Sources:

https://github.com/maxbozza/HOG-SVMlight-trainer/blob/master/train_SVMlight.py

https://github.com/aarcosg/object-detector-svm-hog-python

2-)

- İlk ödevde 'images/jpg/building.tif' görüntüsü ile geliştirdiğiniz ölçek uzayı üzerinde

yerel maksimum/yerel minimum noktaları belirlenecektir.

- Ölçek uzayı olarak DoG filtreleme neticesinde ürettiğiniz 7 görüntüyü kullanabilirsiniz.
- Yerel maksimum noktaları her bir pikselin 26-komşuluğu (9 komşu bir önceki ölçek, 8 komşu mevcut ölçek, 9 komşu bir sonraki ölçek için) kontrol edilerek belirlenecektir.
- Yerel minimum için ise yerel maksimum bulmada kullandığınız sürecin aynısını ölçek uzayının işaretini tersine çevirerek çalıştırınız.
- Elde ettiğiniz yerel maksimum ve minimum noktalarının tam sayı değerleri üzerinde uygun bir eşik değeri kullanınız (0,005 gibi küçük bir değer olabilir veya noktaların tamsayı değerlerine göre belirlenmiş bir değer olabilir, örneğin 0,05 * en büyük tamsayı değeri gibi).
- Son olarak belirlediğiniz yerel noktaların kenar olma olasılığı yüksek olanlarını temizlemek için Hessian matrisine bağlı aşağıdaki koşulu kullanınız.

$0 \leq tr(H\sigma)2 \,/\, \det(H\sigma) \leq 12 \; ki \; burada \; H\sigma = [L\sigma xx \; L\sigma xy \; L\sigma yx \; L\sigma yy] \; ve \; L\sigma xx = \partial 2(L\sigma) \,/\, \partial x2$

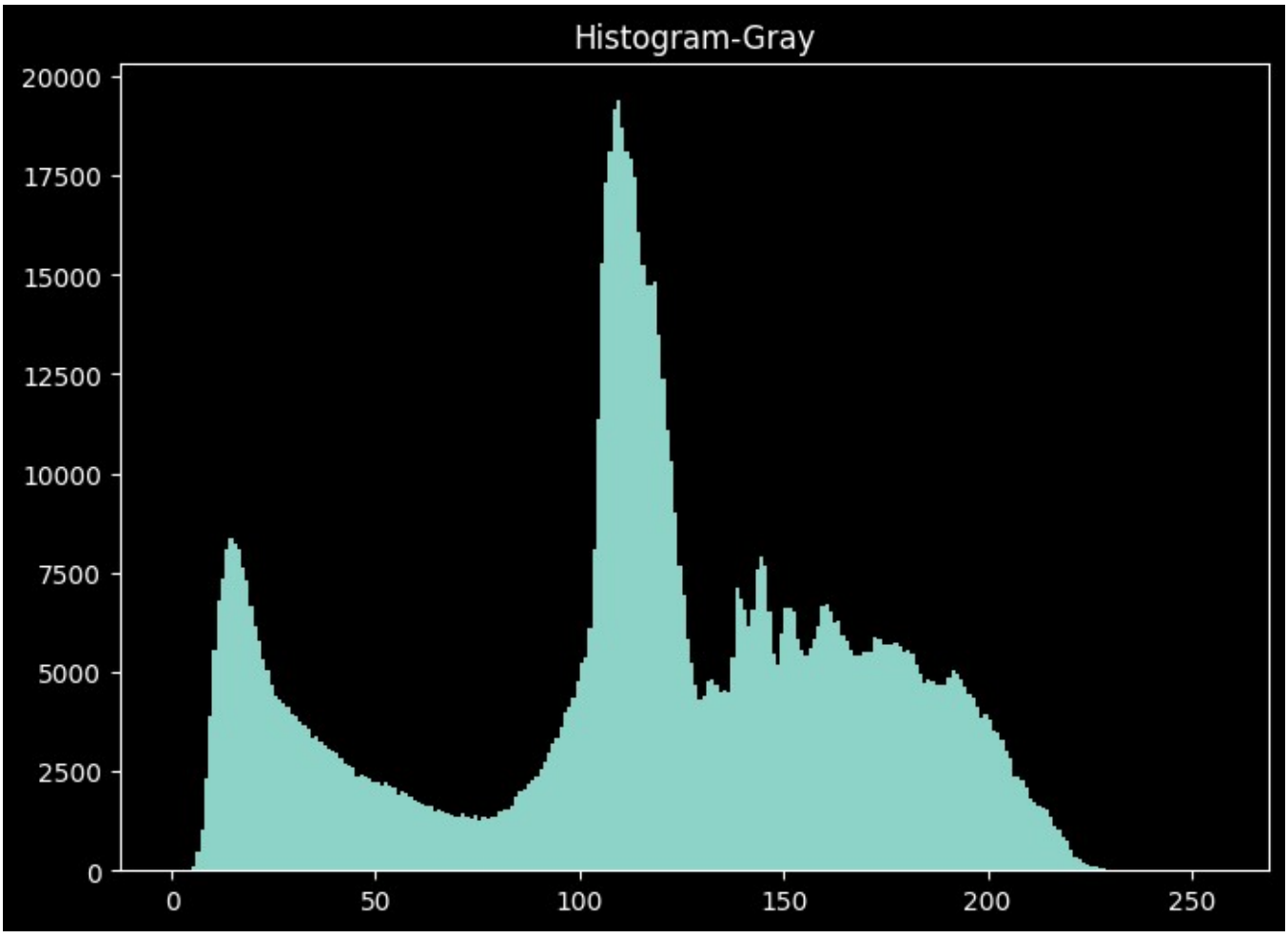
- Elde ettiğiniz ilgi noktalarını 'building.tif' görüntüsü üzerinde genişletme ile gösteriniz.

