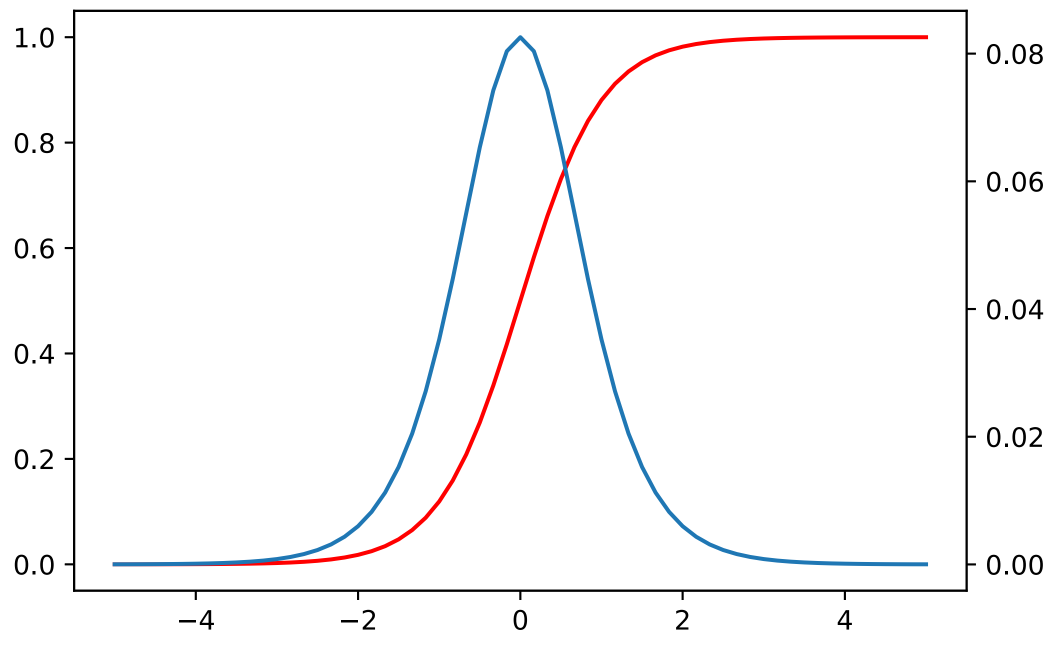
In [6]:
```
import numpy as np #numerical computing module
import matplotlib.pyplot as plt #plotting module
#when using a notebook, the line below will display any plots directly in it:
%matplotlib inline
from __future__ import division
from PIL import Image
import numpy.fft as fft
from scipy import signal
import scipy.io
from mpl_toolkits.mplot3d import Axes3D
```

In [59]:
```
def sigmoid(x):
    return (1/(1+np.exp(-2*x)))
```

In [60]:
```
x = np.linspace(-5,5,61)
plt.plot(x, sigmoid(x), 'r')
n = np.gradient(sigmoid(x))
ax2 = plt.twinx()
ax2.plot(x, n)
```

Out[60]:
```
[<matplotlib.lines.Line2D at 0xd26e288>]
```

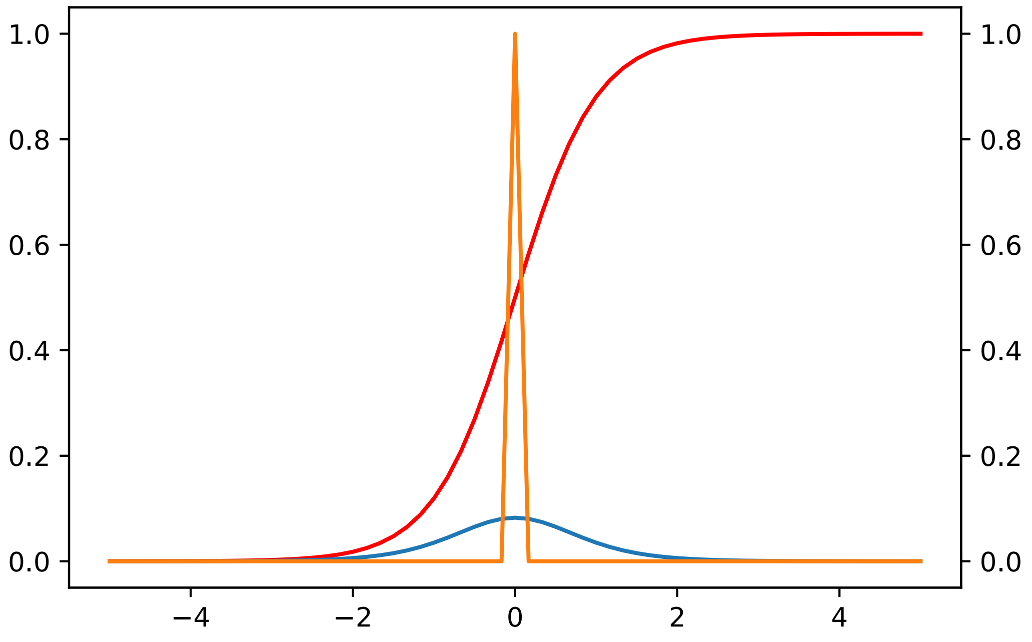


1a.

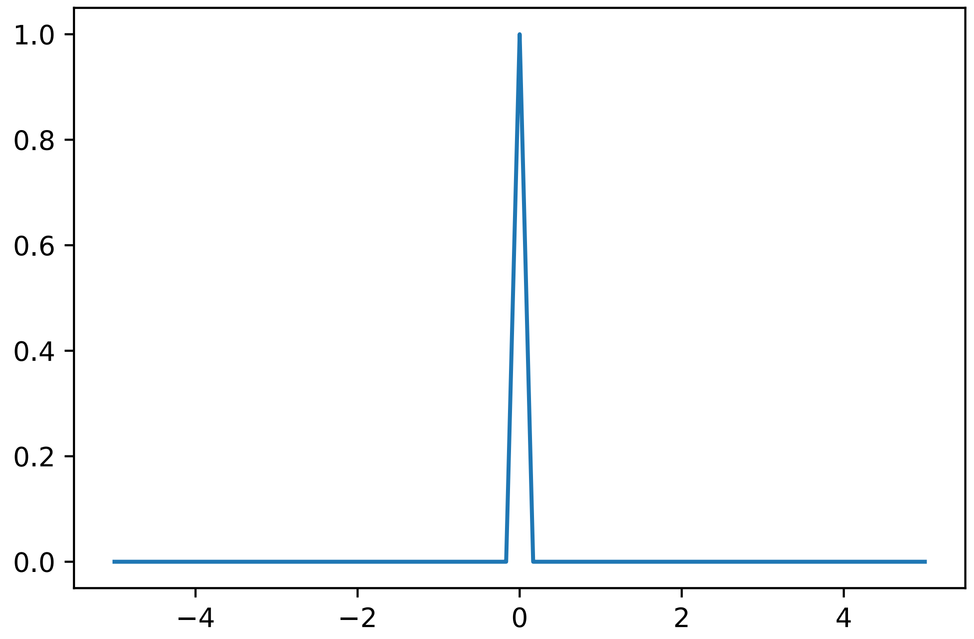
The value of the derivative is highest when the value of the sigmoid function is increasing the fastest. Meanwhile, the value of the derivative is lowest when the value of the sigmoid function is barely changing. The useful information that the derivative can give us is the location of greatest change in an image which indicates an edge

In [61]:
```
x = np.linspace(-5,5,61)
plt.plot(x, sigmoid(x), 'r')
n = np.gradient(sigmoid(x))
ax2 = plt.twinx()
ax2.plot(x, n)
a = np.zeros(len(n))
for i in range(1, len(n) - 1):
    if n[i] >= n[i-1] and n[i] > n[i+1]:
        a[i] = 1
ax2.plot(x, a)
```

Out[61]:
```
[<matplotlib.lines.Line2D at 0xdf50c48>]
```



```
In [62]: xs = np.arange(1, 61 - 1)
         xpeaks = (n[xs-1] <= n[xs]) & (n[xs] > n[xs+1])
         xpeaks = np.pad(xpeaks, (1,1), 'constant', constant_values=(False, False))
         plt.plot(x, xpeaks)

Out[62]: [<matplotlib.lines.Line2D at 0xdfede08>]
```

