EEE482 - Computational Neuroscience

Homework Assignment - 2

Berkan Ozdamar

21602353

Question 1

In this question, we are given a c2p3.mat file which contains the responses of a cat LGN cell to two dimensional images. Stim contains 32767 images with 16x16 dimensions. And the counts is a vector which contains number of spikes in each 15.6 ms bin.

Part a

In this part, we are asked to calculate Spike Triggered Average(STA) of the stimulus for each of the 10 time steps before each spike. Then we will display these each of these STAs with grayscale colormap then discuss what type of spatio-temporal stimulus the LGN cell is selective for.

Scipy.io will be used for loading the matlab file c2p3.mat. Numpy and matplotlib will be used for calculations and plotting.

[1]:
```
import numpy as np
import scipy.io
import matplotlib.pyplot as mpl
```

Data is loaded as c2p3 to python and divided into 2 arrays which are stim and counts. Stim contains 32767 images with 16x16 dimensions. Counts contains the spikes in the neuron for each time bin. Dimension of stim is originally [16, 16, 32767] but to make it easier I have taken the transpose of it to make dimensions [32767, 16, 16]. Counts' dimension is [32767, 1].

[2]:
```
c2p3 = scipy.io.loadmat('c2p3.mat')
stim = c2p3['stim'].T
counts = c2p3['counts']
print(np.shape(stim))
print(np.shape((counts)))
```

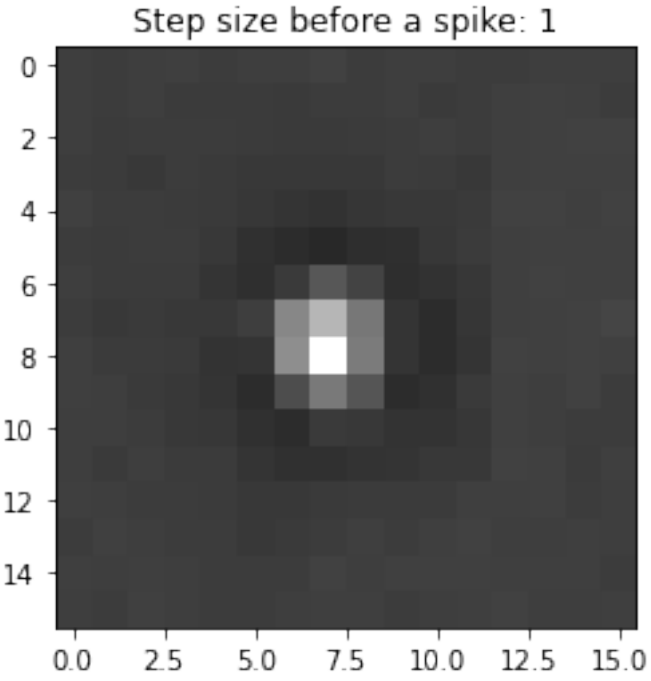
```
(32767, 16, 16)
(32767, 1)
```

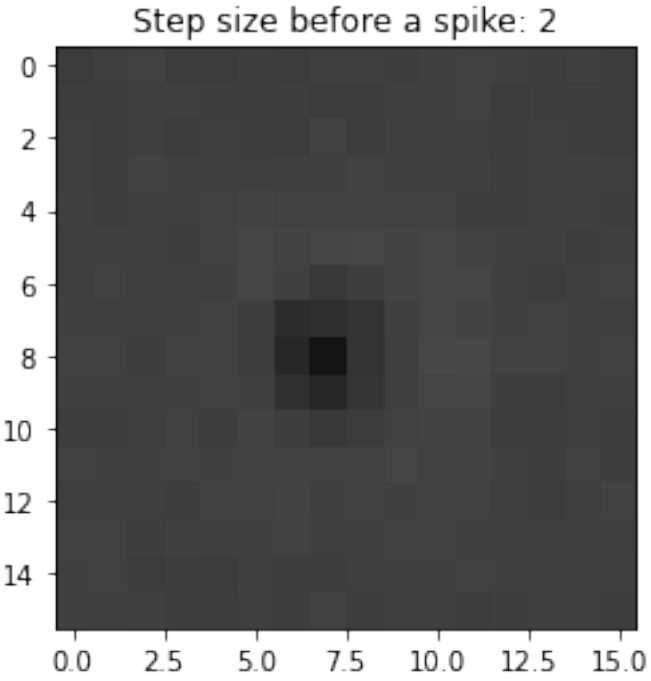
STA function below calculates the bins prior to each spike with time bins being t = -1 to t = -10 steps before a spike. Then create an array of size [10, 16, 16] which 10 represents the each time bin before a spike with 16x16 dimensions. For calculating each time bin, we will sum the corresponding images with corresponding weights. Here, weights represents the spike occurrence which can be accessed from counts array. So, for calculation of each time bin before a spike, corresponding weight for that time bin and the image for each image will be multiplied and will be summed over. Function that calculates STA is given below:

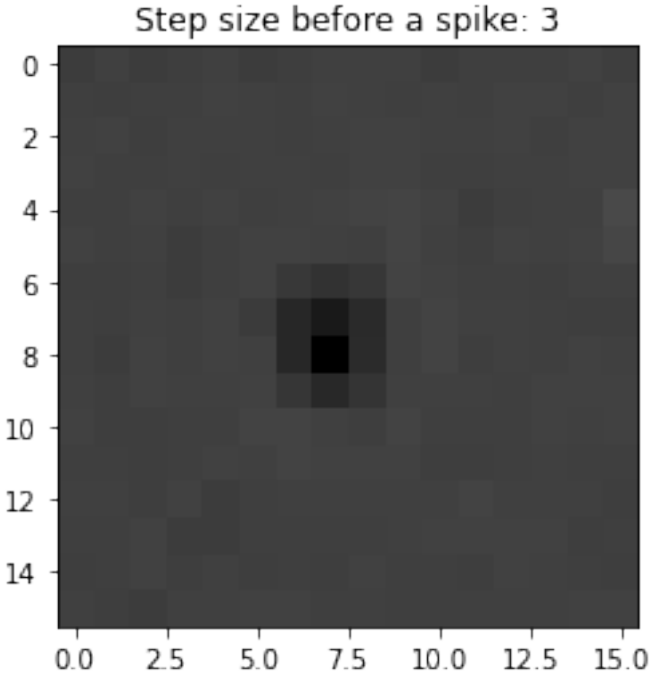
[3]:
```
#Part A

def STA(step, stim, counts):
    total_spike = 0
    result = np.zeros((step, 16, 16))
    for i in range(len(stim[:,0,0])):
        for j in range(step):
            if i > j and counts[i] >= 0:
                result[j,:,:] += stim[i-(j+1),:,:] * counts[i]
        total_spike += counts[i]
    #Normalization
    result[:,:,:] = result[:,:,:] / total_spike
    return result
```

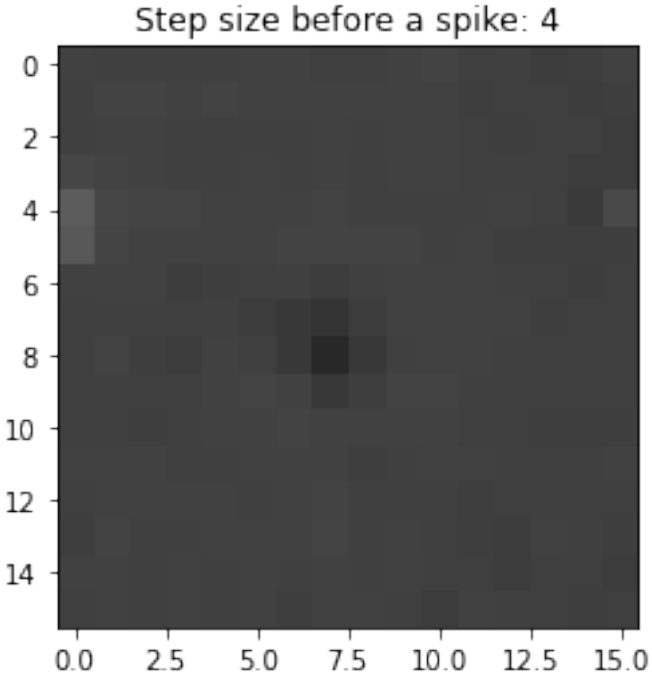
Since we want the time bin from t = -1 to t = 10, steps variable will be set to 10 and we will display all the gray-scale plots for each time bin before a spike.