In [ ]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

from utils import img_read, show_img, show_hist
```

In [ ]:
```
from utils import img_read
img = img_read('../images/jpg/eiffel2-1.jpg', ret_gray=True, show=False)
show_hist(img, channel_order="RGB", cumulative=False, bins=256)
```

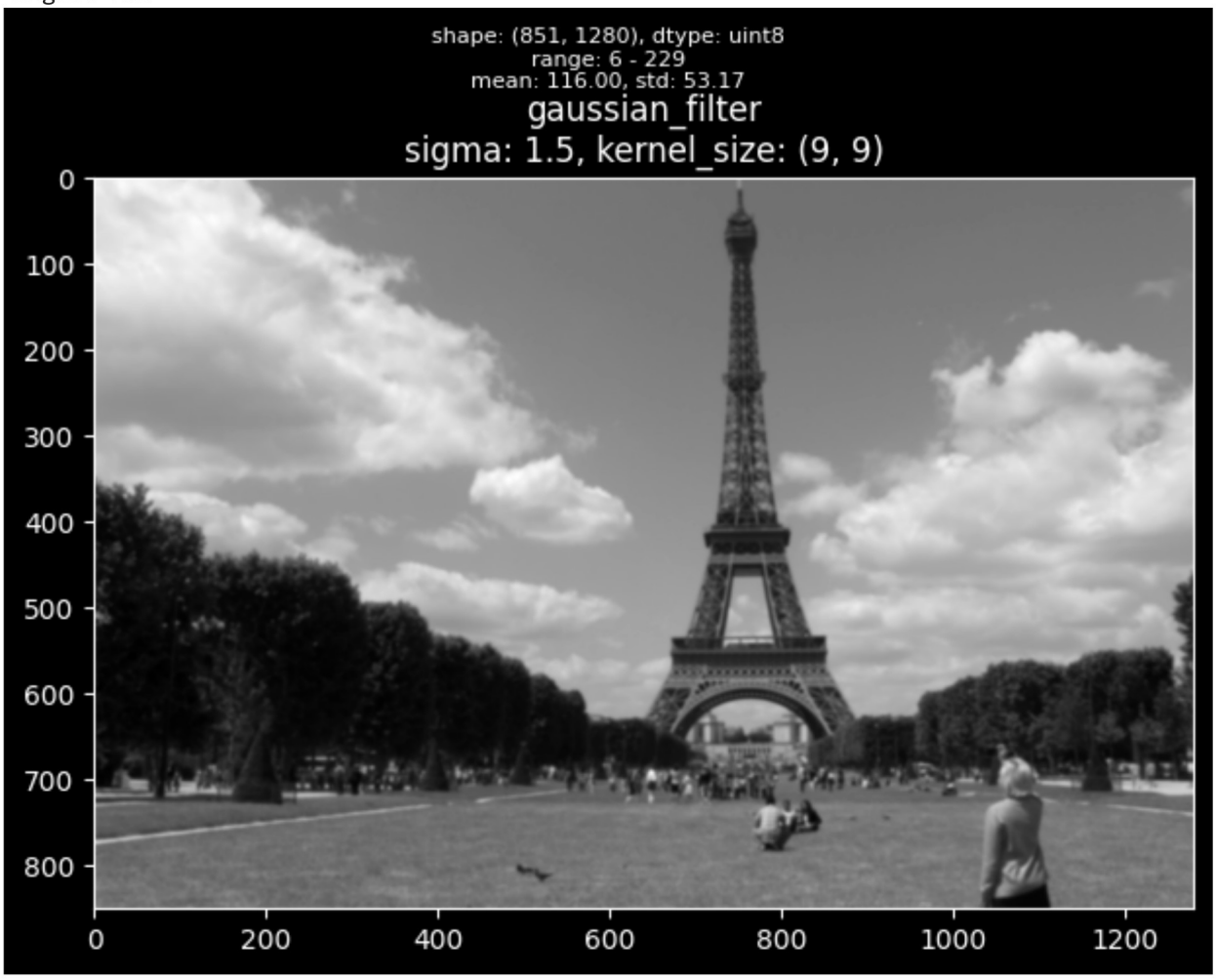

shape: (851, 1280), dtype: uint8
range: 3 - 254
mean: 116.00, std: 54.12
Image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


Histogram-Gray
20000
17500
15000
12500
10000
7500
5000
2500
0
0
50
100
150
200
250

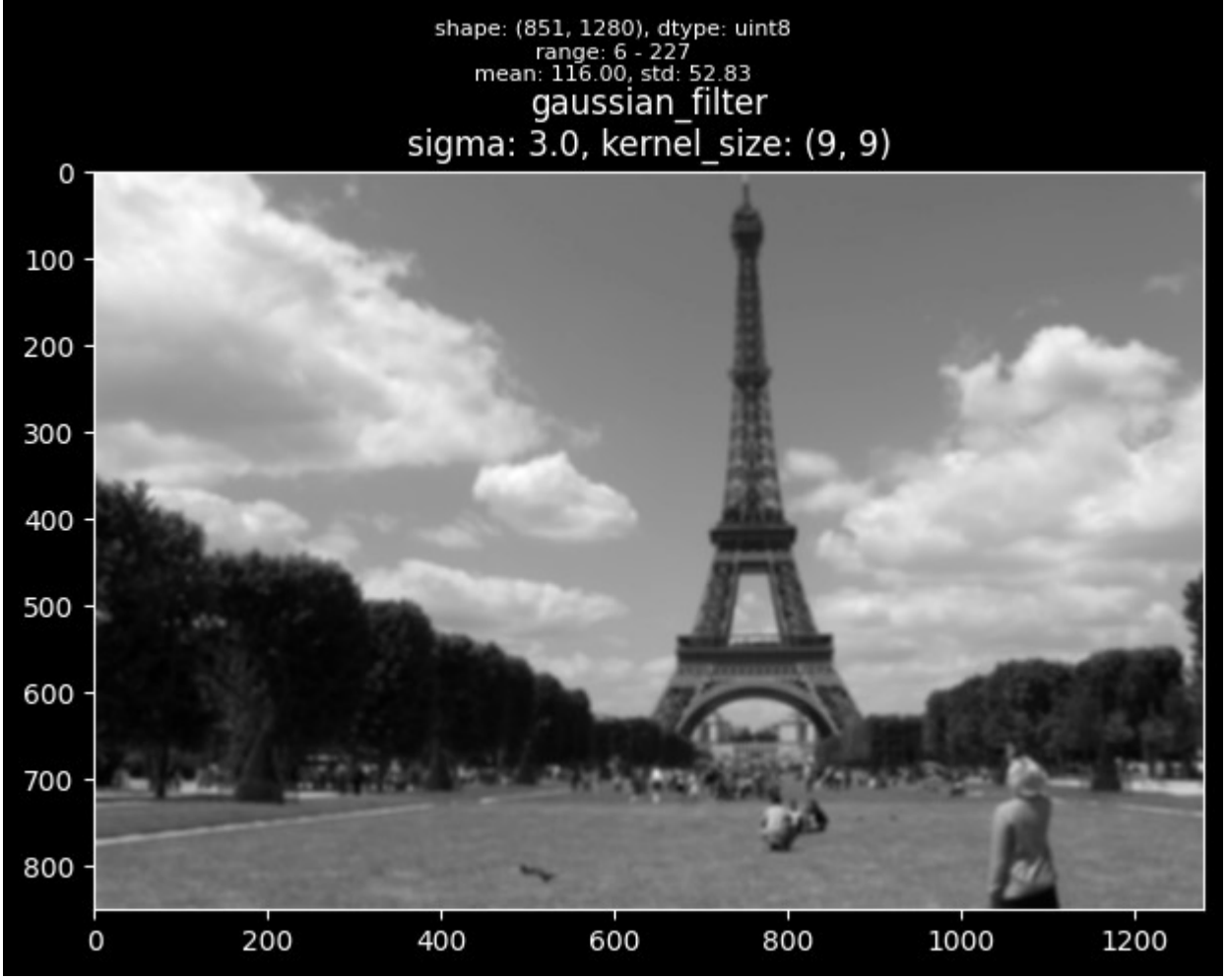
In [ ]:
```
def gaussian_filter(img, kernel_size=5, sigma=0):
    kernel_size = kernel_size if isinstance(kernel_size, tuple) else (kernel_size, kernel_size
    kernel_size = (kernel_size[0] if kernel_size[0]%2 != 0 else kernel_size[0]+1,
                   kernel_size[1] if kernel_size[1]%2 != 0 else kernel_size[1]+1)
    print()
    print("sigma:", sigma)
    # apply gaussian filter
    # https://docs.opencv.org/4.5.2/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb
    img_blur = cv2.GaussianBlur(img, ksize=kernel_size, sigmaX=sigma)
    #show_img(img_blur, title=f"gaussian_filter\nsigma: {sigma}, kernel_size: {kernel_size}",
    return img_blur

