Cornelio, Andrew HW2

March 1, 2020

1 Mathematical Image Analysis (553.493)

Cornelio, Andrew

March 1, 2020

[1]:
```
import cv2
import numpy as np
import math
import matplotlib
import matplotlib.pyplot as plt
import argparse
import itertools
```

1.0.1 Problem 2.1

We can define a look up table and use opencv's look up table function to perform this operation.

[2]:
```
def contrast_helper(p, r_1, s_1, r_2, s_2):
    if p < r_1:
        return (s_1 / r_1) * p
    elif p < r_2:
        return (s_2 - s_1)/(r_2 - r_1) * (p - r_1) + s_1
    else:
        return (255.0 - s_2)/(255.0 - r_2) * (p - r_2) + s_2

# easy way of performing discrete pixel transformation on image is to create a␣
 ↪look up table
def contrast_func(image, r_1, s_1, r_2, s_2):
    contrast = np.vectorize(contrast_helper)
    table = contrast(np.arange(0, 256), r_1, s_1, r_2, s_2).astype("uint8")
    return np.array(cv2.LUT(image, table))
```

[3]:
```
image_airport = cv2.imread('image-airport.tif', 0)
image_airport_enhanc = contrast_func(image_airport, 170, 70, 230, 250)

print(image_airport_enhanc.shape)
```

```
plt.imshow(image_airport_enhanc, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

```
(769, 765)
```

1.0.2 Problem 2.2

We use the same definition for the rotation tranform that we did for the last homework:

$$T^{-1}: \begin{cases} u(x,y) = x\cos\theta + y\,sin\,\theta \\ v(x,y) = -x\,sin\,\theta + x\cos\theta \end{cases}$$

$$f(x,y) = F(u(x,y), v(x,y))$$

Now we can take the gradient of $f(x,y)$ w.r.t. (x,y) >

$$\frac{\partial f}{\partial x} = \frac{\partial F}{\partial u} cos\,\theta - \frac{\partial F}{\partial v} sin\,\theta$$

$$\frac{\partial f}{\partial y} = \frac{\partial F}{\partial u} sin\,\theta + \frac{\partial F}{\partial v} cos\,\theta$$

1.0.3 a

First we show that the magnitude of the gradient is invariant:

$$\begin{aligned}
||\nabla f|| &= \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \\
&= \sqrt{\left(\frac{\partial F}{\partial u} cos\,\theta - \frac{\partial F}{\partial v} sin\,\theta\right)^2 + \left(\frac{\partial F}{\partial u} sin\,\theta + \frac{\partial F}{\partial v} cos\,\theta\right)^2} \\
&= \sqrt{\left(\frac{\partial F}{\partial u}\right)^2 (cos^2\,\theta + sin^2\,\theta) + \left(\frac{\partial F}{\partial v}\right)^2 (cos^2\,\theta + sin^2\,\theta)} \\
&= \sqrt{\left(\frac{\partial F}{\partial u}\right)^2 + \left(\frac{\partial F}{\partial v}\right)^2} \\
&= ||\nabla F||
\end{aligned}$$

Since a rotational tranformation doesn't change the magnitude of the gradient, the magnitude of the gradient must be isotrophic.

1.0.4 b

Now we show that the approximate magnitude of the gradient is not isotrophic:

$$\begin{aligned}
||\nabla f||_A &= \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right| \\
&= \left|\frac{\partial F}{\partial u} cos\,\theta - \frac{\partial F}{\partial v} sin\,\theta\right| + \left|\frac{\partial F}{\partial u} sin\,\theta + \frac{\partial F}{\partial v} cos\,\theta\right|
\end{aligned}$$

To prove this expression is not isotrophic, we just need to find one counter example that shows this expression changes with varying θ. To produce such an example, set both partials to 1. Now we get the expression:

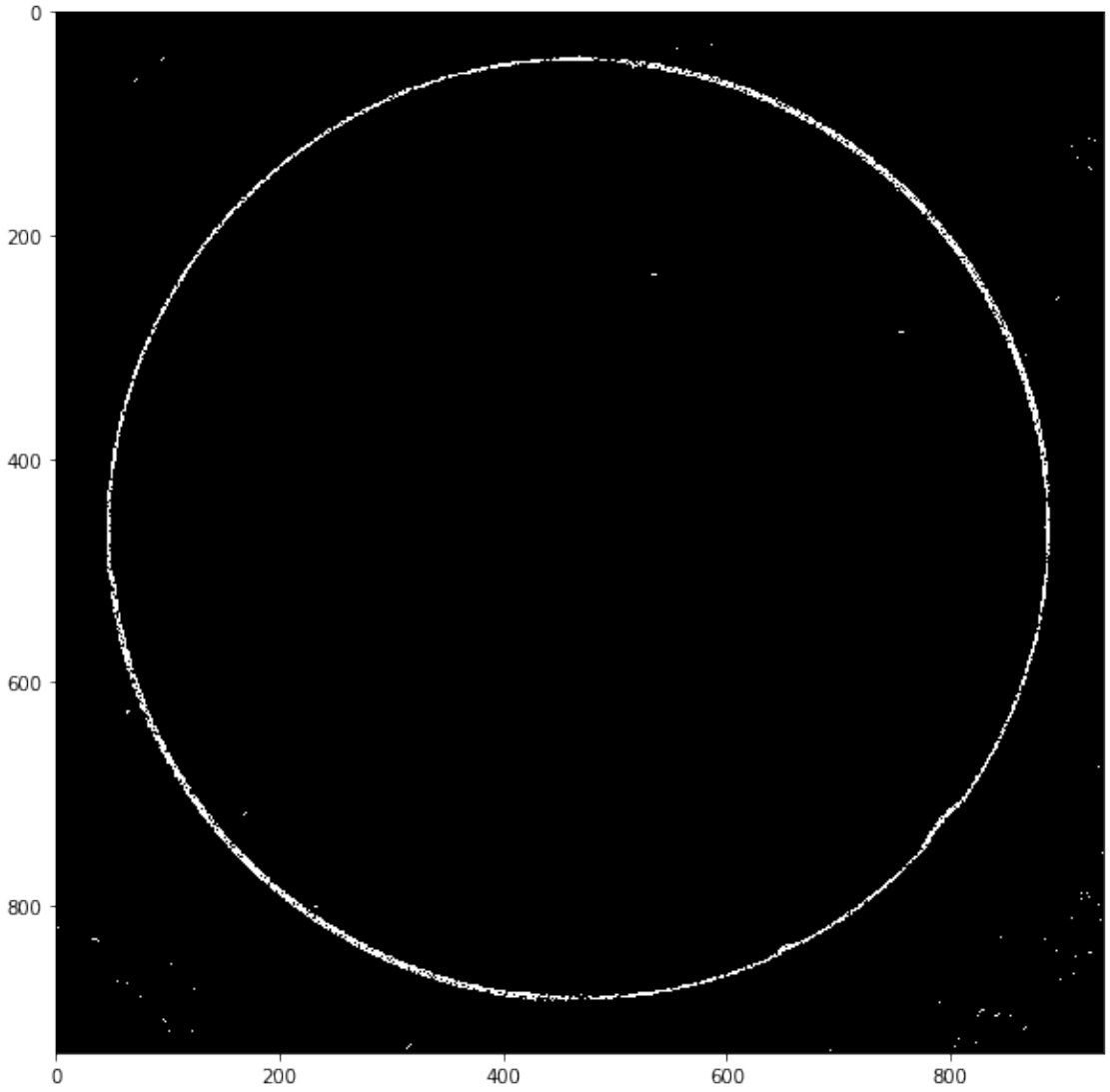
$$||\nabla f||_A = |cos\,\theta - sin\,\theta| + |cos\,\theta + sin\,\theta|$$

Notice that if we choose $\theta = 0$, then $||\nabla f||_A = 2$. However, if we choose $\theta = \frac{\pi}{4}$, then $||\nabla f||_A = \sqrt{2}$. Since different theta produce different results, this formula is not isotrophic.

1.0.5 Problem 2.3.a

We write the two Sobel convolution kernels, convolve the image with each of them and then take the magnitude of the two resulting matrices to get an image. Finally we use the contrast enhancement function from part (a) as a makeshift heaviside function.

[4]:
```
image_contactlens = cv2.imread('image-contactlens.tif', 0)

