(11) (12) (13) (14) (15) (16) (17) (18) (19) (20)



[pattern recognition]

– 이미지의 가운데 64x64 블록을 추출해서 dft를 하고 DC성분을 제외한 일부분을 저장한다
– DC성분은 평균값이기 때문에 패턴의 굴곡에 의미가 없으므로 비교 대상에서 제외한다
– 각 패턴당 5개의 랜덤한 64x64 block을 생성해서 20장의 패턴 중에 무엇인지 맞춘다

```
dft = cv2.dft(np.float32(img[h//2-32:h//2+32, w//2-32:w//2+32]),flags=cv2.DFT_COMPLEX_OUTPUT)
dft[0, 0] = 0  # DC remove

dft_shift = np.fft.fftshift(dft)
magnitude = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

# print(magnitude[h//2,w//2]) # -> DC 성분 위치
# print(mask[32][32]) # DC 성분 위치
```



dft[0, 0]=0 으로 만들어보니 magnitude[h//2,w//2] (=mask[32][32])값이 -inf로 나왔다
이를 통해 해당 위치가 DC성분 위치라는 것을 알 수 있었다 이 위치를 비교 대상에서 제외시켰다

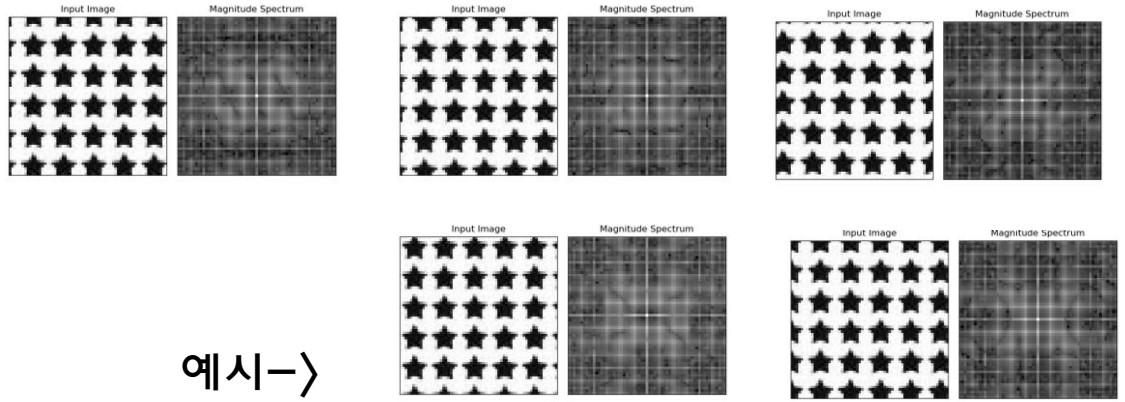
```
save.append(magnitude[33:50,33:50].flatten())
```

20장의 패턴의 magnitude의 오른쪽 위 사분면 일부분을 뽑아서(대칭성 이용) 저장한다
[32,32]가 DC성분 위치이므로 33부터 추출한다

[pattern recognition using vector distance]

```
# 이미지 하나당 5번씩
for n in range(5):
    # 랜덤한 64x64 block 추출
    x = random.randrange(h-64)# 1부터 가로 사이즈 -64 사이의 난수 생성
    y = random.randrange(w-64)
    print('{}:{}, {}:{}'.format(x,x+63,y,y+63))
    mask = testimg[x:x+64,y:y+64]
```

각 이미지당 랜덤한 5개의 64x64 block을 추출한다



예시-〉

```
a=magnitude2[33:50,33:50].flatten()
b=[]
for i in range(20):
    b.append(np.mean(save[i]-a))# 비교
idx = np.argmin(b)
result.append(name[idx])
```

test img도 같은 부분의 계수들을 뽑고
저장된 값과의 차이가 제일 적은 패턴을 찾는다

```
    # 적중률 계산
    if len(result)!=0:
        for n in range(5):
            if result[n] == name[img_num]:
                hit_num += 1

    img_num += 1

print('100번의 횟수 중 패턴을 잘 인식한 횟수:',hit_num)
```

