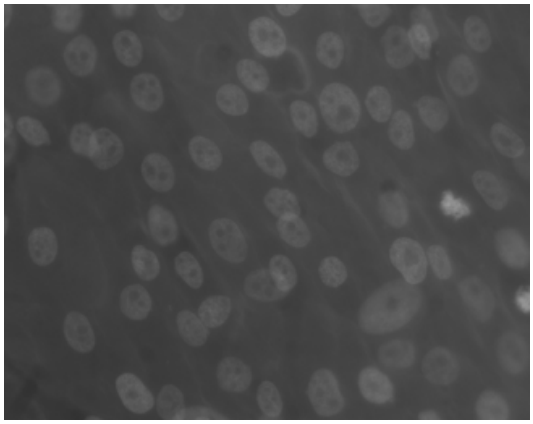
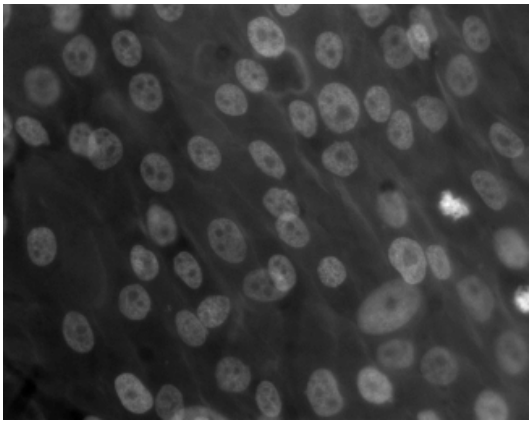
1 Python correction

Figure 1: Gamma transforms.

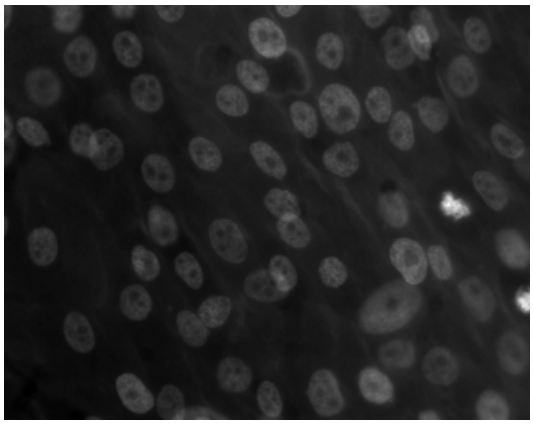
(a) Original image.



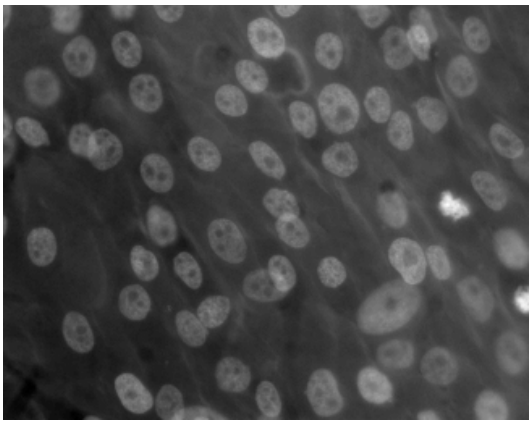
(b) $\gamma = 1$.



(c) $\gamma = 2$.



(d) $\gamma = 0.5$.



```python
from scipy import misc
import matplotlib.pyplot as plt
from skimage import exposure
import numpy as np
import sys
```

1.1 Intensity transformations

1.1.1 γ correction

The function skimage.exposure.adjust_gamma is used in order to adjust the gamma of the image. It is before normalized (values are float between 0 and 1). The results are illustrated in Fig.1.

```
I=imageio.imread("osteoblaste.png")
I = I / np.max(I);

gamma=2;
I2= exposure.adjust_gamma(I, gamma);
plt.imshow(I2);
plt.show();
```

1.1.2 Contrast stretching

The LUT look-up-table is simply a function applied to the original image. In order to avoid division by zero, the smallest float value is introduced. Fig.2 illustrates the results.

```
def contrast_stretching(I, E):
    epsilon = sys.float_info.epsilon;
    m = np.mean(I);
    I = I.astype("float");
    Ar = 1. / (1.+(m/(I+epsilon))**E);
    return Ar;
```

