[3193] 디지털영상처리 과제 2 202012468 김현서

이번 과제는 유독 오래 걸렸다. 특별히 정해진 방식이라거나 조건이 없어서 자유롭게 코드를 작성하다보니 욕심이 생겨 더 그랬던 것 같다.

이번 과제는 크게 2가지, 작게 3가지로 구별됐다.
1. implementation of Basic gray-level transformation
    1) Negative transgormation
    2) Power law transformation
2. implementation of Histogram Equalization

위 코드들은 모두 HSI 값을 수정해야 결과를 도출해낼 수 있을 것 같아 보였다. 그래서 나는 우선 원본 사진의 RGB 값을 추출하여 HSI로 변환, 그리고 그 HSI 값을 RGB 값으로 다시 변환하는 것을 목표로 코드를 짜기 시작했다.

우선 저번 과제에서 RGB 값을 HSI로 변환하는 코드를 짰었기 때문에 반은 이미 완성된 상태로 시작했다.

```
src = cv2.imread('test image/bridge.bmp', cv2.IMREAD_COLOR)
height, width = src.shape[0], src.shape[1]

negative = np.zeros((height, width,3), dtype=np.uint8)
plt = np.zeros((height, width,3), dtype=np.uint8)
HSI = np.zeros((height, width,3), dtype=np.uint8)


I = np.zeros((height, width))
S = np.zeros((height, width))
H = np.zeros((height, width))
hsi2B_negative = np.zeros((height, width))
hsi2G_negative = np.zeros((height, width))
hsi2R_negative = np.zeros((height, width))


#B, G, R = cv2.split(src)
B = src[:,:,0]
G = src[:,:,1]
R = src[:,:,2]



for i in range(height):
    for j in range(width):
        B_tmp, G_tmp, R_tmp = B[i][j]/255., G[i][j]/255., R[i][j]/255.

        I[i][j] = (B_tmp+G_tmp+R_tmp)/3.
        # prevent divide by 0, adding 0.00000001
        S[i][j] = 1 - 3*np.min([B_tmp,G_tmp,R_tmp])/(B_tmp+G_tmp+R_tmp+0.00000001)
        H[i][j] = compute_Hue(B_tmp, G_tmp, R_tmp)
```

위 코드인데, 저번주에 짰던 코드보다는 많이 발전했다.
우선 B, G, R을 받는 코드들을 split이나 src[:,:,0]과 같은 3중 배열을 통해 나타냈다.
이후 B, G, R을 받아 수업시간에 배웠던 I, S, H를 구하는 식을 통해 HSI를 각각 구했다.

1.-i) HSI를 변경하지 않고, 원본사진 반전으로 코드 작성

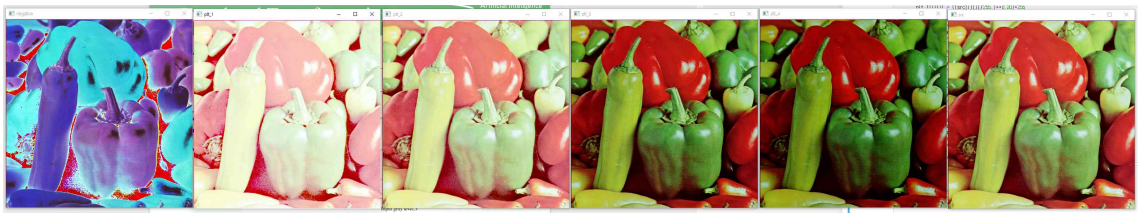
```
#Negative
for i in range(height):
    for j in range(width):
        negative[i][j] = -src[i][j]

#power-law transformation : lambda value = 0.20
for i in range(height):
    for j in range(width):
        plt_1[i][j] = ((src[i][j]/255.)**0.20)*255

#power-law transformation : lambda value = 0.40
for i in range(height):
    for j in range(width):
        plt_2[i][j] = ((src[i][j]/255.)**0.40)*255.

#power-law transformation : lambda value = 1.5
for i in range(height):
    for j in range(width):
        plt_3[i][j] = ((src[i][j]/255.)**1.5)*255.

#power-law transformation : lambda value = 5.0
for i in range(height):
    for j in range(width):
        plt_4[i][j] = ((src[i][j]/255.)**5.0)*255.
```

우선 1주차 과제에서 사용했던 RGB→HSI 변환 코드도 사용하지 않고 Negative, power-law transformation을 구현해보려고 했다.

Power-Law Transformation에서 원본사진 값만 건드린인 흑백사진

Negative 0.2 lambda 0.45 람다 1.5 람다 4.0 람다 원본

Power-Law Transformation에서 원본사진 값만 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 1.5 람다 4.0 람다 원본

사진을 보면 보이지만, RGB→HSI, HSI→RGB 값 변환을 안 하고 원본 값만 이용하려고 하다보니 결과가 제대로 나오지 않았다. 흑백사진의 경우 '-scr' 값을 사용해서 0에는 부호가 붙지 않아, 오른쪽 아래 검정색 부분이 흰색으로 바뀌지 않았다. 컬러사진의 경우 흑백 사진과 마찬가지로 검정색 부분에 제곱 값을 취해봤자 0이 나오기 때문에 빨강색으로 나오게 되었다. 또, 1보다 작은 람다를 Hue, Saturation, Indensity에 각각 취했을 때 Hue, Saturation은 더 진한 값, Indensity는 밝은 값이 나와서 밝지만 색이 진하게 나와야하는데, 원본 사진 자체에 람다를 취해서 사진의 전체적인 값이 밝아져서 아쉬웠다.

