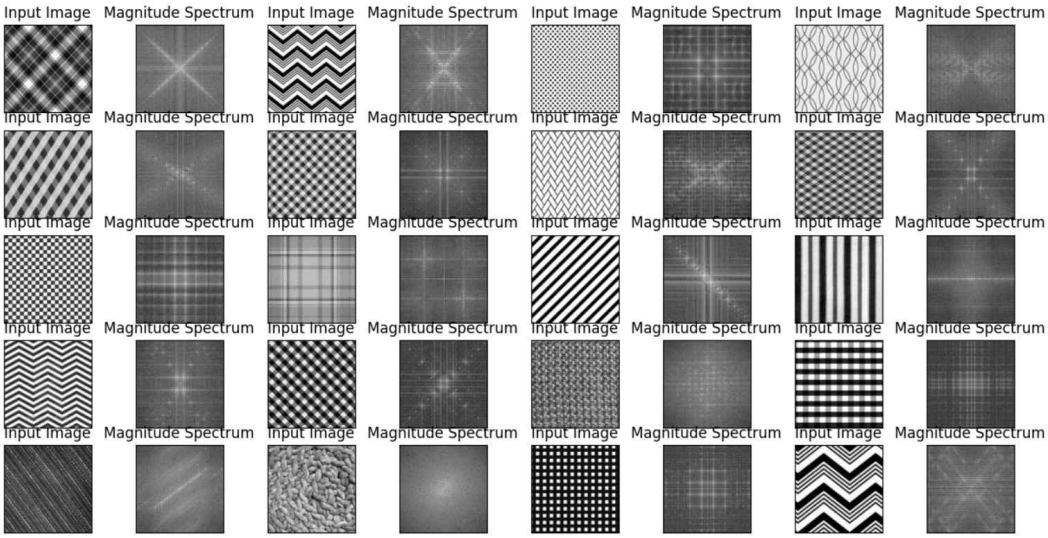
다음의 코드를 통해 각각의 패턴들을 dft에 적용시켜 magnitude spectrum을 보았습니다.

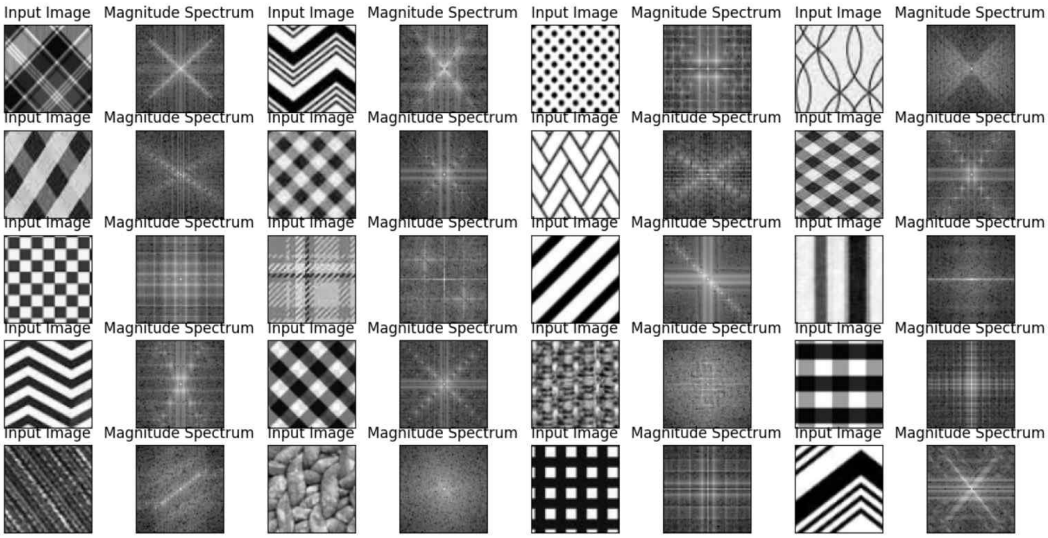
```
i = 0
j=1

fig = plt.figure()
while i < 20:
f = np.fft.fft2(img_list[i])
fshift = np.fft.fftshift(f)
mag = 20*np.log(np.abs(fshift))
plt.subplot(5,8,j),plt.imshow(img_list[i], cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(5,8,j+1), plt.imshow(mag, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
i+=1
j= i*2+1
plt.show()
```



모든 패턴의 크기는 192*192의 크기로 캡쳐하였고 우선 magnitude의 spectrum 또한 192*192의 크기로 설정하였습니다. 패턴인식을 위해 다음과 같이 64*64의 영역을 랜덤으로 지정하여 magnitude spectrum을 다시 구해보았습니다.