Kernel_hxS = 1/8 * np.array([[ 1, 2, 1],
                             [ 0, 0, 0],
                             [-1, -2, -1]])


Kernel_hyS = np.transpose(Kernel_hxS)

J_x = cv2.filter2D(image_contactlens, -1, Kernel_hxS)
J_y = cv2.filter2D(image_contactlens, -1, Kernel_hyS)

def H(image, t):
    return contrast_func(image, t, 0, t, 255)


J = np.sqrt(J_x*J_x + J_y*J_y).astype("uint8")
J_enhanc = H(J, int(np.amax(J) / 3))


plt.imshow(J_enhanc, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

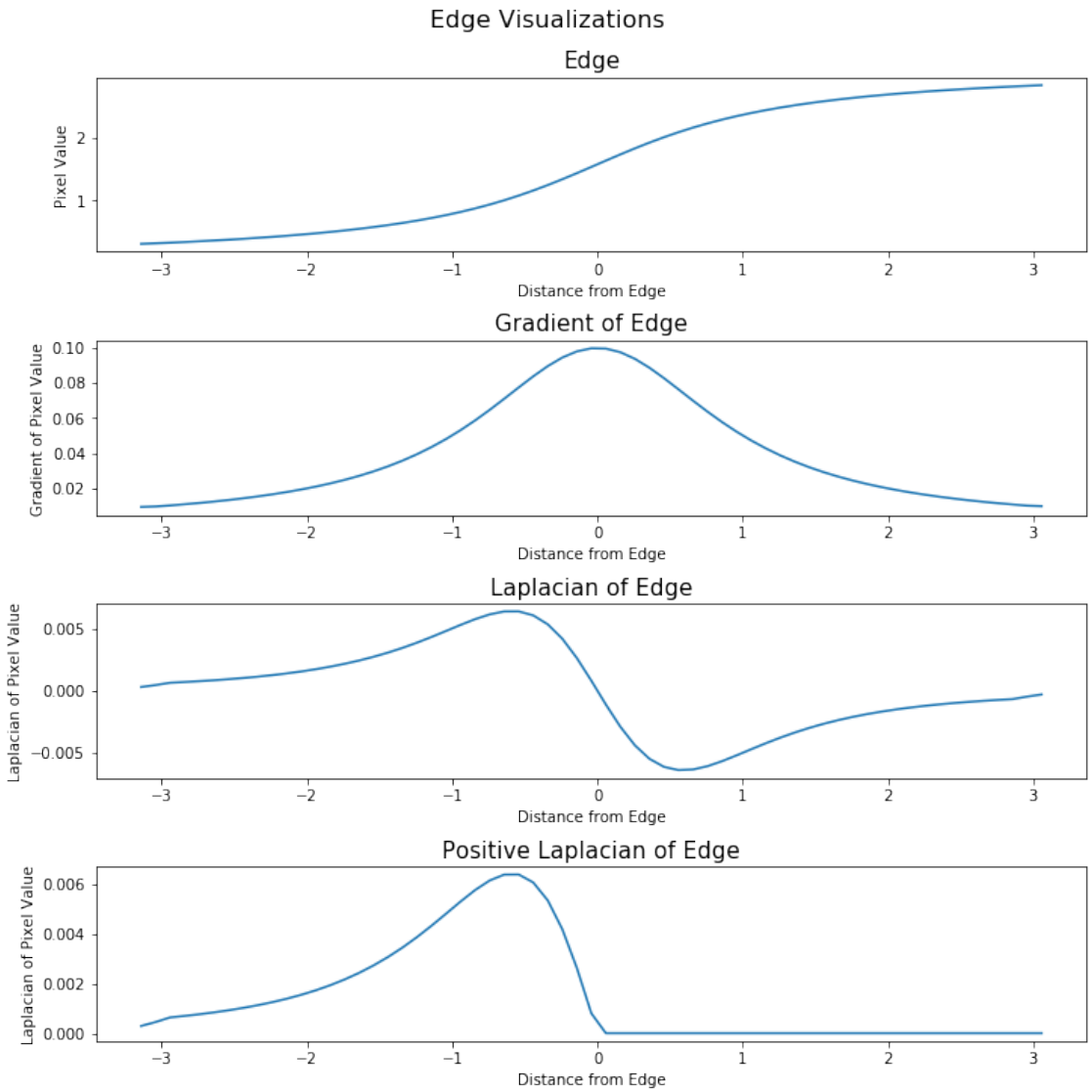


1.0.6 Problem 2.2.b

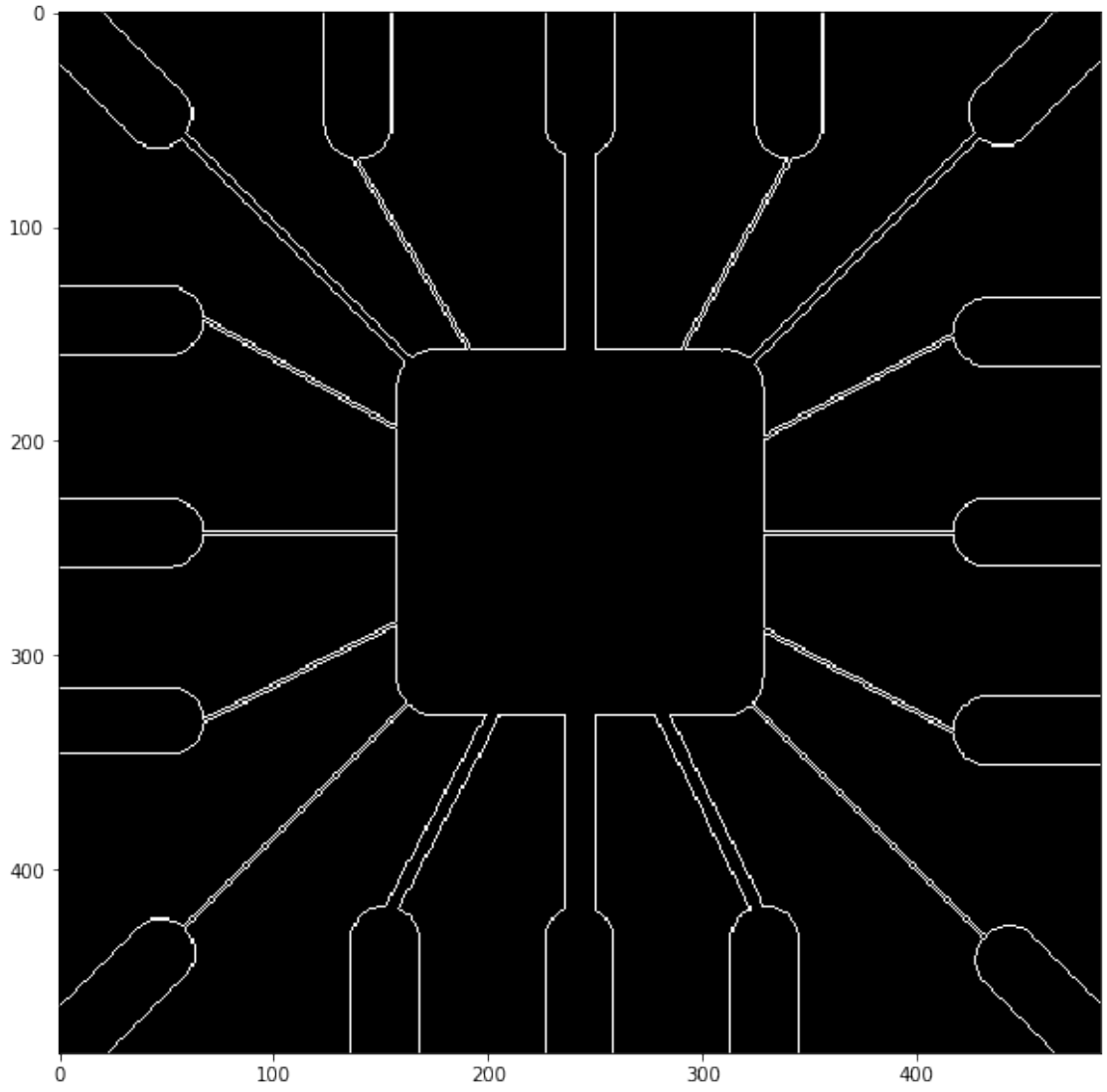
We can think about an a cross section of an edge that between a dark and bright region of the image as a sigmoid. It it important to use a sigmoid since there is a clear demarkation between the end of one region and the start of the other. Had it just been a linear function, there would not be that demarcation. In the following visualizatins, we can see that the laplacian of the edge will have a zero crossing in the middle of the edge. However, if we simply take the maximum of the laplacian with 0, we can still recognize the zero crossing because of the sharp transition from the laplacian to the zeroed portion.