1.-ii) indensity만 변경해 Power-law Transformation 구하기

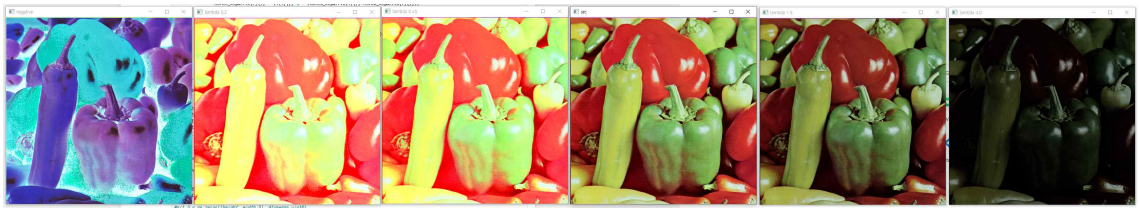
```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                I[i][j] = 1 - I[i][j]
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))
                I[i][j] = 1 - I[i][j]

                I[i][j] = I[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)

negative = cv2.merge((1-hsi2B_negative,1-hsi2G_negative,1-hsi2R_negative))


cv2.imshow("negative", negative)
cv2.imshow("src", src)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

위 코드는 Negative와 Power-law transformation을 한 번에 구하는 코드이다.
Power-law transformation의 경우 람다의 값을 0.2, 0.45, 1.5, 4.0으로 임의로 설정했다.
for문을 사용해 람다(감마)값을 자동으로 변경해가며 lambda 0.2, lambda 0.45, lambda 1.5, lambda 4.0가 plt에 잠깐씩 저장해 보였다.
람다 값에 따라 plt(Power-law transformation)이 변하고, Negative의 경우 HSI를 RGB로 변환한 값에 마이너스를 적용해 Hue, Saturation, Indensity에 모두 negative 값을 적용해주었다. 이렇게 하면 Hue, Saturation, Indensity 모두 고려되기 때문에 negetive로 반전시킬 때 어떤 값도 빠짐 없이 변환될 것 같았다.
밑에 나오는 사진을 보면 알겠지만, 다행히도 예상이 맞아떨어졌고, 가장 마음에 드는 Negative 값과 power-law Transformation 값이 나왔다. Negative 값의 경우 이 코드 그대로 끝까지 가기 때문에 1.-iii)부터는 Power-Law Translation 위주로 서술하겠다.

Power-Law Transformation에서 Indensity만 건드린 흑백사진

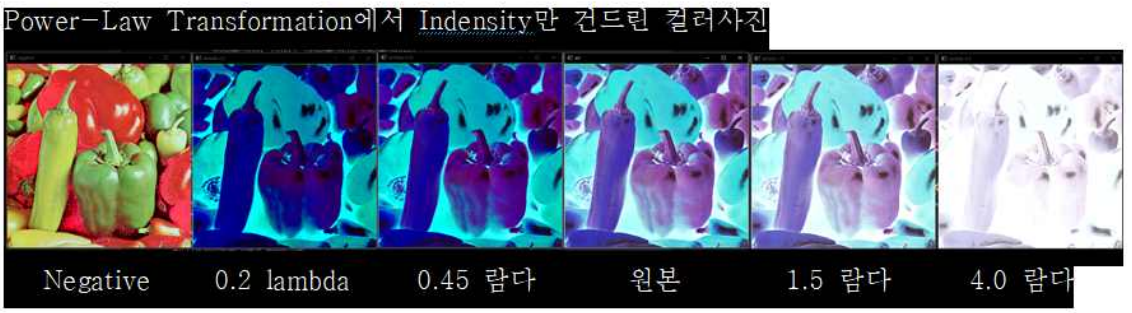
Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Indensity만 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

현재 한글에 보고서를 쓰는 중인데, 보고서를 쓰면서 마우스로 드래그를 하면 위 사진들 모두 Negative 값으로 반전된다. 신기해서 한 번 공유한다.



↑ 위 사진이 방금 말 했듯 드래그 한 사진이다. 신기하다...!!

다시 보고서로 돌아와서, Power-Law Transformation에서 Indensity 값만 변경 시 흑백 사진의 경우 제대로 잘 출력된 것 같았고, 컬러사진의 경우 Hue와 Saturation은 그대로인 채 Indensity가 밝아지고 어두워져서 밝을 때의 사진이 너무 노란 기가 도는 건 아닌가 싶었다.

1.-iii) Saturation만 변경해 Power-law Transformation 구하기

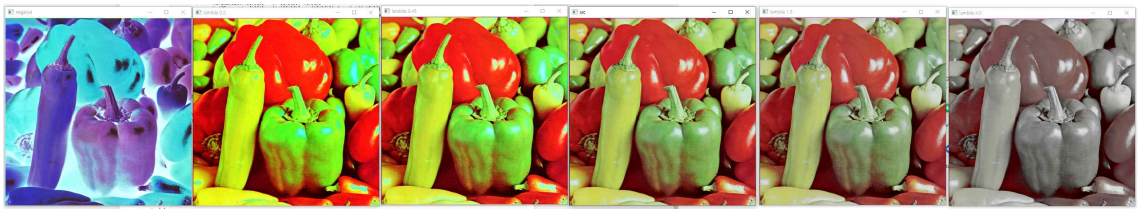
```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

