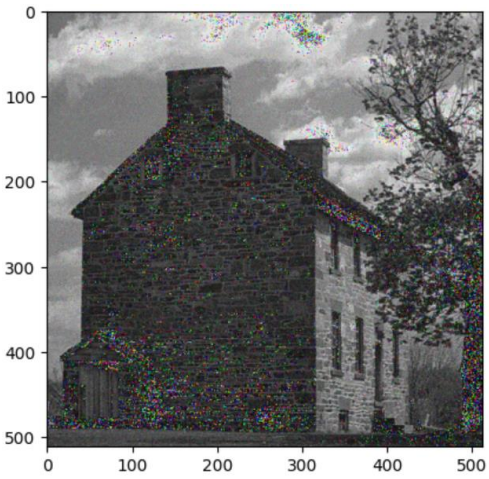
HW2

2022019734 김민서

1. Mean Filtering

먼저, np.random.normal()을 사용해서 평균이 0이고, 표준편차가 10인 정규분포를 따르는 noise를 만든 뒤, 이를 원래 이미지에 더하여 아래와 같은 이미지를 만들었다.

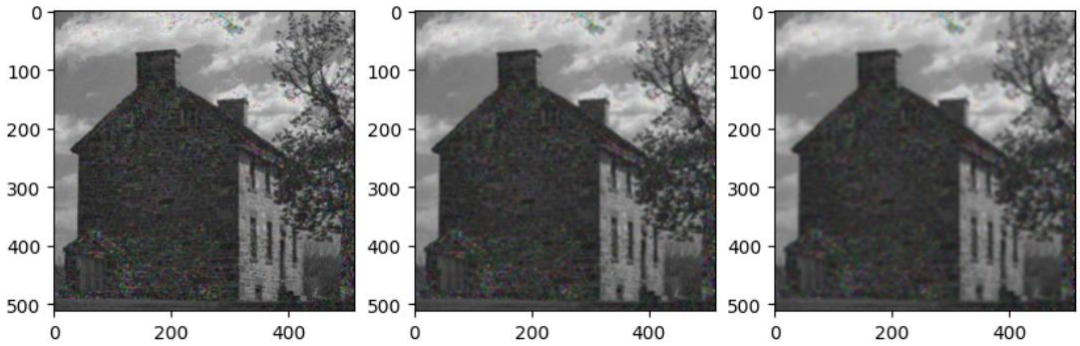


그 후, Convolution을 적용하기 위한 apply_filter을 다음과 같이 정의했다. 각각의 채널에 대해서, 이미지의 기준점인 (x, y)에서 (kernel_size) / 2 이내로 떨어진 점들에 대해서 elementwise multiplication을 한 뒤, 그 합을 최종적으로 저장하는 것을 볼 수 있다.

```
def apply_filter(img, kernel, kSize):
  offset = kSize // 2
  ret = np.zeros((h, w, 3))
  for ch in range(3):
    for i in range(h):
      for j in range(w):
        if i < offset or i > h - offset - 1 or j < offset or j > w - offset - 1
          ret[i][j][ch] = img[i][j][ch]
          continue
        #decide val for ret[i][j]
        i_st = i - offset
        j_st = j - offset
        val = 0
        for k in range(kSize):
          for l in range(kSize):
            val += img[k + i_st][l + j_st][ch] * kernel[k][l]
        ret[i][j][ch] = val
  return ret.astype(np.uint8)
```

그리고, 3 * 3, 5 * 5, 7 * 7 크기의 mean filter을 만든 뒤, 이를 apply_filter()을 통해 noisy한 이미지와 convolution함으로써 noise가 줄어들었지만 blurry한 결과를 얻을 수 있었다. 아래 그림에서 볼 수 있듯이, 커널 사이즈가 커질수록 noise는 줄어들지만, blur가 점점

심해지는 것을 확인할 수 있다.



아래의 수식을 이용하여 아래와 같이, psnr을 계산하는 함수를 만들었다. np.mean()을 이용해서 mse를 계산한 후, uint8 형식이므로 pixel range에 255를 대입했다.