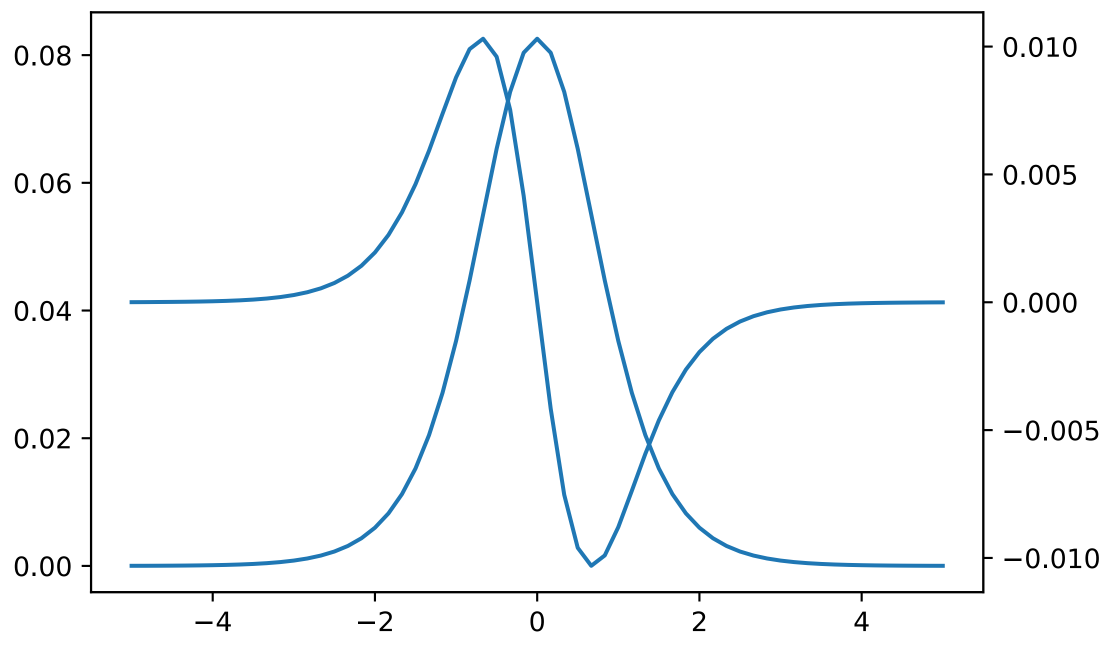


1c.

Yes, they look identical

In [69]:
```
sd = np.gradient(n)
plt.plot(x, n)
ax2 = plt.twinx()
ax2.plot(x, sd)
```

Out[69]:
```
[<matplotlib.lines.Line2D at 0xe82fdc8>]
```

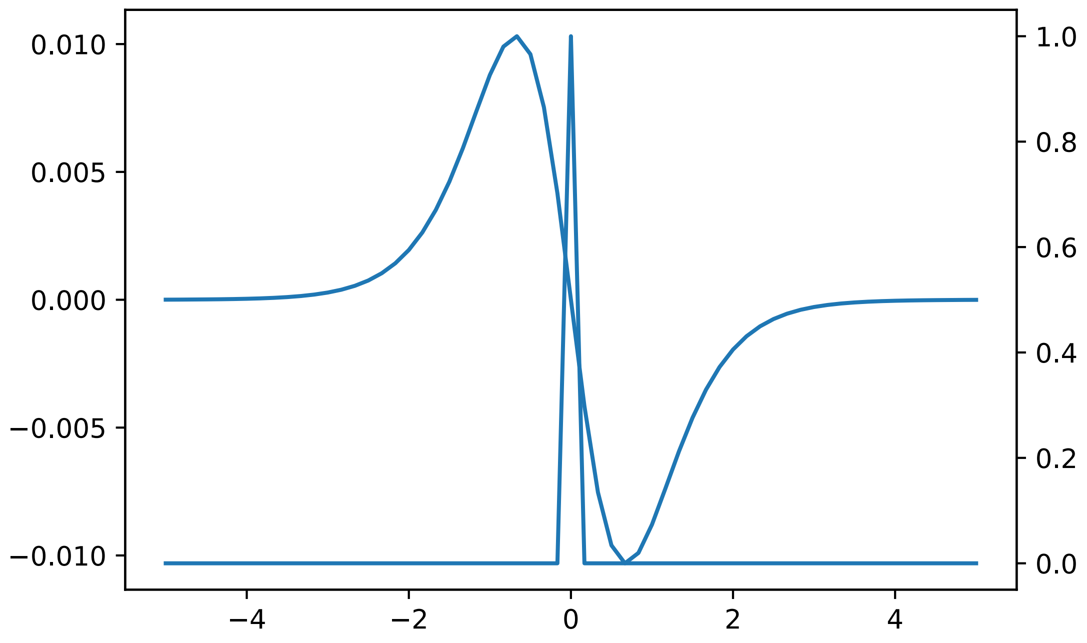


1d.

The point where the second derivative crosses the zero value is where the maximum of the first derivative is, so it is the point where the first derivative switches from increasing to decreasing

In [112]:
```
xs = np.arange(1, 61 - 1)
second_peaks = (sd[xs-1] > 0) & (sd[xs] <= 0)
sp = np.pad(second_peaks, (1,1), 'constant', constant_values=(0,0))
plt.plot(x, sd)
ax2 = plt.twinx()
ax2.plot(x, sp)
```

Out[112]:
```
[<matplotlib.lines.Line2D at 0x109bedc8>]
```



1e.

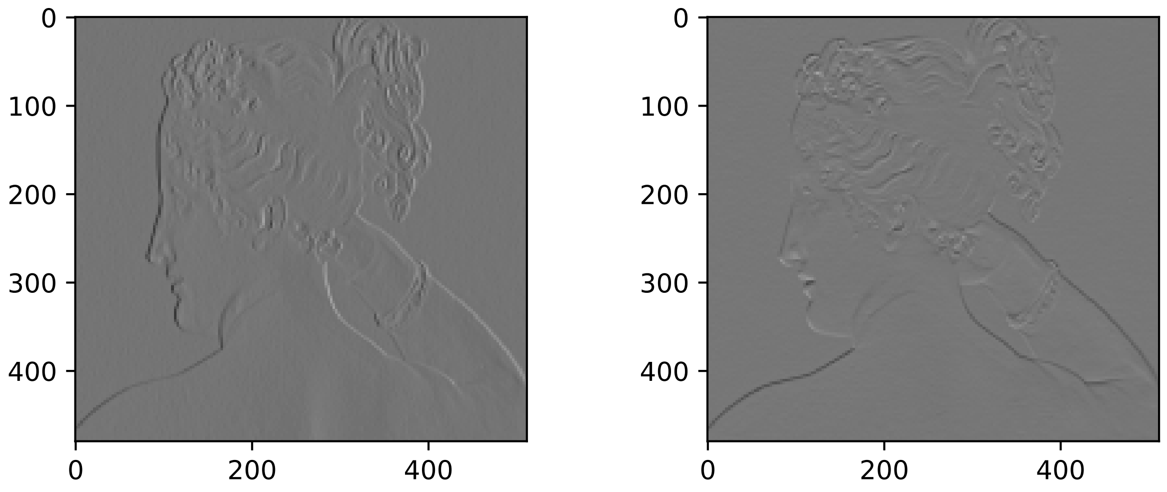
I used a procedure very similar the one in part c. I wanted to find the value where the one before it was greater than 0 and the current one was less than or equal to 0, because this would signify that the second derivative had crossed the x-axis in descending fashion which would mean the first derivative had a maximum. When plotting this over the graph of the second derivative, we see it yields the same result as in part 1c.

In [241]:
```
Sx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])/8
Sy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])/8
```

In [240]:
```
image_file = 'Paolina.tiff'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
```

In [242]:
```
imx = scipy.signal.convolve2d(im,Sx,'same')
imy = scipy.signal.convolve2d(im,Sy,'same')
f,axes = plt.subplots(1,2,figsize=(8,3))
axes[0].imshow(imx, cmap='gray')
axes[1].imshow(imy, cmap='gray')
```

Out[242]:
```
<matplotlib.image.AxesImage at 0x23ed04c8>
```

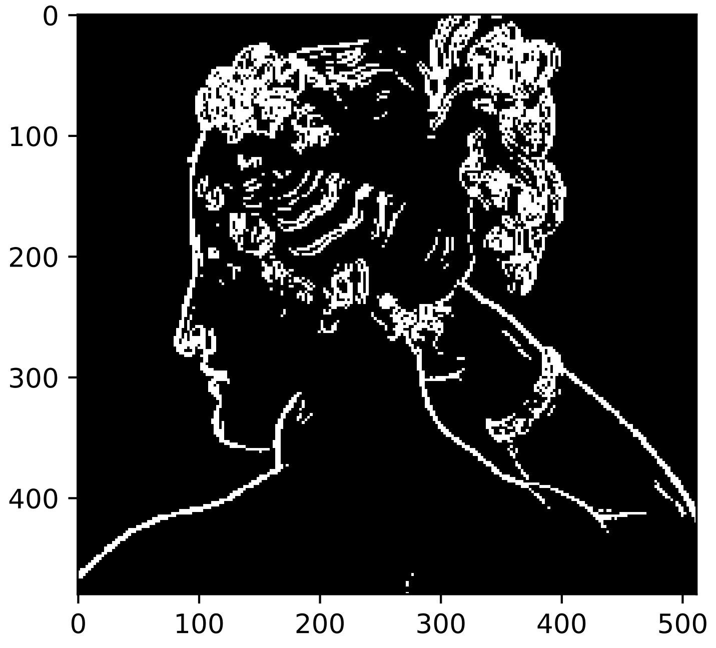


2a.