인식된 이미지가 맞으면 hit_num+=1

[pattern recognition] **패턴 인식 : 약 %98**

```
< 1.jpg >
315:378, 164:227
146:209, 68:131
72:135, 369:432
268:331, 280:343
98:161, 386:449
['1.jpg', '1.jpg', '1.jpg', '1.jpg', '1.jpg']
< 10.jpg >
106:169, 55:118
33:96, 69:132
16:79, 123:186
121:184, 19:82
7:70, 126:189
['10.jpg', '10.jpg', '10.jpg', '10.jpg', '10.jpg']
< 11.jpg >
16:79, 5:68
29:92, 31:94
31:94, 7:70
35:98, 22:85
15:78, 1:64
['11.jpg', '11.jpg', '11.jpg', '11.jpg', '11.jpg']
< 12.jpg >
67:130, 93:156
38:101, 182:245
122:185, 3:66
13:76, 90:153
73:136, 65:128
['12.jpg', '12.jpg', '12.jpg', '12.jpg', '12.jpg']
< 13.jpg >
4:67, 29:92
19:82, 22:85
5:68, 10:73
29:92, 13:76
35:98, 31:94
['13.jpg', '13.jpg', '13.jpg', '13.jpg', '13.jpg']
< 14.jpg >
52:115, 35:98
76:139, 97:160
71:134, 66:129
94:157, 66:129
133:196, 33:96
['14.jpg', '14.jpg', '14.jpg', '14.jpg', '14.jpg']
< 15.jpg >
5:68, 21:84
75:138, 116:179
12:75, 44:107
115:178, 23:86
103:166, 97:160
['15.jpg', '15.jpg', '15.jpg', '15.jpg', '15.jpg']
< 16.jpg >
14:77, 59:122
45:108, 36:99
111:174, 3:66
96:159, 24:87
128:191, 5:68
['16.jpg', '16.jpg', '16.jpg', '16.jpg', '16.jpg']
< 17.jpg >
7:70, 23:86
18:81, 6:69
34:97, 0:63
30:93, 5:68
4:67, 17:80
['17.jpg', '17.jpg', '17.jpg', '17.jpg', '17.jpg']
< 18.jpg >
48:111, 108:171
98:161, 51:114
83:146, 35:98
7:70, 118:181
75:138, 74:137
['18.jpg', '18.jpg', '18.jpg', '18.jpg', '18.jpg']
< 19.jpg >
154:217, 467:530
97:160, 240:303
115:178, 365:428
115:178, 130:193
244:307, 303:366
['19.jpg', '19.jpg', '19.jpg', '19.jpg', '19.jpg']
< 2.jpg >
151:214, 18:81
139:202, 22:85
43:106, 73:136
68:131, 67:130
139:202, 146:209
['2.jpg', '2.jpg', '2.jpg', '2.jpg', '2.jpg']
< 20.jpg >
77:140, 134:197
59:122, 135:198
40:103, 7:70
114:177, 28:91
6:69, 59:122
['20.jpg', '20.jpg', '20.jpg', '20.jpg', '20.jpg']
< 3.jpg >
86:149, 174:237
76:139, 54:117
101:164, 68:131
41:104, 117:180
36:99, 70:133
['3.jpg', '3.jpg', '3.jpg', '3.jpg', '3.jpg']
< 4.jpg >
108:171, 80:143
123:186, 76:139
122:185, 94:157
145:208, 156:219
118:181, 16:79
['4.jpg', '4.jpg', '4.jpg', '4.jpg', '4.jpg']
< 5.jpg >
22:85, 11:74
19:82, 1:64
22:85, 10:73
26:89, 3:66
12:75, 25:88
['5.jpg', '5.jpg', '5.jpg', '5.jpg', '5.jpg']
< 6.jpg >
12:75, 175:238
371:434, 223:286
60:123, 722:785
18:81, 322:385
69:132, 105:168
['6.jpg', '6.jpg', '6.jpg', '6.jpg', '5.jpg']
< 7.jpg >
5:68, 31:94
23:86, 33:96
25:88, 34:97
9:72, 12:75
2:65, 31:94
['7.jpg', '7.jpg', '7.jpg', '14.jpg', '7.jpg']
< 8.jpg >
44:107, 137:200
47:110, 72:135
52:115, 23:86
18:81, 150:213
62:125, 75:138
['8.jpg', '8.jpg', '8.jpg', '8.jpg', '8.jpg']
< 9.jpg >
52:115, 110:173
67:130, 89:152
32:95, 102:165
132:195, 45:108
85:148, 62:125
['9.jpg', '9.jpg', '9.jpg', '9.jpg', '9.jpg']
```

패턴 인식이 상당히 잘 되는 것을 볼 수 있다