이 때, magnitude의 값이 DC의 값이 매우 크고 AC의 값이 매우 작아 상대적으로 표현하기 매우 비효율적이어서 log함수를 통해 scale의 값을 어느정도 조정했으며 0~255사이에 matching하기 위해 20을 곱해 표현하였습니다.

```
x = rd.randint(0,128)
y = rd.randint(0,128)

def crop(img):
w = 64
h = 64
img_crop = img[y:y+h,x:x+w]
return img_crop
```



64*64크기로 crop할 경우 패턴이 많이 반복되지 않는, 패턴의 크기가 큰 것들은 두 번 이상 반복되지 않아 인식이 힘들 것입니다. 패턴의 crop부분을 크게하면 되겠지만 우선 이상태에서 분석을 해보도록하겠습니다. 신기한점은 많이 반복되지 않았음에도 불구하고 magnitude specrum의 밝기만 달라질 뿐 모양은 변함이 없다는 점입니다.

우선 vector distance로 구분하기 위해 magnitude의 평균값을 구했습니다. 그 이후, 평균값과 인식할 이미지의 차이를 통해 구분하려 했습니다. 하지만, 위의 이미지처럼, 2번, 4번같은 이미지는 한 block사이즈 안에 두 번 이상의 패턴이 반복되지 않아 다른 이미지로 인식할 수 있어 64*64단위의 crop을 100번 진행하여 평균을 내었습니다.

```
################### <magnitude average 함수> ##############################

x = range(-2048,2048) # 4096차원magnitude 계수 살펴보기
def avgmag(img):
avg1 = np.zeros((64,64))
count = 0
while count < 5:
img2 = crop2(img)
f = np.fft.fft2(img2)
shifted = np.fft.fftshift(f)
mag = 20*np.log(np.abs(shifted))
avg1 += mag
count += 1
avg1 /= count
return np.resize(avg1,(64*64))
```

```
def vecdis(avg,img):
count = 0
min2 =0
min = 0
while count < 10:
img2 = crop2(img)
f = np.fft.fft2(img2)
vecdisshi = np.fft.fftshift(f)
```

```
mag = 20 * np.log(np.abs(vecdisshi))
img2_re = np.resize(mag,(64*64))
min = np.abs((avg)-(img2_re))
min2 += np.sum(min)
count+=1
return min2/count
```

recognize과정은 다음과 같이 구했습니다.

$$threshold = \frac{\text{인식할 이미지의 평균벡터와 그 이미지의 벡터거리}}{\text{비교대상이미지의 평균벡터와 인식할 이미지의 벡터거리}}$$

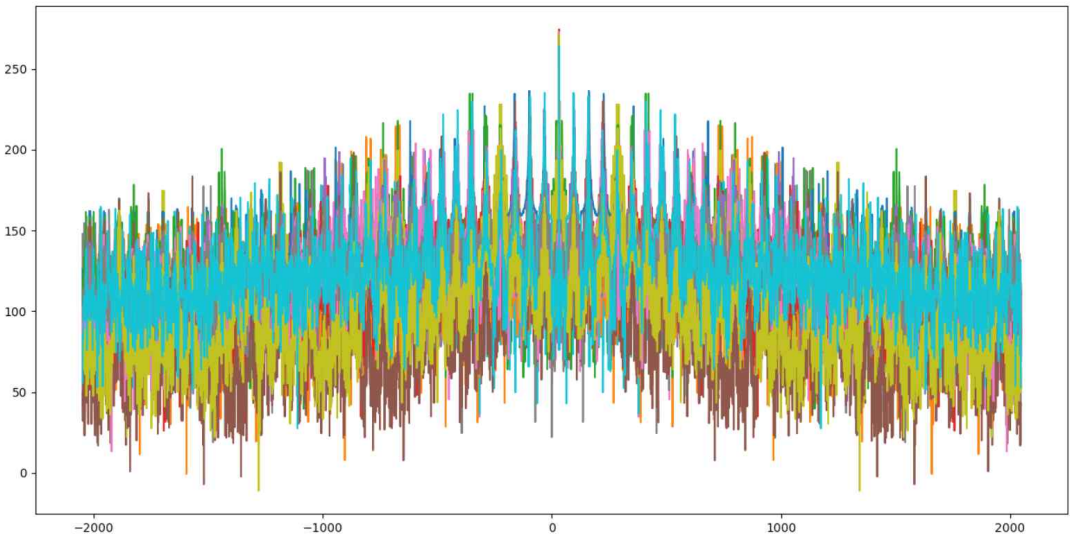
```
################### < recognize 함수> ######################

def rec(img1,img2):
avg = avgmag(img1)
thres = 0.10
origin = vecdis(avg,img1)
pick = vecdis(avg,img2)
result = np.abs(origin-pick)

if (result / origin) < thres:
print("same pattern")
else :
print("different pattern")
```

위 실험의 결과는 다음과 같습니다.