1.-ii)와 달라진 점은 'if (H범위)'문 안에 있는 값 중 power에 해당하는 값은 변화된 Saturation을 사용하는 점이다. 해당 gamma 값을 제곱시킨 후 hsi2B/G/R_power에 넣은 직후 다시 1/gamma 승을 해서 S 값을 원상복구 시켜둬야한다. 그러지 않으면 gamma 값이 변하고 다시 for문을 돌 때, 원래의 S 값이 아닌 S**gamma이 된 값을 가지고 다음 사진이 출력된다. 이를 막기 위해 꼭! 원상복구를 시켜줘야한다는 것을 명심하자.

Power-Law Transformation에서 Saturation만 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Saturation만 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

이 경우 흑백사진은 0.2 lambda 값이 살짝 누렇게 나온 것을 확인할 수 있다.
그 이유는 Saturation 값은 1보다 작은 수로 제곱할 경우 1에 가까워지기 때문이다. Saturation은 0에 가까울수록 채도가 낮아지고, 1에 가까울수록 채도가 높아짐을 의미한다. 따라서 흑백 사진도 0에 제일 가까운 lambda 0.2는 다른 사진에 비해 채도를 띠고 있고, 컬러시진의 경우 명확하게 0에 가까울 수록 색이 진해지고 1보다 큰 람다 값을 가지만 그 값이 커질 수록 채도를 잃어 점점 흑백 사진에 가까워진다는 것을 확인할 수 있다.

1.-iv) Hue만 변경해 Power-law Transformation 구하기

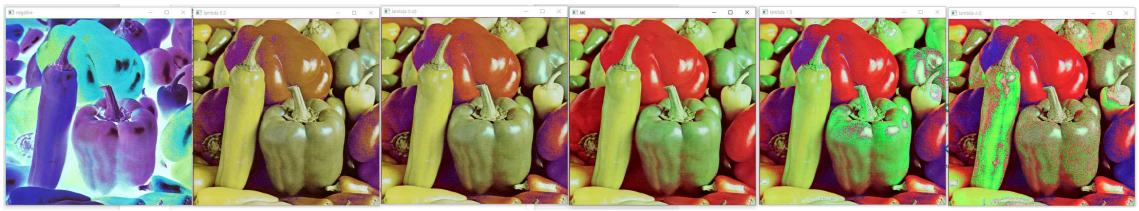
```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                H[i][j] = H[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                H[i][j] = H[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                H[i][j] = S[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                H[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                H[i][j] = S[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                H[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

이 코드 역시 알고리즘은 이전 코드들과 동일하다. 단지 1.-ii)의 Indensity, 1.-iii)의 Saturation 대신 Hue의 값에 gamma 승을 해서 Powe-law Transformation을 한 코드이다.

Power-Law Transformation에서 Hue만 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Hue만 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Indensity와 Saturation을 건들였을 때와 달리 흑백 사진에서 Power-Law Transformation은 아무런 변화가 없다. 그도 그럴 것이 위 사진은 정말 흑과 백, 명도에 따라 회색으로만 이뤄진 사진이기 때문이다. 반면 컬러 사진의 경우 색이 진해질수록 빨강, 초록, 파랑 부분이 두드러지게 나타난다. 이유는 명확하게는 모르겠지만, 아마 제곱 값을 가지게 되면서 B, G, R이 가질 수 있는 최댓값 이상을 가지게 되어 빨강 부분은 완전히 빨간 최대값으로 표현, 초록 부분과 파랑 부분도 이와 마찬가지로 최댓값으로 표현되는 게 아닌가 예상한다. 아마 맞을 것이다!

1.-v) Indensity, Saturation 변경해 Power-law Transformation 구하기

```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

ii), iii), iv)에서는 Indensity, Saturation, Hue 각각 한 개의 값들만 건드렸고, v), vi), vii)에서는 두 개의 값씩, viii)에서는 최종적으로 모든 값을 건드려보려고 한다.

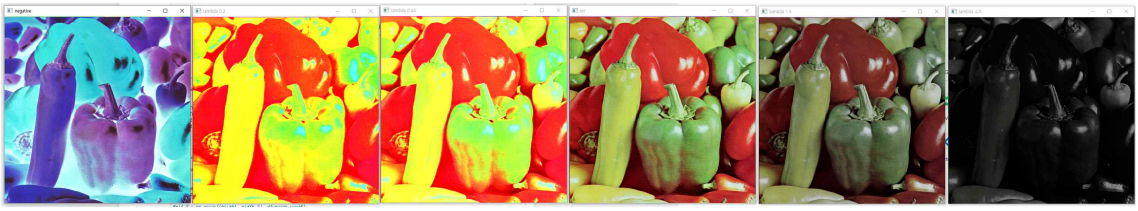
우선 지금은 Indensity와 Saturation을 건드린 값이다.