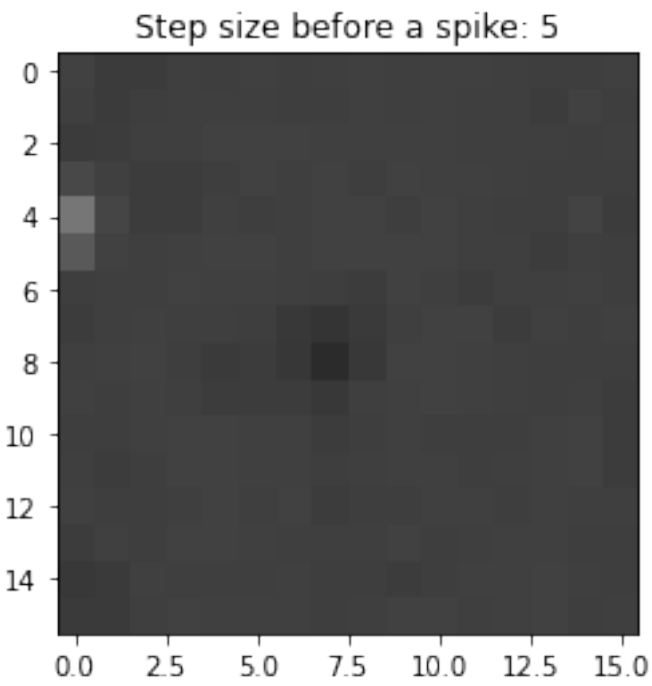
[4]:
```
STA_image = STA(10,stim,counts)
figure = 0
for i in range(np.shape(STA_image)[0]):
    figure += 1
    mpl.figure(figure)
    mpl.title("Step size before a spike: " +str(i+1) )
    mpl.imshow(STA_image[i,:,:], cmap='gray', vmin=np.min(STA_image), vmax=np.
 ↪max(STA_image))
```

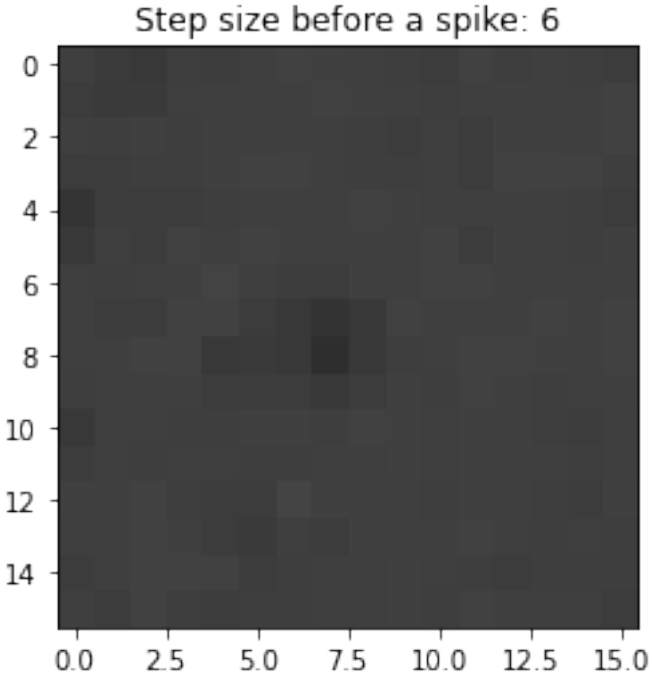


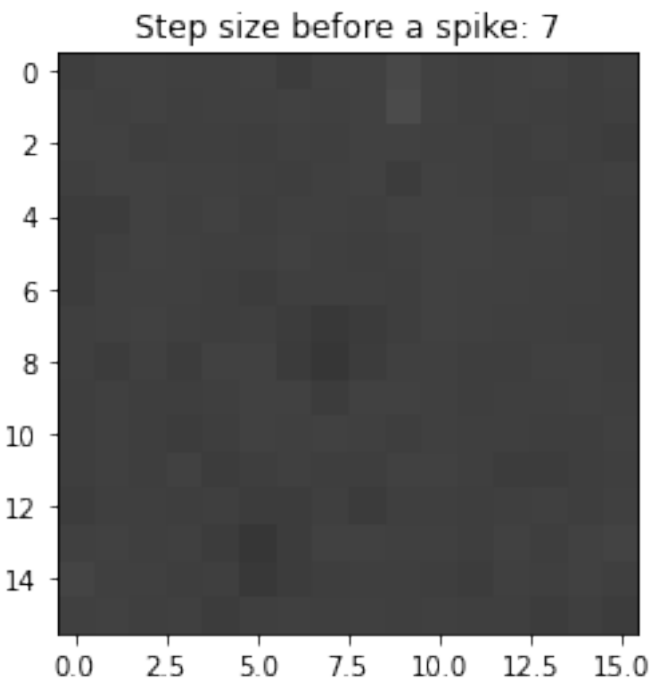


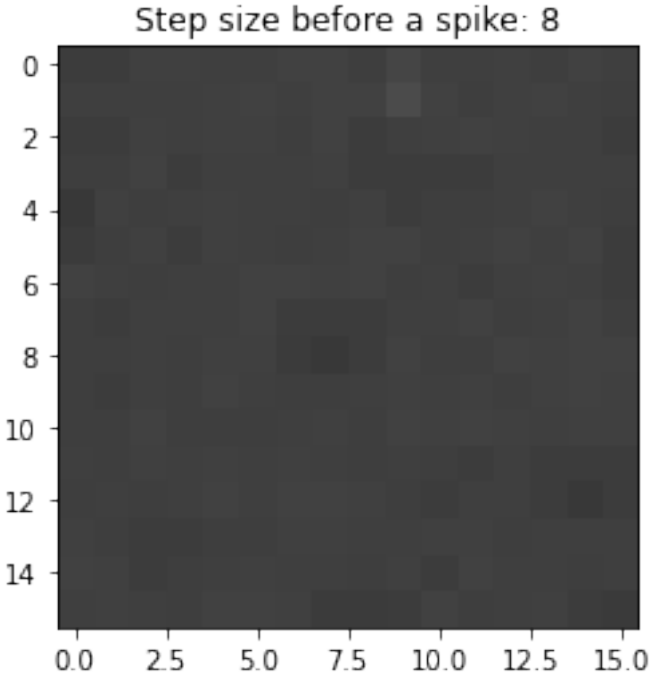

Step size before a spike: 3
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0

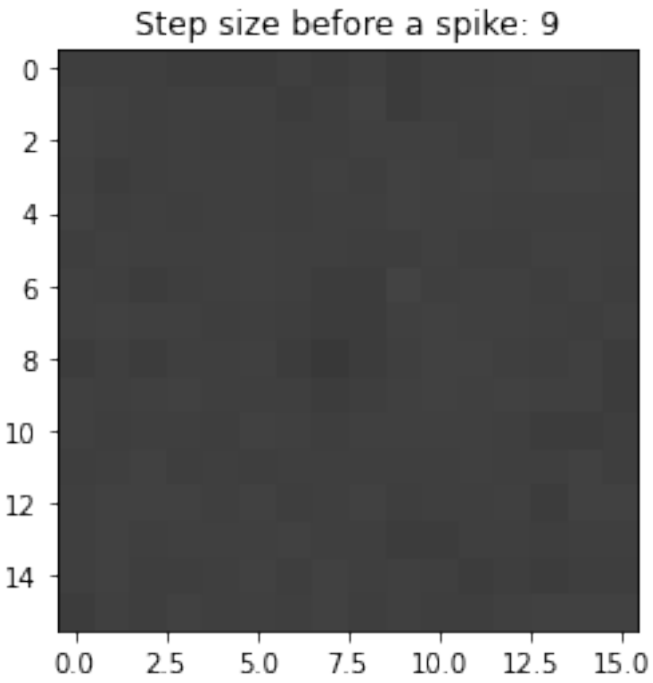

Step size before a spike: 4
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0

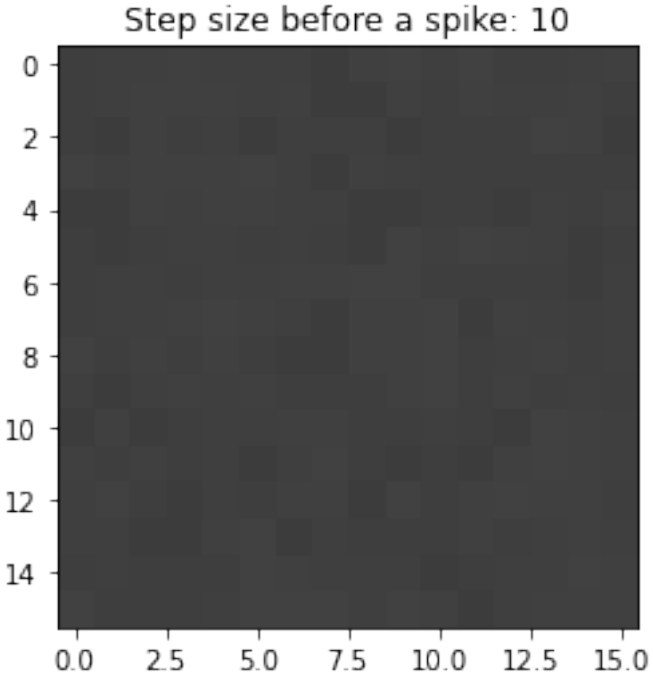

Step size before a spike: 5
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0


