Matthew Mullin
Introduction to Computer Vision
Problem Set 2:

#1

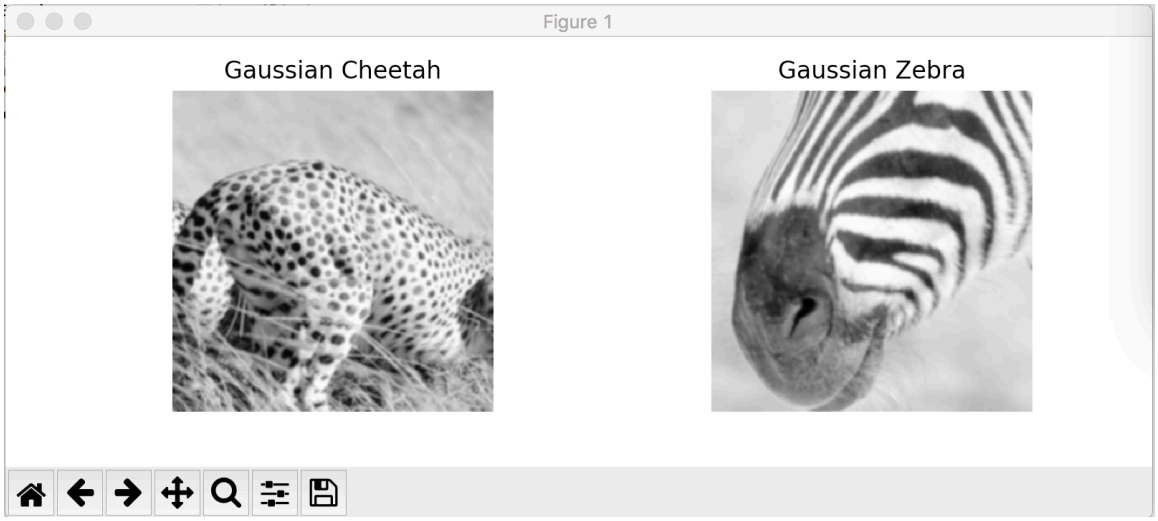
```
import numpy as np
from scipy import ndimage
from scipy import misc
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import skimage
```

After loading in the correct packages (shown above), the two images 'cheetah.png' and 'zebra.png' were read into python using cv2.imread. These would be the two images that would have a Gaussian blur applied to them in question #1.

```
cheetahblurred = cv2.GaussianBlur(cheetah, (7,7),0)
zebrablurred = cv2.GaussianBlur(zebra, (7,7),0)
```

A 7 x 7 kernel was chosen to be applied to the images to make them have a Gaussian blur. Commands from the matplotlib library were used to plot the two images in the same figure. Pictured below shows how the figure was set up with one row and two columns.

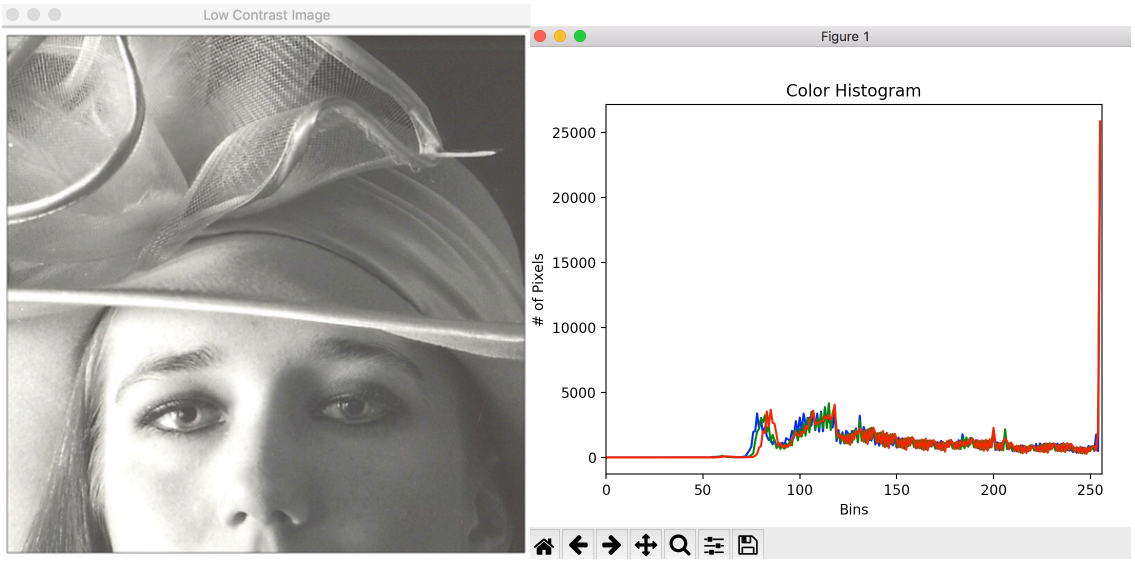
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 3),
                                    sharex=True, sharey=True)