코드를 실행하기 이전의 결과를 한 번 예상해보았다. Indensity는 1 이하일 때 값이 작아질수록 이미지 색이 밝아지고 1 이상일 때는 값이 커질수록 이미지 색이 어두워질 것이고, Saturation은 1 이하일 때 값이 작아질수록 이미지 색의 채도가 높아지고 1 이상일 때는 값이 커질수록 이미지 색의 채도가 낮아질 것이다. 때문에 1 이하일 때 값이 작아질수록 색은 Indensity만 건드렸을 때보다 진하지만 밝은 이미지를, 1 이상일 때 값이 커질 수록 색은 Indensity만 건드렸을 때보다 옅지만 어두운 이미지 즉 거의 검정에 가까운 이미지를 출력할 것으로 예상했다.

Power-Law Transformation에서 Indensity, Saturation을 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Indensity, Saturation을 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

흑백 사진은 예상과는 조금 빗나갔다. iii)에서 Saturation 값만 바꿨을 때를 고려하지 않아서 또 놀랐다. 0.2의 경우 Indensity에 의해 이미지가 밝아지고, Saturation에 의해 채도가 높아지기 때문에 살짝 노랑색으로 보인다.
컬러 사진은 예상보다 채도가 더 진해져서 놀랐다. 거의 열화상 탐지기정도로 진해진 이유는 0.2라는 값이 1과 상당히 가깝기 때문에 Saturation 값이 거의 0에 가까워졌기 때문이라고 생각한다. 또, 4.0 람다의 경우 예상처럼 흰색이 거의 보이지 않을 정도로 까맣고, 채도가 거의 없었다. 이보다 람다 값이 더 커진다면 정말 검정색으로 보일 것이다.

1.-vi) Indensity, Hue 변경해 Power-law Transformation 구하기

```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

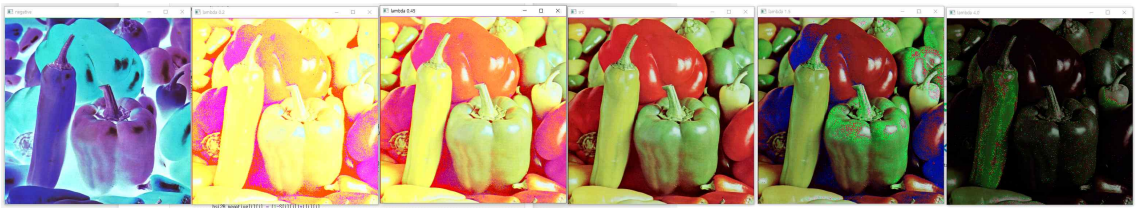
이번에는 Indensity와 Hue를 동시에 건드려보았다.

코드를 실행하기 전 또 결과를 한 번 예상해보았다. Indensity는 1 이하일 때 값이 작아질수록 이미지 색이 밝아지고 1 이상일 때는 값이 커질수록 이미지 색이 어두워질 것이고, Hue는 각도의 값이라 1 이하일 때 값이 작아질수록 이미지 색이 붉은 색으로 가까워질 것이고 1 이상일 때는 값이 커질수록 이미지 색이 푸른 색에 가까워질 것이라 예상했다. 때문에 1 이하일 때 값이 작아질수록 색은 Indensity만 건드렸을 때보다 붉지만 밝은 이미지를, 1 이상일 때 값이 커질 수록 색은 Indensity만 건드렸을 때보다 파랗지만 어두운 이미지 즉 거의 검정에 가까운 이미지를 출력할 것으로 예상했다.

Power-Law Transformation에서 Indensity, Hue를 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Indensity, Hue를 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

예상과는 조금 다른 이미지들이 나왔다. 우선 흑백 사진의 경우 Hue는 이미 0에 가깝다는 것을 iv) Hue만을 사용한 코드에서 확인을 했다. 따라서 Indensity만 이용해 Power-Law Transformation을 한 것처럼 Indensity 값에 많은 영향을 받은 것을 결과를 통해 확인할 수 있다.

하지만, 컬러의 경우 달랐다. 원본에서도 빨강, 초록, 노랑이 명확하게 보이기 때문에 값이 흑백과는 다를 것이라 예상했다. 하지만 람다 값이 엄청 작은 0.2에서는 파랑색은 핑크색으로, 초록색은 노랑색으로, 명도가 높고 약간 빨강 빛을 떠서 살짝 핑크색으로 보이는 곳은 연파랑(민트색)으로 변했다. 이 값을 보니 '이게 바로 강의에서 교수님께서 설명하신 RGB와 보색관계인 Subtractive colors인가?' 하는 생각이 들었다. 정확한 이유는 모르겠지만 Indensity의 값이 굉장히 올라가면서 전체적으로 명도가 올라가고, 보색관계가 된 것 같다.

람다 값이 크고 나름 색이 잘 구별되는 1.5에서는 내가 작성한 코드가 좀 잘못되었다는 것을 알았다. 아마 Hue 구간 설정을 잘못한 것 같다. if (Hue) 들로 Hue에 따라 RGB가 결정된다는 사실을 간과하고 if문 안에서 Hue 값을 임의로 지정해주는 바람에 Hue 값이 커질 때 빨강 부분은 완전 빨갛게, 초록 부분은 완전 초록색으로, 파랑 부분은 완전 파랗게 변한 것 같다. 코드를 수정하면, if문에 들어가기 전에 Hue = Hue**gamma 한 후에

```
while (Hue<0 or Hue>2*pi):
        if (Hue<0): Hue = Hue+2*pi
        elif (Hue>2*pi):  Hue = Hue-2*pi
```

라는 조건을 추가로 달 것 같다. Hue는 각도라 [0, 355]의 값이라 360도를 더하거나 빼면 원래 값 그대로의 값이 추출되기 때문이다.

1.-vii) Saturation, Hue 변경해 Power-law Transformation 구하기

```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

이번에는 Saturation과 Hue를 동시에 건드려봤다.

