Assignment 1

Philip Lassen, Luise Costa

February 10, 2019

Below are the functions and libraries that we use throughout the assignment

```
In [8]: import matplotlib.pyplot as plt
        import skimage.io as ski
        from skimage.io import imread
        from pylab import ginput
        import numpy as np
        import numpy.random as npr
        from skimage import img_as_ubyte
        from skimage import img_as_uint
        from skimage.color import rgb2hsv
        from skimage.transform import resize
        from skimage.color import rgba2rgb
        import time
```

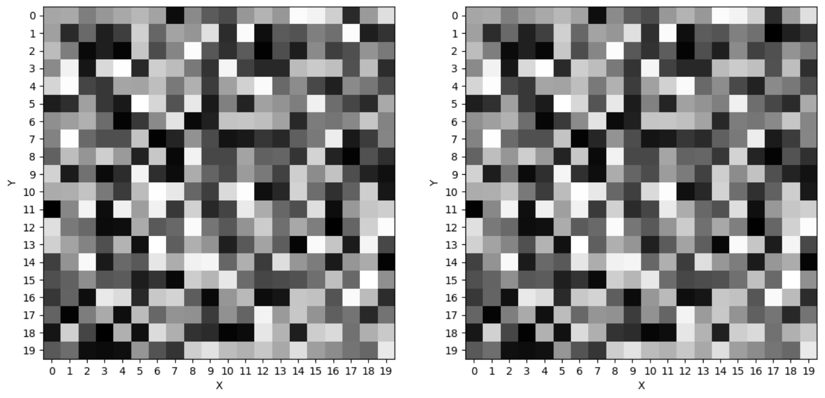
1 1.1

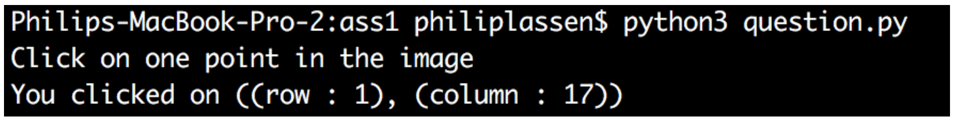
```
In [6]: def pixel():

            I = np.random.rand(20, 20)*255

            plt.figure()
            plt.subplot(1,2,1)
            plt.imshow(I, cmap = 'gray')
            plt.xlabel('X')
            plt.ylabel('Y')
            plt.xticks(np.arange(0, 20))
            plt.yticks(np.arange(0, 20))

            print('Click on one point in the image')
            coord = ginput(1)
            (row, column)  = (int(round(coord[0][1])), int(round(coord[0][0])))
            print('You clicked on ((row : %s), (column : %s))' % (row, column))
            I[row, column] = 0
            plt.subplot(1,2,2)
            plt.imshow(I, cmap = 'gray')
            plt.xlabel('X')
```





```
Philips-MacBook-Pro-2:ass1 philiplassen$ python3 question.py
Click on one point in the image
You clicked on ((row : 1), (column : 17))
```

Figure 1: (left) Random 20-by-20 pixel gray scale image (right) The colour of the selected pixel (r, c = 1, 17) was changed to black

```
    plt.ylabel('Y')
    plt.xticks(np.arange(0, 20))
    plt.yticks(np.arange(0, 20))
```

2 1.2

```
In [69]: def bit_slicing(path):
           plt.figure()
           plt.subplot(3,3,1)
           I = img_as_ubyte(imread(path, as_gray = True))
           plt.imshow(I, cmap = 'gray')
           plt.axis('off')

           for b in range(0, 8):
             plt.subplot(3,3,b+2)
             J = np.bitwise_and(I, 2**b)
             plt.imshow(J, cmap = 'gray')
             plt.axis('off')
         bit_slicing('Images/bigben.png')
```