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\
&= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)
\end{aligned}
$$

```python
def get_psnr(img1, img2):
  mse = np.mean((img1 - img2) ** 2)
  if mse == 0:
    raise Exception("MSE is 0")
  psnr = 10 * math.log10(255 **2 / mse)
  return psnr
```

그 결과, noisy 이미지는 약 30.3, mean filtered 이미지는 각각 29.1, 28.9, 28.8의 psnr을 얻을 수 있었다. 이는 원래 이미지의 noise가 그렇게 심하지 않았기에 noise를 없앰으로써 얻는 이미지의 품질 향상보다, blur로 인한 품질 하락이 더 컸다고 해석할 수 있다.

2. Unsharp Masking

먼저, unsharp masking을 수행하기 위해서, low pass filter인 gaussian filter을 구현하는 코드를 아래와 같이 작성했다. Low pass filter는 sum to 1이 되어야 하기에 최종적으로 ret.sum()으로 나눠주는 것을 확인할 수 있다.

```python
def getGaussian(size, std):
  ret = np.zeros((size, size))
  for i in range(size):
    for j in range(size):
      x = (i - size // 2)
      y = (j - size // 2)
      ret[i][j] = np.exp(-(x ** 2 + y ** 2) / (2 * std ** 2)) / (2 * np.pi * std ** 2)
  ret /= ret.sum()
  return ret
```

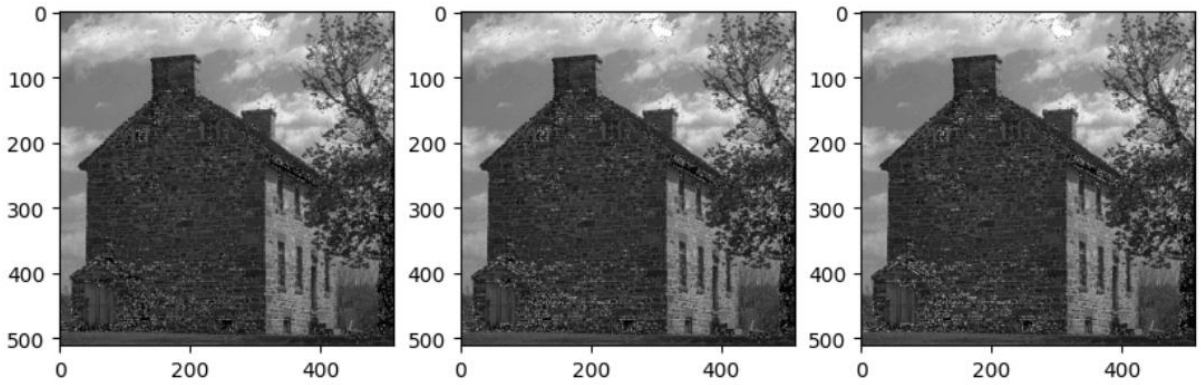
그 후, 아래 수식에 α = 0.1을 대입하고, low pass filter H로 위의 gaussian filter을 사용하여 sharpen filter을 만들 수 있었다.

$$F + \alpha(F - \underbrace{F * H}_{\text{blurred image}}) = (1+\alpha)F - \alpha(F * H) = F * ([1+\alpha]e - \alpha H)$$

image (↑ F)

(Identity kernel) (↑ e)

아래 그림에서 확인할 수 있듯이, kernel size가 커짐에 따라, 집 앞쪽의 edge가 더 부각되는 것을 알 수 있다. 또한, psnr은 각각 43.1, 41.4, 40.7을 얻었는데, 이는 edge가 부각될수록 원본 이미지와는 멀어져서 psnr이 떨어진 것으로 보인다.



3. Contrast Stretching

1) Contrast stretching

아래 수식에 따라, contrast stretching을 하는 코드를 작성했다. a, b, c는 각각 기울기를 나타내고, x1과 x2는 아래 수식에서의 a와 b를, 마지막으로 u는 input을 나타낸다.