[5]:
```
X = np.arange(-math.pi,math.pi,0.1);
sigmoid = np.arctan(X) + math.pi/2;
```

```
d_sigmoid = np.gradient(sigmoid)
dd_sigmoid = np.gradient(d_sigmoid);
pos_dd_sigmoid = dd_sigmoid.clip(min=0);

fig, axs = plt.subplots(4, 1, figsize=(10, 10), constrained_layout=True)
fig.suptitle('Edge Visualizations', fontsize=16)


axs[0].plot(X, sigmoid)
axs[0].set_title('Edge', fontsize = 15)
axs[0].set_xlabel('Distance from Edge',fontsize = 10) #xlabel
axs[0].set_ylabel('Pixel Value', fontsize = 10)#ylabel

axs[1].plot(X, d_sigmoid)
axs[1].set_title('Gradient of Edge', fontsize = 15)
axs[1].set_xlabel('Distance from Edge',fontsize = 10) #xlabel
axs[1].set_ylabel('Gradient of Pixel Value', fontsize = 10)#ylabel

axs[2].plot(X, dd_sigmoid)
axs[2].set_title('Laplacian of Edge', fontsize = 15)
axs[2].set_xlabel('Distance from Edge',fontsize = 10) #xlabel
axs[2].set_ylabel('Laplacian of Pixel Value', fontsize = 10)#ylabel

axs[3].plot(X, pos_dd_sigmoid)
axs[3].set_title('Positive Laplacian of Edge', fontsize = 15)
axs[3].set_xlabel('Distance from Edge',fontsize = 10) #xlabel
axs[3].set_ylabel('Laplacian of Pixel Value', fontsize = 10)#ylabel

plt.show()
```



