

1 Intensity-level (monkey) entropy

```
1 from PIL import Image, ImageFilter, ImageOps
2 import numpy as np
3 import numpy.linalg as linalg
4 import matplotlib.pyplot as plt
```

Given a discrete random variable X on a probability space (Ω, \mathcal{F}, P) with image $\chi = \text{im}(X)$, the *Shannon entropy* is

$$H = \sum_{x \in \chi} -P(x) \ln P(x).$$

The *intensity-level entropy* is the Shannon entropy of the empirical distribution of intensity values.

```
1 def shannon_entropy(h):
2     """The Shannon entropy in bits"""
3     return -sum(p*np.log2(p) if p > 0 else 0 for p in h)
4
5 def intensity_entropy(data):
6     hist, _ = np.histogram(data, bins=range(256+1), density=True)
7     return shannon_entropy(hist)
8
9 def uniform(n):
10    return np.array([1] * n) / n
11
12 def rescale(data):
13    b = np.max(data)
14    a = np.min(data)
15    return (b - data) / (b - a)
```

1.1 Natural image

```
1 img = ImageOps.grayscale(Image.open('test.jpg'))
2 scale = max(np.shape(img))
3 data = np.array(img)
4 img
```



```
1 intensity_entropy(img)
```

7.51132356216608

The problem with the intensity entropy is that it is usually near maximum (8 bits for these grayscale images).

```
1 def intensity_blur(img, scales, display=True):
2     scale = max(np.shape(img))
3
4     results = []
5     for k in scales:
6         simg = img.filter(ImageFilter.GaussianBlur(k * scale))
7         data = np.array(simg)
8         ihist, ibins = np.histogram(data, bins=range(256+1), density=True)
9         S = shannon_entropy(ihist)
10        if display:
11            hist = plt.hist(ibins[:-1], ibins, weights=ihist, alpha=0.5)
12            results.append((k, simg, hist, S))
13        else:
14            results.append((k, S))
15
```

```

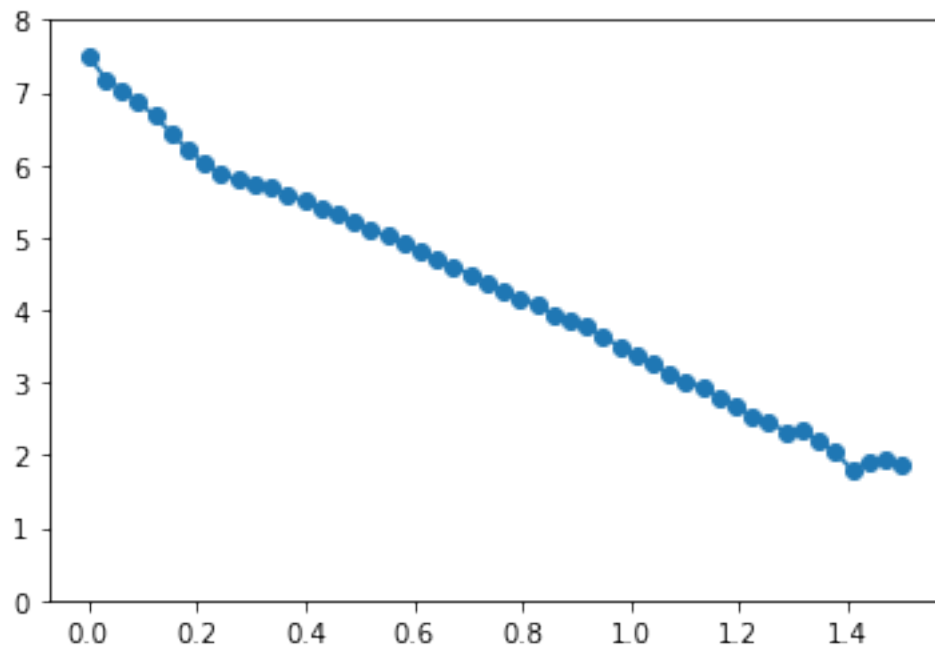
16     if display:
17         plt.axvline(x=np.mean(np.array(img)))
18
19     return results