They convey the lines and edges within the Paolina image. These relate to the results from part 1-a because we took the derivatives in the x and y directions which gave us higher values for areas where they were great change between the pixels in the image which leads to this etching effect. In part 1-a, we did the same thing but in just 1 dimension to find the edge in the sigmoid.

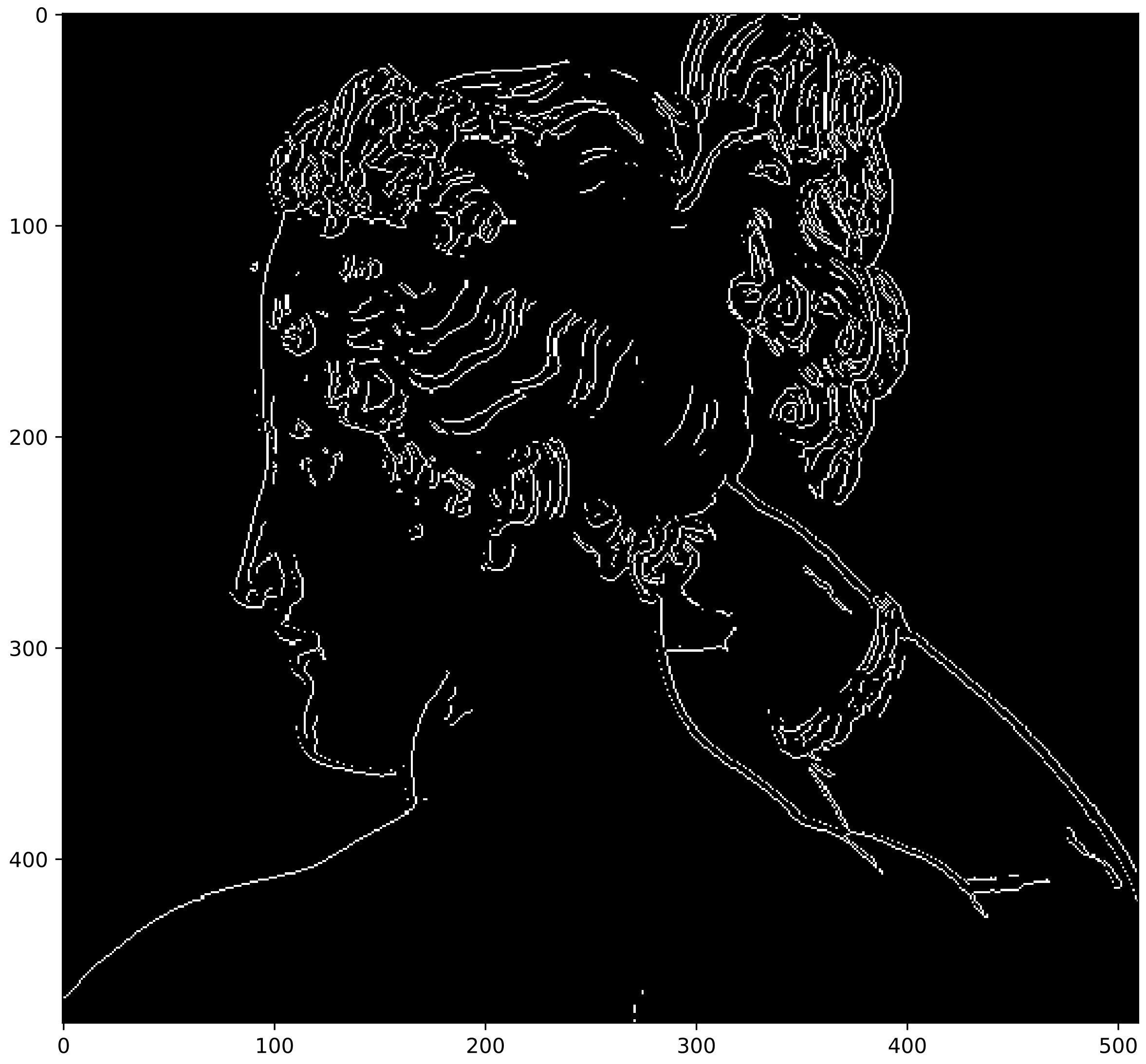
In [243]:
```
gmag = np.sqrt(imx**2.0 + imy**2.0)
im = gmag > 2*np.mean(gmag)
plt.imshow(im, cmap='gray')
```

Out[243]:
```
<matplotlib.image.AxesImage at 0x26589ac8>
```



In [244]:
```python
m, n = im.shape
xs = np.arange(1, n-1)
ys = np.arange(1, m-1)
xpeaks = (im[ys][:,xs-1] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys][:,xs+1])
ypeaks = (im[ys-1][:,xs] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys+1][:,xs])
im = im[ys][:,xs].astype(bool) & (xpeaks | ypeaks)
plt.figure(0,(12,9))
plt.imshow(im, cmap='gray')
```

Out[244]:
```
<matplotlib.image.AxesImage at 0x266cf7c8>
```

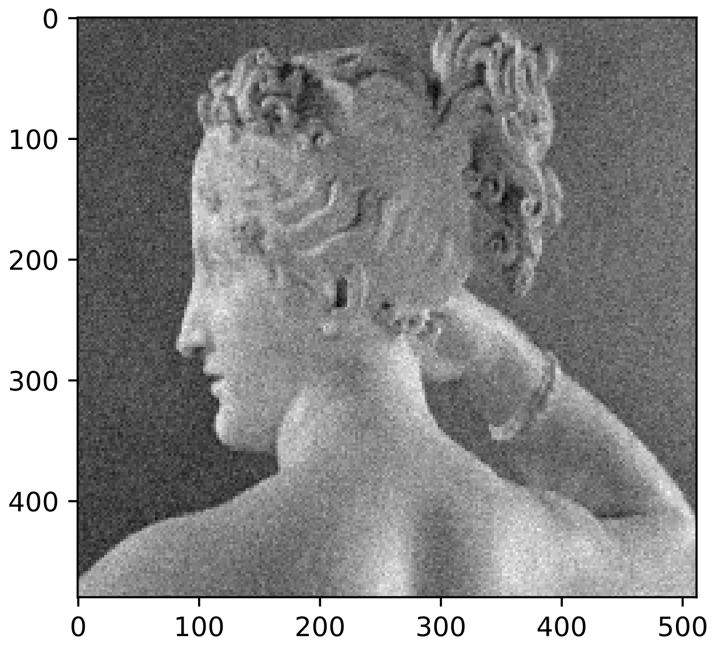


2c.

I see a more pixelated form of the edges from the cleaned-up magnitude image because they are thinner. The purpose of this post-processing is to have definitive points that are either true or false as to whether or not they are an edge, and these edges are only 1 pixel wide so they can be processed more easily. This compares to what we did in part 1-b,c in that we wanted to find the maximums of the derivative, because we know this is where the distinct edges were in the image since the maximum is where the image pixel values are changing the most; the difference is that we used the partial derivatives in the x and y direction this time around because we were in 2D. We need to do this to get an edge map because we want all the edges to be of the same thickness.

