2016125003
소프트웨어학과
구영민

IPython과 Numpy로 이미지 가지고 놀기

0. 사용할 라이브러리 import하기

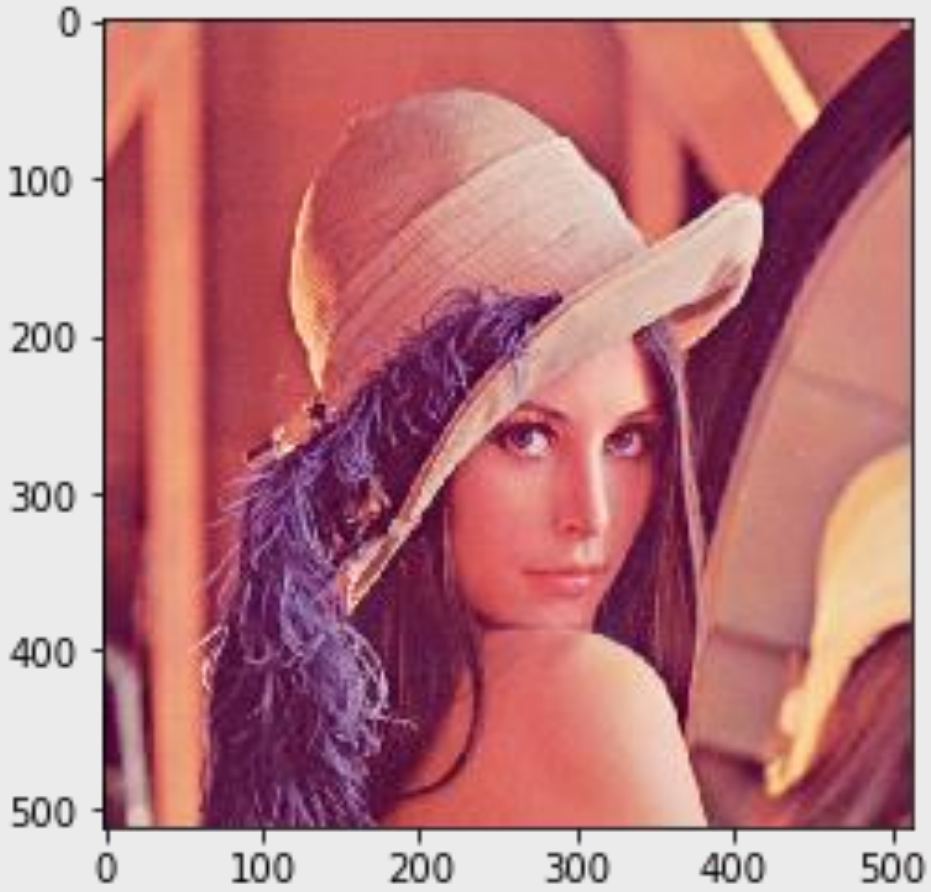
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread

from scipy.stats import norm
    # Normal Distribution의 PDF를 그리기 위해 사용합니다.
```

1. Image를 읽어 Numpy Array로 읽기