ax1.imshow(cheetahblurred, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('Gaussian Cheetah')

ax2.imshow(zebrablurred, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title('Gaussian Zebra')
fig.tight_layout()

plt.show()
```



This is the result for applying a Gaussian Blur to the images.

#2
Problem #2 was challenging. The low contrast image was able to be split into it three color channels of Blue, Green & Red and plotted as a histogram but I was unable to fulfill the other requirements.

```
#Problem 2
#the low contrast images is read in
lowConImage = cv2.imread("lowcontrast.jpg")

#the image is shown and given a command to remain on the screen until another key is h
cv2.imshow("Low Contrast Image", lowConImage)
cv2.waitKey()
cv2.destroyAllWindows()

#cv.split breaks the image into its three color channels, which in this case are arrar
chans = cv2.split(lowConImage)

#The three channels are converted in the following order: blue, green, red
colors = ("b", "g", "r")
plt.figure()
plt.title("Color Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")

for (chan, color) in zip(chans, colors):
    hist = cv2.calcHist([chan], [0], None, [256], [0, 256])
    plt.plot(hist, color = color)
    plt.xlim([0, 256])

# Show our plots
plt.show()
```



#3

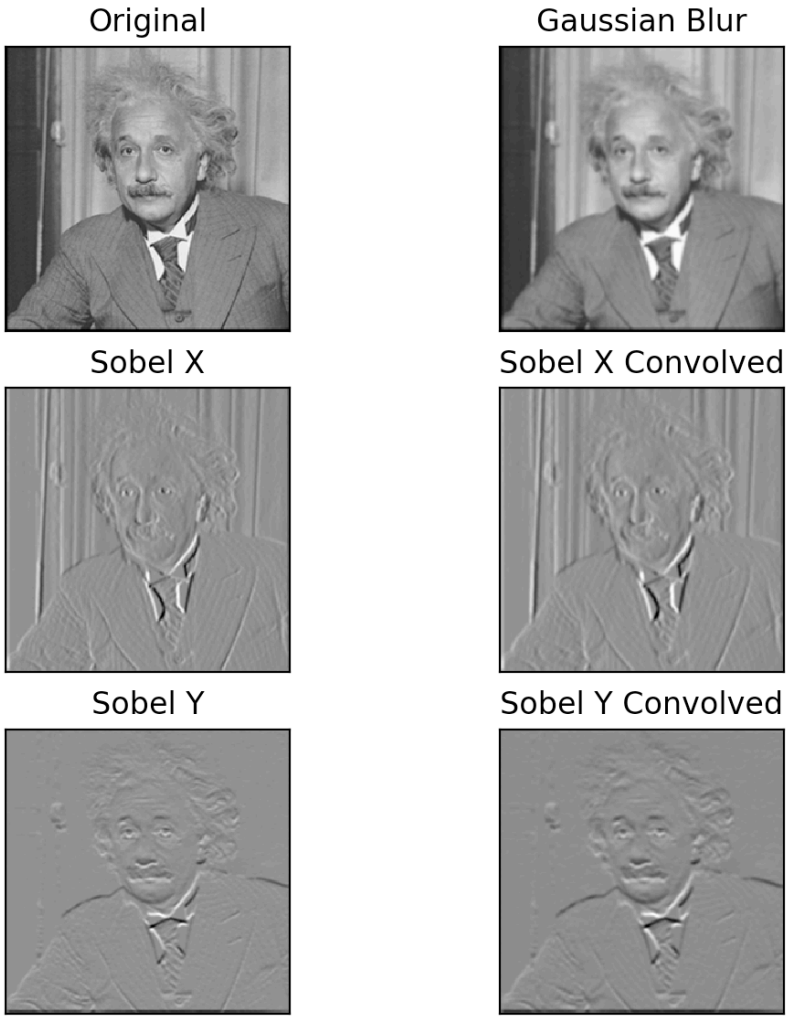
For problem #3 the link provided in the question sheet was used to create the convolved sobel filters and to find out what other variables were necessary such as “weight” which was represented by w as a 3x3 numpy array. The cv2 separable filters were found by looking through the documentation for openCV and plugging in the correct parameters. The Einstein picture was used and read into the python script as “smartman.”