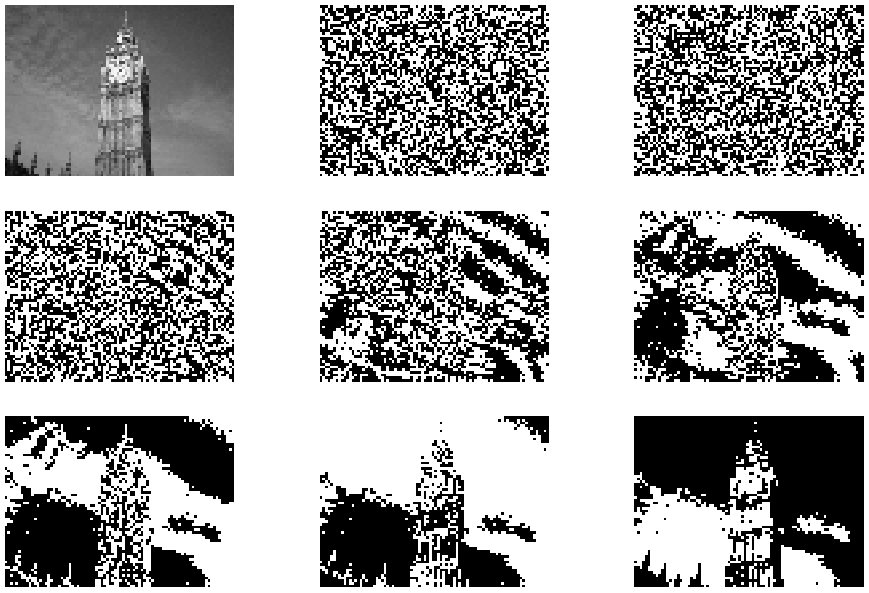


Figure 2: Bit-slicing. The first image shows the original black and white image. The following images represent the bit slices beginning with the less important slice.

3 1.3

```
In [70]: def HSV(path):

             plt.figure()
             I = imread(path)
             if len(I.shape) != 3:
               print('ERROR: image not in RGB')
               exit
             plt.subplot(2,2,1)
             plt.imshow(I)
             plt.axis('off')
             plt.title('original')
             I_hsv = rgb2hsv(I)
             plt.subplot(2,2,2)
             plt.imshow(I_hsv[:, :, 0], cmap = 'hsv')
             plt.axis('off')
             plt.title('hue channel')
             plt.subplot(2,2,3)
             plt.imshow(I_hsv[:, :, 1], cmap = 'gray')
             plt.axis('off')
             plt.title('saturation channel')
             plt.subplot(2,2,4)
             plt.imshow(I_hsv[:, :, 2], cmap = 'gray')
             plt.axis('off')
```

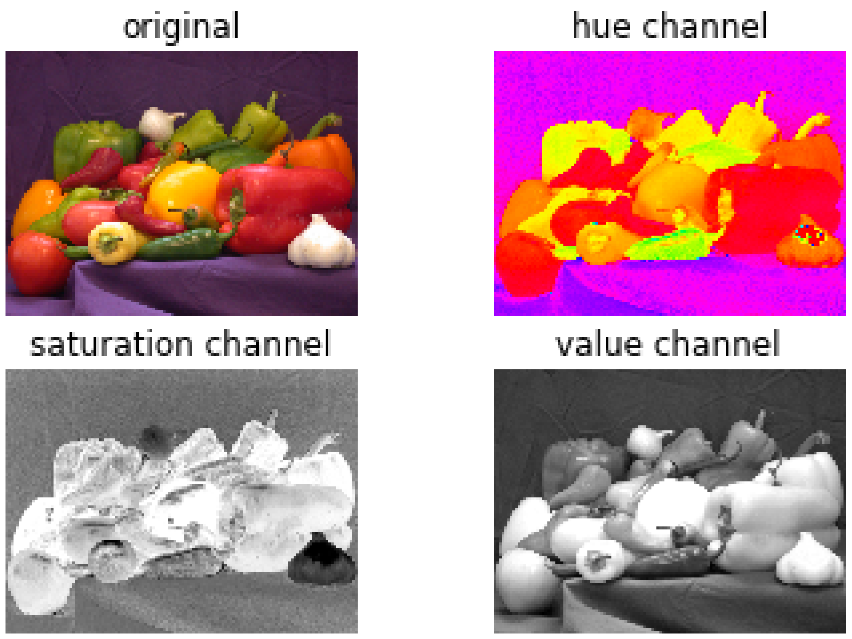


Figure 3: Perceptual color space (HSV): each image represents an other channel (hue, saturation and value)

```
    plt.title('value channel')
    plt.show()

HSV("Images/peppers.png")
```