```
In [1]:
 img = imread("lena.tiff")
 plt.imshow(np.uint8(img))
 plt.show()
```

scipy.misc.imread 모듈을 이용하여 이미지 파일을 읽을 수 있습니다.

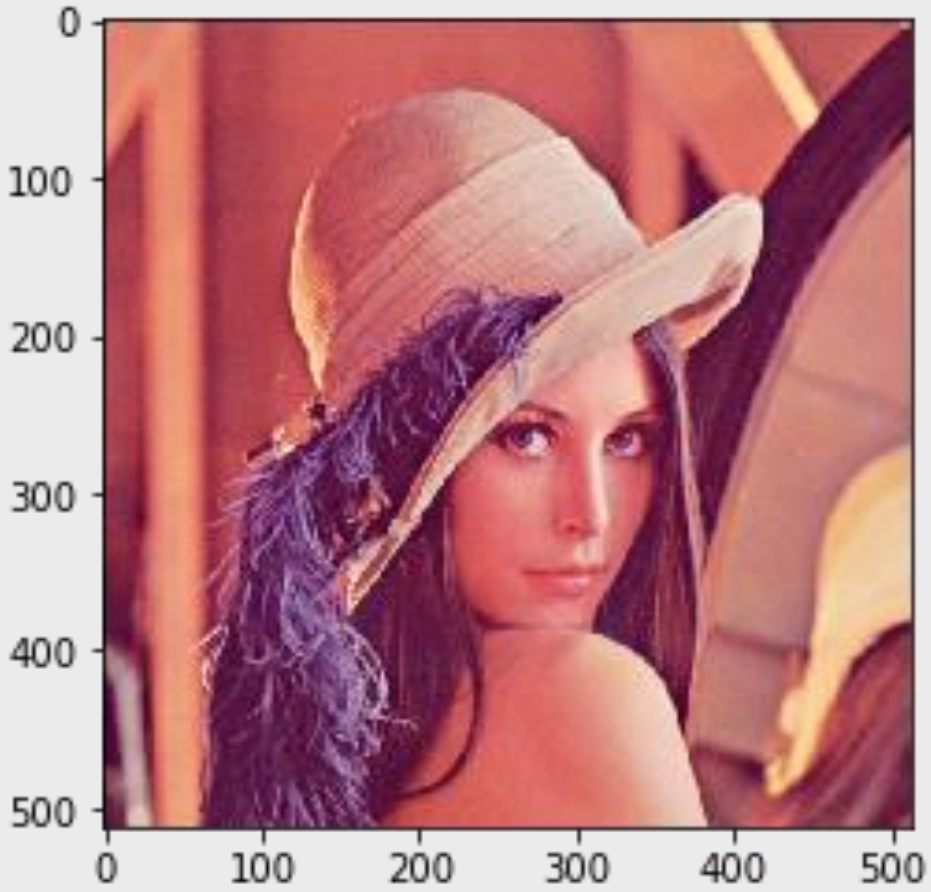
Out[1]:



1. Image를 읽어 Numpy Array로 읽기

```
In [2]: img.dtype, img.shape
Out[2]: (dtype('uint8'),
         (512, 512, 3))
```

어떤 모양을 가지는지 확인해 보았더니,
512 by 512 by 3 모양을 가지고 있음을
알 수 있습니다. 마지막의 3개에는 각각
Red, Green, Blue 값이 저장되어 있습니다.

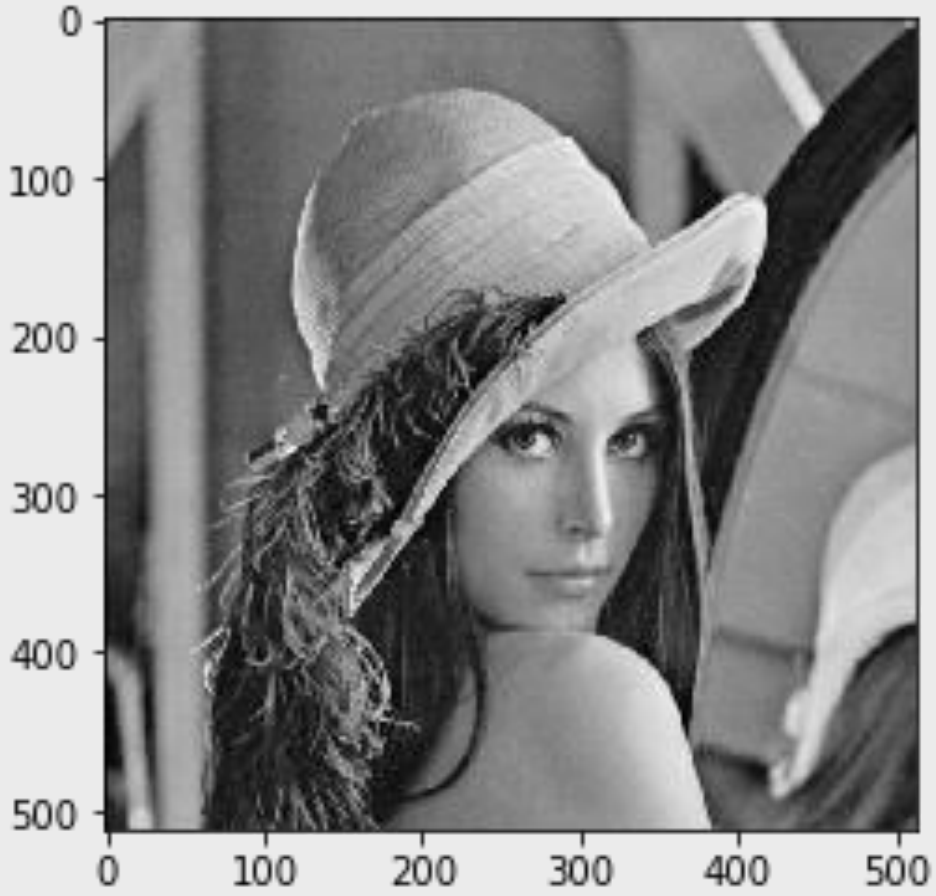


2. Image를 grayscale하기