gauss_imgs = []
sigmas = [i * 1.5 for i in range(1,8)]
for sigma in sigmas:
    img_blur = gaussian_filter(img=img, kernel_size=9, sigma=sigma)
    gauss_imgs.append(img_blur)
```

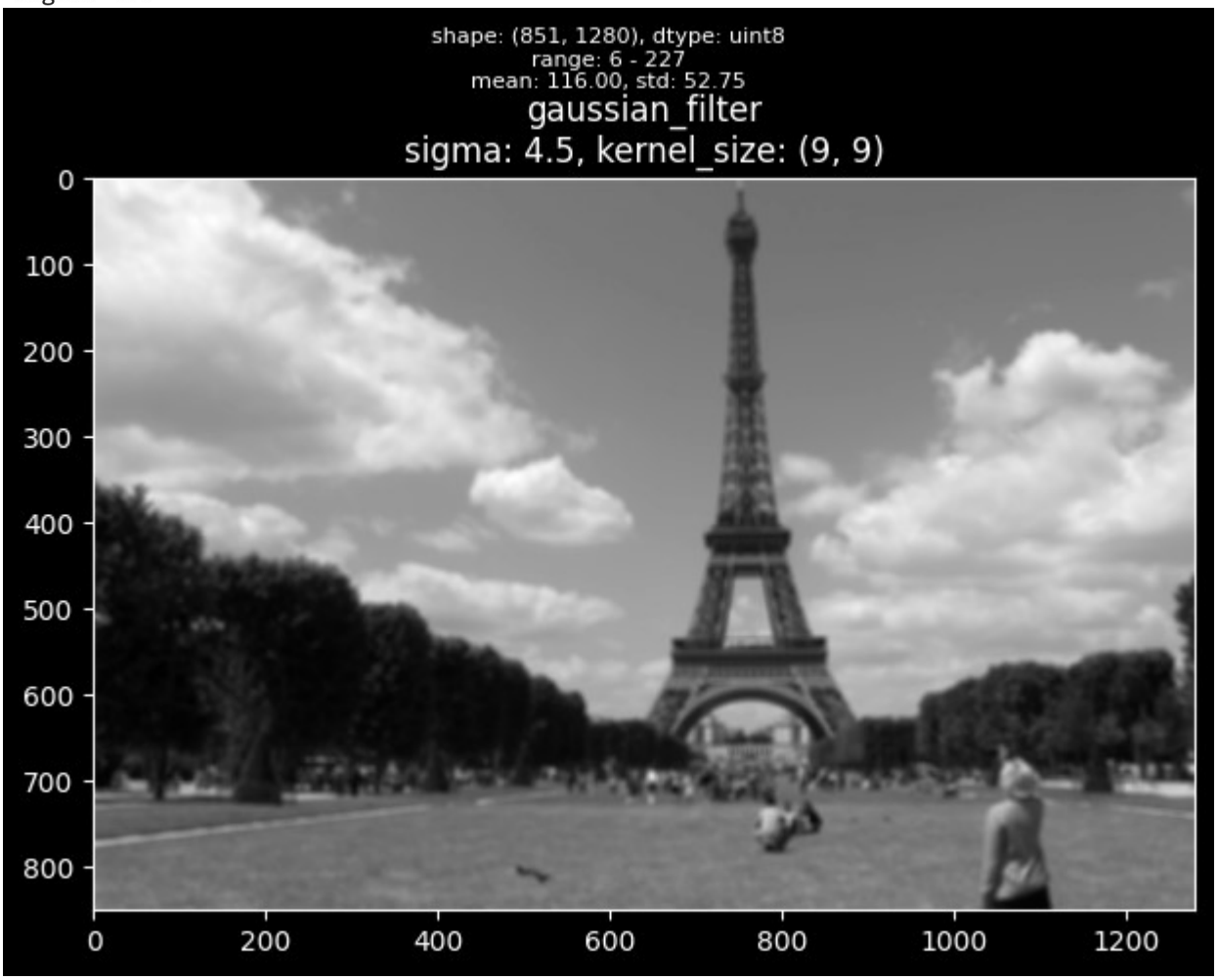
```
sigma: 1.5
```



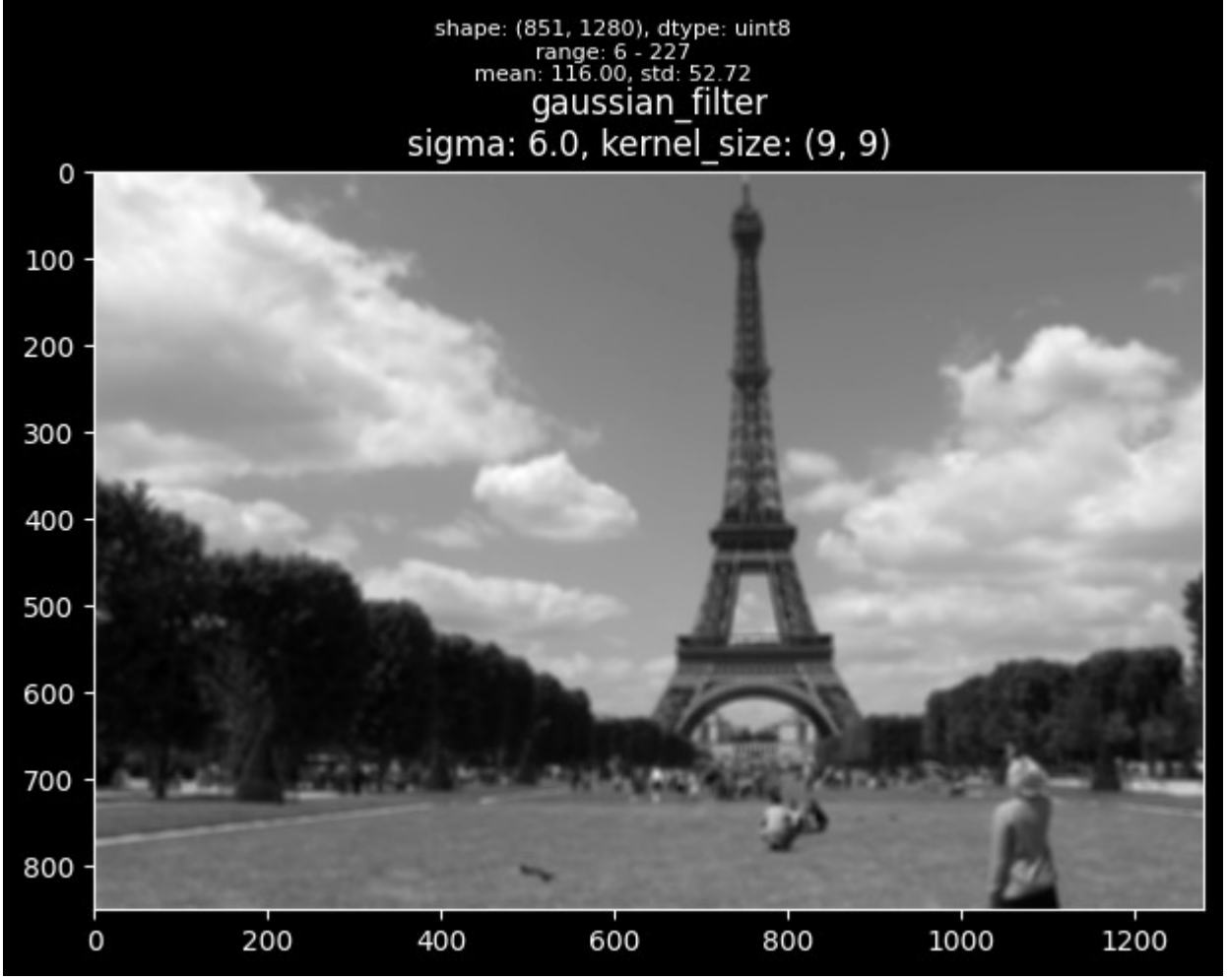

