Gray-scale images

For this topic, we will assume that all images are gray-scale. This means that they have only one real number which signifies their color, rather than a vector of three numbers (RGB). For practical purposes we still store those images as having three channels, but they all have the same value. When transforming a color image to a gray-scale image, we can use the average value of the three channels, or a weighted average, such as the luminosity method, which uses the weighting $0.21R + 0.72G + 0.07B$.

Our methods and code is easiest to understand when we treat images as 2D arrays that contain single real numbers (between 0 and 1) for each pixel, so most of the time, we first convert the 3D ($n \times m \times 3$) object into a 2D ($n \times m$) object, and then back again at the end of the function. (No information is lost if we assume all three color channels are equal in every pixel.)

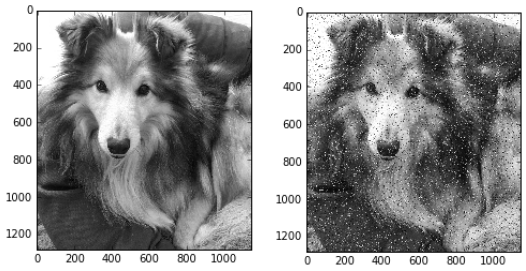
Adding Noise to an image

Because we will write a denoising function, it is good to start with writing a function that adds noise to an image. This way, we can keep track of the "ground truth" of the image, and control the type and amount of noise that we add. We look at two different types of noise: salt & pepper noise, and Gaussian noise.

Salt & Pepper Noise

Salt and pepper noise is a type of noise that replaces pixels with white or black pixels, independently of other pixels and independently of their original color. It is a type of noise that results from pieces of the signal being lost entirely, for example during transmission or conversion. We let our function replace each pixel by a white pixel with probability p_s, and by a black pixel with probability p_p, and keep its color with probability $1 - p_s - p_b$.

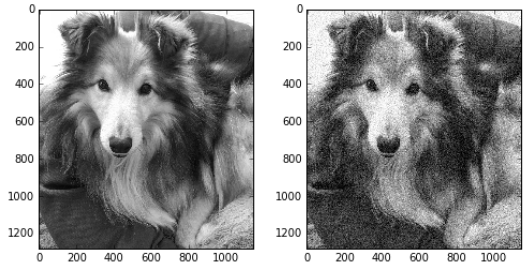
```
def salt_pepper(im,ps=.1,pp=.1):
    im1=im[:,:,0].copy()
    n,m=im1.shape
    for i in range(n):
        for j in range(m):
            b=np.random.uniform()
            if b<ps:
                im1[i,j]=0
            elif b>1-pp:
                im1[i,j]=1
    noisy_im=[[[im1[i,j]]*3 for j in range(m)] for i in range(n)]
    return noisy_im
```



Gaussian Noise

Gaussian noise is additive noise with a Gaussian distribution, usually introduced by the image sensor or during transmission. With Gaussian noise we have a small deviation around the original signal, rather than a complete loss of the signal.

```
def gauss_noise(im,p=1/4):
    im1=im[:,:,0].copy()
    n,m=im1.shape
    noise=np.random.normal(0,np.var(im1)**(1/2)*p,[n,m])
    noisy_im1=im1+noise
    for i in range(n):
        for j in range(m):
            if noisy_im1[i,j]>1:
                noisy_im1[i,j]=1
            elif noisy_im1[i,j]<0:
                noisy_im1[i,j]=0
    noisy_im=[[[noisy_im1[i,j]]*3 for j in range(m)] for i in range(n)]
    return noisy_im
```



Note that we can generate the noise array in one line, and then simply add it to the image. This works for numpy arrays, and is one of the differences between numpy arrays and python lists. For example, see the difference between the following two commands:

```
In:   [1,2]+[3,4]
Out:  [1,2,3,4]
In:   np.array([1,2])+np.array([3,4])
Out:  array([4, 6])
```

Also, note the line `im1=im[:,:,0].copy()`. This ensures that when we alter the noisy image, we do not change the original input image. When we set two variables equal in python, they **stay** equal. For example:

```
In:   a=[1,2]
In:   b=a
In:   b.append(3)
In:   a
out:  [1,2,3]
```

```
In:   a=[1,2]
In:   b=a.copy()
In:   b.append(3)
In:   a
out:  [1,2]
```

Exercises

- Write a function that adds noise only to dark areas of an image.
- Write a function that adds periodic noise, for example in horizontal bands.