```
In [3]:
 grayscale =
  np.uint8(
   img.dot([1./3, 1./3, 1./3]))
 plt.imshow(grayscale,
  cmap=plt.get_cmap('gray'))
 plt.show()
```

RGB 부분과 적절한 가중치를 주는 행렬을 dot product 해 주면 이미지가 grayscale됩니다. Color map을 지정하지 않으면 이상한 색으로 출력되기 때문에 gray로 설정합니다.

Out[3]:

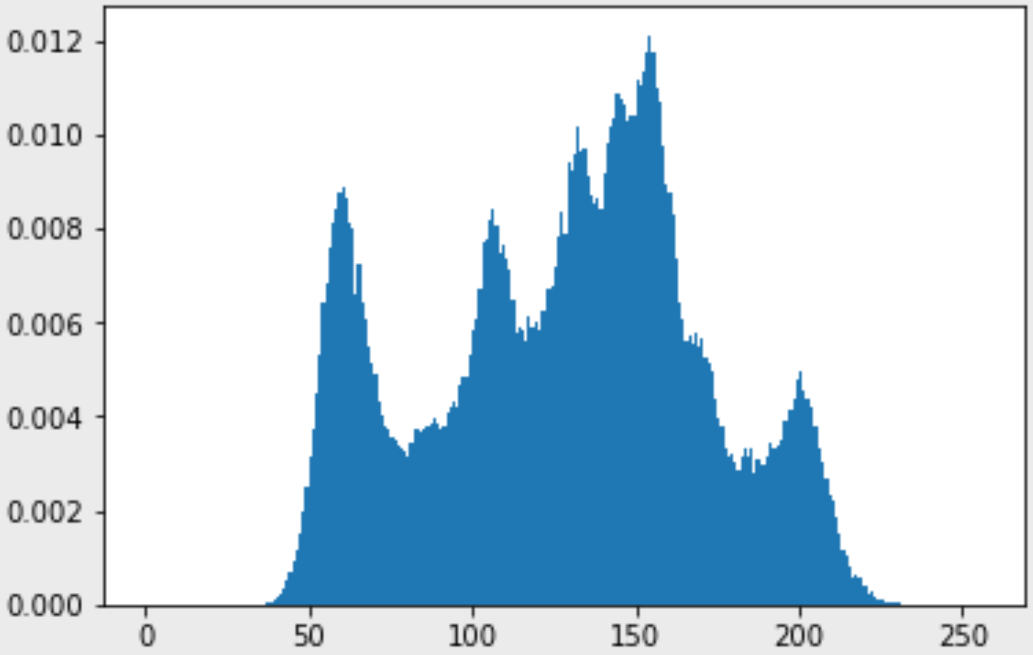


출처: http://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python

3. 히스토그램 생성

```
In [4]:
 flat = grayscale.flatten()
 plt.hist(flat, bins=256,
  range=[0, 256], normed=True)
 plt.show()
```

Out[4]:



grayscale은 512 by 512 배열입니다. flatten() 함수는 이를 크기가 262144인 1차원 배열로 만들어줍니다. 이렇게 만들어진 배열을 plt.hist에 넘기면 친절하게도 빈도수를 세어서 히스토그램을 그려 줍니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