실행 전 예상으로는 Saturation에 의해서 lambda의 값이 1보다 작을 때 값이 작아질수록 살짝씩 노랑 빛을 띠게 될 것 같았고, 1보다 클 때는 값이 커질수록 채도를 잃어 생기 없는 회색이 될 것으로 예상했다.

여기서도 vi)와 같은 실수가 있었다. Hue 값을 Hue**gamma로 변경시키는 문장을 if문 이전으로 빼서 R, G, B 값이 보다 정확하게 나오게 해야한다.

while (Hue<0 or Hue>2*pi):

    if (Hue<0): Hue = Hue+2*pi

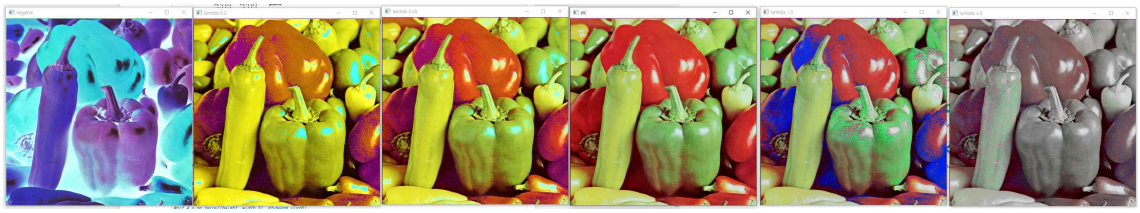
    elif (Hue>2*pi): Hue = Hue-2*pi

이 문장을 if문 이전에 넣어줘야한다!

Power-Law Transformation에서 Saturation, Hue를 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Saturation, Hue를 건드린 컬러사진



Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

예상과 같이 흑백사진의 경우 1보다 큰 값은 거의 변화가 없고, lambda(gamma)의 값이 거의 0에 가까워졌을 때 Saturation 값에 의해 노란 기가 보이는 결과를 출력한다.
컬러사진의 경우 0에 가까워질수록 엄청 노랗고, 핑크, 노랑, 민트색 위주의 보색이 보인다. 반면 1보다 큰 값 중 값이 더 커질수록 생기를 잃고 거의 회색으로 변하는 것을 확인할 수 있다. Sautration 값이 충분히 작아지지 않았을 때는 Hue값에 의해 빨강 값은 더 Hue 값이 0과 120 사이에 있었던 값은 Hue 값이 120으로(완전 빨강), 120과 240 사이에 있었던 값은 240으로(완전 초록), 240과 360 사이에 있었던 값은 360으로(완전 파랑) 가까워지는 것을 확인할 수 있다.

1.-viii) Indensity, Saturation, Hue를 모두 변경해 Power-law Transformation 구하기

```
for gamma in [0.2, 0.45, 1.5, 4.0]:
    hsi2B_power = np.zeros((height, width))
    hsi2G_power = np.zeros((height, width))
    hsi2R_power = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            if H[i][j] >= 0 and H[i][j] < 2*pi/3: #0도~120도
                hsi2B_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_negative[i][j] = (I[i][j]*3. - (hsi2R_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2B_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2R_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2G_power[i][j] = (I[i][j]*3. - (hsi2R_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

            if H[i][j] >= 2*pi/3 and H[i][j] < 4*pi/3:
                H[i][j] = H[i][j]-2*pi/3
                hsi2R_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_negative[i][j] = (I[i][j]*3 - (hsi2R_negative[i][j]+hsi2G_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2R_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2G_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2B_power[i][j] = (I[i][j]*3 - (hsi2R_power[i][j]+hsi2G_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+2*pi/3

            if H[i][j] >= 4*pi/3 and H[i][j] <= 2*pi:
                H[i][j] = H[i][j] - 4*pi/3
                hsi2G_negative[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_negative[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_negative[i][j] = (I[i][j]*3 - (hsi2G_negative[i][j]+hsi2B_negative[i][j]))

                I[i][j] = I[i][j] ** gamma
                S[i][j] = S[i][j] ** gamma
                H[i][j] = H[i][j] ** gamma
                hsi2G_power[i][j] = (1-S[i][j])*I[i][j]
                hsi2B_power[i][j] = (1+(S[i][j]*cos(H[i][j]) / cos(pi/3-H[i][j])))*I[i][j]
                hsi2R_power[i][j] = (I[i][j]*3 - (hsi2G_power[i][j]+hsi2B_power[i][j]))
                I[i][j] = I[i][j] ** (1/gamma)
                S[i][j] = S[i][j] ** (1/gamma)
                H[i][j] = H[i][j] ** (1/gamma)

                H[i][j] = H[i][j]+4*pi/3

    plt = cv2.merge((hsi2B_power, hsi2G_power, hsi2R_power))
    cv2.imshow("lambda "+str(gamma), plt)
```

이번에는 Indensity와 Saturation, Hue를 모두 동시에 건드려봤다.