```

```

1 results = intensity_blur(img, np.linspace(0, 1.5, num=50), False)
2
3 plt.plot(*np.transpose(results), 'o-')
4 plt.ylim((0, 8))
5 plt.xlabel = "Smoothing"
6 plt.ylabel = "Intensity Entropy (bits)"

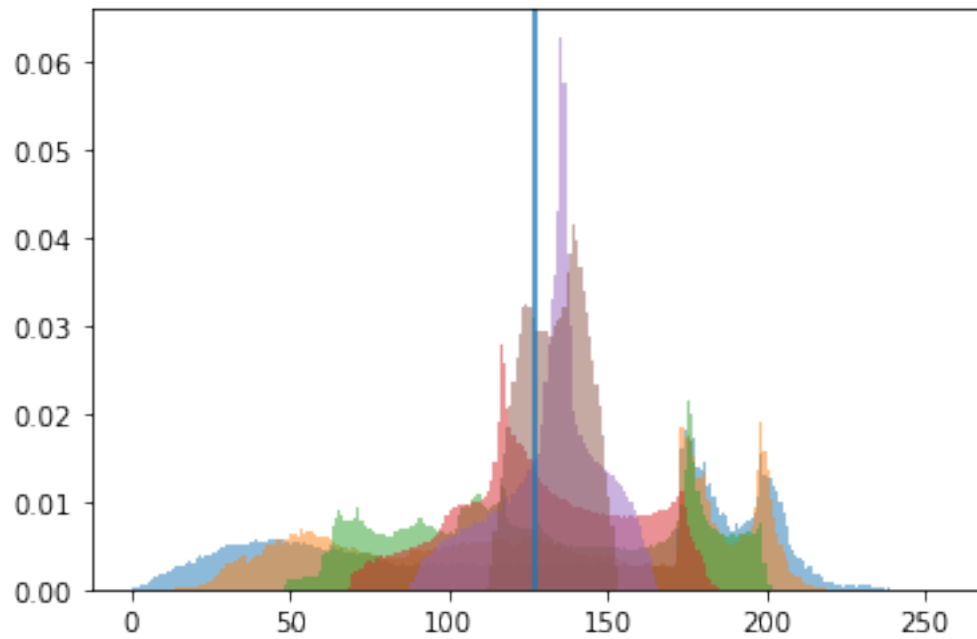
```



```

1 rings = [img for _, img, _ in intensity_blur(img, [0, 0.01, 0.05, 0.125, 0.25,
↪ 0.5])]
2 plt.show()

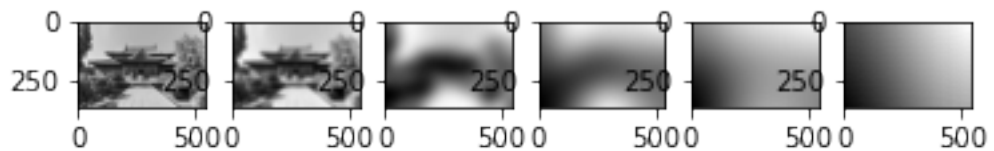
```



```

1 _, axarr = plt.subplots(1, len(rings))
2 for i, subimg in enumerate(rings):
3     axarr[i].imshow(subimg, cmap='gray')
4 plt.show()