Step size before a spike: 6
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0


Step size before a spike: 7
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0


Step size before a spike: 8
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0





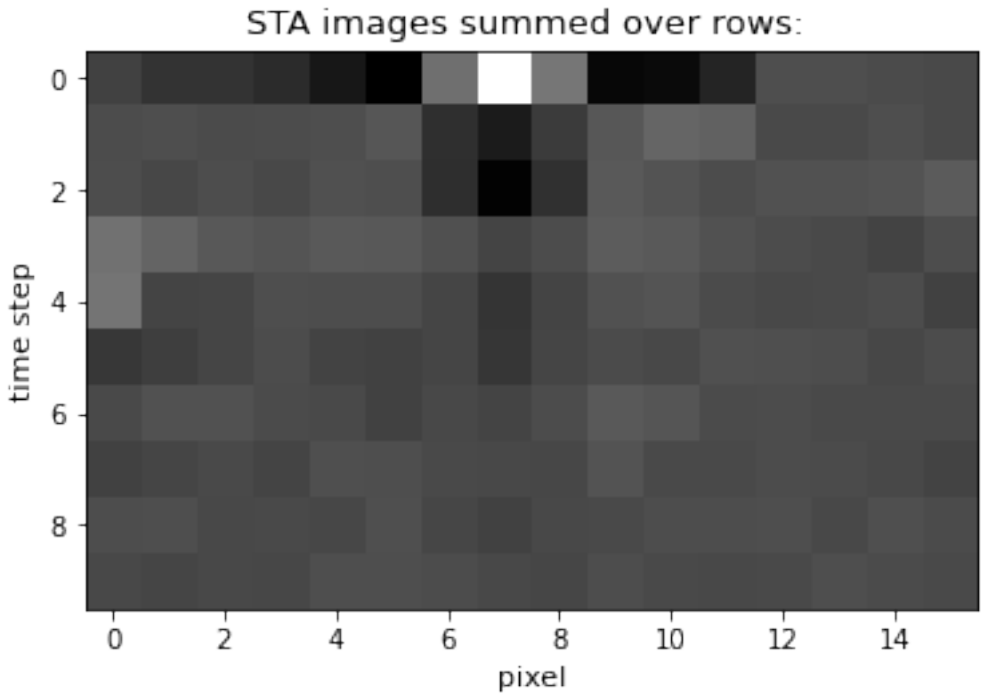
As it can be seen from the images, in time step t = -1, there is a flash in the middle which represents the sudden increase of neuron's response. After time t = -1, we see a darkening in the middle until it becomes normal when it comes near to time step t = -10 before a spike. That means that near -10 time steps before a spike, neuron is almost idle. When it get closer to t = -5, the darkening may represent the falling edge and it seems that LGN neuron requires the darkening before the sudden flash in t = -1. The LGN neuron is selective for a sudden flash in the middle preceded by a time-varying decrease.

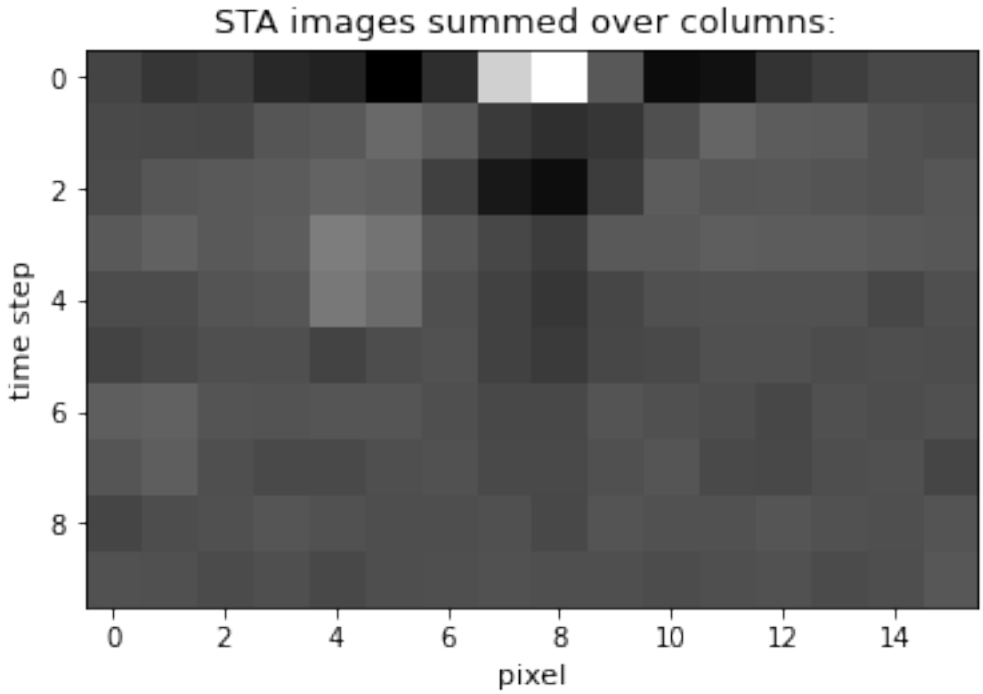
Part b

In this part, to see how STA reacts to spatial direction, we sum the calculated STA image in a chosen spatial direction. For that, first we will sum the STA images over direction of rows then images will be summed over direction of columns. To do that, numpy's sum fucntion with determined axis is used. Since gathered STA images are in a shape of [10, 16, 16], 16x16 represents the rows and columns of image respectively. So for summation over rows, axis=1 will be chosen and for summation over columns, axis=2 will be chosen.

[5]:
```
#Part B

row_sum = np.sum(STA_image, axis=1)
col_sum = np.sum(STA_image, axis=2)

figure += 1
mpl.figure(figure)
mpl.title("STA images summed over rows: ", fontsize=13)
mpl.xlabel('pixel', fontsize=11)
mpl.ylabel('time step', fontsize=11)
mpl.imshow(row_sum, cmap='gray')
mpl.show(block=False)

figure += 1
mpl.figure(figure)
mpl.title("STA images summed over columns: ", fontsize=13)
mpl.xlabel('pixel', fontsize=11)
mpl.ylabel('time step', fontsize=11)
mpl.imshow(col_sum, cmap='gray')
mpl.show(block=False)
```

The results shows row_sum and col_sum which are both 10x16 matrices.





As can be seen from both figures, the neuron's response is brighter in the middle pixels(7-9) preceded by a darker response in pixels 4-7. Finally from 7th pixel to 10th, there is almost no response. Since both spatial direction summations shows very similar results, we can say that the LGN neuron's response is not sensitive to spatial direction. Moreover, the results from these figures support the argument made about how the neuron reacts according to each time step since they both show almost identical responses.

Part c

Frobenius inner product is like vector inner product but it is applied to two matrices. In the same manner with vector inner product, from to matrices points that represent the same coordinates will be multiplied and they will all be summed. Frobenius product can also be shown as:

$$\sum_{i,j} A_{ij}.B_{ij}$$

For projecting vector onto another vector, vector inner product is used. In same manner, for projecting a matrix onto another matrix, we will use Frobenius inner product. Stimulus will be projected onto STA images prior to time steps before a spike.