[6]:
```
laplacian = np.array([[0, 1, 0],
                      [1, -4, 1],
                      [0, 1, 0]])


image_circuit2 = cv2.imread('image-circuit2.tif', 0)
image_circuit2_lap = cv2.filter2D(image_circuit2, -1, laplacian)
image_circuit2_lap_pos = image_circuit2_lap.clip(min=0);

plt.imshow(image_circuit2_lap_pos, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

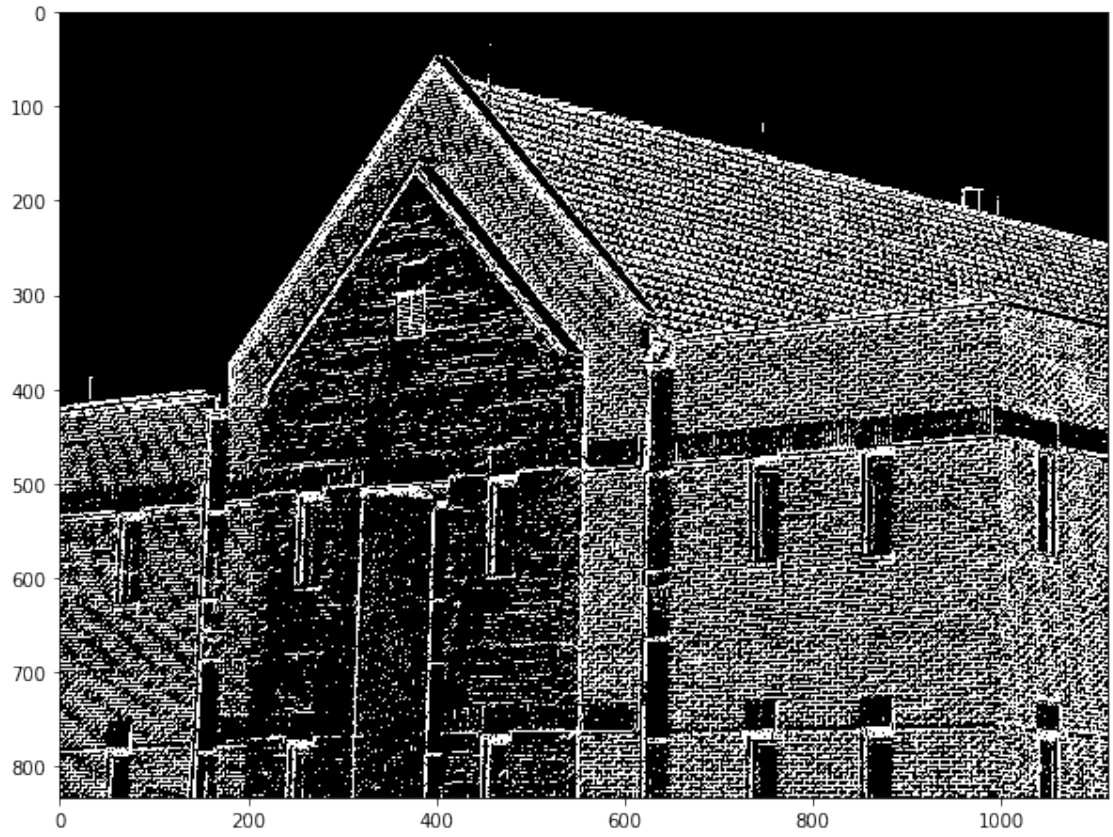


1.0.7 Problem 2.2.c

For the first part of this question, we can copy and paste our previous code. For the second part, we first use the mean blur and then the gradient edge edection.

[7]:
```
image_building = cv2.imread('image-building.tif', 0)

Kernel_hxS = 1/8 * np.array([[ 1, 2, 1],
                             [ 0, 0, 0],
                             [-1, -2, -1]])
```

```
Kernel_hyS = np.transpose(Kernel_hxS)

J_x = cv2.filter2D(image_building, -1, Kernel_hxS)
J_y = cv2.filter2D(image_building, -1, Kernel_hyS)

J = np.sqrt(J_x*J_x + J_y*J_y).astype("uint8")
J_enhanc = H(J, int(np.amax(J) / 3))