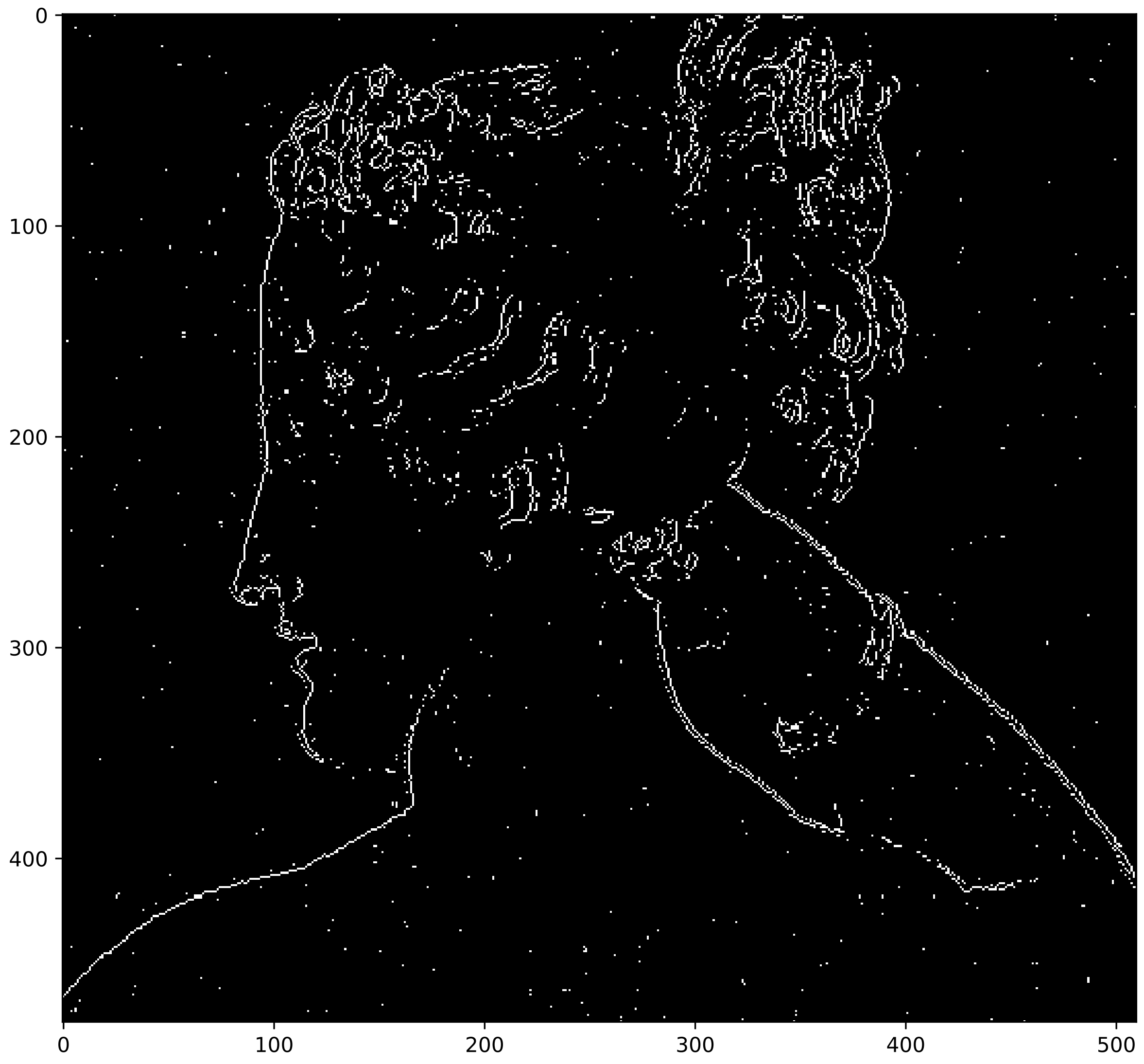
In [352]:
```
image_file = 'Paolina.tiff'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
im = im + np.random.randn(480,512)*.05
plt.imshow(im, cmap='gray')
```

Out[352]:
```
<matplotlib.image.AxesImage at 0x63f62a08>
```



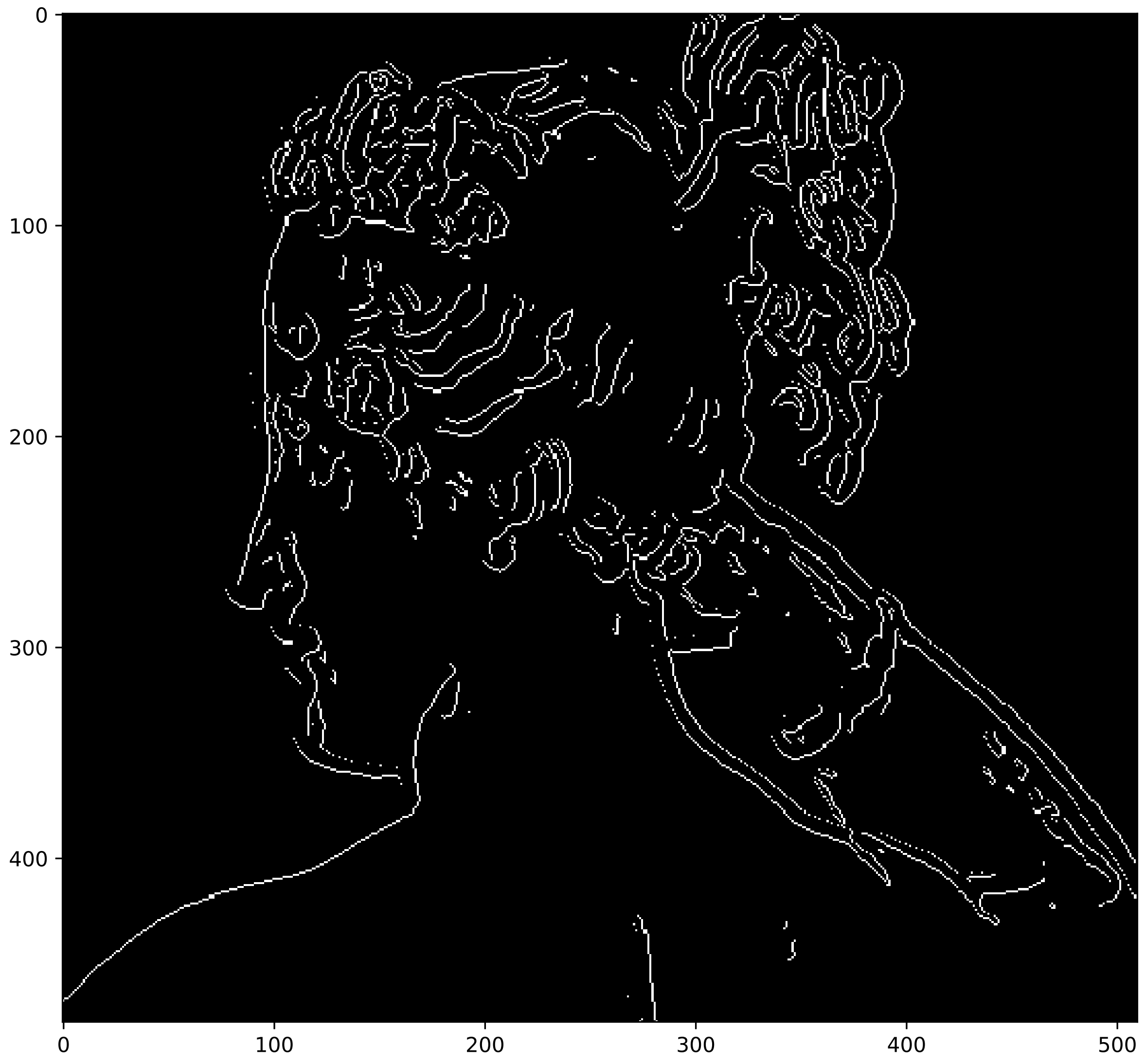
In [353]:
```
imx = scipy.signal.convolve2d(im,Sx,'same')
imy = scipy.signal.convolve2d(im,Sy,'same')
gmag = np.sqrt(imx**2.0 + imy**2.0)
im = gmag > 2.2*np.mean(gmag)
m, n = im.shape
xs = np.arange(1, n-1)
ys = np.arange(1, m-1)
xpeaks = (im[ys][:,xs-1] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys][:,xs+1])
ypeaks = (im[ys-1][:,xs] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys+1][:,xs])
im = im[ys][:,xs].astype(bool) & (xpeaks | ypeaks)
plt.figure(0,(12,9))
plt.imshow(im, cmap='gray')
```

Out[353]:
```
<matplotlib.image.AxesImage at 0x5c963108>
```



In [348]:
```
image_file = 'Paolina.tiff'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
im = im + np.random.randn(480,512)*.05

im = scipy.ndimage.gaussian_filter(im, 2)

imx = scipy.signal.convolve2d(im,Sx,'same')
imy = scipy.signal.convolve2d(im,Sy,'same')
gmag = np.sqrt(imx**2.0 + imy**2.0)
im = gmag > 1.5*np.mean(gmag)
m, n = im.shape
xs = np.arange(1, n-1)
ys = np.arange(1, m-1)
xpeaks = (im[ys][:,xs-1] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys][:,xs+1])
ypeaks = (im[ys-1][:,xs] <= im[ys][:,xs]) & (im[ys][:,xs] > im[ys+1][:,xs])
im = im[ys][:,xs].astype(bool) & (xpeaks | ypeaks)

plt.figure(0,(12,9))
plt.imshow(im, cmap='gray')
```

Out[348]:
```
<matplotlib.image.AxesImage at 0x63bd91c8>
```



2e.

The pre-processing with blur vastly improves the result as it smooths the image and reduces the amount of salt and pepper noise in the output.

2f.