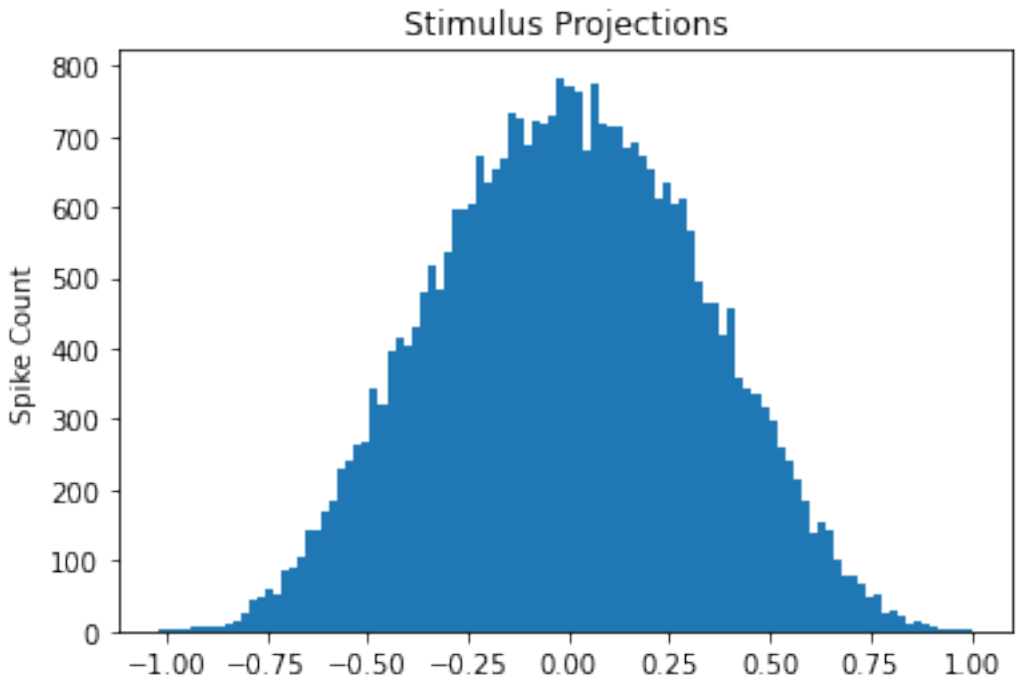
In the code, function takes allSpikes boolean. This is used since we require 2 projections in this part. One is including all the spikes and the other one is only non-zero spikes. So instead of creating two different functions, the condition is checked with a single boolean. Also, since maximum value is required to be setted to 1, i have normalized all the values prior to max value. The code for frobenius inner product is given below:

[6]:
```
#Part C

def frobenius(STA, stim, counts, allSpikes):
    if allSpikes == True:
        result = np.zeros(len(counts))
        normalizer = 0
        for i in range(len(counts)):
```

```
            result[i] = np.sum(np.multiply(STA[0,:,:],stim[i,:,:]))
            if result[i] > normalizer:
                normalizer = result[i]
        result[:] = result[:] / normalizer
    else:
        result = []
        normalizer = 0
        for i in range(len(counts)):
            if counts[i] != 0:
                result.append(np.sum(np.multiply(STA[0,:,:],stim[i,:,:])))
        normalizer = max(result)
        result[:] = result[:] / normalizer
    return result
```

The code below is for plotting the histograms that represent the projection of stimulus onto STA images with all spikes included.

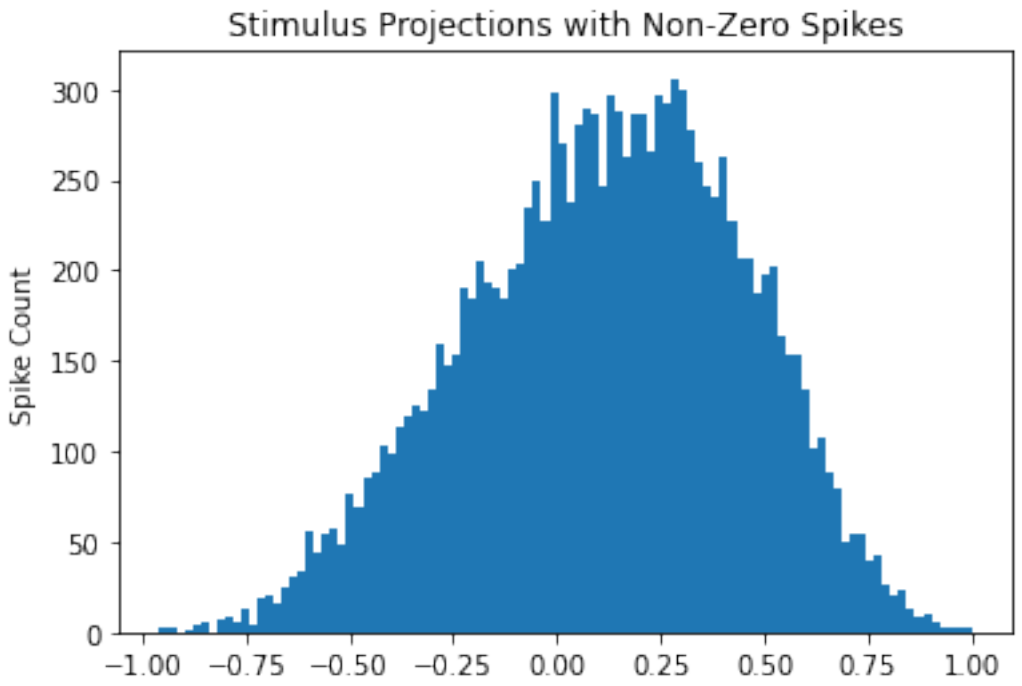
[7]:
```
histo_frobenius = frobenius(STA_image, stim, counts, True)
figure += 1
mpl.figure(figure)
mpl.title("Stimulus Projections")
mpl.ylabel('Spike Count')
mpl.hist(histo_frobenius, bins=100)
mpl.show()
```



For the stimulus projection with only non-zero spikes, the histogram plot code is shown below:

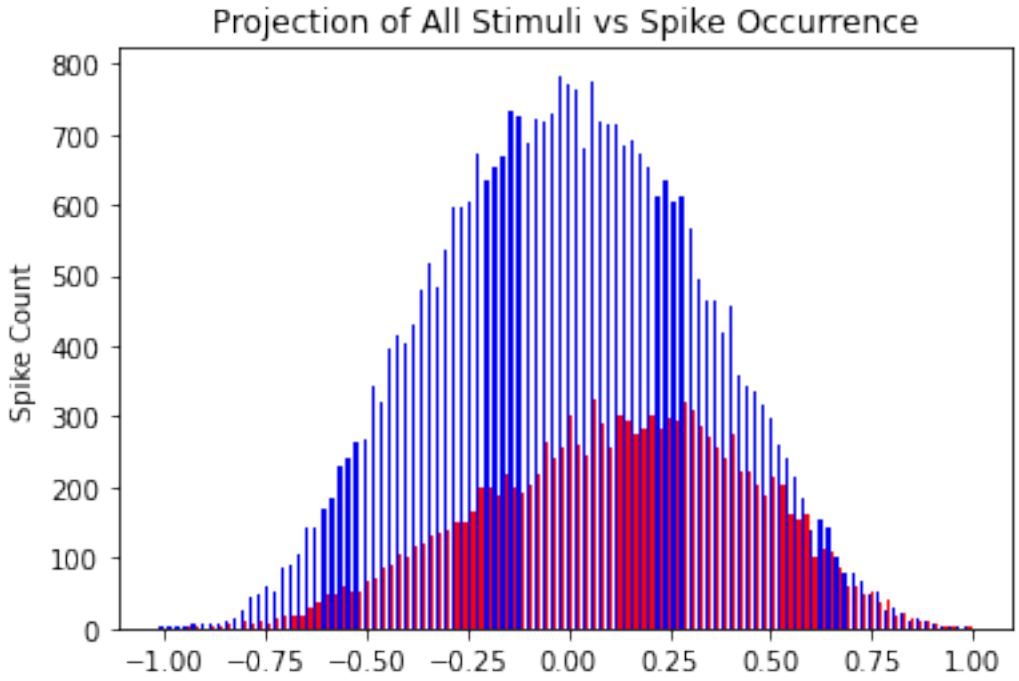
[8]:
```
histo_frobenius_nonzero_spikes = frobenius(STA_image, stim, counts, False)
figure += 1
mpl.figure(figure)
mpl.title("Stimulus Projections with Non-Zero Spikes")
mpl.hist(histo_frobenius_nonzero_spikes, bins=100)
mpl.ylabel('Spike Count')
```

```
mpl.show()
```



For representing these two projections together, which are containing all spikes vs only non-zero spikes, they are plotted in the same histogram figure.

[9]:
```
figure += 1
mpl.figure(figure)
mpl.
 ↪hist([histo_frobenius,histo_frobenius_nonzero_spikes],bins=100,color=['blue','red'])
mpl.title("Projection of All Stimuli vs Spike Occurrence")
mpl.ylabel('Spike Count')
```



In the figure, blue histogram represents the projection including all the spikes. Contrary, red histogram represents the projection with only non-zero spikes. It can be seen that the shape of the projection with all spikes is a Gaussian distribution with mean around zero. It means that most of the projection values are close to zero. But red histogram which means the projection data with only non-zero spikes, is also like a Gaussian but with a mean around 0.25. Thus, we can see that only excluding zero spikes, the shift of projection is significant. It means STA discriminates spike eliciting stimulus.

Question 2

In this question, it is explained that LGN and V1 neuron cells are responsible for image recognition and image processing. It is given that the LGN neuron cells use Difference of Gaussians(DOG) for their receptive fields and V1 neuron cells use Gabor as their receptive fields. We are asked to construct these Difference of Gaussians and Gabor models and apply them on a given image. Also we are asked to apply edge detection for the same image.