```
In [5]:
 hist, bin_edges =
  np.histogram(flat, bins=256,
   range=[0, 256], density=True)
 cumsum = np.cumsum(hist)
```

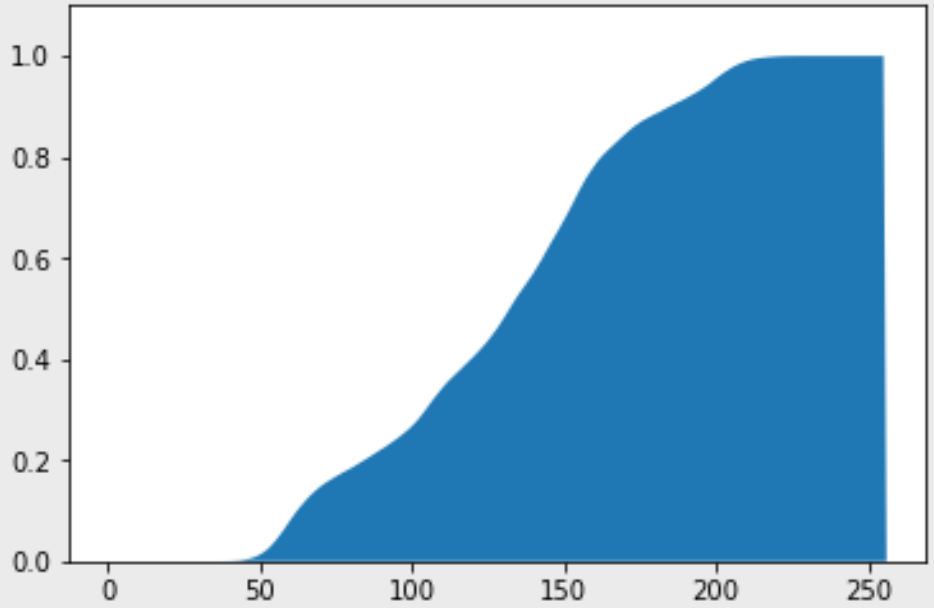
CDF를 먼저 구해야 합니다. 고맙게도, Numpy에는 누적 합을 구하는 함수가 준비되어 있습니다. 이를 이용하면 CDF를 매우 쉽게 구할 수 있습니다.

matplotlib에도 hist 함수가 있고, numpy에도 histogram 함수가 있습니다. matplotlib.hist은 즉시 그래프를 그려주지만, numpy.histrogram은 히스토그램의 구성 요소인 배열 두 개를 반환합니다. 여기서는 누적 합을 구하기 위해 numpy.histrogram을 사용하였습니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

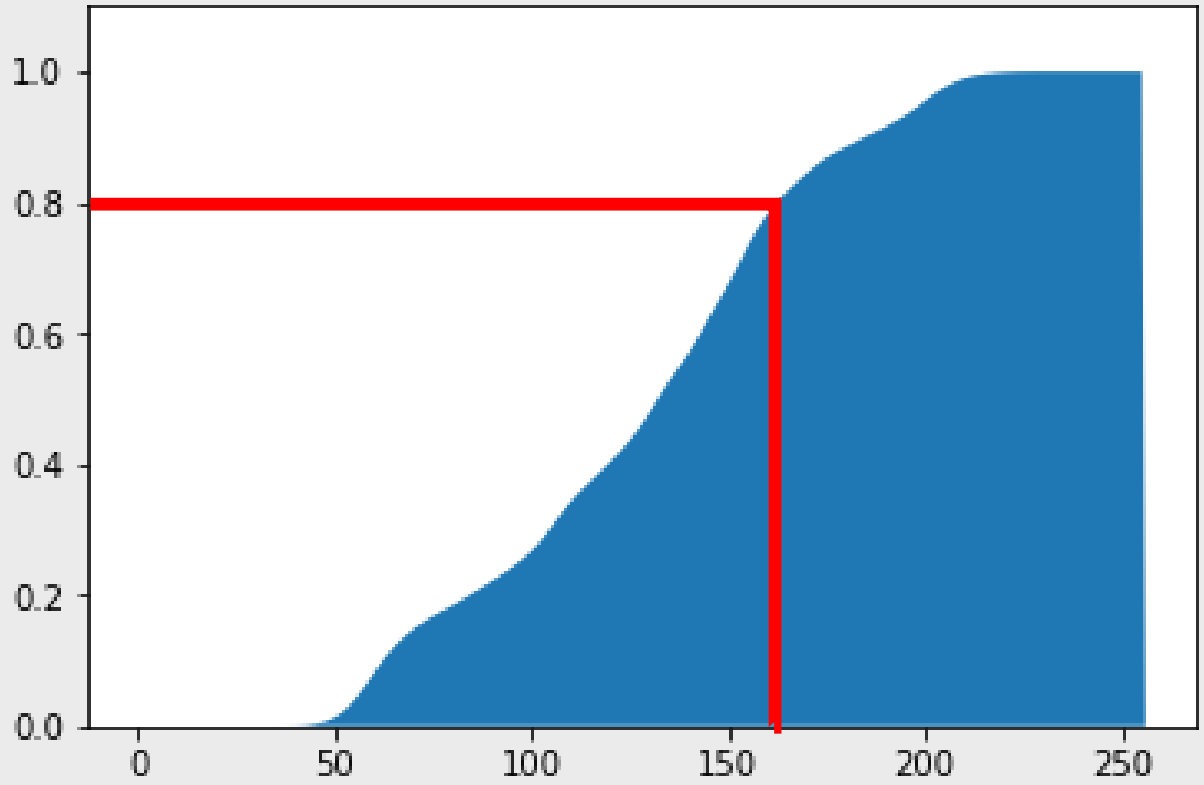
```
In [6]:
 plt.fill(bin_edges, cumsum)
 plt.ylim(0, 1.1)
 plt.show()
```

Out[6]:



CDF를 시각화해 보았습니다.

4. 생성한 히스토그램으로 Normal Distribution 생성



히스토그램에서 많이 나온 부분은 많이 뽑고, 적게 나온 부분은 적게 뽑기 위하여,
0에서 1 사이의 값을 넣으면 x축과 만나는 부분의 값을 반환하는 함수를 만들어야 합니다.
예를 들면, CDF(0.8) = 163와 같은 함수를 만들면 됩니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