1.2 Histogram equalization

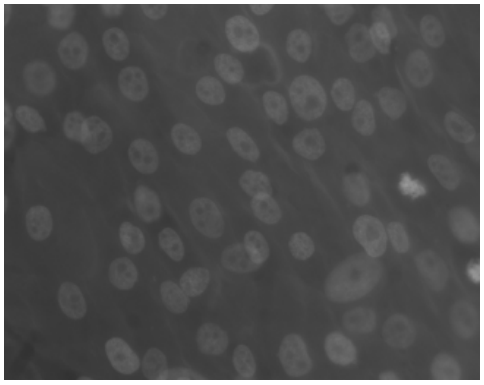
The histogram is displayed with the following function:

```
def displaySaveHisto(I, filename=None):
    """
    Display and save pdf (if filename provided) of histogram of image I
    If values are between 0 and 1, they are multiplied by 255
    """
    if np.max(I)<=1:
        I = 255 * I;
    hist,bins = np.histogram(I.flatten(), 256, range=(0,255))
    fig = plt.figure()
    plt.bar(bins[:-1], hist, width=1);
    plt.show();
    if filename!=None:
        fig.savefig(filename, bbox_inches='tight')
```

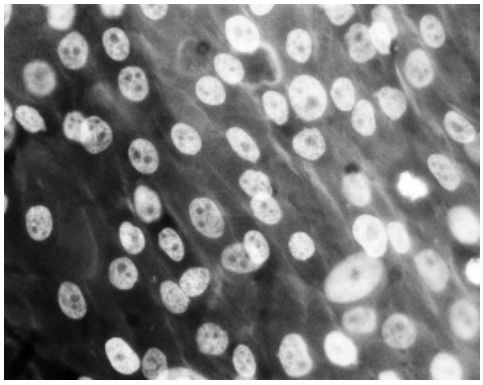
The histogram equalization is done via the skimage.exposure.equalize_hist function. The results are illustrated in Fig.3.

Figure 2: Constrast stretching.

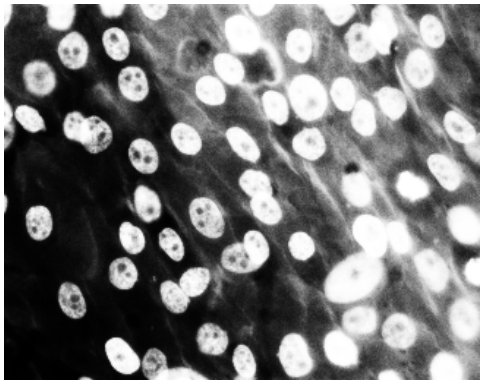
(a) Original image.



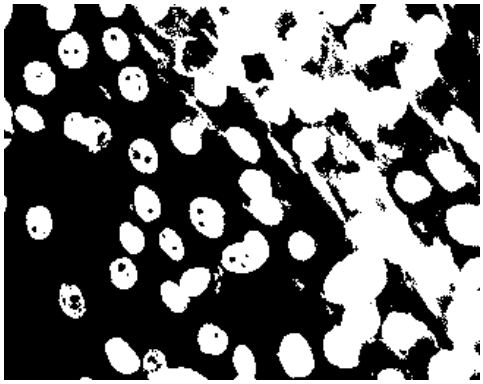
(b) $E = 10$.



(c) $E = 20$.



(d) $E = 1000$.



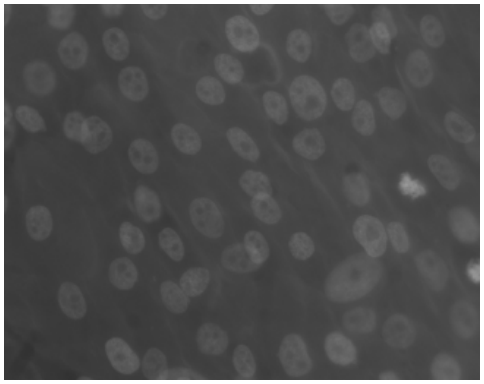
```
# histogram equalization
I2 = exposure.equalize_hist(I);
plt.imshow(I2);
plt.show()
```

The look-up-table (LUT) or cumulative distribution function (cdf) is computed and used in the next function:

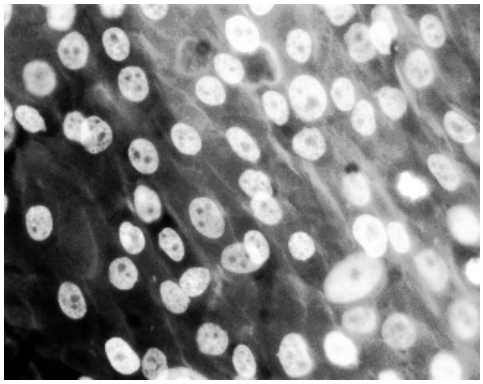
```
def histeq(I):
    """
    histogram equalization, version with look-up-table
    I: original image, with values in 8 bits integer
    """
    hist,bins=np.histogram(I.flatten(), 256, range=(0,255))
    cdf = hist.cumsum();
    cdf = (cdf / cdf[-1]);
    return cdf[I];
```

Figure 3: Histogram equalization.

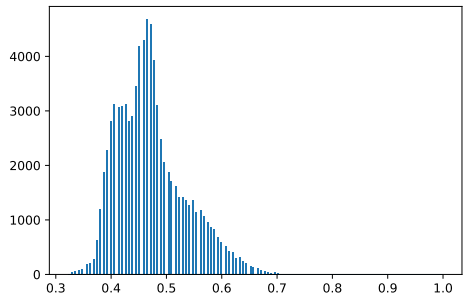
(a) Original image.



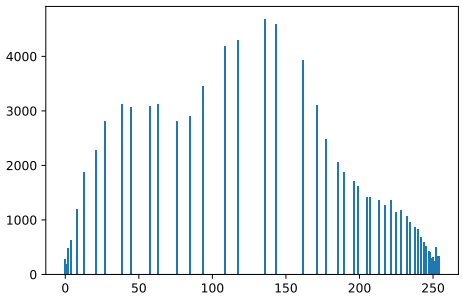
(b) Histogram equalization.



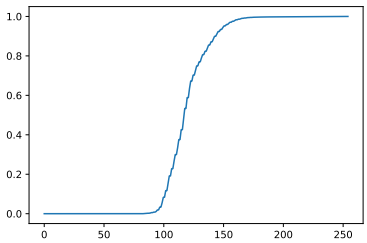
(c) Histogram of original image.



(d) Histogram after equalization.



(e) LUT (cumulative distribution function).



1.3 Histogram matching

This method is only an approximate method. It requires an interpolation of the cumulative distribution functions in order to find the transformation. Results are shown in Fig.4

```python
def hist_matching(I, cdf_dest):
    """
    Histogram matching of image I, with cumulative histogram cdf_dest
    This should be normalized, between 0 and 1.
    This version uses interpolation
    """
    imhist,bins = np.histogram(I.flatten(), len(cdf_dest), density=True)
    cdf = imhist.cumsum() #cumulative distribution function
    cdf = (cdf / cdf[-1]) #normalize between 0 and 1
    # first: histogram equalization
    im2 = np.interp(I.flatten(), bins[:-1], cdf)
    # 2nd: reverse function
    im3 = np.interp(im2, cdf_dest, bins[:-1])
    # reshape into image
    imres = im3.reshape(I.shape)
    return imres;
```

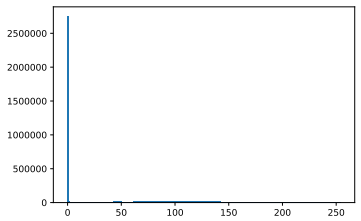
Figure 4: Histogram matching.

(a) Original image.

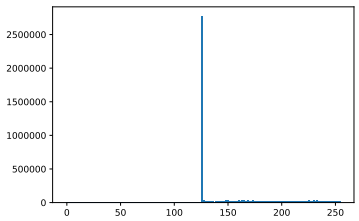
(b) Histogram equalization.

(c) Histogram matching.

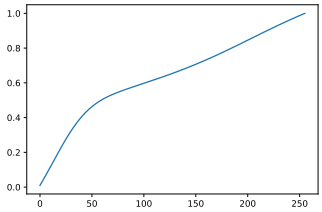
(d) Histogram of original image.



(e) Histogram after equalization.



(f) Target histogram (cumulative distribution function).



(g) Histogram after histogram matching.