Part a

In this part, we are asked to construct an on-center Difference of Gaussians center surround receptive field centered at 0. Difference of Gaussians is:

$$D(x,y) = \frac{1}{2\pi\sigma_c^2} e^{-(x^2+y^2)/2\sigma_c^2} - \frac{1}{2\pi\sigma_s^2} e^{-(x^2+y^2)/2\sigma_s^2}$$

where σ_c represents the central Gaussian standart deviation and σ_s represents surround Gaussian standart deviation. Python code for Difference of Gaussians is given below:

[1]:
```
import numpy as np
import math
import scipy.io
import matplotlib.pyplot as mpl
from mpl_toolkits import mplot3d
from scipy import signal
```

[2]:
```
#Part A

def DOG(sigma_c, sigma_s, x, y):
    gaussian_c = (1/(2*math.pi*(sigma_c**2)))*math.exp(-(x**2+y**2)/
 ↪(2*(sigma_c**2)))
    gaussian_s = (1/(2*math.pi*(sigma_s**2)))*math.exp(-(x**2+y**2)/
 ↪(2*(sigma_s**2)))
    return gaussian_c - gaussian_s
```

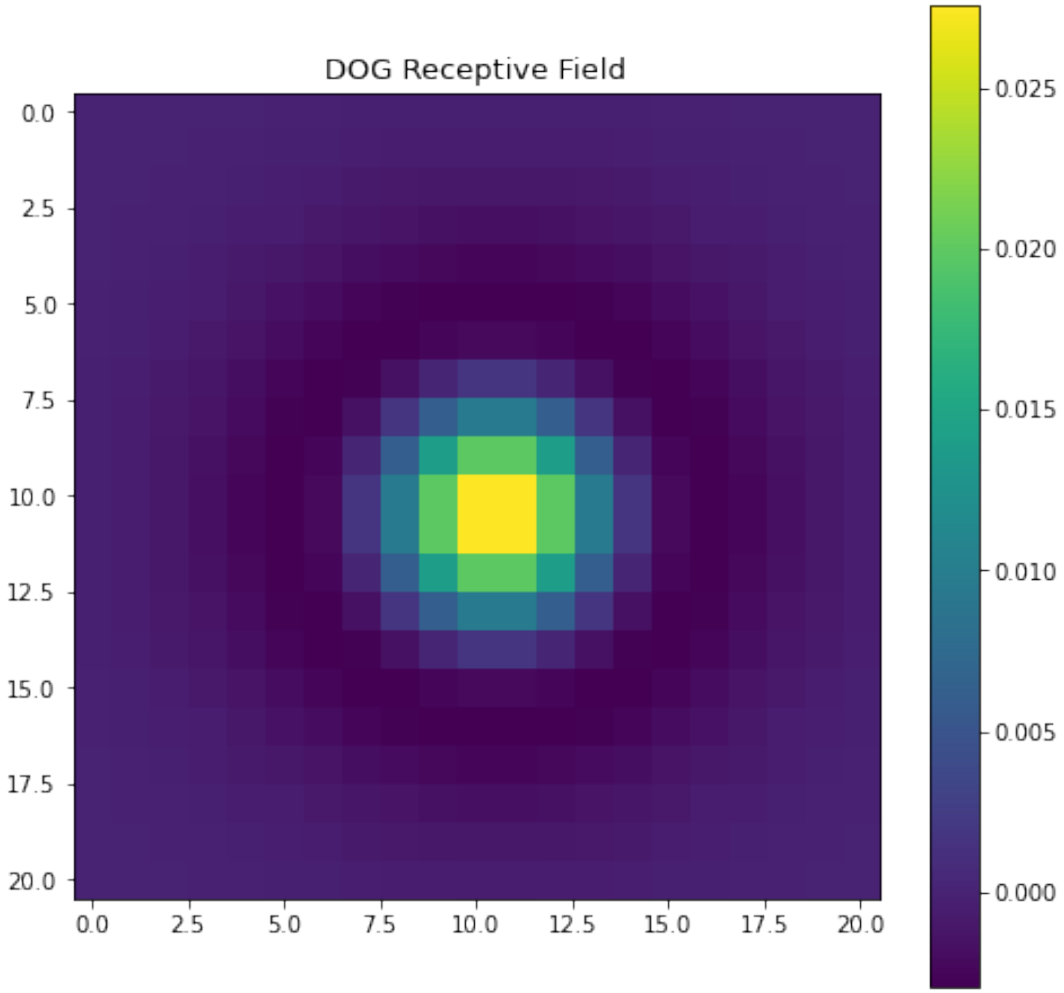
Now, we will construct Difference of Gaussians center-surround receptive fields with a sample size of 21x21 matrix. In the DOG_receptive_field, resultRowSize and resultColSize are the row and column size of the sample matrix which is 21x21 in our case. What the function does is, it constructs the 21x21 matrix in the center so the with center standart deviation σ_c and surround standart deviation σ_s and takes the Gaussian Difference of them for every row,column pair.

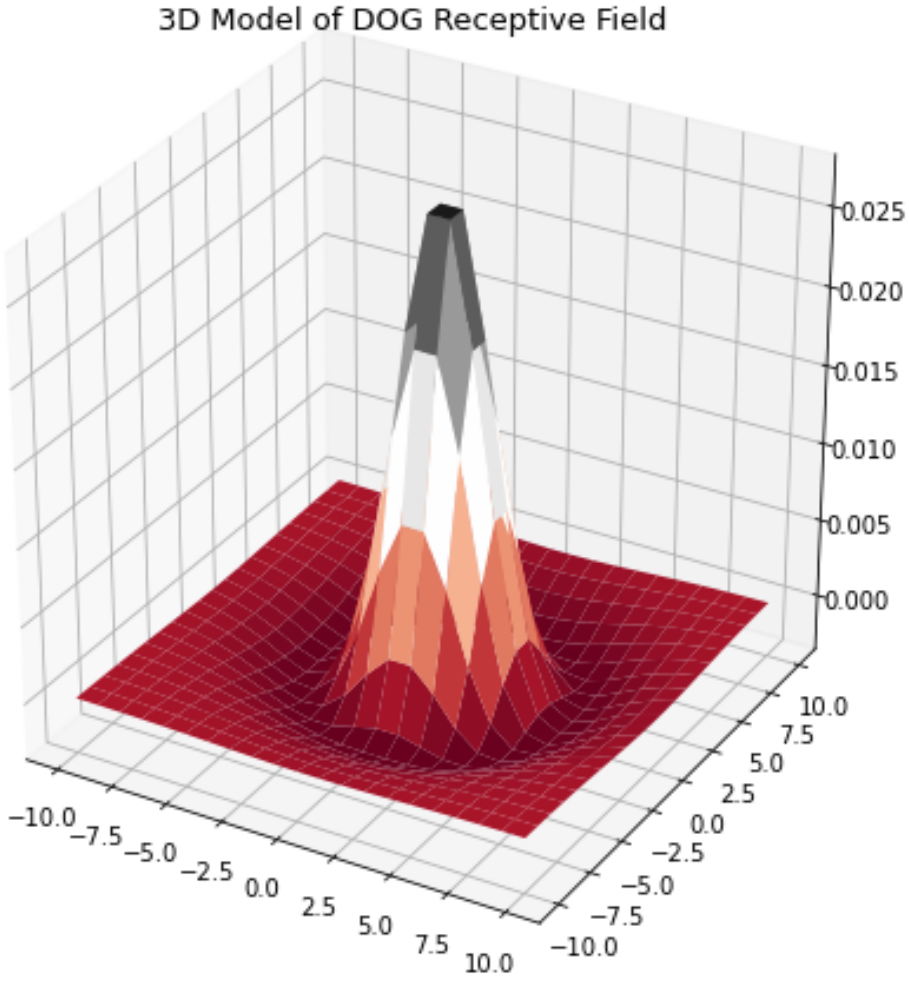
[3]:
```
def DOG_receptive_field(sigma_c, sigma_s, resultRowSize, resultColSize):
    result = np.zeros((resultRowSize, resultColSize))
    for i in range(resultRowSize):
        for j in range(resultColSize):
            result[i][j] = DOG(sigma_c, sigma_s, i-(resultRowSize/2),␣
 ↪j-(resultRowSize/2))
    return result
```

The σ_c is given 2 and σ_s is given as 4. So the 2D and 3D view of the DOG_receptive_field is given below:

[4]:
```
Dog_receptive_field = DOG_receptive_field(2,4,21,21)

# Plot for DOG Receptive Field
figure = 0
mpl.figure(figure)
mpl.figure(figsize=(8,8))
mpl.title('DOG Receptive Field', fontsize=13)
```

```
mpl.imshow(Dog_receptive_field)
mpl.colorbar()
mpl.show(block=False)

# 3D Plot for Gabor Receptive Field with Theta = pi/2
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
X = Y = np.linspace(-10, 10, 21)
X, Y = np.meshgrid(X, Y)
mpl.axes(projection='3d').plot_surface(X, Y, Dog_receptive_field, cmap='RdGy',␣
 ↪edgecolor='none')
mpl.title('3D Model of DOG Receptive Field', fontsize=13)
mpl.show()
```





As can be seen from the 2D and 3D figures, DOG_receptive_field, focuses the center pixels and as it goes to sides the response is fading. To be able to achieve that kind of response, central standart deviation must be smaller than surround standart deviation.