plt.imshow(J_enhanc, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```



[8]:
```
image_building = cv2.imread('image-building.tif', 0)

kernel1 = 1/25 * np.ones((5,5))
image_building_blur = cv2.filter2D(image_building, -1, kernel1)

Kernel_hxS = 1/8 * np.array([[ 1, 2, 1],
                             [ 0, 0, 0],
```

```
                              [-1, -2, -1]])


Kernel_hyS = np.transpose(Kernel_hxS)

J_x = cv2.filter2D(image_building_blur, -1, Kernel_hxS)
J_y = cv2.filter2D(image_building_blur, -1, Kernel_hyS)

J = np.sqrt(J_x*J_x + J_y*J_y).astype("uint8")
J_enhanc = H(J, int(np.amax(J) / 3))


plt.imshow(J_enhanc, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

1.0.8 Problem 2.4.a

We must use the properties of vectorspaces as well as properties of norms. Specifically we will use the property of additive inverses and scalar distributivity from vaector spaces, as well as property

(N2) of norms. Assume there is some arbtratry vector $\mathbf{v} \in \mathbf{V}$:

$$||\mathbf{0}|| = ||\mathbf{v} - \mathbf{v}|| = ||(1-1)\mathbf{v}|| = |0|||\mathbf{v}|| = 0$$

We see that this is simply the gamma function that uses the reciprocal value as the original.

Now we use property of the norm we just proved and along with the ones we used before, and (N2) to show the next property:

$$0 = ||\mathbf{0}|| = ||\mathbf{v} - \mathbf{v}|| \leq ||\mathbf{v}|| + ||-\mathbf{v}|| = ||\mathbf{v}|| + ||\mathbf{v}|| = 2||\mathbf{v}||$$

So we see that

$$0 \leq ||\mathbf{v}||$$

1.0.9 Problem 2.4.b

First we will show property (N1):

$$\begin{aligned}
\big|\big|aI\big|\big|_{TV} &= \frac{1}{Area(D)} \int\int_D ||\nabla(aI(x,y))||dA \\
&= \frac{1}{Area(D)} \int\int_D \sqrt{\left(\frac{\partial}{\partial x} aI(x,y)\right)^2 + \left(\frac{\partial}{\partial y} aI(x,y)\right)^2} dA \\
&= \frac{1}{Area(D)} \int\int_D \sqrt{(aI_X)^2 + (aI_Y)^2} dA \\
&= \frac{1}{Area(D)} \int\int_D |a|\sqrt{I_X^2 + I_Y^2} dA \\
&= \frac{|a|}{Area(D)} \int\int_D ||\nabla I(x,y)||dA \\
&= |a|\big|\big|I\big|\big|_{TV}
\end{aligned}$$

Now we show property (N2). Note that gradients are vectors in $\mathbb{R}^2$, which are subject to the triangle inequality:

$$\begin{aligned}
\big|\big|I+J\big|\big|_{TV} &= \frac{1}{Area(D)} \int\int_D ||\nabla(I+J)||dA \\
&= \frac{1}{Area(D)} \int\int_D ||\nabla I + \nabla J||dA \\
leq &\frac{1}{Area(D)} \int\int_D ||\nabla I|| + ||\nabla J||dA \\
&= \frac{1}{Area(D)} \int\int_D ||\nabla I||dA + \frac{1}{Area(D)} \int\int_D ||\nabla J||dA \\
&= \big|\big|I\big|\big|_{TV} + \big|\big|J\big|\big|_{TV}
\end{aligned}$$

Now we will show a counter example to (N3). Suppose $I = c$ where c is an arbitrary constant not necessarily equal to 0:

$$\begin{aligned} ||c||_{TV} &= \frac{1}{Area(D)} \int\int_D ||\nabla c|| dA \\ &= \frac{1}{Area(D)} \int\int_D ||0|| dA \\ &= \frac{1}{Area(D)} \int\int_D 0 dA \\ &= 0 \end{aligned}$$

So we see that the total variatioal norm does not satisfy (N3), so it is in fact a semi norm. We write a function to computer the TV norm in the next cell.

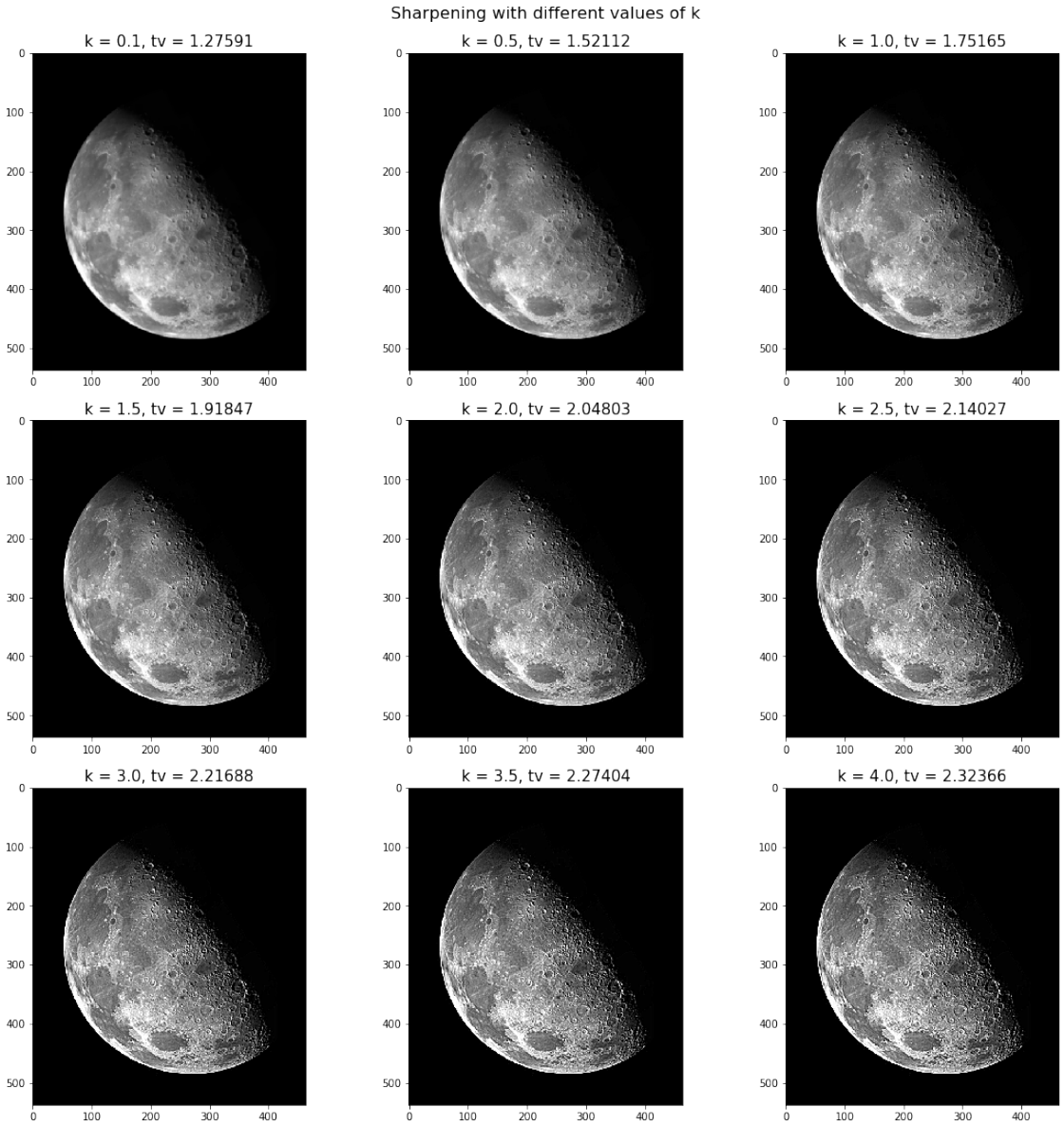
[9]:
```
def calc_TV(image):
    Kernel_hxS = 1/8 * np.array([[ 1, 2, 1],
                                 [ 0, 0, 0],
                                 [-1, -2, -1]])


    Kernel_hyS = np.transpose(Kernel_hxS)

    J_x = cv2.filter2D(image, -1, Kernel_hxS)
    J_y = cv2.filter2D(image, -1, Kernel_hyS)

    J = np.sqrt(J_x*J_x + J_y*J_y).astype("uint8")

    return 1/J.size * np.sum(J)
```

1.0.10 Problem 2.5.a

We define a sharpening function and then use it to create different versions of the image with various degrees of sharpening

[10]:
```
def sharp_k(image, k) :
    delta = np.array([[0, 0, 0],
                      [0, 1, 0],
                      [0, 0, 0]])

    laplacian = np.array([[0, 1, 0],
                          [1, -4, 1],
                          [0, 1, 0]])

    h_Sh = delta - k*laplacian
    sharp = image_moon_lap = cv2.filter2D(image, -1, h_Sh)
    return sharp
```