실행 전 예상으로는 lambda의 값이 1보다 작을 때 값이 작아질수록 살짝 노랑 빛을 띠면서 엄청 밝게 될 것 같았고, 1보다 클 때는 값이 커질수록 채도를 잃어 생기 없는 회색에 명도가 더해져 까맣게 될 것으로 예상했다.

여기서도 vi), vii)와 같은 실수가 있었다. Hue 값을 Hue**gamma로 변경시키는 문장을 if 문 이전으로 빼서 R, G, B 값이 보다 정확하게 나오게 해야한다.

```
while (Hue<0 or Hue>2*pi):
        if (Hue<0): Hue = Hue+2*pi
        elif (Hue>2*pi):  Hue = Hue-2*pi
```

이 문장을 if문 이전에 넣어줘야한다!

Power-Law Transformation에서 Indensity, Saturaion, Hue를 건드린 흑백사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

Power-Law Transformation에서 Indensity, Saturaion, Hue를 건드린 컬러사진

Negative 0.2 lambda 0.45 람다 원본 1.5 람다 4.0 람다

예상대로 흑백사진은 0에 가까워질수록 더 밝고 노란 빛을 띠고, 1에 가까울수록 거의 검정색을 떴다.

컬러사진의 경우 0에 가까워질수록 명도가 높아져 거의 흰색에 가까워지니 컬러는 보색에 가까운 색을 띠게 되었고, 1에 가까워질수록 색과 흰색이 거의 없는 검정색으로 가까워졌다.

2. implementation of Histogram Equalization

```
# histogram - low
def histogram_low(value):
    height, width = value.shape[0], value.shape[1]
    value_low = 255
    value_high = 0
    for i in range(height):
        for j in range(width):
            if value_low >= value[i][j]:
                value_low = value[i][j]
            if value_high <= value[i][j]:
                value_high = value[i][j]

    for i in range(height):
        for j in range(width):
            if value[i][j] <= value_low:
                value[i][j] = 0
            elif value[i][j] > value_low and value[i][j] < value_high:
                value[i][j] = 255*(value[i][j] - value_low)/(value_high - value_low)
            else: #value[i][j] >= value_high
                value[i][j] = 255

    return value
```

histogram 형태로, min값과 max값을 사용해 자르기

```
# implementation of Histogram Equalization

src = cv2.imread('test image/dark.jpg', cv2.IMREAD_COLOR)
height, width = src.shape[0], src.shape[1]

hist = np.zeros((height, width,3), dtype=np.uint8)
inv_hist = np.zeros((height, width,3), dtype=np.uint8)

I = np.zeros((height, width))
S = np.zeros((height, width))
H = np.zeros((height, width))
inv_I = np.zeros((height, width))
inv_S = np.zeros((height, width))
inv_H = np.zeros((height, width))
hsi2B = np.zeros((height, width))
hsi2G = np.zeros((height, width))
hsi2R = np.zeros((height, width))

B = src[:,:,0]
G = src[:,:,1]
R = src[:,:,2]

for i in range(height):
    for j in range(width):
        B_tmp, G_tmp, R_tmp = B[i][j]/255., G[i][j]/255., R[i][j]/255.

        I[i][j] = (B_tmp+G_tmp+R_tmp)/3.
        S[i][j] = 1 - 3*np.min([B_tmp,G_tmp,R_tmp])/(B_tmp+G_tmp+R_tmp+0.00000001)
        H[i][j] = compute_Hue(B_tmp, G_tmp, R_tmp)

I = I*255

inv_H = H
inv_S = S
inv_I = histogram_low(I)/255

for i in range(height):
    for j in range(width):
        if inv_H[i][j] >= 0 and inv_H[i][j] < 2*pi/3: #0도~120도
            hsi2B[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2R[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2G[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2B[i][j]))

        if inv_H[i][j] >= 2*pi/3 and inv_H[i][j] < 4*pi/3:
            inv_H[i][j] = inv_H[i][j]-2*pi/3
            hsi2R[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2G[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2B[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2G[i][j]))

        if inv_H[i][j] >= 4*pi/3 and inv_H[i][j] <= 2*pi:
            inv_H[i][j] = inv_H[i][j] - 4*pi/3
            hsi2G[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2B[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2R[i][j] = (inv_I[i][j]*3 - (hsi2G[i][j]+hsi2B[i][j]))

inv_hist = cv2.merge((hsi2B, hsi2G, hsi2R))

cv2.imshow("1st histogram_", inv_hist)
cv2.imshow("1st photo", src)
```