```
In [7]:
 def cdf(x):
     for index, value in enumerate(cumsum):
         if x <= value:
             return index
     return 0
```

cumsum 배열에서 x보다 커지는 가장 작은 index를 반환하는 함수를 구현하는 식으로 간단하게 구현하였습니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

```
In [8]:
 def clt():
     return np.vectorize(cdf)(np.random.sample(10)).mean()
 clt_values = np.array([clt() for _ in range(20000)])
```

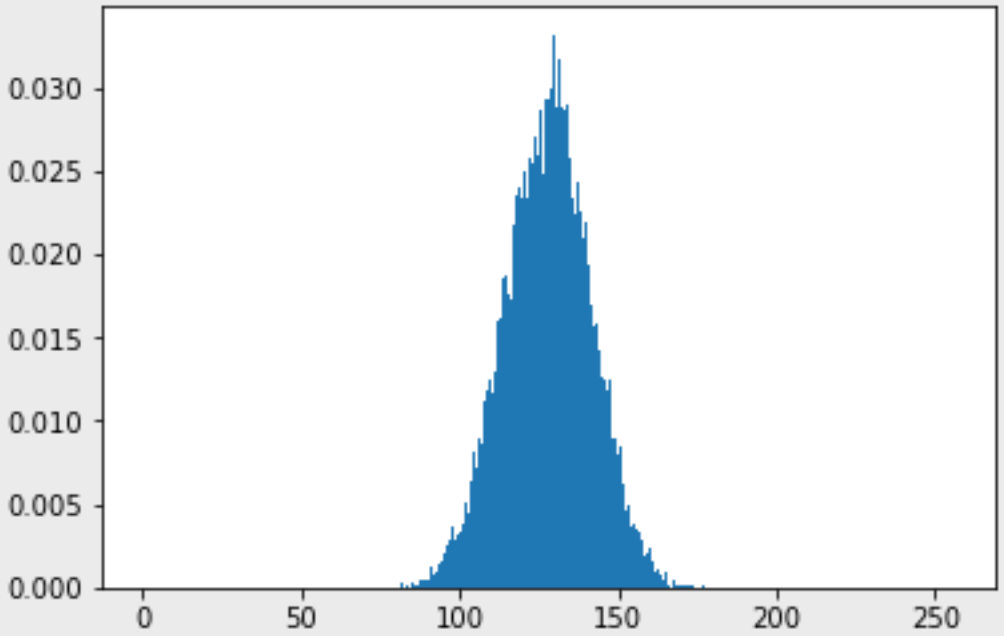
이제 본격적으로 랜덤으로 추출하였을 때 CLT의 내용처럼 Normal Distribution 모양으로 나오는지 확인하기 위해 코드를 작성했습니다. clt() 함수는 cdf() 함수를 10번 호출해 그 결과값의 평균을 내는 함수입니다. clt_values에는 clt() 함수를 20,000번 호출해 나온 결과를 저장합니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

In [9]:

```
plt.hist(clt_values, bins=512,
 range=[0, 256], normed=True)
plt.show()
```

Out[9]:

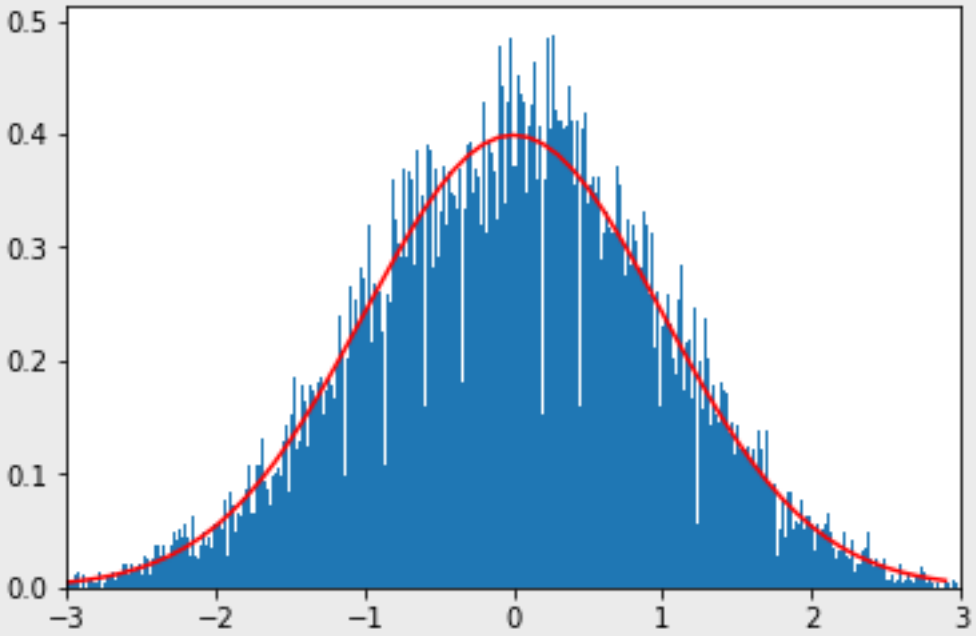


clt_values로 히스토그램을 그렸더니, 깔끔한 종 모양이 나왔습니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

Out[10]:

```
In [10]:
 clt_mean = clt_values.mean()
 clt_var = clt_values.var()
 clt_Z = (clt_values - clt_mean) / (clt_var ** 0.5)
 plt.hist(clt_Z, bins=512, normed=True)
 plt.plot(np.arange(-3, 3, 0.1),
  norm.pdf(np.arange(-3, 3, 0.1), 0, 1), 'r')
 plt.xlim(-3, 3)
 plt.show()
```



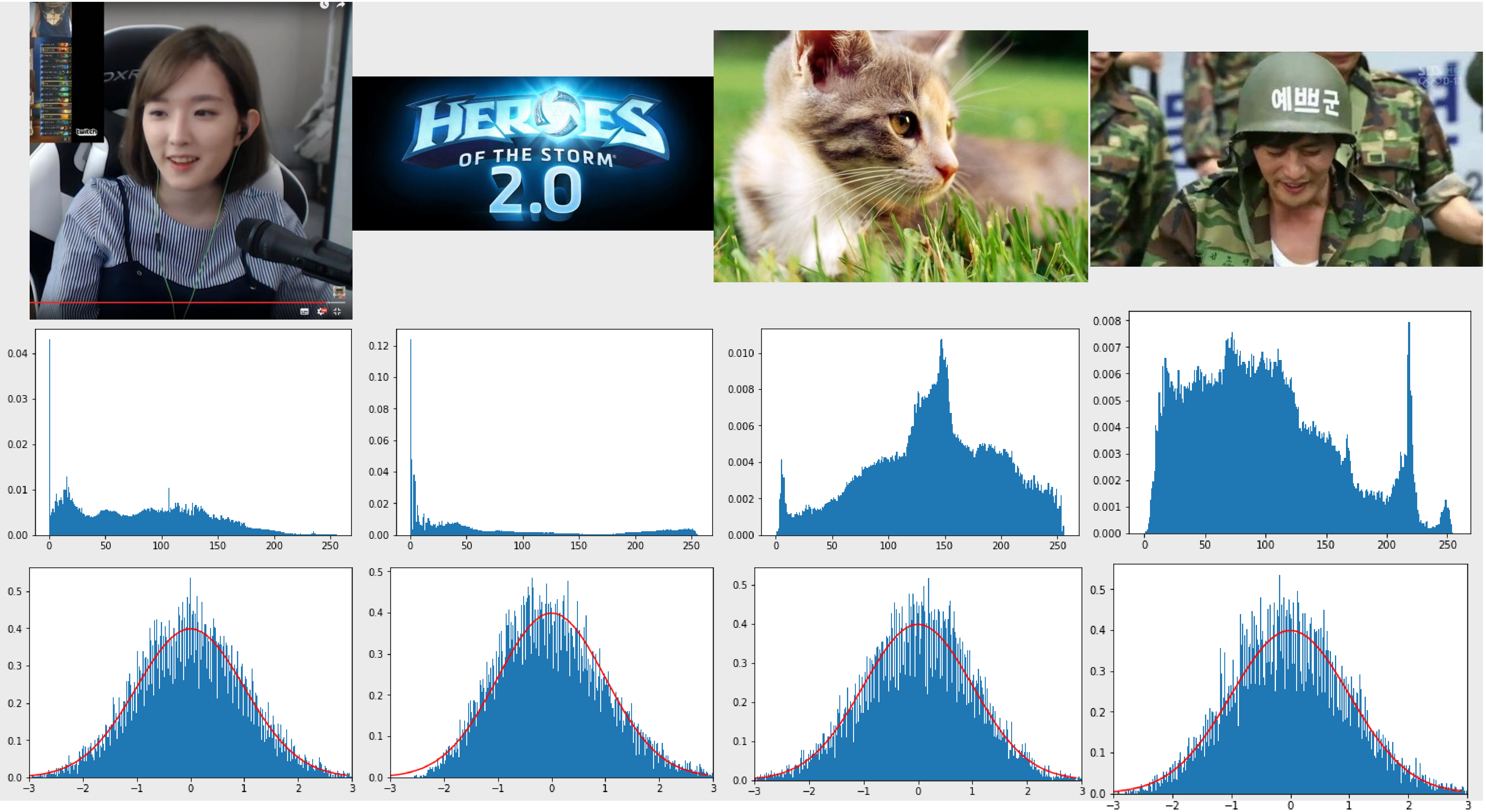
나온 히스토그램이 얼마나 정확하게 종 모양을 따르는지 확인해 보고 싶었습니다. 그러기 위해서는 데이터 집단을 정규화하면 됩니다. 정규화한 후 Normal Distribution을 같이 그려보았습니다. 표본의 크기가 커서 그런지, 상당히 정확하게 잘 따르는 모습을 볼 수 있었습니다.

4. 생성한 히스토그램으로 Normal Distribution 생성

Lena 이미지는 색분포가 비교적 균일하기 때문에,
CLT를 취해도 종 모양에 가깝게 결과가 나왔습니다.

우연의 일치일까요?

다른 이미지로도 실험해 보았습니다.


twitch
HEROES
OF THE STORM
2.0

4. 생성한 히스토그램으로 Normal Distribution 생성

여러 가지 분포를 가진 히스토그램으로 확인해 보아도 결국 종 모양이 만들어짐을 확인할 수 있었습니다.