```

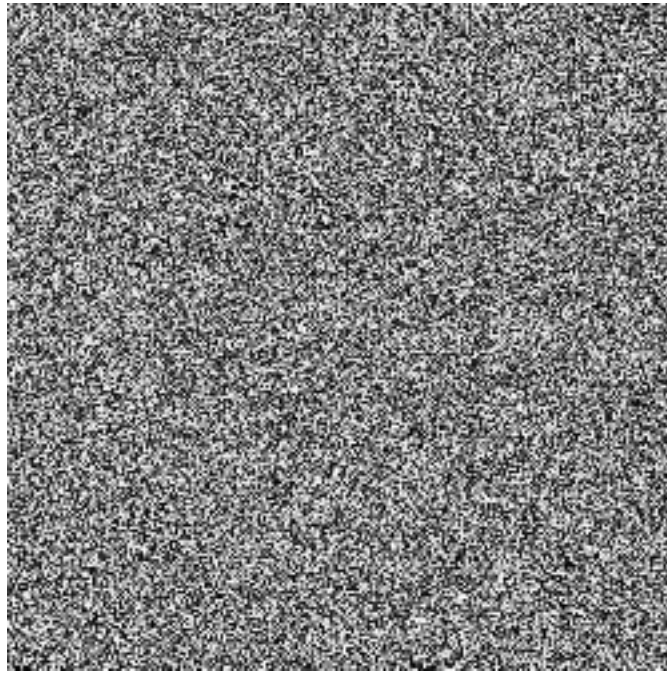


1.2 Random pixel values

```

1 rsize = 250
2 randimg = Image.fromarray((256*np.random.rand(*2*[rsize])).astype('uint8'))
3 randimg

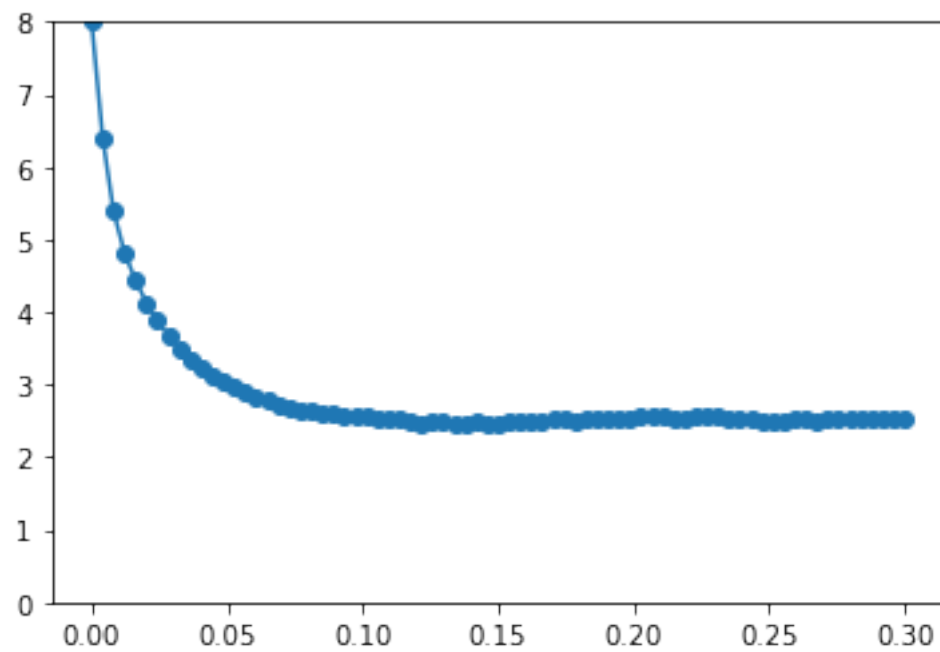
```



1.2.1 Beware: GIGO

The boundary effects and discrete kernel of `ImageFilter.GaussianBlur` renders the data unreliable after the “minimum” of the intensity entropy with smoothing. This is immediately clear after even small smoothing for random pixel values, since there are no spatial correlations.