1번째 사진 추출 코드

```
# Implementation of Histogram Equalization

src = cv2.imread('HE test/Fig0316(2)(2nd_from_top).jpg', cv2.IMREAD_COLOR)
height, width = src.shape[0], src.shape[1]

hist = np.zeros((height, width,3), dtype=np.uint8)
inv_hist = np.zeros((height, width,3), dtype=np.uint8)

I = np.zeros((height, width))
S = np.zeros((height, width))
H = np.zeros((height, width))
inv_I = np.zeros((height, width))
inv_S = np.zeros((height, width))
inv_H = np.zeros((height, width))
hsi2B = np.zeros((height, width))
hsi2G = np.zeros((height, width))
hsi2R = np.zeros((height, width))

B = src[:,:,0]
G = src[:,:,1]
R = src[:,:,2]

for i in range(height):
    for j in range(width):
        B_tmp, G_tmp, R_tmp = B[i][j]/255., G[i][j]/255., R[i][j]/255.

        I[i][j] = (B_tmp+G_tmp+R_tmp)/3.
        S[i][j] = 1 - 3*np.min([B_tmp,G_tmp,R_tmp])/(B_tmp+G_tmp+R_tmp+0.00000001)
        H[i][j] = compute_Hue(B_tmp, G_tmp, R_tmp)

I = I*255

inv_H = H
inv_S = S
inv_I = histogram_low(I)/255

for i in range(height):
    for j in range(width):
        if inv_H[i][j] >= 0 and inv_H[i][j] < 2*pi/3: #0도~120도
            hsi2B[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2R[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2G[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2B[i][j]))

        if inv_H[i][j] >= 2*pi/3 and inv_H[i][j] < 4*pi/3:
            inv_H[i][j] = inv_H[i][j]-2*pi/3
            hsi2R[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2G[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2B[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2G[i][j]))

        if inv_H[i][j] >= 4*pi/3 and inv_H[i][j] <= 2*pi:
            inv_H[i][j] = inv_H[i][j] - 4*pi/3
            hsi2G[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2B[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2R[i][j] = (inv_I[i][j]*3 - (hsi2G[i][j]+hsi2B[i][j]))

inv_hist = cv2.merge((hsi2B, hsi2G, hsi2R))

cv2.imshow("2nd histogram", inv_hist)
cv2.imshow("2nd photo", src)
```

2번째 사진 추출 코드(1번째와 동일, 사진 위에 뜨는 문구만 바꾼 것)

```
# Implementation of Histogram Equalization

src = cv2.imread('HE test/Fig0316(3)(third_from_top).jpg', cv2.IMREAD_COLOR)
height, width = src.shape[0], src.shape[1]

hist = np.zeros((height, width,3), dtype=np.uint8)
inv_hist = np.zeros((height, width,3), dtype=np.uint8)

I = np.zeros((height, width))
S = np.zeros((height, width))
H = np.zeros((height, width))
inv_I = np.zeros((height, width))
inv_S = np.zeros((height, width))
inv_H = np.zeros((height, width))
hsi2B = np.zeros((height, width))
hsi2G = np.zeros((height, width))
hsi2R = np.zeros((height, width))

B = src[:,:,0]
G = src[:,:,1]
R = src[:,:,2]

for i in range(height):
    for j in range(width):
        B_tmp, G_tmp, R_tmp = B[i][j]/255., G[i][j]/255., R[i][j]/255.

        I[i][j] = (B_tmp+G_tmp+R_tmp)/3.
        S[i][j] = 1 - 3*np.min([B_tmp,G_tmp,R_tmp])/(B_tmp+G_tmp+R_tmp+0.00000001)
        H[i][j] = compute_Hue(B_tmp, G_tmp, R_tmp)

I = I*255

inv_H = H
inv_S = S
inv_I = histogram_low(I)/255

for i in range(height):
    for j in range(width):
        if inv_H[i][j] >= 0 and inv_H[i][j] < 2*pi/3: #0도~120도
            hsi2B[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2R[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2G[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2B[i][j]))

        if inv_H[i][j] >= 2*pi/3 and inv_H[i][j] < 4*pi/3:
            inv_H[i][j] = inv_H[i][j]-2*pi/3
            hsi2R[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2G[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2B[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2G[i][j]))

        if inv_H[i][j] >= 4*pi/3 and inv_H[i][j] <= 2*pi:
            inv_H[i][j] = inv_H[i][j] - 4*pi/3
            hsi2G[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2B[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2R[i][j] = (inv_I[i][j]*3 - (hsi2G[i][j]+hsi2B[i][j]))

inv_hist = cv2.merge((hsi2B, hsi2G, hsi2R))

cv2.imshow("3rd histogram", inv_hist)
cv2.imshow("3rd photo", src)
```