```
fig, axs = plt.subplots(3,3, figsize=(15, 15), constrained_layout=True)
fig.suptitle('Sharpening with different values of k', fontsize=16)

image_moon = cv2.imread('image-moon.jpg', 0)

ks = np.array([[0.1, 0.5, 1],
               [1.5, 2, 2.5],
               [3, 3.5, 4]])

s = sharp_k(image_moon, 1);


for i in range(0, 3):
    for j in range(0,3):
        s = sharp_k(image_moon, ks[i][j]).clip(min=0,max=255)
        tv_string = '%.5f' % calc_TV(s)
        axs[i][j].imshow(s, cmap='gray', vmin=0, vmax=255)
        axs[i][j].set_title('k = ' + str(ks[i][j]) + ', tv = ' + tv_string,␣
 ↪fontsize = 15)


plt.show()
```



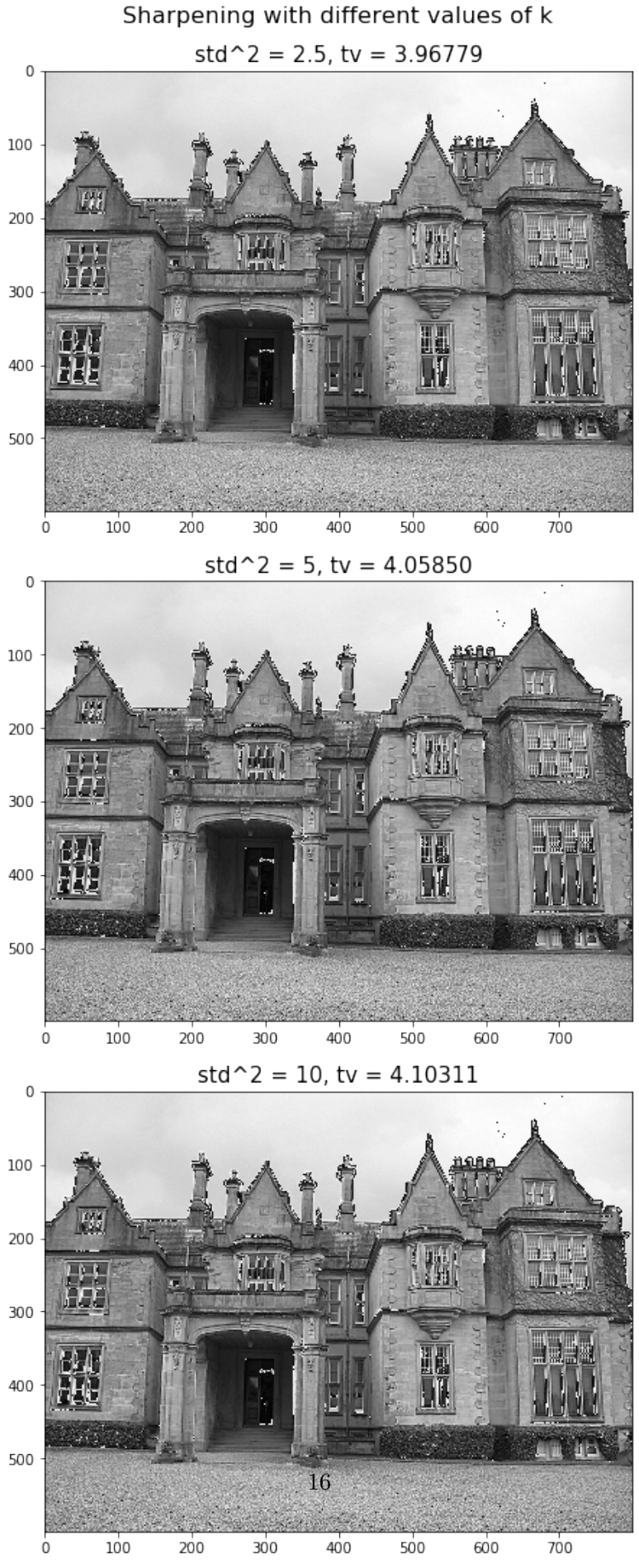
1.0.11 2.5.b

We define a function that calculaed the gaussian blur and then subtracts the blurred image from the original to find the sharp component. Notice that while the image appears sharper, it also appears noisier. This is because blurring removes the sharp components but also removes noise. So by boosting the sharp posrtion relative to the unsharp portion, we are also boosting the noise.

[11]:
```
def sharp_mask(img, a, sig2, n) :
    img_M = cv2.GaussianBlur(img,(n,n),math.sqrt(sig2))
    img_S = img - img_M
```

```
    return a*img_S + img


image_irish = cv2.imread('image-irish.tif', 0)
```

[12]:
```
fig, axs = plt.subplots(3,1, figsize=(15, 15), constrained_layout=True)
fig.suptitle('Sharpening with different values of k', fontsize=16)


m1 = sharp_mask(image_irish, 1, 2.5, 7).clip(min=0, max=255)
tv_string = '%.5f' % calc_TV(m1)
axs[0].imshow(m1, cmap='gray', vmin=0, vmax=255)
axs[0].set_title('std^2 = 2.5, tv = ' + tv_string, fontsize = 15)

m2 = sharp_mask(image_irish, 1, 5, 7).clip(min=0, max=255)
tv_string = '%.5f' % calc_TV(m2)
axs[1].imshow(m2, cmap='gray', vmin=0, vmax=255)
axs[1].set_title('std^2 = 5, tv = ' + tv_string, fontsize = 15)

m3 = sharp_mask(image_irish, 1, 10, 7).clip(min=0, max=255)
tv_string = '%.5f' % calc_TV(m3)
axs[2].imshow(m3, cmap='gray', vmin=0, vmax=255)
axs[2].set_title('std^2 = 10, tv = ' + tv_string, fontsize = 15)

plt.show();
```


Sharpening with different values of k
std^2 = 2.5, tv = 3.96779
std^2 = 5, tv = 4.05850
std^2 = 10, tv = 4.10311
0
100
200
300
400
500
600
700

1.0.12 2.6

We blur the image and then apply the directional derivative.We notice that the front face of the building, which is the darkest region, is highlighted.

[13]:
```
h_n = 1/8 * np.array([[2, 1, 0],
                      [1, 0, -1],
                      [0, -1, 2]])


image_building = cv2.imread('image-building.tif', 0)
kernel1 = 1/25 * np.ones((5,5))
image_building_blur = cv2.filter2D(image_building_blur, -1, kernel1)

image_building_dif = cv2.filter2D(image_building, -1, h_n)
J_enhanc = H(image_building_dif, int(np.amax(image_building_dif) / 3))


plt.imshow(J_enhanc, cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

1.0.13 2.7

We use the definition of the discrete convolution on the two kernels.

$$(g * h)[i, j] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} g(i-k, j-l)h(k, l)$$

Let's just consider the rows of the image, but the same logic applies the the columns. When considering image g, the row can range from $i - K_1$ to $i + K_1$. If we consider the image h, the row can range between $-K_2$ and K_2. The largest and smallest possible i for which the resulting image will be one for which both indeices are defined on the original images. This maximum and mimumum will define the boundaries of the image. When it comes to the maximum, we cannot do any better than $i = K_1 + K_2$, where we set $k = K_2$. If it was any larger, either the rows in h would be out of range, or the rows of g would be out of range. Similarly the minimum must be $i = -K_1 - K_2$, where $k = -K_2$. Hence we can see that i, the rows of the output, has $2(K_1 + K_2) + 1$ possible values. If we use the same exact reasoning for the columns, we see that the result of the covolution should be of size $[2K_1 + 2K_2 + 1] \times [2K_1 + 2K_2 + 1]$.