We can say that DOG_receptive_field, is a good model for LGN neuron cells since they also focus on the central pixels and slowly fades to sides which can be seen from Question1.

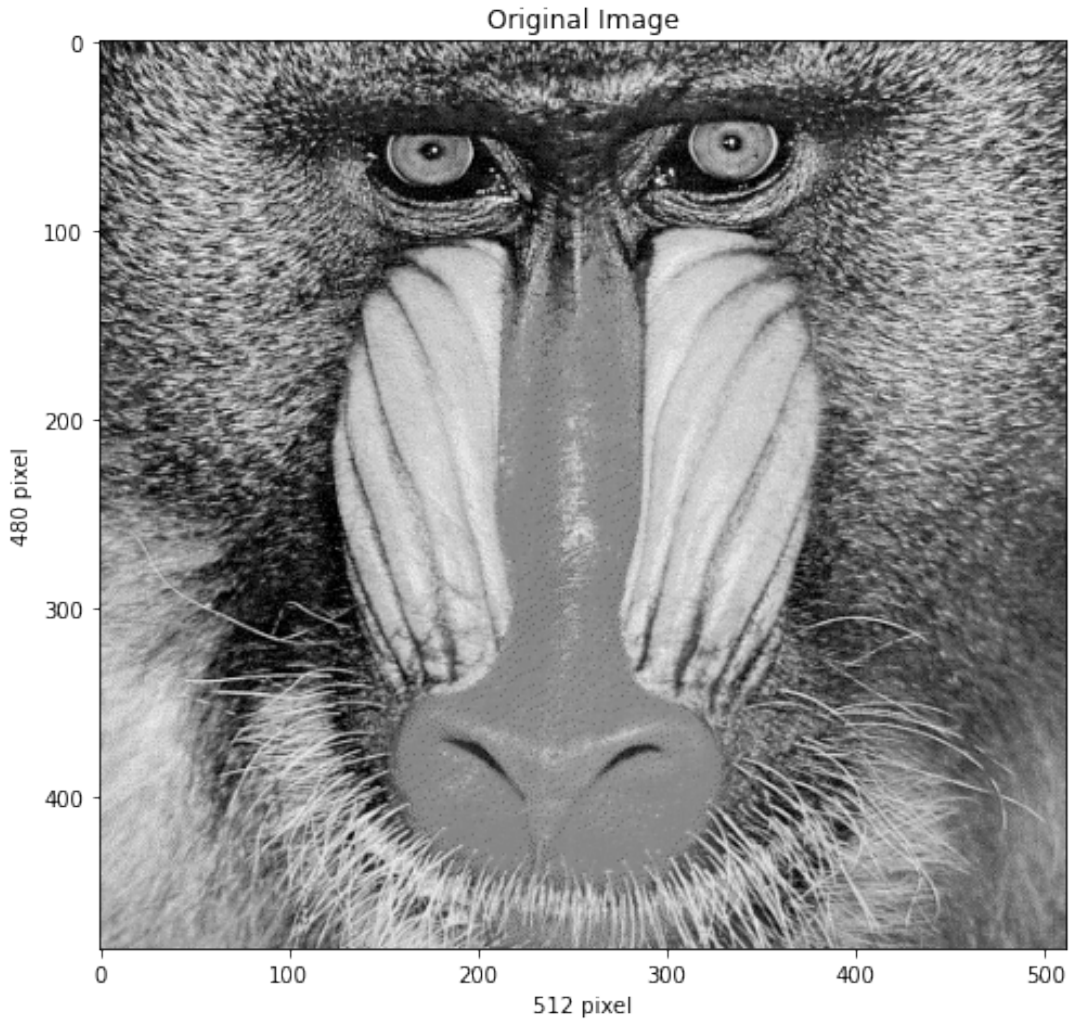
Part b

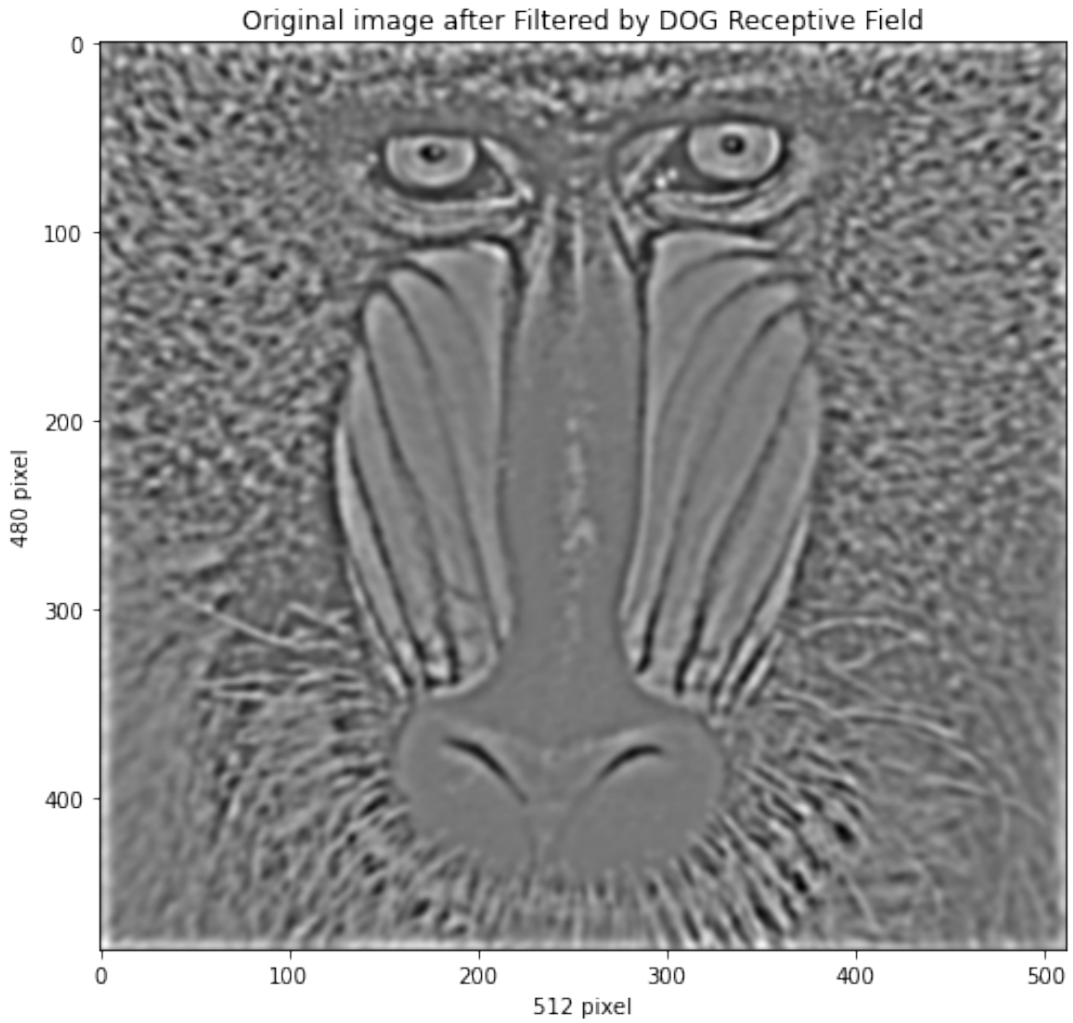
In this part, we are asked to model LGN cells with a DOG_receptive_field on a given image. The way to represent this LGN cell is to use convolution of the DOG_receptive_field over the given image. Given image has a shape of 480x512x3, the last element represent the RGB values of the pixels. Since the image is grayscaled, values of R, G and B are same so only one of them is selected while using convolution for easier calculations. For convolution, scipy's convolve method is used. First the original image, then the convolved image with DOG_receptive_field, is displayed.