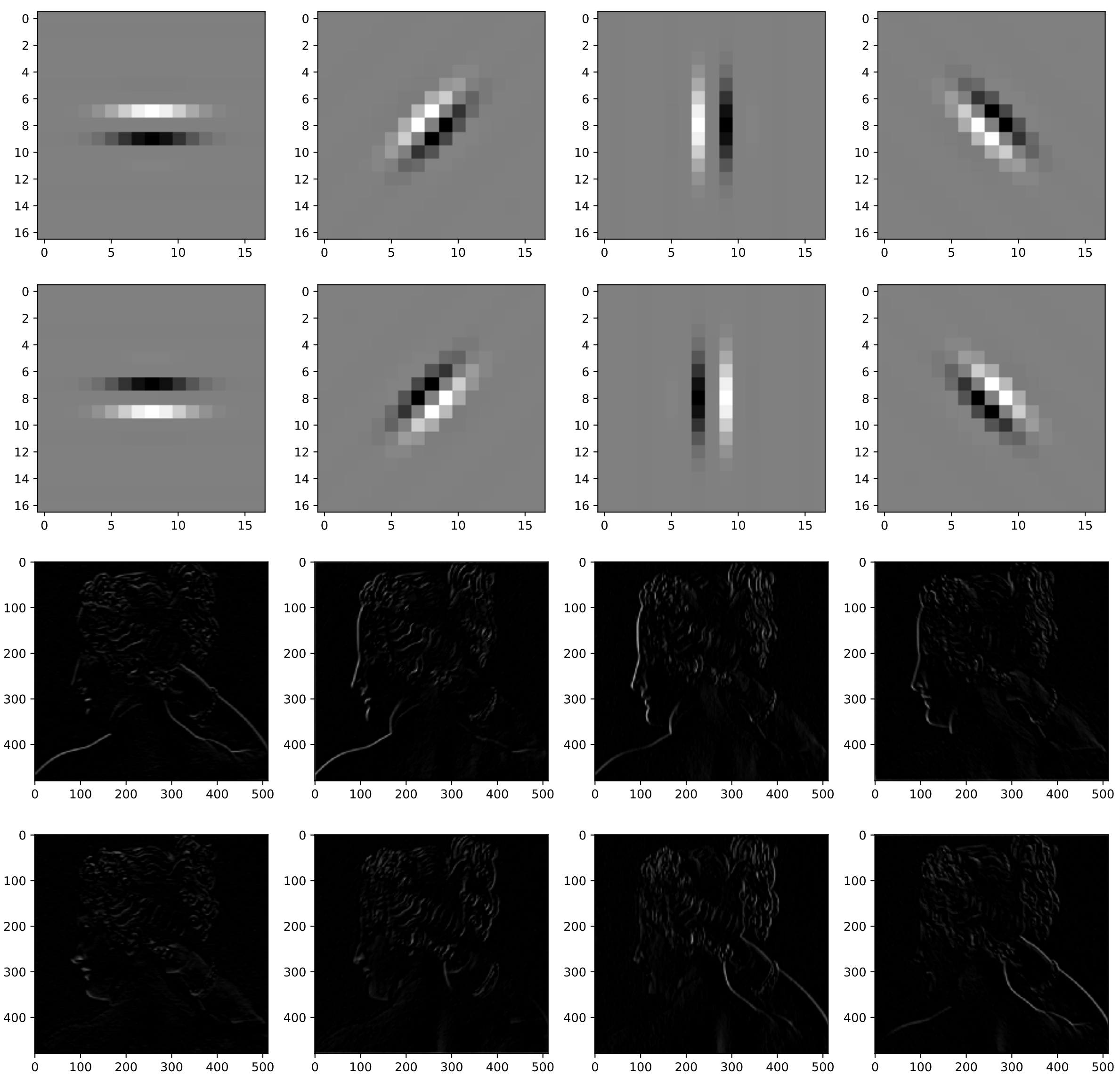
I expect this operator to find all the places where the 2nd derivative changes from positive to negative values because this is the same as finding the maximum of the first derivative. The role of composing a Gaussian with the Laplacian in this filter would be to smooth the image out because the Laplacian is sensitive to noise. The gaussian is the blurring filter so we use it to eliminate the noise. Without using it, we will have the salt and pepper noise corrupting the image so the edge detector will fail with lots of false positives. So, it's smoothing then detecting edges to increase the fidelity of edge detection

In [7]:
```
import numpy as np
def gaborfilter(theta, wavelength, phase, sigma, aspect, ksize=None):

        """
        GB = GABORFILTER(THETA, WAVELENGTH, PHASE, SIGMA, ASPECT, KSIZE)
        creates a Gabor filter GB with orientation THETA (in radians),
        wavelength WAVELENGTH (in pixels), phase offset PHASE (in radians),
        envelope standard deviation SIGMA, aspect ratio ASPECT, and dimensions
        KSIZE x KSIZE. KSIZE is an optional parameter, and if omitted default
        dimensions are selected.
         """

        if ksize is None:
                ksize = 8*sigma*aspect


        if type(ksize) == int or len(ksize) == 1:
                ksize = [ksize, ksize]


        xmax = np.floor(ksize[1]/2.)
        xmin = -xmax
        ymax = np.floor(ksize[0]/2.)
        ymin = -ymax

        xs, ys = np.meshgrid(np.arange(xmin,xmax+1), np.arange(ymax,ymin-1,-1
))

    # Your code here
        xprime = np.cos(theta)*xs + np.sin(theta)*ys
        yprime = -np.sin(theta)*xs + np.cos(theta)*ys


        G = np.sin(2*np.pi*yprime/wavelength + phase)*np.exp(-(xprime**2/(aspe
ct**2) + yprime**2)/(2*sigma**2))
        G = G-np.mean(G)
        G /= np.sqrt(((G**2).sum()))
        return G
```

In [44]:
```
image_file = 'Paolina.tiff'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
```

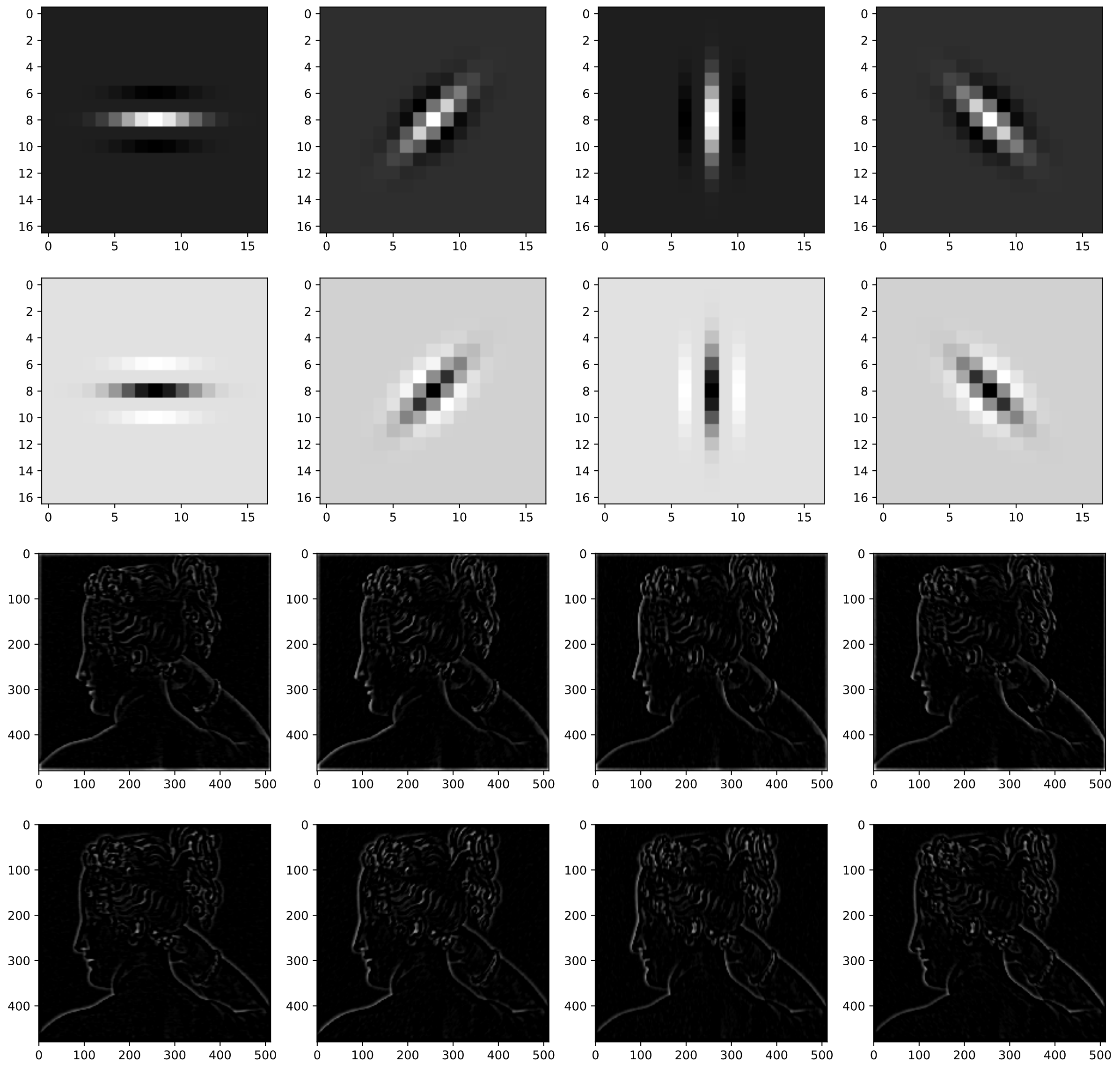
In [18]:
```
gb1 = gaborfilter(0,4, 0, 1, 2)
gb2 = gaborfilter(np.pi/4,4, 0, 1, 2)
gb3 = gaborfilter(np.pi/2,4, 0, 1, 2)
gb4 = gaborfilter(3*np.pi/4,4, 0, 1, 2)
gb5 = gaborfilter(np.pi,4, 0, 1, 2)
gb6 = gaborfilter(5*np.pi/4,4, 0, 1, 2)
gb7 = gaborfilter(3*np.pi/2,4, 0, 1, 2)
gb8 = gaborfilter(7*np.pi/4,4, 0, 1, 2)
f, axes = plt.subplots(4,4, figsize=(16,16))
axes[0][0].imshow(gb1, cmap='gray')
axes[0][1].imshow(gb2, cmap='gray')
axes[0][2].imshow(gb3, cmap='gray')
axes[0][3].imshow(gb4, cmap='gray')
axes[1][0].imshow(gb5, cmap='gray')
axes[1][1].imshow(gb6, cmap='gray')
axes[1][2].imshow(gb7, cmap='gray')
axes[1][3].imshow(gb8, cmap='gray')
axes[2][0].imshow(scipy.signal.convolve2d(im,gb1,'same'), vmin=0, vmax=1, cmap
='gray')
axes[2][1].imshow(scipy.signal.convolve2d(im,gb2,'same'), vmin=0, vmax=1, cmap
='gray')
axes[2][2].imshow(scipy.signal.convolve2d(im,gb3,'same'), vmin=0, vmax=1, cmap
='gray')
axes[2][3].imshow(scipy.signal.convolve2d(im,gb4,'same'), vmin=0, vmax=1, cmap
='gray')
axes[3][0].imshow(scipy.signal.convolve2d(im,gb5,'same'), vmin=0, vmax=1, cmap
='gray')
axes[3][1].imshow(scipy.signal.convolve2d(im,gb6,'same'), vmin=0, vmax=1, cmap
='gray')
axes[3][2].imshow(scipy.signal.convolve2d(im,gb7,'same'), vmin=0, vmax=1, cmap
='gray')
axes[3][3].imshow(scipy.signal.convolve2d(im,gb8,'same'), vmin=0, vmax=1, cmap
='gray')
```

```
Out[18]: <matplotlib.image.AxesImage at 0x187c2d48>
```