1.0.14 2.8

We can apply the definition of the volume to the definition of the convolution:

$$\begin{aligned} Vol(f*g) &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}(g*h)[i,j] \\ &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} g(i-k,j-l)h(k,l) \end{aligned}$$

We can use the following change of variables:

$$x = i-k \qquad y = j-l$$

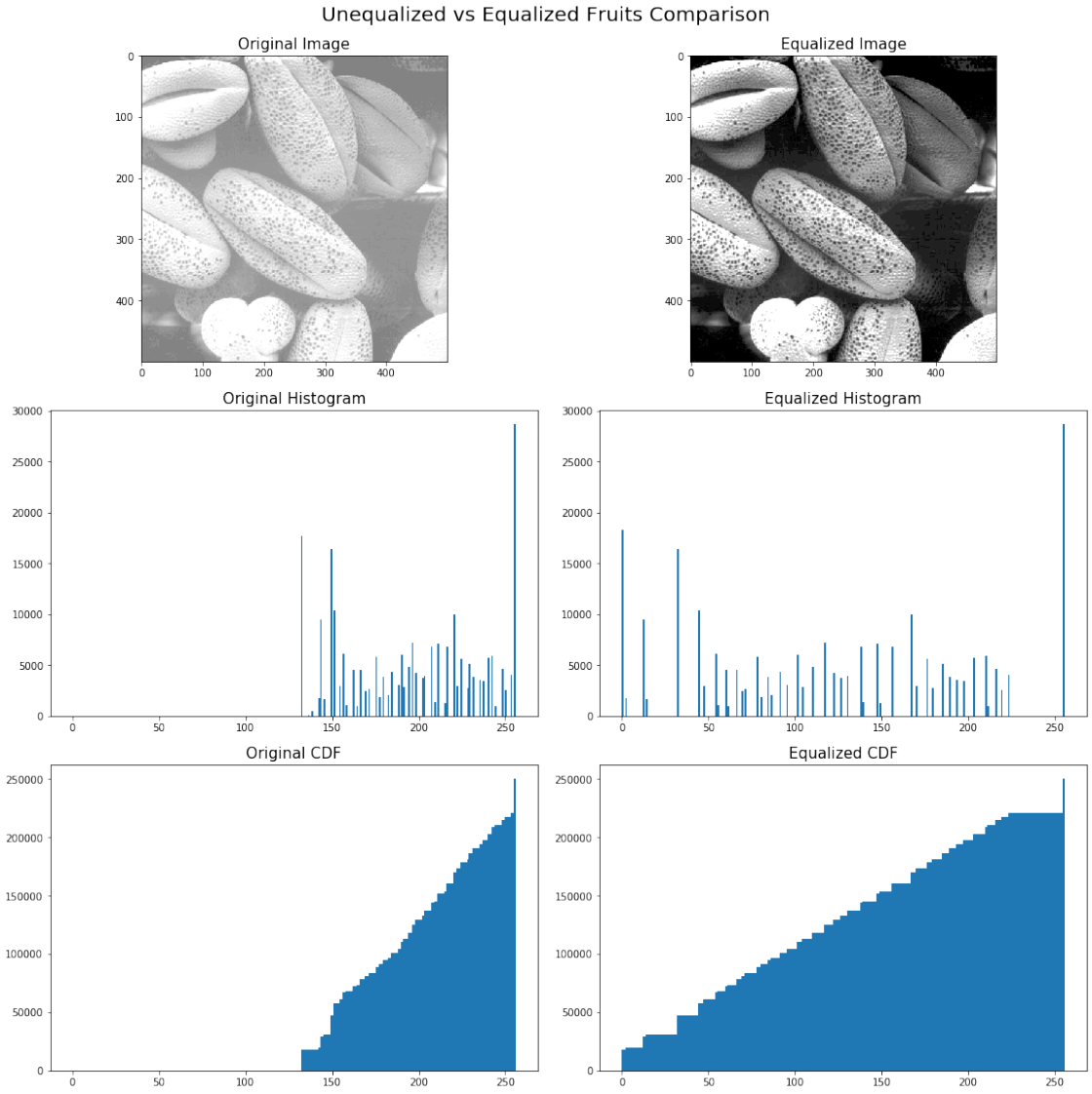
Using this substitution we get

$$\begin{aligned} Vol(f*g) &= \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty}\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} g(x,y)h(k,l) \\ &= \sum_{x=-\infty}^{\infty}\sum_{y=-\infty}^{\infty} g(x,y)\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} h(k,l) \\ &= Vol(g)Vol(h) \end{aligned}$$