4 1.4

```
In [27]: def blend(A, B, w_A, w_B):
           result = np.multiply(w_A, A) + np.multiply(w_B, B)
           return np.round(result).astype('int')

         image1 = imread("Images/toycars1.png")
         image2 = imread("Images/toycars2.png")
         w_b = image1.astype('float')
         oneMatrix = image1.astype('float')
         oneMatrix.fill(1.0)
         plt.figure(figsize = (15, 15))
         for i in range(5):
           plt.subplot(1, 5, i + 1)
           w_b.fill(i / 4.0)
           w_a = oneMatrix - w_b
```

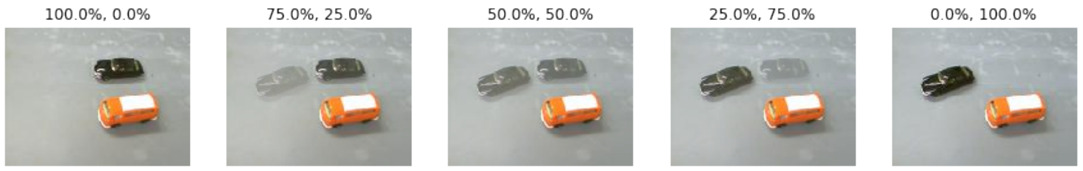


Figure 4: Changing Image Weights for Blending

```
    result = blend(image1, image2, w_a, w_b)
    plt.axis('off')
    plt.title("%s%%, %s%%" % ((1 - i / 4) * 100, i /4 * 100))
    plt.imshow(result)
```

Blending Images may be helpful in the case of noise due to oscillations in a camera. Thus it may be helpful to get the average of photos in order to get a more accurate single image representation. It also can be used for representing a moving object in a single image.

5 1.5

```
In [9]: def rgb2gray(img):
            img_gray = 0.2989 * img[:,:,0] + 0.5870 * img[:,:,1] + 0.1140 * img[:,:,2]
            return img_gray

        def plot_rgb_gray(path):

            I_rgb = imread(path)
            I_gray = rgb2gray(I_rgb)
            plt.subplot(1, 2, 1)
            plt.imshow(I_rgb)
            plt.axis('off')
            plt.title('orginal')

            plt.subplot(1, 2, 2)
            plt.imshow(I_gray, cmap = 'gray')
            plt.axis('off')
            plt.title('convert to gray')

        plot_rgb_gray('Images/sunandsea.jpg')
```

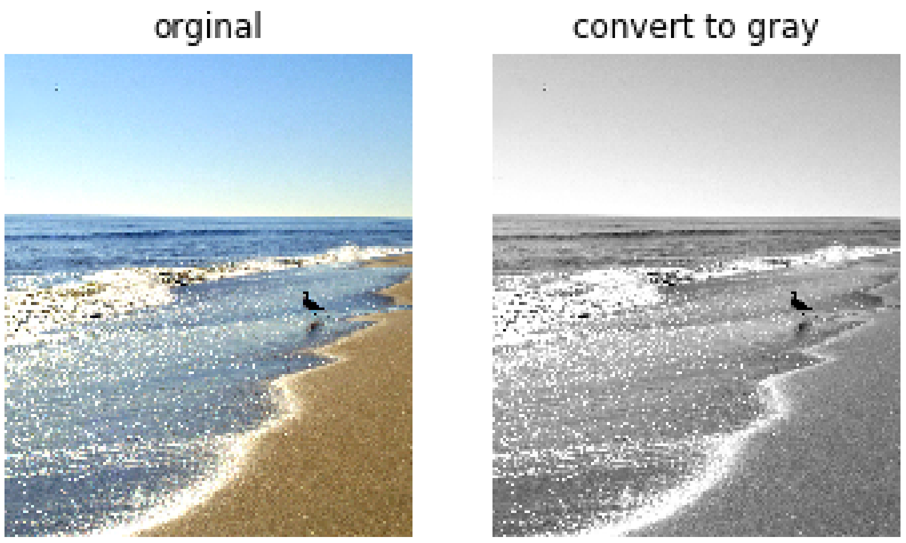


Figure 5: The original RGB image an the converted black and white one.