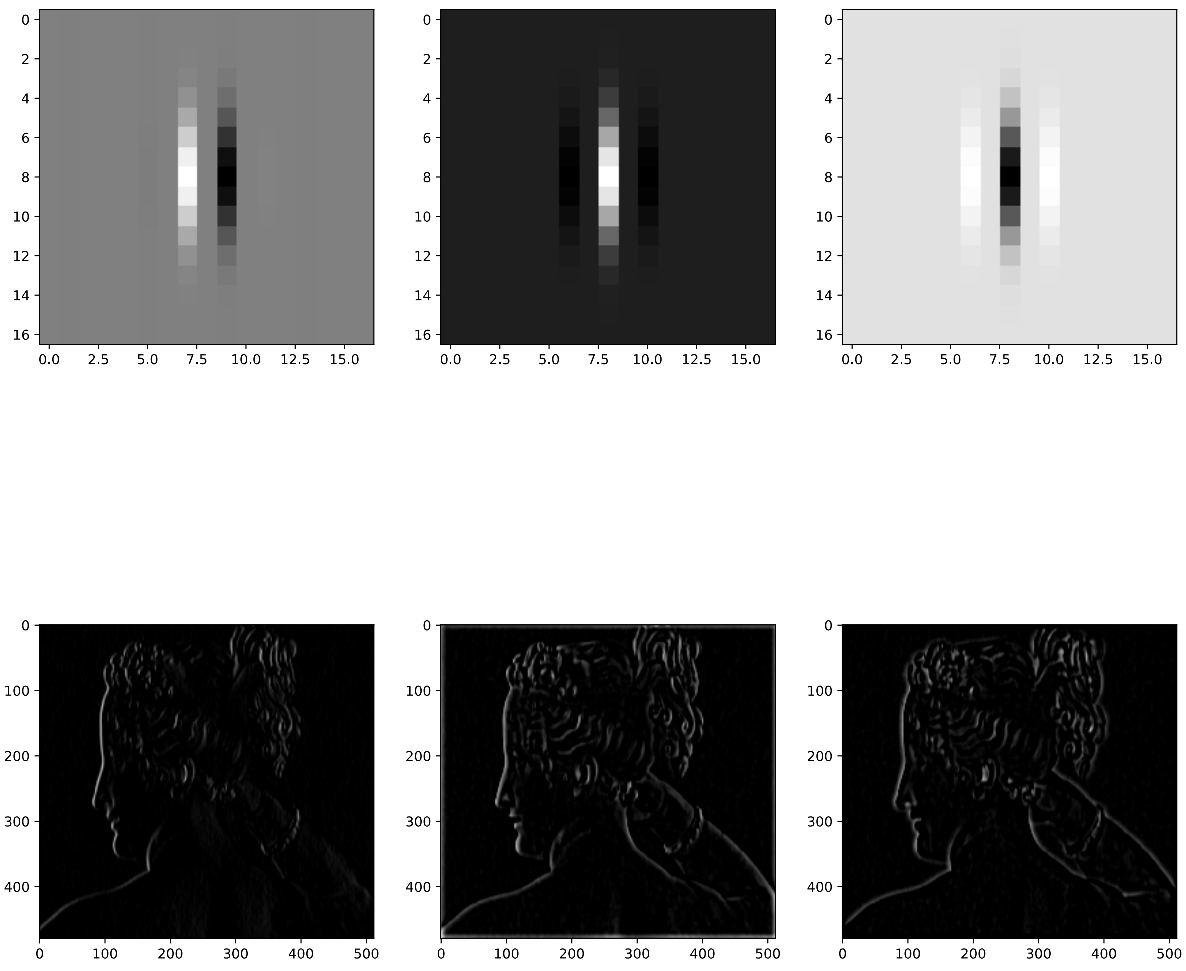
In [46]:
```
gb9 = gaborfilter(0,4, np.pi/2, 1, 2)
gb10 = gaborfilter(np.pi/4,4, np.pi/2, 1, 2)
gb11 = gaborfilter(np.pi/2,4, np.pi/2, 1, 2)
gb12 = gaborfilter(3*np.pi/4,4, np.pi/2, 1, 2)
gb13 = gaborfilter(0,4, 3*np.pi/2, 1, 2)
gb14 = gaborfilter(np.pi/4,4, 3*np.pi/2, 1, 2)
gb15 = gaborfilter(np.pi/2,4, 3*np.pi/2, 1, 2)
gb16 = gaborfilter(3*np.pi/4,4, 3*np.pi/2, 1, 2)
f, axes = plt.subplots(4,4, figsize=(16,16))
axes[0][0].imshow(gb9, cmap='gray')
axes[0][1].imshow(gb10, cmap='gray')
axes[0][2].imshow(gb11, cmap='gray')
axes[0][3].imshow(gb12, cmap='gray')
axes[1][0].imshow(gb13, cmap='gray')
axes[1][1].imshow(gb14, cmap='gray')
axes[1][2].imshow(gb15, cmap='gray')
axes[1][3].imshow(gb16, cmap='gray')
axes[2][0].imshow(scipy.signal.convolve2d(im,gb9,'same'), vmin=0, vmax=1, cmap
='gray')
axes[2][1].imshow(scipy.signal.convolve2d(im,gb10,'same'), vmin=0, vmax=1, cma
p='gray')
axes[2][2].imshow(scipy.signal.convolve2d(im,gb11,'same'), vmin=0, vmax=1, cma
p='gray')
axes[2][3].imshow(scipy.signal.convolve2d(im,gb12,'same'), vmin=0, vmax=1, cma
p='gray')
axes[3][0].imshow(scipy.signal.convolve2d(im,gb13,'same'), vmin=0, vmax=1, cma
p='gray')
axes[3][1].imshow(scipy.signal.convolve2d(im,gb14,'same'), vmin=0, vmax=1, cma
p='gray')
axes[3][2].imshow(scipy.signal.convolve2d(im,gb15,'same'), vmin=0, vmax=1, cma
p='gray')
axes[3][3].imshow(scipy.signal.convolve2d(im,gb16,'same'), vmin=0, vmax=1, cma
p='gray')
```

```
Out[46]: <matplotlib.image.AxesImage at 0x42b68188>
```



```
In [39]: edge = scipy.signal.convolve2d(im,gb3,'same')
         bright = scipy.signal.convolve2d(im,gb11,'same')
         dark = scipy.signal.convolve2d(im,gb15,'same')
         f, axes = plt.subplots(2,3,figsize=(15,15))
         axes[0][0].imshow(gb3, cmap='gray')
         axes[0][1].imshow(gb11, cmap='gray')
         axes[0][2].imshow(gb15, cmap='gray')
         axes[1][0].imshow(edge, vmin=0, vmax=1, cmap='gray')
         axes[1][1].imshow(bright, vmin=0, vmax=1, cmap='gray')
         axes[1][2].imshow(dark, vmin=0, vmax=1, cmap='gray')
```

```
Out[39]: <matplotlib.image.AxesImage at 0x3a1072c8>
```