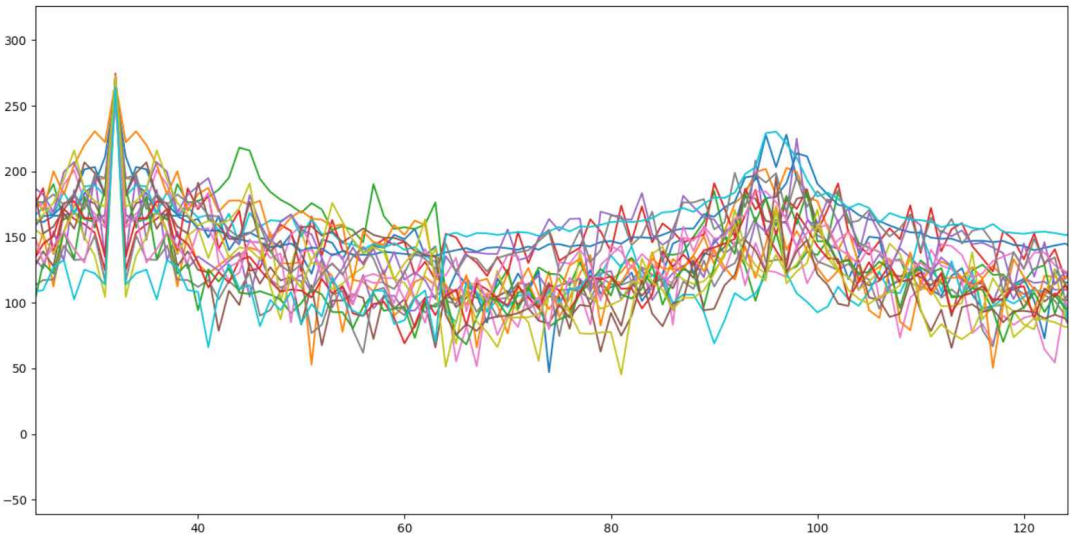
same pattern
different pattern
different pattern
different pattern
same pattern
different pattern
different pattern
different pattern
different pattern
different pattern
different pattern
different pattern
different pattern
different pattern
different pattern
same pattern
different pattern
different pattern
different pattern

1번째 그림, 5번째, 17번째 그림이 same pattern으로 나오게 되는데 이로써 평균벡터들과의

거리로는 일정부분은 판별이 가능하지만 정확하지 않다는 점이 확인이 되었습니다. 그리고 계산값을 확인해본 결과 nan, inf가 나옴에 따라 vector distance를 구할 때 평균을 내는 숫자를 10으로 줄여 다시 해본 결과 오차가 줄어들었습니다. 또한, vector distance로 구하는 것만으로는 판별이 불가능하다 생각되었습니다. 따라서, dominant한 coefficient들 중 5개를 선별하여 추가적으로 확인하는 작업을 하였습니다. 우선, coefficient의 그래프를 통해 확인해보았습니다.



[그림 3] 전체 magnitude의 coefficient그래프



[그림 4] 높은 coefficient를 구하기 위한 확대 그래프

그래프를 살펴보았을 때, 45번, 96번, 162번, 226번, 288번이 상대적으로 높은 계수값을 갖는 것을 확인하고 이들의 10번의 평균벡터를 구하여 이들의 차가 일정비율이상 높아지면 일치하는 pattern임을 확인할 수 있는 계수를 1씩 높여 결과에 한가지 기준을 더 세워주는 방법을

사용하였습니다.

```
#################### < average 2함수> ####################
def avgmag2(img1):
avg1 = np.zeros((64,64))
avg2 = np.zeros(5)
count = 0
while count < 10:
img2 = crop2(img1)
avg1 += img2
count +=1
avg1 /= count
avg1 = np.resize(avg1,(64*64))
count = 0
while count < 5:
avg2[count] = avg1[choice[count]]
count +=1

return avg2
```

```
#################### < recognize 함수> ######################

def rec(img1,img2):
ret = 0

avg1 = avgmag(img1)
avg2 ori = avgmag2(img1)
avg2 = avgmag2(img1)
avg3 = avgmag2(img2)

thres = 0.20
origin1 = vecdis(avg1,img1)
pick1 = vecdis(avg1,img2)
result = np.abs(origin1-pick1)

origin2 = differ(avg2 ori,avg2)
p = differ(avg2,avg3)


if (result / origin1) < thres:
ret+=1

if (np.abs(p-origin2) / origin2) < 2:
ret+=1

return ret
```

	1회차	2회차	3회차	4회차	5회차	인식률
0번	O	O	O	O	O	100
1번	O	O	O	X	O	80
2번	O	O	O	X	O	80
3번	O	X	O	O	X	60
4번	X	O	O	O	O	80
5번	X	O	O	O	X	60
6번	O	O	O	O	O	100
7번	O	O	X	X	O	60
8번	X	O	O	O	O	80
9번	X	O	O	O	O	80

10번	O	O	O	O	O	100
11번	O	X	X	X	O	40
12번	O	O	O	O	O	100
13번	X	O	X	O	X	60
14번	O	O	O	O	O	100
15번	X	X	X	X	X	0
16번	O	O	X	O	X	60
17번	X	O	O	O	O	80
18번	O	O	X	O	O	80
19번	X	X	O	O	O	60

총인식률 : 73%

인식률은 굉장히 아쉬운 편입니다. 벡터평균거리와의 차이를 잴 때 무한대와 not a number 이 일어나는 오버플로우만 잡는다면 더 좋은 결과가 있을 것이라 생각합니다.