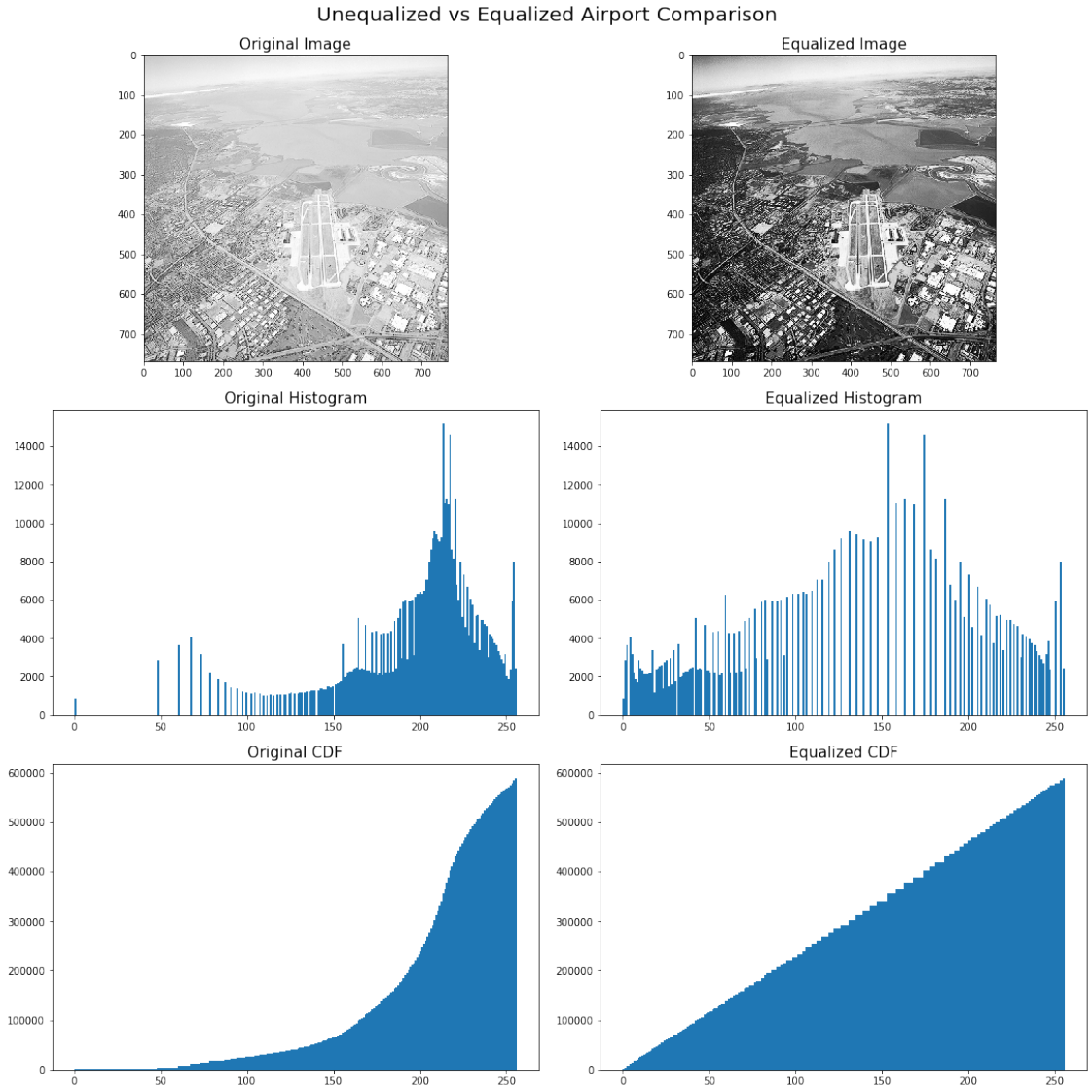
[14]:
```
def equalizing_func(image):
    pix, freq = np.unique(image, return_counts=True)
    cdf = 255/image.size * np.array(list(itertools.accumulate(freq)))
    new_pix = ((cdf - cdf[0])/(255 - cdf[0]) * 255).astype("uint8")
    table = np.zeros(256)
    for i in range(len(pix)):
        table[pix[i]] = new_pix[i]
    return np.array(cv2.LUT(image, table))
```

[15]:
```
image_fruits = cv2.imread('image-fruits.tif', 0)
image_fruits_eq = equalizing_func(image_fruits)

fig, axs = plt.subplots(3, 2, figsize=(15, 15), constrained_layout=True)
fig.suptitle('Unequalized vs Equalized Fruits Comparison', fontsize=20)

axs[0][0].imshow(image_fruits, cmap='gray', vmin=0, vmax=255)
axs[0][0].set_title('Original Image', fontsize = 15)

axs[0][1].imshow(image_fruits_eq, cmap='gray', vmin=0, vmax=255)
axs[0][1].set_title('Equalized Image', fontsize = 15)

axs[1][0].hist(image_fruits.ravel(),256,[0,256])
axs[1][0].set_title('Original Histogram', fontsize = 15)
```

```
axs[1][1].hist(image_fruits_eq.ravel(),256,[0,256])
axs[1][1].set_title('Equalized Histogram', fontsize = 15)

axs[2][0].hist(image_fruits.ravel(),256,[0,256], cumulative=True)
axs[2][0].set_title('Original CDF', fontsize = 15)

axs[2][1].hist(image_fruits_eq.ravel(),256,[0,256], cumulative=True)
axs[2][1].set_title('Equalized CDF', fontsize = 15)

plt.show()
```



[16]:
```
image_airport = cv2.imread('image-airport.tif', 0)
image_airport_eq = equalizing_func(image_airport)

fig, axs = plt.subplots(3, 2, figsize=(15, 15), constrained_layout=True)
fig.suptitle('Unequalized vs Equalized Airport Comparison', fontsize=20)

axs[0][0].imshow(image_airport, cmap='gray', vmin=0, vmax=255)
axs[0][0].set_title('Original Image', fontsize = 15)

axs[0][1].imshow(image_airport_eq, cmap='gray', vmin=0, vmax=255)
axs[0][1].set_title('Equalized Image', fontsize = 15)

axs[1][0].hist(image_airport.ravel(),256,[0,256])
axs[1][0].set_title('Original Histogram', fontsize = 15)

axs[1][1].hist(image_airport_eq.ravel(),256,[0,256])
axs[1][1].set_title('Equalized Histogram', fontsize = 15)

axs[2][0].hist(image_airport.ravel(),256,[0,256], cumulative=True)
axs[2][0].set_title('Original CDF', fontsize = 15)

axs[2][1].hist(image_airport_eq.ravel(),256,[0,256], cumulative=True)
axs[2][1].set_title('Equalized CDF', fontsize = 15)

plt.show()
```



[17]:
```
image_spine = cv2.imread('image-spine.tif', 0)
image_spine_eq = equalizing_func(image_spine)

fig, axs = plt.subplots(3, 2, figsize=(15, 15), constrained_layout=True)
fig.suptitle('Unequalized vs Equalized Spine Comparison', fontsize=20)

axs[0][0].imshow(image_spine, cmap='gray', vmin=0, vmax=255)
axs[0][0].set_title('Original Image', fontsize = 15)

axs[0][1].imshow(image_spine_eq, cmap='gray', vmin=0, vmax=255)
axs[0][1].set_title('Equalized Image', fontsize = 15)

axs[1][0].hist(image_spine.ravel(),256,[0,256])
```

```
axs[1][0].set_title('Original Histogram', fontsize = 15)

axs[1][1].hist(image_spine_eq.ravel(),256,[0,256])
axs[1][1].set_title('Equalized Histogram', fontsize = 15)

axs[2][0].hist(image_spine.ravel(),256,[0,256], cumulative=True)
axs[2][0].set_title('Original CDF', fontsize = 15)

axs[2][1].hist(image_spine_eq.ravel(),256,[0,256], cumulative=True)
axs[2][1].set_title('Equalized CDF', fontsize = 15)

plt.show()
```