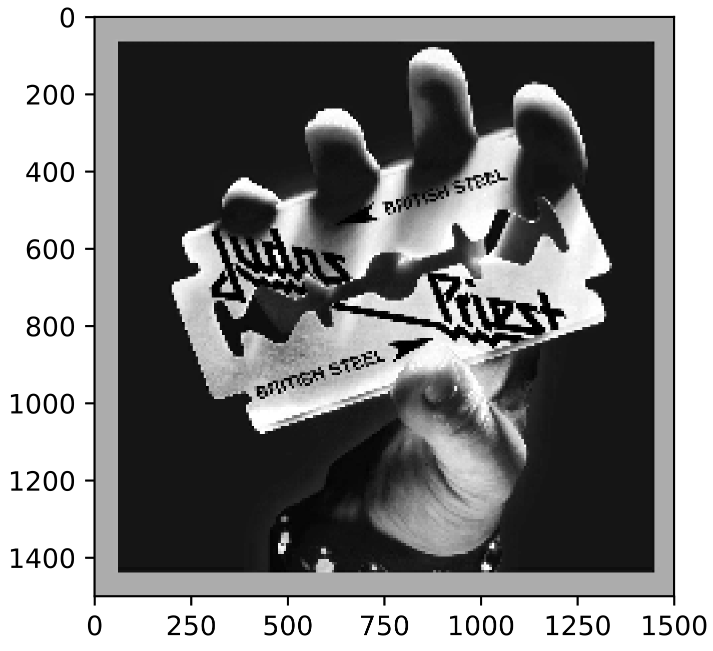


3b.

The vertical edge enhancer highlights edges that are straight up and down which is especially apparent on the profile of Paolina's face (forehead, nose, and chin). The bright line enhancer adds an illuminating effect on the inside edges of Paolina because those are the areas where it goes from the brightness of her skin to the darkness of the background. The bright line enhancer also highlights the bright stripes in her hair. The dark line enhancer does the opposite as it adds an illuminating effect on the outside edges of Paolina because these are the places where it goes from the darkness of the background to the brightness of her skin. The dark line enhancer also highlights the dark areas in the folds of Paolina's hair.

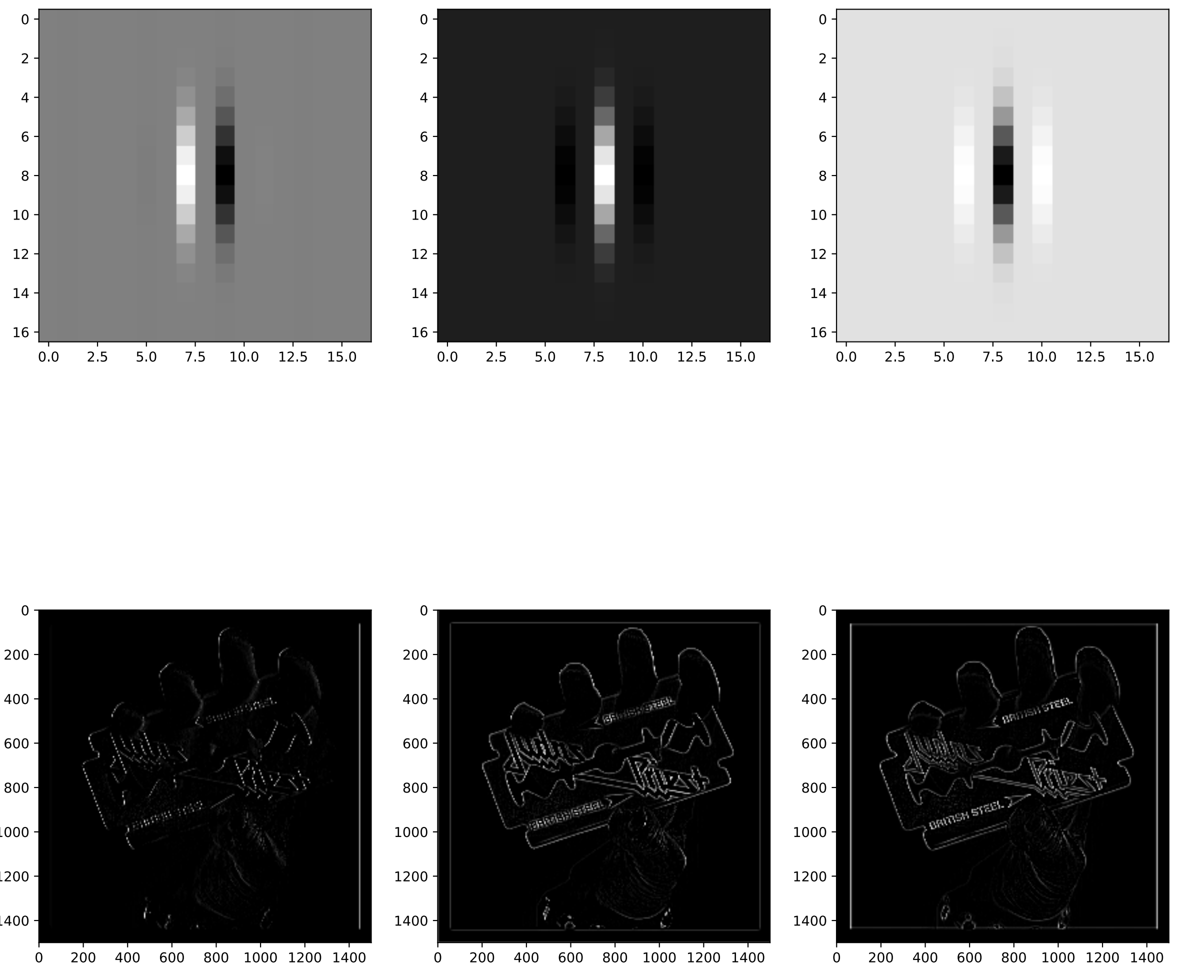
In [40]:
```
image_file = 'British Steel.tiff'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
im = im[:,:,0]
plt.imshow(im, cmap='gray')
```

Out[40]:
```
<matplotlib.image.AxesImage at 0x3a6dd308>
```



In [41]:
```
edge = scipy.signal.convolve2d(im,gb3,'same')
bright = scipy.signal.convolve2d(im,gb11,'same')
dark = scipy.signal.convolve2d(im,gb15,'same')
f, axes = plt.subplots(2,3,figsize=(15,15))
axes[0][0].imshow(gb3, cmap='gray')
axes[0][1].imshow(gb11, cmap='gray')
axes[0][2].imshow(gb15, cmap='gray')
axes[1][0].imshow(edge, vmin=0, vmax=1, cmap='gray')
axes[1][1].imshow(bright, vmin=0, vmax=1, cmap='gray')
axes[1][2].imshow(dark, vmin=0, vmax=1, cmap='gray')
```

Out[41]:
```
<matplotlib.image.AxesImage at 0x3a9a0808>
```

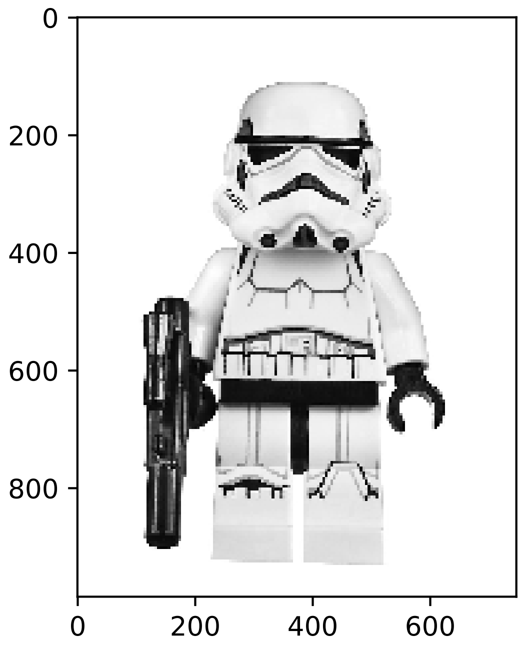


3b. continued:

In this new loaded image (the album cover of Judas Priest's British Steel), we can also see the difference between the three types of enhancers. The vertical edge enhanceres highlights those more vertical edges in the image which are the outside of the razorblade and the inside cutouts of the words. As for the line enhancers, the difference we see here is more due to how they are detecting edges which isn't really their focus but it is still cool. The bright line enhancer is enhancing the outline of the word cutouts in the razor blade because those are the bright areas on the silver against the black words themselves while the dark line enhancer is enhancing the inside bits of the word cutout because those are the dark areas against the brighter surrounds.