3번째 사진 추출 코드(1번째와 동일, 사진 위에 뜨는 문구만 바꾼 것)

```
# Implementation of Histogram Equalization

src = cv2.imread('HE test/Fig0316(4)(bottom_left).jpg', cv2.IMREAD_COLOR)
height, width = src.shape[0], src.shape[1]

hist = np.zeros((height, width,3), dtype=np.uint8)
inv_hist = np.zeros((height, width,3), dtype=np.uint8)

I = np.zeros((height, width))
S = np.zeros((height, width))
H = np.zeros((height, width))
inv_I = np.zeros((height, width))
inv_S = np.zeros((height, width))
inv_H = np.zeros((height, width))
hsi2B = np.zeros((height, width))
hsi2G = np.zeros((height, width))
hsi2R = np.zeros((height, width))

B = src[:,:,0]
G = src[:,:,1]
R = src[:,:,2]

for i in range(height):
    for j in range(width):
        B_tmp, G_tmp, R_tmp = B[i][j]/255., G[i][j]/255., R[i][j]/255.

        I[i][j] = (B_tmp+G_tmp+R_tmp)/3.
        S[i][j] = 1 - 3*np.min([B_tmp,G_tmp,R_tmp])/(B_tmp+G_tmp+R_tmp+0.00000001)
        H[i][j] = compute_Hue(B_tmp, G_tmp, R_tmp)

I = I*255

inv_H = H
inv_S = S
inv_I = histogram_low(I)/255

for i in range(height):
    for j in range(width):
        if inv_H[i][j] >= 0 and inv_H[i][j] < 2*pi/3: #0도~120도
            hsi2B[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2R[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2G[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2B[i][j]))

        if inv_H[i][j] >= 2*pi/3 and inv_H[i][j] < 4*pi/3:
            inv_H[i][j] = inv_H[i][j]-2*pi/3
            hsi2R[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2G[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2B[i][j] = (inv_I[i][j]*3 - (hsi2R[i][j]+hsi2G[i][j]))

        if inv_H[i][j] >= 4*pi/3 and inv_H[i][j] <= 2*pi:
            inv_H[i][j] = inv_H[i][j] - 4*pi/3
            hsi2G[i][j] = (1-inv_S[i][j])*inv_I[i][j]
            hsi2B[i][j] = (1+(inv_S[i][j]*cos(inv_H[i][j]) / cos(pi/3-inv_H[i][j])))*inv_I[i][j]
            hsi2R[i][j] = (inv_I[i][j]*3 - (hsi2G[i][j]+hsi2B[i][j]))

inv_hist = cv2.merge((hsi2B, hsi2G, hsi2R))

cv2.imshow("4th histogram_", inv_hist)
cv2.imshow("4th photo", src)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

4번째 사진 추출 코드(1번째와 동일, 사진 위에 뜨는 문구만 바꾼 것)

여기서는 cv2.waitKey(0)와 cv2.detroyAllWindows() 사용해서 모든 창을 지우기 전에는 아무것도 실행하지 않고 잠시 결과창을 띠운 상태로 가만히 있으라는 것을 지시한다.

histogram으로 보정한 커피콩 사진들

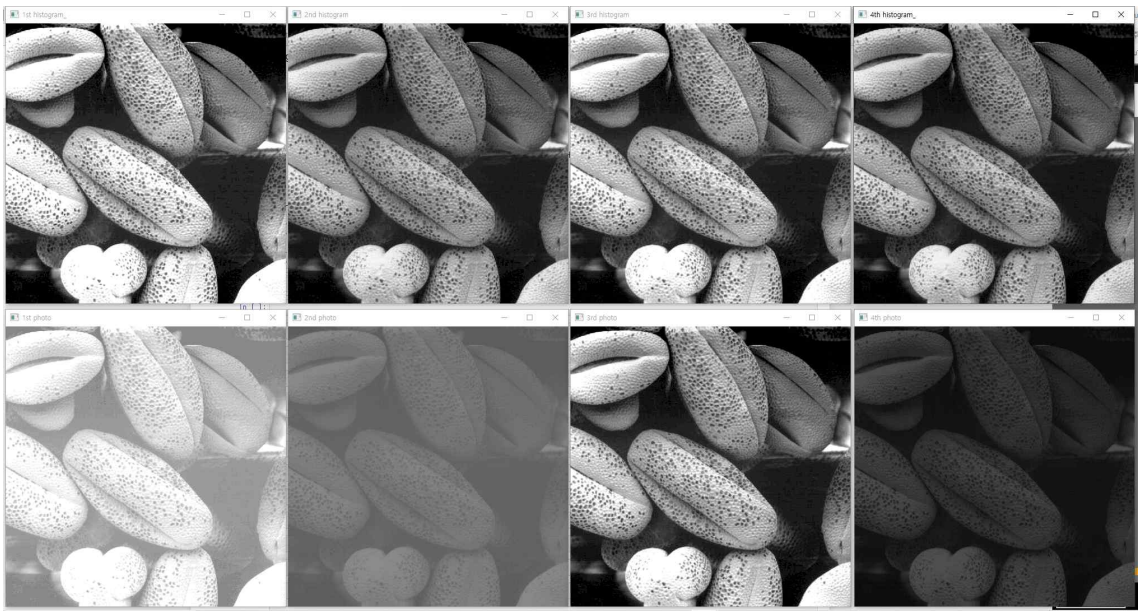


사진1(보정 후)	사진2(보정 후)	사진3(보정 후)	사진4(보정 후)
사진1(보정 전)	사진2(보정 전)	사진3(보정 전)	사진4(보정 전)

위와 같은 결과가 나온다. 한 번에 출력해서 비교해보려고 긴 코드들을 총 4개의 cell에 걸쳐 썼는데, 생각해보니 def로 '함수명(src)'을 선언해서 src 안에 사진만 바꿔 넣어주는 방법으로 했어도 될 것 같다. 그럼 확실히 코드의 길이가 짧아질 것 같다.
커피콩 사진을 비교해서 보자면, 각 사진별로 보정 후의 사진 명도 값이 조금씩 다르다. 그 이유는 원본 자체에서 갖고있던 기존의 HSI 값이 달랐기 때문인데, Indensity가 한 눈에 볼 때도 제일 밝아보이는 맨 왼쪽, 사진 1이 보정 후에도 가장 밝게 나왔고, Indensity가 가장 어두워보이는 사진 4가 보정 후에도 가장 어둡게 아놨다. 하지만 사진 2와 사진 4의 보정 값에 큰 차이가 있어보이진 않는다.
웬만하면 같은 사진의 경우, 원본이 어떻든 비슷한 값으로 정규화되는 것으로 보인다.