```
sigma: 3.0
```



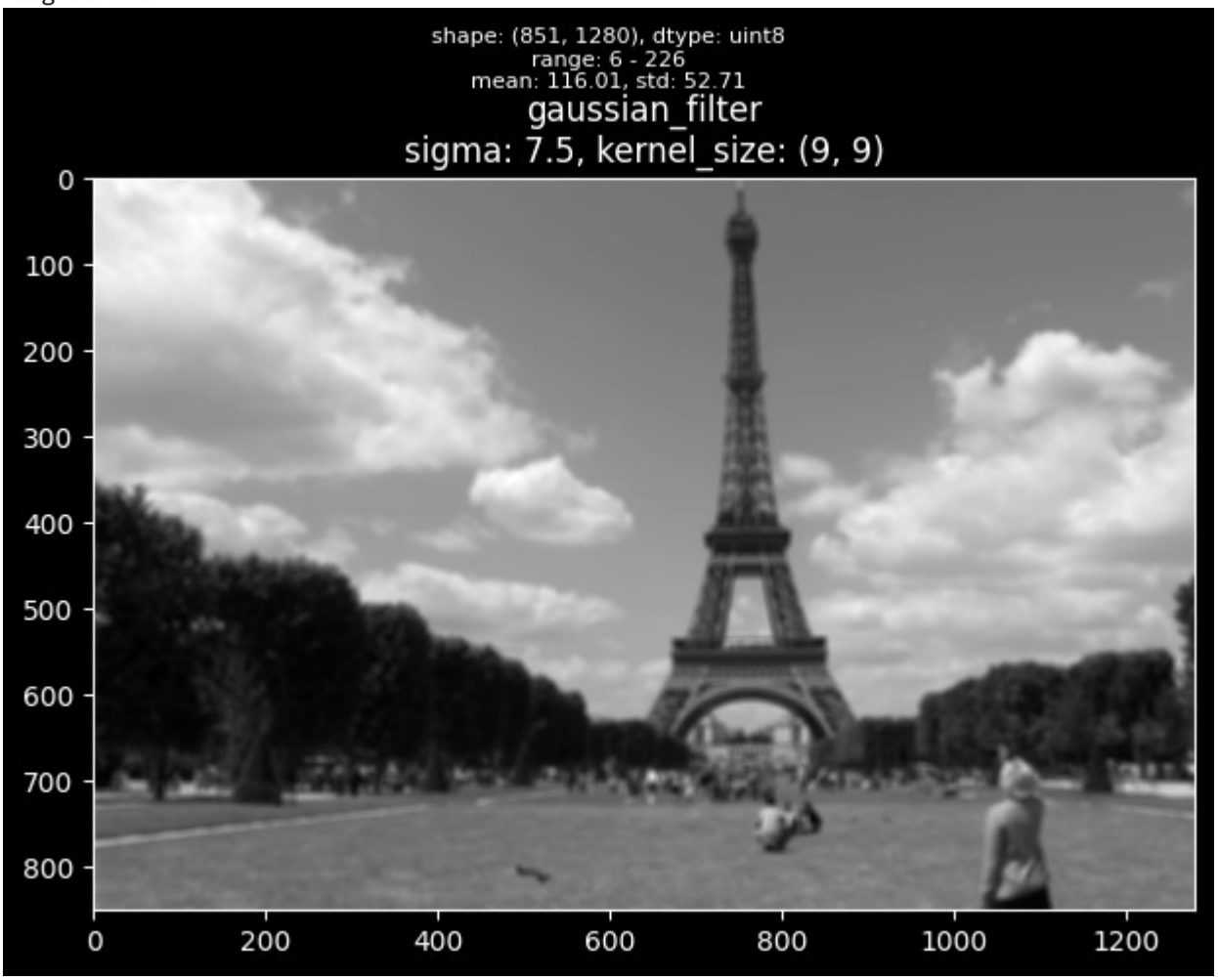
sigma: 4.5



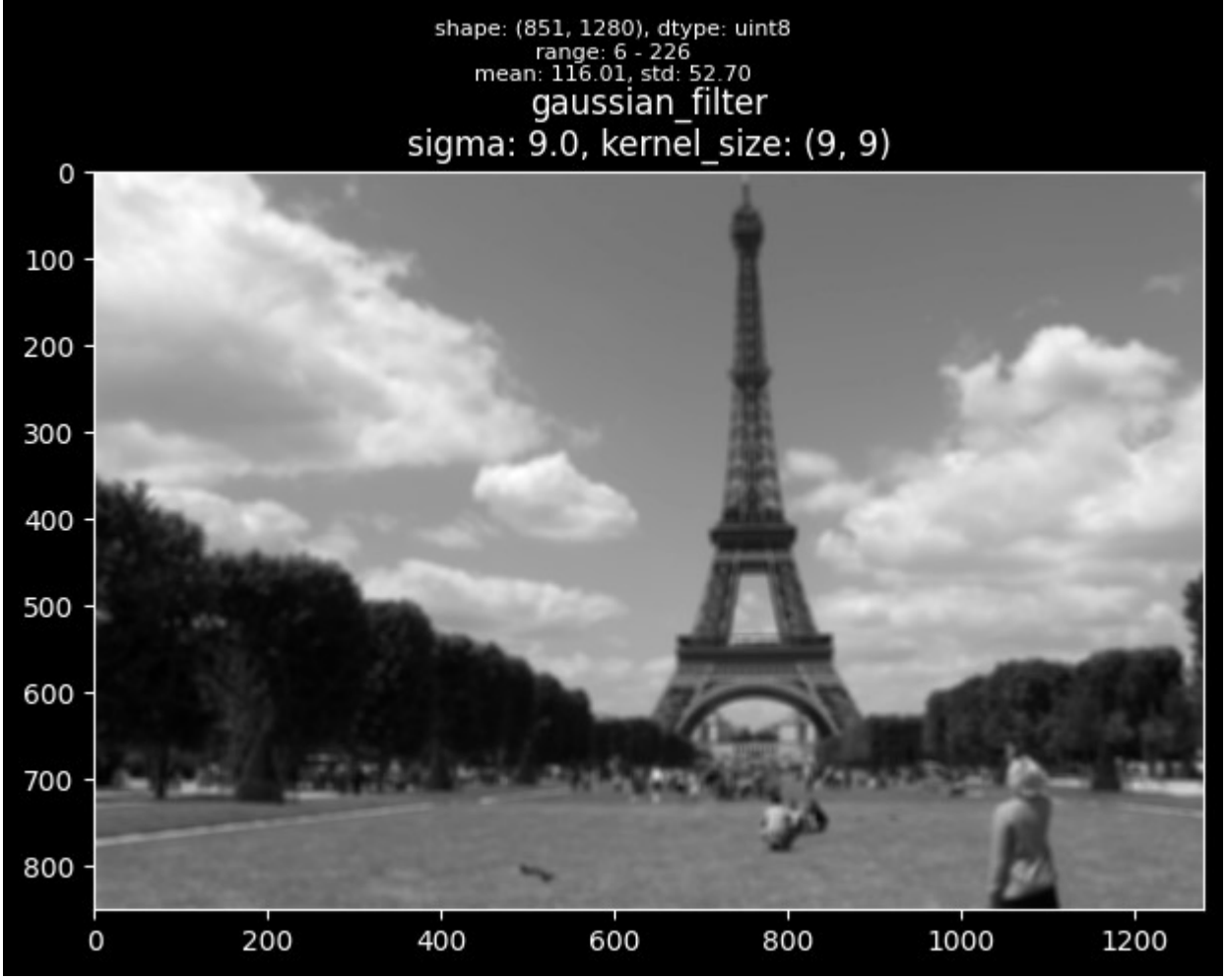
sigma: 6.0



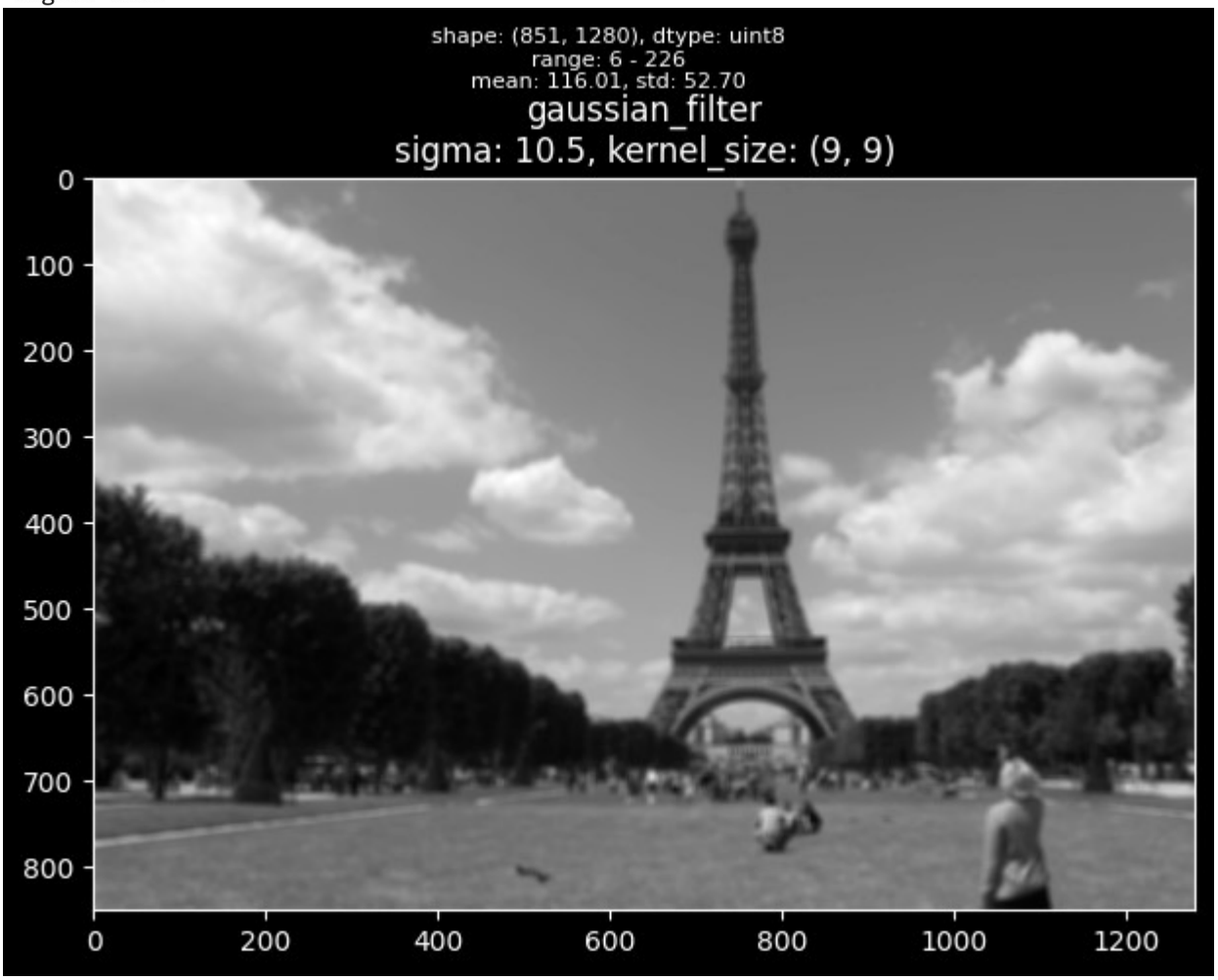
sigma: 7.5



sigma: 9.0


shape: (851, 1280), dtype: uint8
range: 6 - 226
mean: 116.01, std: 52.70
gaussian_filter
sigma: 9.0, kernel_size: (9, 9)
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

```
sigma: 10.5
```


shape: (851, 1280), dtype: uint8
range: 6 - 226
mean: 116.01, std: 52.70
gaussian_filter
sigma: 10.5, kernel_size: (9, 9)
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

In [ ]:

```
import numpy as np
import scipy
import scipy.ndimage as ndimage
import matplotlib.pyplot as plt

neighborhood_size = 26
threshold = 0.5 * img.max()

data_max = ndimage.maximum_filter(img, neighborhood_size)
maxima = (img == data_max)
data_min = ndimage.minimum_filter(img, neighborhood_size)
diff = ((data_max - data_min) > threshold)
maxima[diff == 0] = 0

labeled, num_objects = ndimage.label(maxima)
xy = np.array(ndimage.center_of_mass(img, labeled, range(1, num_objects+1)))

plt.imshow(img, cmap="gray")

plt.autoscale(False)
plt.plot(xy[:, 1], xy[:, 0], 'ro', alpha=0.1)

plt.show()
```