6 1.6

```
In [71]: def resize_exp(path):

             I = imread(path)
             I = rgba2rgb(I)
             height = I.shape[0]
             length = I.shape[1]
             plt.subplot(2, 2, 1)
             plt.imshow(I)
             plt.axis('off')
             plt.title('orginal: (%i, %i)' %(height, length))

             I1 = resize(I, (np.floor(0.5*height), np.floor(0.5*length), 3))
             plt.subplot(2, 2, 2)
             plt.imshow(I1)
             plt.axis('off')
             plt.title('resize: 0.5')
             I2 = resize(I, (np.floor(0.2*height), np.floor(0.2*length), 3))
             plt.subplot(2, 2, 3)
             plt.imshow(I2)
             plt.axis('off')
             plt.title('resize: 0.2')
             I3 = resize(I, (np.floor(0.1*height), np.floor(0.1*length), 3))
             plt.subplot(2, 2, 4)
             plt.imshow(I3)
             plt.axis('off')
             plt.title('resize: 0.1')
```

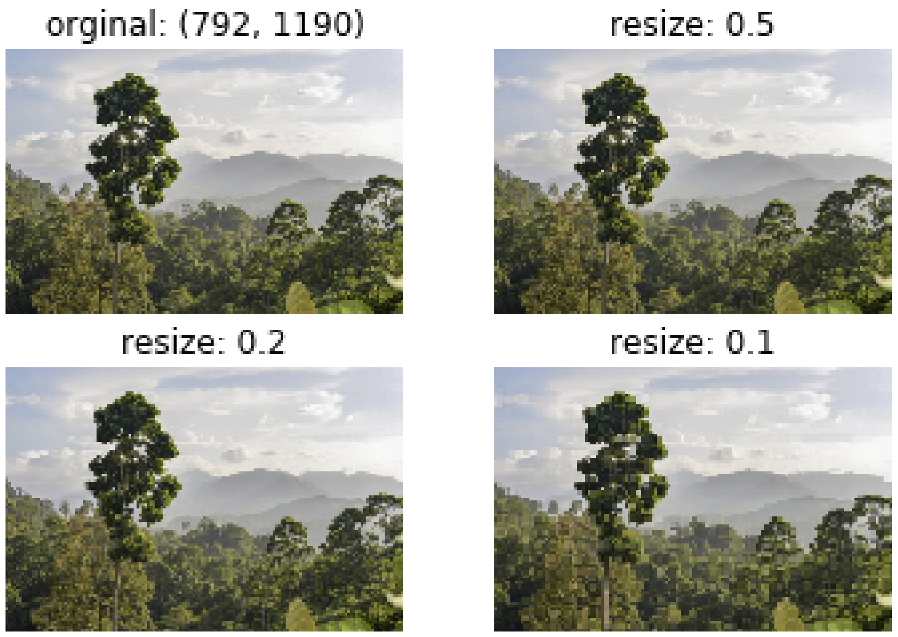


Figure 6: Downsized images

```
resize_exp('Images/tree.png')
```

We can see that the resolution of the image begins to deteriorate as we resize the image to smaller formats. In the image above we begin to lose the ability to distinguish between branches and leaves.

```
In [72]: def enlarge(img):

           height = img.shape[0]
           length = img.shape[1]
           factor = 5

           plt.subplot(2, 4, 1)#, sharex=True, sharey=True)
           plt.imshow(img)
           plt.axis('off')
           plt.title('orginal', fontsize = 12)

           location = [2, 3, 4, 6, 7, 8]
           for inter in range(6):
             time1 = time.time()*1000
             I1 = resize(img, (height*factor, length*factor, 3), order = inter)
             time2 = time.time()*1000
             diff = time2 - time1
```