[5]:
```
# Part B

# Plot of original image
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
```

```
original_image = mpl.imread('hw2_image.bmp')
print('The size of the image is: %s' % str(np.shape(original_image)))
mpl.imshow(original_image)
mpl.title("Original Image")
mpl.xlabel('512 pixel')
mpl.ylabel('480 pixel')
mpl.show()

# Plot of original image convolved with DOG receptive field
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
convolved_image = scipy.signal.convolve(original_image[:, :, 1],␣
 ↪Dog_receptive_field, mode='same')
mpl.imshow(convolved_image, cmap='gray')
mpl.title("Original image after Filtered by DOG Receptive Field")
mpl.xlabel('512 pixel')
mpl.ylabel('480 pixel')
mpl.show()
```

```
The size of the image is: (480, 512, 3)
```





DOG_receptive_field emphasized edges on an image, makes the edges more visible and darker while it blurs the area if the tone difference is small which makes edges stand out more clearly.

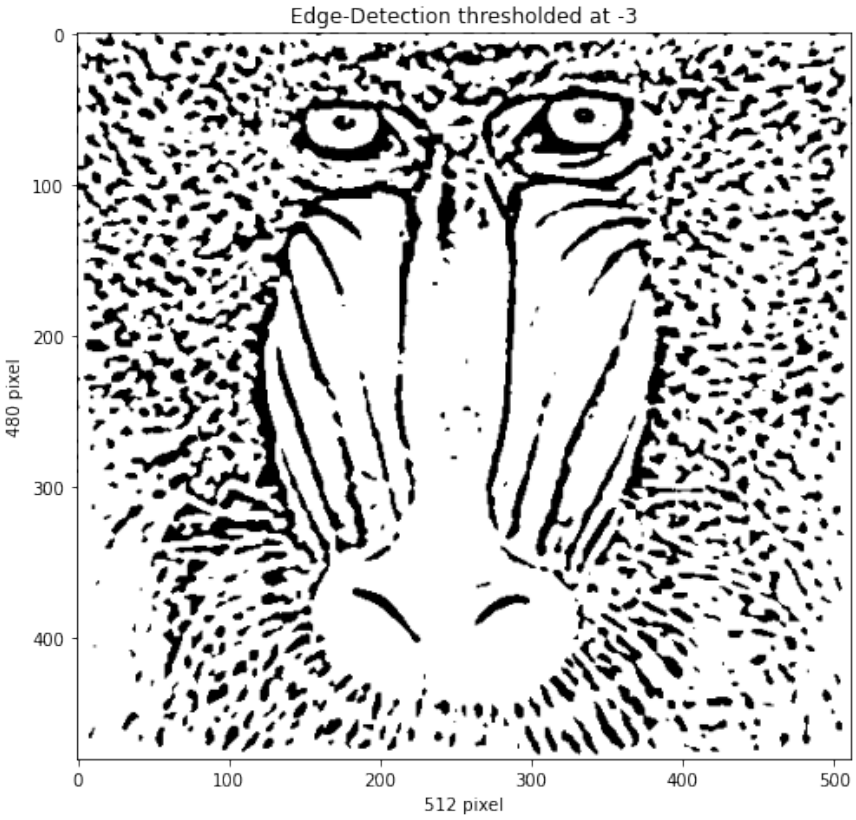
Part c

In this part, we will construct edge detection with threshold method. Since in the last part, with convolution we got a grayed out image with more clear edges. So, if we build a threshold method were edges of the image stays under some certain threshold and the other parts are above it, we can redraw the image with only edges remaining. The code for checking each pixel in the image for a given threshold is as below:

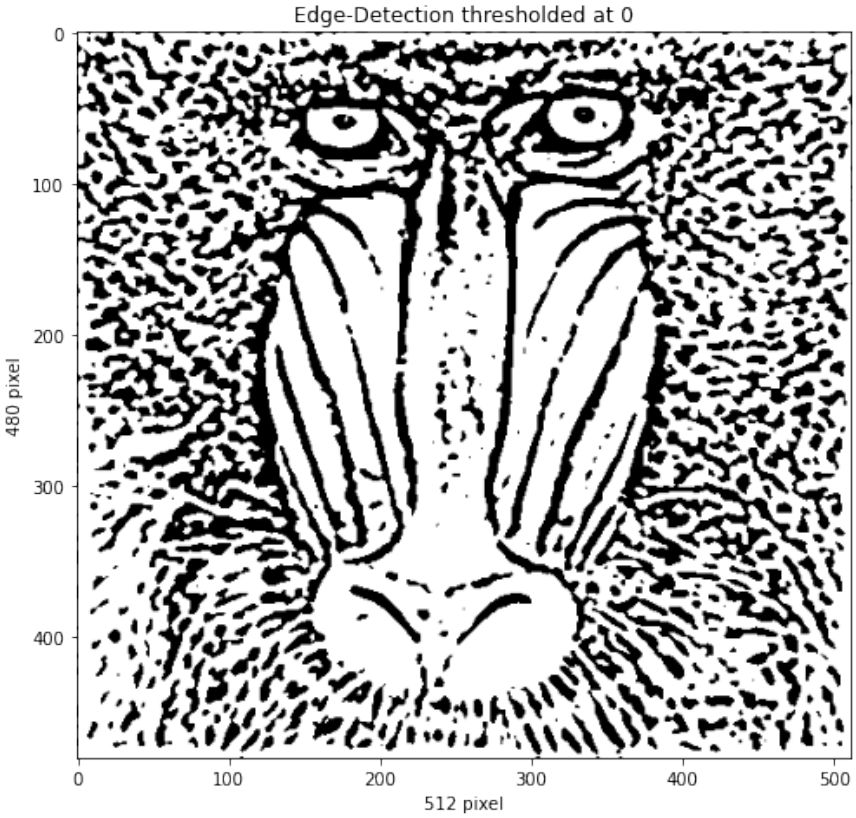
[6]:
```
#Part C

def edge_detect(image, threshold):
    result = np.zeros(np.shape(image))
    for i in range(len(image[:,0])):
        for j in range(len(image[0])):
            if image[i,j] > threshold:
                result[i,j] = 1
            else:
                result[i,j] = 0
    return result
```

Since there is no exact way to find the best threshold, i will try some threshold values and check how they perform. For that i have used -3, 0 and 3 for threshold values and displayed them respectively.

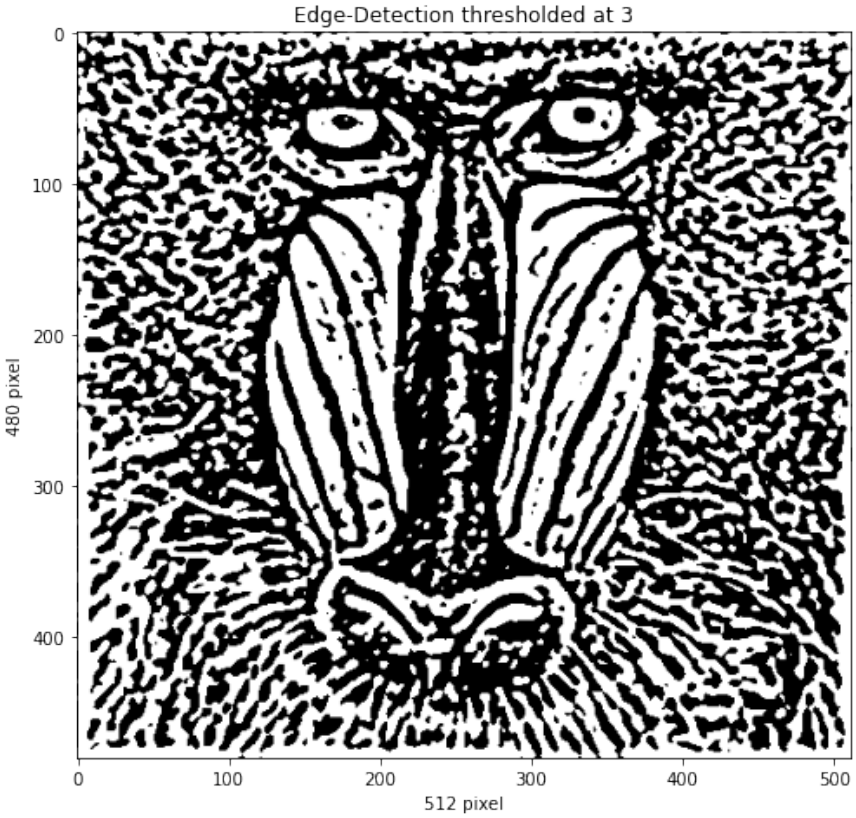
[7]:
```
# Plot for edge detection with different thresholds (-3, 0, 3)
for i in range(-1,2):
    edged_image = edge_detect(convolved_image,i*3)
    figure += 1
    mpl.figure(figure)
    mpl.figure(figsize=(8,8))
    mpl.imshow(edged_image, cmap='gray')
    mpl.title("Edge-Detection thresholded at %s" % str(i*3))
    mpl.xlabel('512 pixel')
    mpl.ylabel('480 pixel')
    mpl.show()
```



Threshold set to -3 almost gets major part of the edges but misses some minor edges around the nose. If the difference of pixel values around nose are larger than -3, that might be the reason why it could not detect some minor edges but overall it work great.