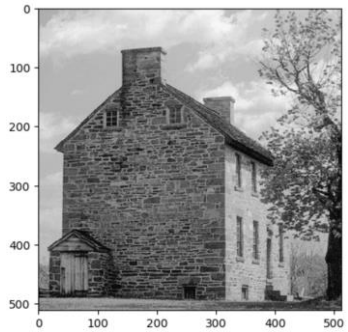
- **Contrast Stretching**

$$v = f(u), u \in [0, L], v \in [0, L]$$

$$v = \begin{cases} \alpha u, & 0 \le u < a \\ \beta(u-a) + v_a & a \le u < b \\ \gamma(u-b) + v_b & b \le u < L \end{cases}$$

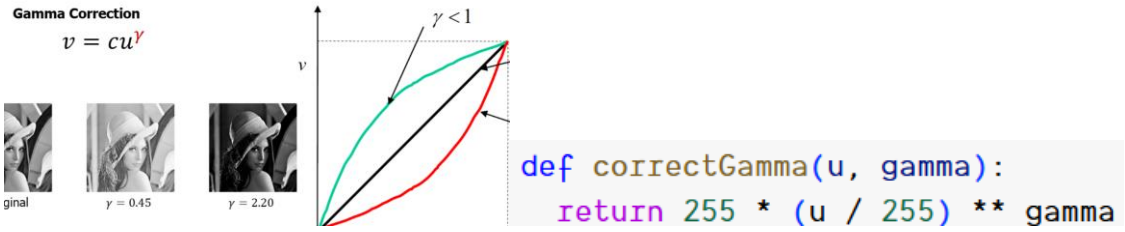
```
def stretchContrast(u, a, b, c, x1, x2):
  if u < x1:
    return a * u
  elif u < x2:
    return b * (u - x1) + a * x1
  else:
    return c * (u - x2) + b * (x2 - x1) + a * x1
```

a = 2, b = c = 0.5로 설정한 후, 위의 함수를 원본 이미지에 적용해보니, 전체적으로 intensity가 상승해서 밝아진 것을 확인할 수 있었다(psnr = 27.8).

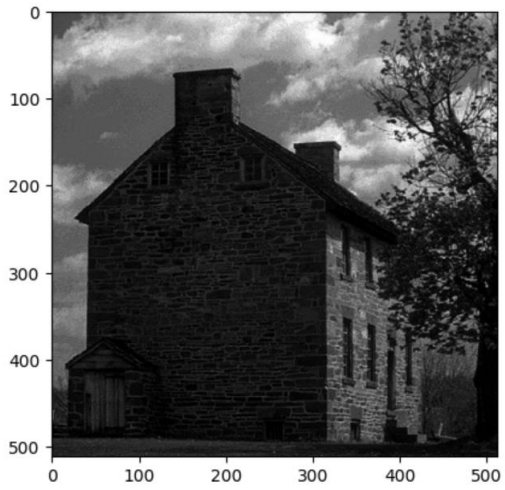


2) Gamma Correction

아래 수식에 따라서, 아래와 같이 코드를 작성했다. 유의할 점은, u가 0 ~ 1 범위가 되도록, 255를 나눠준 뒤, 다시 255를 곱해서 값을 얻는다는 것이다.

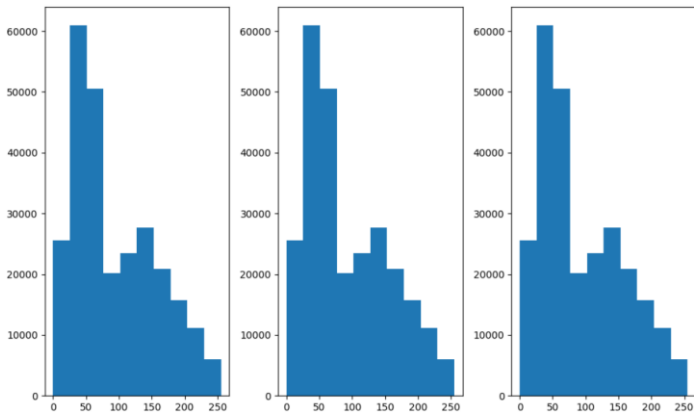


Gamma 값으로 1보다 큰 1.5를 사용했더니, 아래와 같이 원본보다 전체적으로 어두운 이미지를 얻을 수 있었다. 밝은 부분의 contrast가 어두운 곳의 contrast보다 크다는 점을 유의할 만하다. (psnr = 27.8)