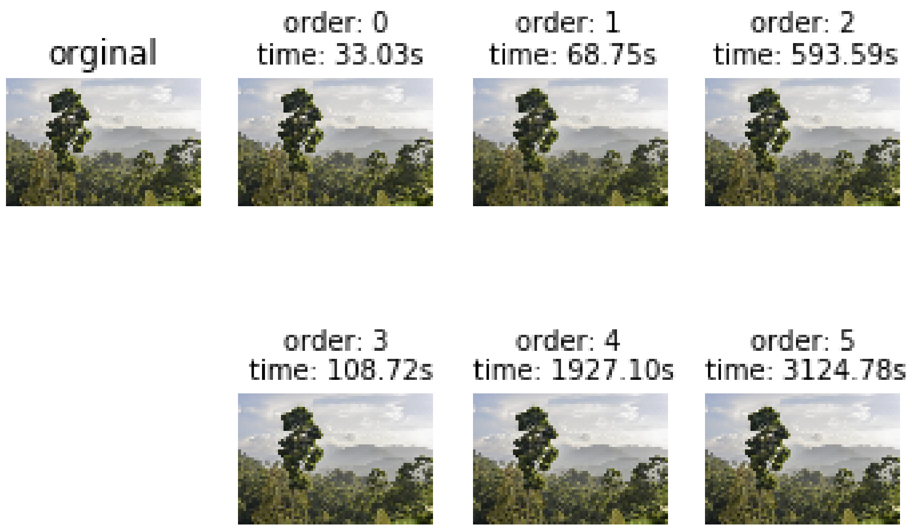


Figure 7: Upscaling the original image with different interpolation orders and the time needed for the operation.

```
            plt.subplot(2, 4, location[inter])
            plt.imshow(I1)
            plt.axis('off')
            plt.title('order: %i\n time: %.2fs' %(inter, diff), fontsize = 10)

        I = imread('Images/tree.png')
        I = rgba2rgb(I)
        img = resize(I, (np.floor(0.2*I.shape[0]), np.floor(0.2*I.shape[1]), 3))
        enlarge(img)
```

The order terms are different methods used to extrapolate the colors of the new added pixels in the resizing. The order terms correspond to:
0 = Nearest-neighbor, 1 = Bi-linear, 2 = Bi-quadratic, 3 = Bi-cubic, 4 = Bi-quartic, 5 = Bi-quintic.

The time complexity increases with the order term respectively

7 1.7

```
In [13]: path = 'Images/railway.png'
         I = imread(path)
         plt.imshow(I)
         #manually measured:
         ycoords_front = [37, 215]
         length_front = ycoords_front[1] - ycoords_front[0] +1
         ycoords_rear = [137, 140]
```

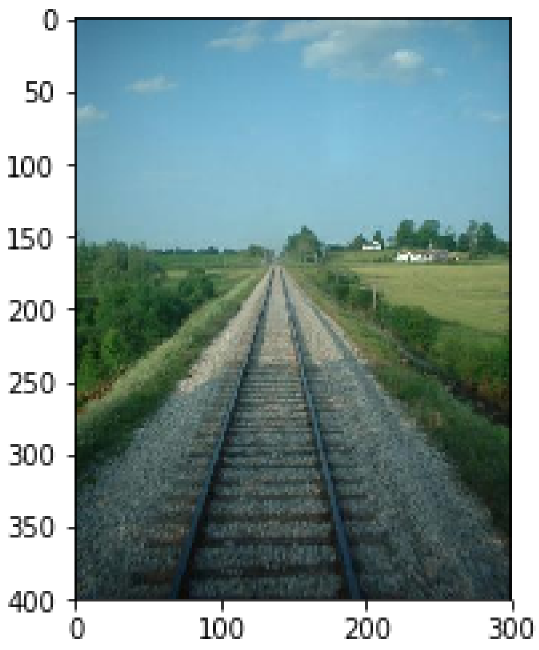


Figure 8: Example image of a railway, where the perspective projection is shown.

```
    length_rear = ycoords_rear[1] - ycoords_rear[0] +1
    print('length of the sleepers in pixel: %i frontground, %i background' %(length_front,
    #what can we determine about the relative distance to the camera of the two sleepers yo
    z_front = 1/length_front
    z_rear = 1/length_rear
    abs_dist = length_front/length_rear
    print('the slepper in the frontground is %.2f times closer to the camera than the sleep
    print('Q : If we assume the standard length of a railway sleeper to be 2.5 m, what addi
          'information would be needed to determine the absolute distance of these items fr
    print('A : Focal length')
    print('Q : How could this be applied to other areas, such as estimating the distance of
    print('A : distance = size_real/size_in_image * focal length')
```

```
length of the sleepers in pixel: 179 frontground, 4 background
the sleeper in the frontground is 44.75 times closer to the camera than the sleeper in the backg
Q : If we assume the standard length of a railway sleeper to be 2.5 m, what additional informati
A : Focal length
Q : How could this be applied to other areas, such as estimating the distance of people or cars
A : distance = size_real/size_in_image * focal length
```