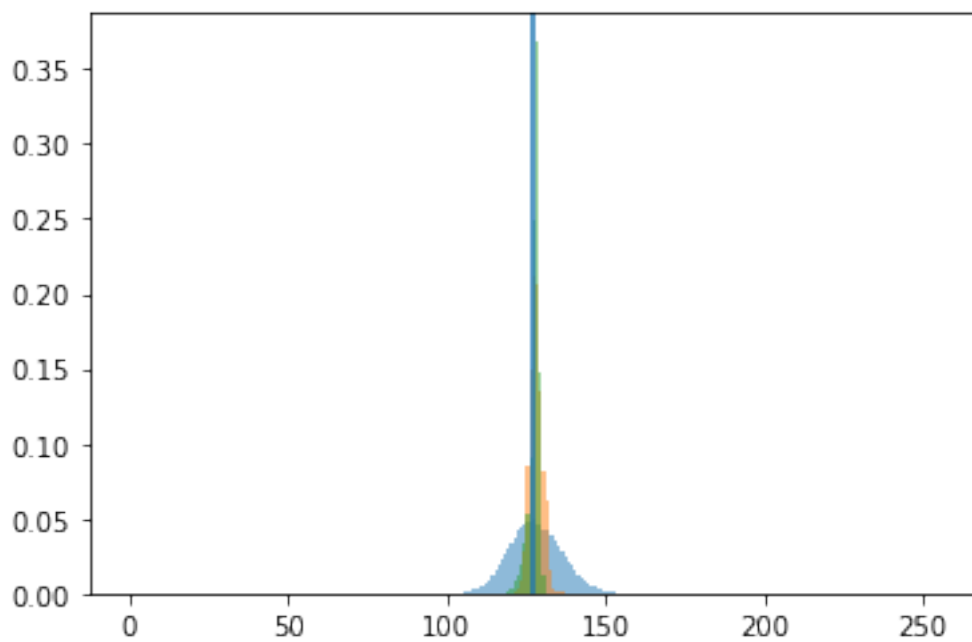
```
1 results = intensity_blur(randimg, np.linspace(0, 0.3, num=75), False)
2
3 plt.plot(*np.transpose(results), 'o-')
4 plt.ylim((0, 8))
5 plt.xlabel = "Smoothing"
6 plt.ylabel = "Intensity Entropy (bits)"
```



```

1 rings = [img for _, img, _, _ in intensity_blur(randing, [0.01, 0.05, 0.25])]
1 plt.show()

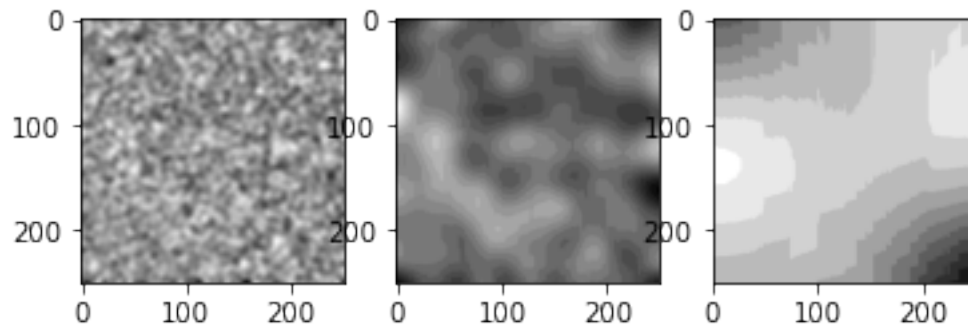
```



```

1 _, axarr = plt.subplots(1, len(rings))
2 for i, subimg in enumerate(rings):
3     axarr[i].imshow(subimg, cmap='gray')
4 plt.show()

```



The rightmost image should be uniform: the renormalization emphasizes incorrect deviations. These are what keep the intensity entropy from vanishing.

1.3 Comparing different levels of smoothing

Is composing n Gaussian blurs with variance σ^2 the same as doing one with variance $n\sigma^2$ (considering the boundary effects and discrete kernel)?

```

1 nsmooths = 10
2 cimg = img
3 oneimg = cimg.filter(ImageFilter.GaussianBlur(np.sqrt(nsmooths)*2))
4 oneimg

```



```
1 nimg = cimg
2 for _ in range(nsmooths):
3     nimg = nimg.filter(ImageFilter.GaussianBlur(2))
4 nimg
```




Answer: **No**

The differences between results at different scales can be pretty wack.

```
1 Image.fromarray((255*rescale(np.array(nimg) - np.array(oneimg))).astype('uint8'))
```



```
1 smimg = img
2 smdiff = np.array(sming.filter(ImageFilter.GaussianBlur(2))) -
   ↪ np.array(sming.filter(ImageFilter.GaussianBlur(100)))
3 diffimg = Image.fromarray((255 * rescale(smdiff)).astype('uint8'))
4 diffimg
```