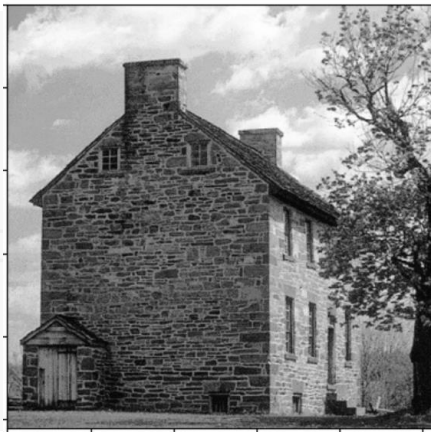
4. Histogram Equalization

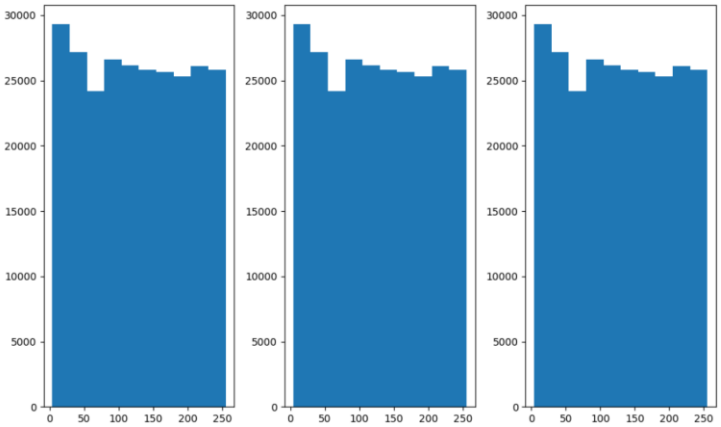
원본 이미지의 histogram을 확인해보니 intensity가 50 근처에서 집중되는 것을 확인할 수 있었다.



아래와 같이 함수를 작성하여 histogram equalization을 수행했다. Freq에 각 intensity에 대한 cdf를 담은 뒤, 이를 255와 곱해서, 최종적인 결과값을 얻어낼 수 있었고, 그 결과 전반적으로 어두웠던, 원본 이미지와 달리 intensity가 균일하게 분포된 것을 알 수 있다. 이는 equalized된 이미지의 histogram을 확인하면 더욱 분명해진다. (psnr = 28)

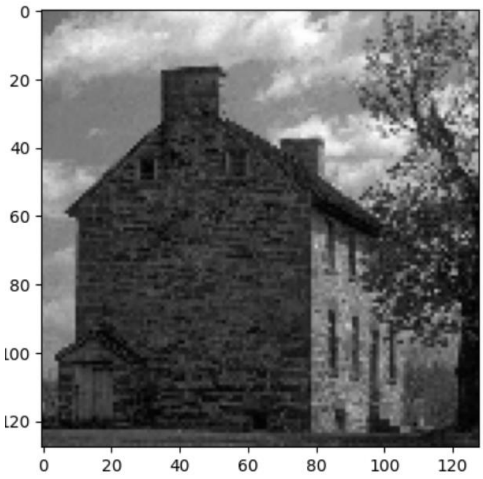
```
def equalHist(img):
  freq = np.zeros(256)
  ret = np.zeros((h, w))
  for i in range(h):
    for j in range(w):
      freq[img[i][j]] += 1
  for i in range(1, 256):
    freq[i] += freq[i - 1]
  freq /= w * h
  for i in range(h):
    for j in range(w):
      ret[i][j] = freq[img[i][j]] * 255
  return ret.astype(np.uint8)
```





5. Image Upsampling

먼저, cv2.resize()를 이용해서 원래 이미지보다 4배 작은 이미지를 아래와 같이 얻었다.



그 후, cv2.resize()의 interpolation에 각각 nearest neighbor, bilinear, bicubic interpolation을 할당함으로써 upsampling을 하고, 아래와 같은 이미지를 얻을 수 있었다. Psnr은 각각 28.9, 29, 28.9가 나왔는데, 원래 이미지 자체가 크게 복잡하지 않았기에 세 방법 사이의 유의미한 차이가 나타나지 않았다고 판단된다.