```
w = np.array([[1,1,1],[1,2,1],[1,1,1]])

laplacian = cv2.Laplacian(smartman,cv2.CV_64F)
sobelx = cv2.Sobel(smartman,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(smartman,cv2.CV_64F,0,1,ksize=5)
gaussianBlur = cv2.GaussianBlur(smartman, (5, 5), 0)

sobelxd = ndimage.convolve(sobelx, w, mode='constant', cval=0.0)
sobelyd = ndimage.convolve(sobely, w, mode='constant', cval=0.0)
```

The images were then plotted using plt.subplot with 2 columns and 3 rows.

```
plt.subplot(3,2,1),plt.imshow(smartman,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(3,2,2),plt.imshow(gaussianBlur,cmap = 'gray')
plt.title('Gaussian Blur'), plt.xticks([]), plt.yticks([])
plt.subplot(3,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(3,2,4),plt.imshow(sobelxd,cmap = 'gray')
plt.title('Sobel X Convolved'), plt.xticks([]), plt.yticks([])
plt.subplot(3,2,5),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.subplot(3,2,6),plt.imshow(sobelyd,cmap = 'gray')
plt.title('Sobel Y Convolved'), plt.xticks([]), plt.yticks([])
plt.show()
```

The results for the plot:



#4
These were the additional packages that needed to be loaded in, in order to create the edge outlines.

```
#Problem number 4
from skimage import feature
from scipy import ndimage as ndi
```

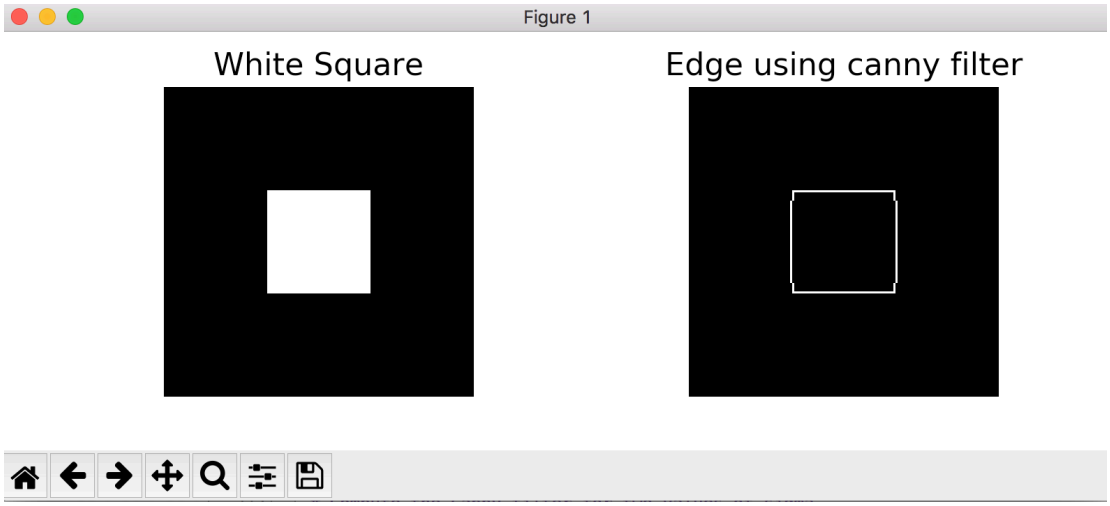
A square was made by creating a numpy zero array and then selecting a square region within the image to have a value of 1 (in terms of binary this would make the color white)

```
# Generating the square
square = np.zeros((150, 150))
square[50:-50, 50:-50] = 1
```

The edges were created by using a canny filter which was loaded in from the *skimage* library. The results were then plotted using the matplotlib method used for the other questions.

```
# Compute the Canny filter for two values of sigma
edges1 = feature.canny(square)
edges2 = feature.canny(square, sigma=3)

# display results
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 3),
                                    sharex=True, sharey=True)

ax1.imshow(square, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('White Square', fontsize=16)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title('Edge using canny filter', fontsize=16)
fig.tight_layout()

plt.show()
```

Result :



Problem #5:

Sources:
The code used to answer this problem set was greatly influenced by the eBook, *"Practical Python and OpenCV" written by Dr. Adrian Rosebrock* and from the online documentation for openCV.