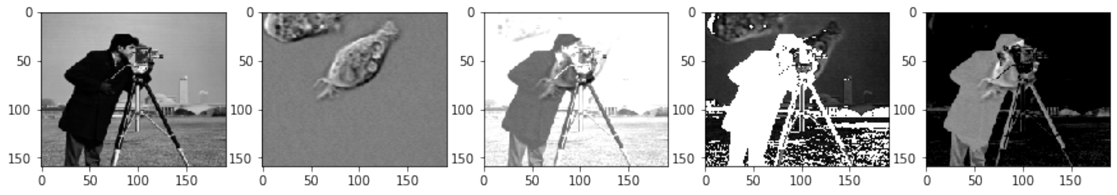


Figure 9: Cell and Cameraman sum and differences

8 1.8

```
In [29]: im1 = imread("Images/cameraman.tif")
         im2 = imread("Images/cell.tif")
         im1 = resize(im1, im2.shape)
         im1 = imread("Images/cameraman.tif")
         im2 = imread("Images/cell.tif")
         im1 = resize(im1, im2.shape)
         im1 = np.round(im1 * 255).astype(int)
         result_plus  = np.clip(im1.astype('uint16') + im2, 0, 255).astype('uint8')
         result_minus1 =  np.clip(im1.astype('uint16') - im2, 0, 255).astype('uint8')
         result_minus2 =  np.clip(im2.astype('uint16') - im1, 0, 255).astype('uint8')
         plt.figure(figsize = (15, 15))
         plt.subplot(1, 5, 1), plt.imshow(im1, cmap = "gray")
         plt.subplot(1, 5, 2), plt.imshow(im2, cmap = "gray")
         plt.subplot(1, 5, 3), plt.imshow(result_plus, cmap = "gray")
         plt.subplot(1, 5, 4), plt.imshow(result_minus1, cmap = "gray")
         plt.subplot(1, 5, 5), plt.imshow(result_minus2, cmap = "gray")
         plt.show()
```

The backgrounds of both photos typically become maximal intensity in the case of addition. And in the case of subtraction the backgrounds get minimal intensity. Because the cell image is mainly background we predominantly see the cameraman in the image and cam see a little outline of the cell.

9 1.9

```
In [30]: arr_path = ["Images/AT3_1m4_0" + str(i) + ".tif" for i in range(1, 10)]
         arr_path += ['Images/AT3_1m4_10.tif']
         def motion(arr_path):
           plt.figure()
           number = len(arr_path)
           for i in range(number-1):
             img1 = imread(arr_path[i])
```

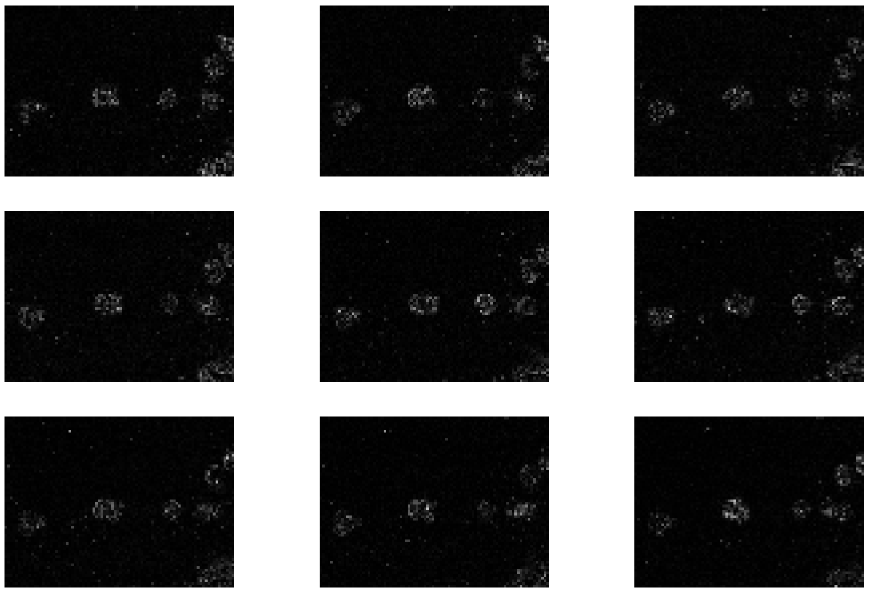


Figure 10: Differences between 10 cell images.

```
    img2 = imread(arr_path[i+1])
    img = img_as_uint(np.abs(img1.astype('int16') - img2))
    plt.subplot(3,3,i+1)
    plt.imshow(img, cmap="gray")
    plt.axis('off')
  plt.show()

motion(arr_path)
```

This could be extremely helpful if we are tracking moving objects against a still background. There are many techniques for doing this, but this simple technique of tracking the difference can actually perform very well in practice provided there isn't too much noise.