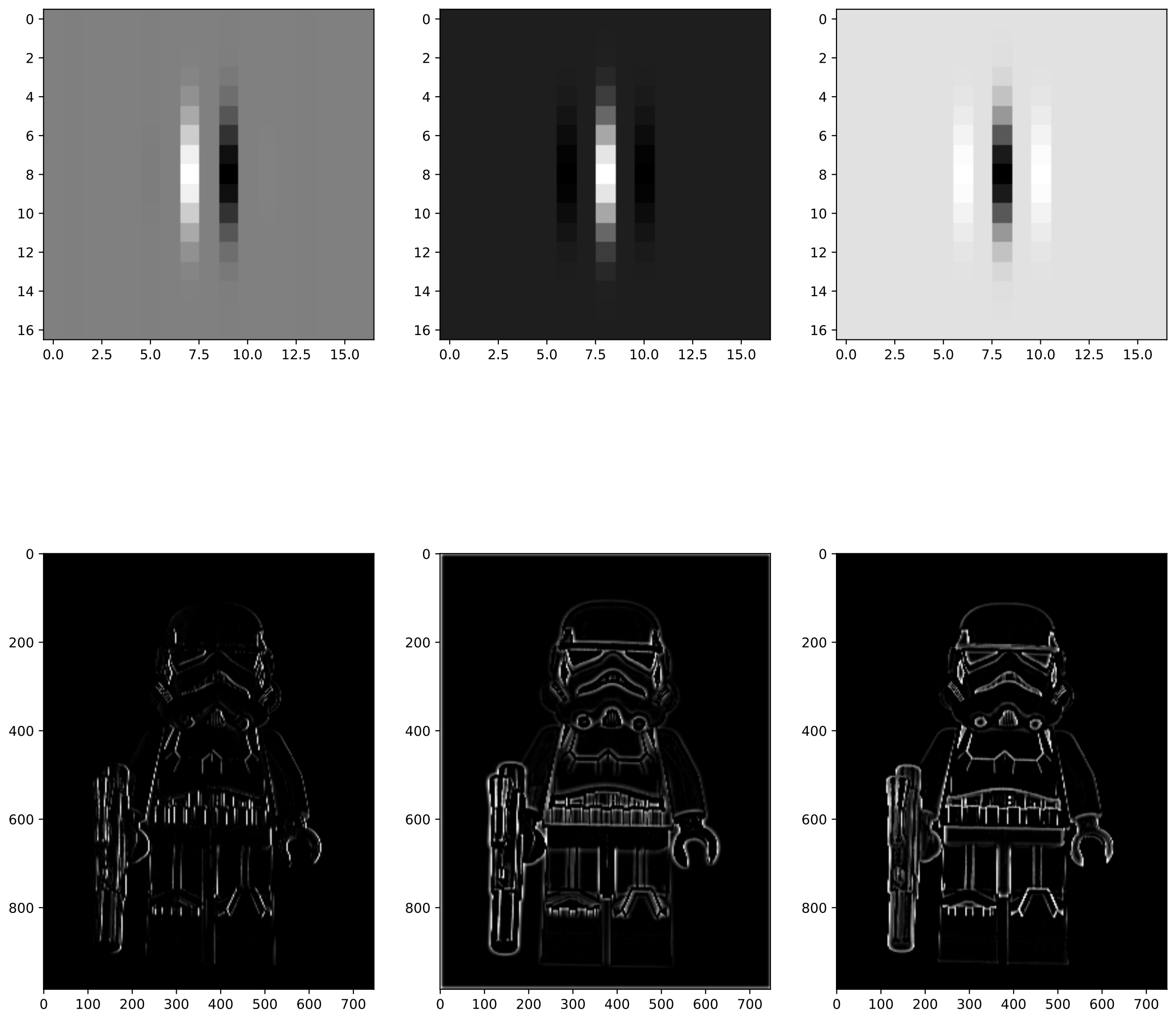
In [42]:
```
image_file = 'Stormtrooper2015.jpg'
im = np.array(Image.open(image_file))
im = im.astype('float')/255
im = im[:,:,0]
plt.imshow(im, cmap='gray')
```

Out[42]:
```
<matplotlib.image.AxesImage at 0x3a8adb48>
```



In [43]:
```
edge = scipy.signal.convolve2d(im,gb3,'same')
bright = scipy.signal.convolve2d(im,gb11,'same')
dark = scipy.signal.convolve2d(im,gb15,'same')
f, axes = plt.subplots(2,3,figsize=(15,15))
axes[0][0].imshow(gb3, cmap='gray')
axes[0][1].imshow(gb11, cmap='gray')
axes[0][2].imshow(gb15, cmap='gray')
axes[1][0].imshow(edge, vmin=0, vmax=1, cmap='gray')
axes[1][1].imshow(bright, vmin=0, vmax=1, cmap='gray')
axes[1][2].imshow(dark, vmin=0, vmax=1, cmap='gray')
```

Out[43]:
```
<matplotlib.image.AxesImage at 0x40474248>
```



3b. continued further:

In this image of the lego stormtrooper, we get a good look at how the vertical enhancers work because it has lots of vertical lines. The edge enhancer enhances the specific places on the stormtrooper that are edges which go from bright to dark. Meanwhile, the bright line enhancer enhances the white areas on his armour that are up against dark areas. The dark line enhancer does the opposite and highlights the dark areas that are up against bright areas. We see that the dark line enhancer fills in the dark areas of the stormtrooper while the bright line enhancer illuminates the outlines of those dark areas, but that is more of an issue with the filter because it is picking up on edges. The stormtrooper's left kneecap shows a very cool distinction between the bright and dark line enhancers because the dark line enhancer enhances the dark line on the armour super well while that same line is super dark in the bright line enhancer. Since the stormtrooper has so many little dark lines like that, the dark line enhancer works really well here

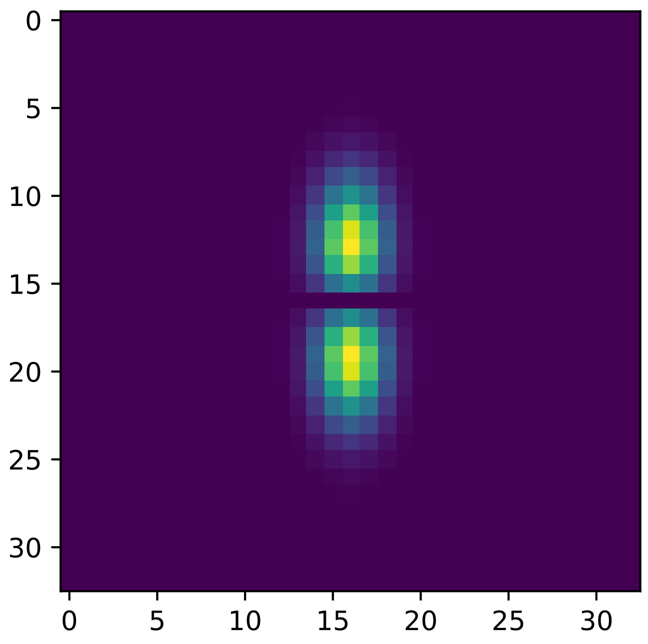
3c.

In our brain, we want to be able to differentiate between the different orientation so if we did it all at once, we couldn't tell what angle the edges are. Multiple edge detectors give a richer encoding of the scene. Since you can tell more directions, your judgement will be more precise. In v1 of the brain, you have simple, complex, and hypercomplex cells. So, we use gabor filter to approximate the receptor field of the simple cell.

Having a single edge map is poor form because it cannot possibly cover all angles, whereas the Gabor can build up the complete image by matching the parts it is looking for. The computational/perceptual advantages of the latter representation are that it is much more accurate.

```
In [239]: plt.imshow(abs(fft.fftshift(fft.fft2(gb1))))
```

```
Out[239]: <matplotlib.image.AxesImage at 0x25fdf048>
```



3d.

The transform of a Gaussian is another Gaussian which looks like a hump. The transform of a sin function with a single frequency is the delta function so it's a 1 at one point, but it's symmetric so there is a 1 at the positive and 1 at the negative. The delta function is the identity in the frequency domain. So since we have a pair of delta functions, we get two copies of the gaussian which are symmetric. Therefore, we should see two bumps in the frequency domain which is what we see in the heat map above of the resulting fourier transform of the gabor filter.