When threshold is set to 0, it detects almost all edges and works better than threshold at -3, since pixel differences around the nose is probably between (-3, 0), so 0 threshold gives better result.



Lastly, threshold is set to 3, and as it can be seen from the image, it detects whole nose as an edge which is a really bad model. The reason for that is because the nose area is thick gray and the pixel values probably fall in interval (0, 3) so it detects them as edges.

Part d

From this part on, we will repeat the processes done for LGN cells to V1 cells. V1 neuron cells are modeled with Gabor receptive field. To construct the Gabor receptive field, we must first intriduce the Gabor model as:

$$D(\vec{x}) = \exp\left(-\left(\vec{k}(\theta)\cdot\vec{x}\right)^2/2\sigma_l^2 - \left(\vec{k}_\perp(\theta)\cdot\vec{x}\right)^2/2\sigma_w^2\right)\cos\left(2\pi\frac{\vec{k}_\perp(\theta)\cdot\vec{x}}{\lambda}+\phi\right)$$

In this function, $\vec{x} =< x, y >$ is the position vector with respect to axises x and y. $\vec{k}$ is a function of θ which is the orientation. Since it is given that $\vec{k}$ is a unit vector then we can write $\vec{k}$ as:
$\vec{k} =< \cos\theta, \sin\theta >$
And $\vec{k_\perp}$ is orthogonal to And $\vec{k}$ so we can write it as:
$\vec{k_\perp} =< -\sin\theta, \cos\theta >$

Python code for the Gabor function is given below:

[8]:
```
#Part D

def Gabor(x, sigma_l, sigma_w, lambdaa, theta, phi):
    k = [math.cos(theta),math.sin(theta)]
    k_orthogonal = [- math.sin(theta), math.cos(theta)]
    k = np.array(k)
    k_orthogonal = np.array(k_orthogonal)
    result = math.exp(-(k.dot(x)**2)/(2*sigma_l**2) - (k_orthogonal.dot(x)**2) / (2␣
 ↪* (sigma_w**2))) * math.cos(2* math.pi * (k_orthogonal.dot(x))/lambdaa + phi)
    return result
```

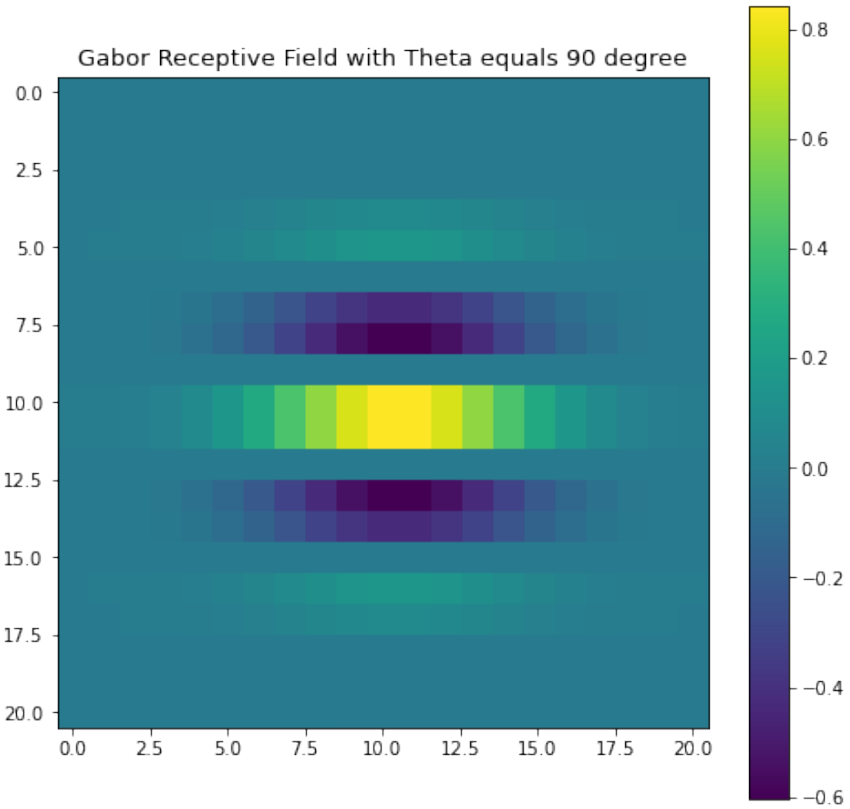
Then, like we did with DOG_receptive_field, we will construct Gabor_receptive_field. Parameters resultRowSize and resultColSize are the row and column size of the sample matrix which is also 21x21 in this case. Gabor_receptive_field, applies function Gabor for every row,column pair for given σ_l, σ_w, λ, θ and ϕ. Here σ_l, σ_w represents the standart deviation of length and width of the model respectively. Python code for Gabor_receptive_field is given below:

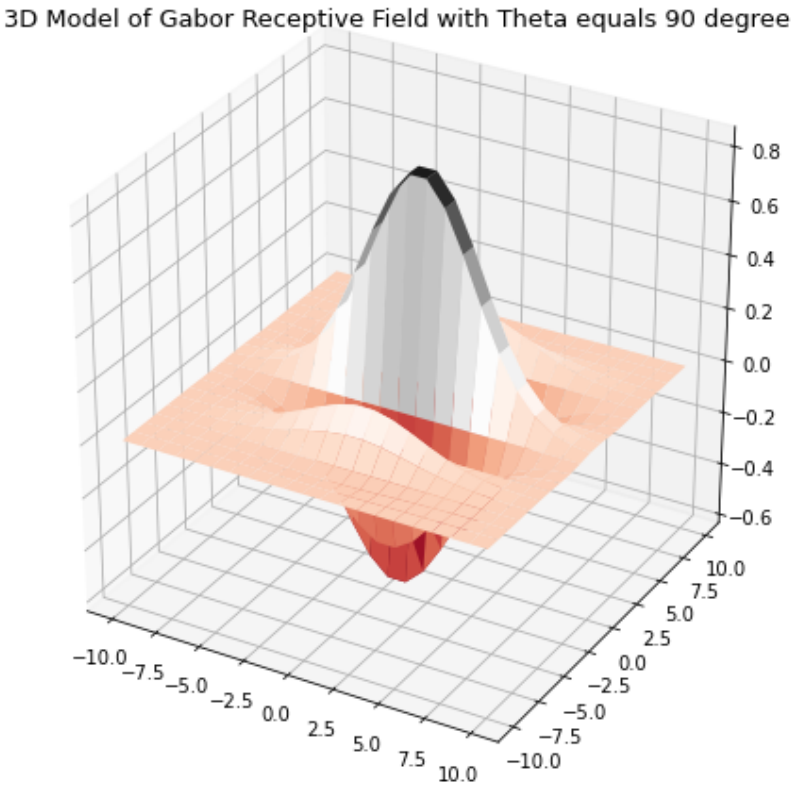
[9]:
```
def Gabor_receptive_field(resultRowSize, resultColSize, sigma_l, sigma_w, lambdaa,␣
 ↪theta, phi):
    result = np.zeros((resultRowSize, resultColSize))
    for i in range(resultRowSize):
        for j in range(resultColSize):
            result[i][j] = Gabor([i-(resultRowSize/2), j-(resultRowSize/2)],␣
 ↪sigma_l, sigma_w, lambdaa, theta, phi)
    return result
```

Code for 2D and 3D model of Gabor_receptive_field with $\sigma_l = 3$, $\sigma_w = 3$, $\lambda = 6$, $\theta = \frac{\pi}{2}$ and $\phi = 0$.

[10]:
```
gabor_field = Gabor_receptive_field(21, 21, 3, 3, 6, math.pi/2, 0)

# Plot for Gabor Receptive Field with Theta = pi/2
figure += 1
mpl.figure(figure)
```

```
mpl.figure(figsize=(8,8))
mpl.title('Gabor Receptive Field with Theta equals 90 degree', fontsize=13)
mpl.imshow(gabor_field)
mpl.colorbar()
mpl.show()

# 3D Plot for Gabor Receptive Field with Theta = pi/2
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
X = Y = np.linspace(-10, 10, 21)
X, Y = np.meshgrid(X, Y)
mpl.axes(projection='3d').plot_surface(X, Y, gabor_field, cmap='RdGy',␣
 ↪edgecolor='none')
mpl.title('3D Model of Gabor Receptive Field with Theta equals 90 degree',␣
 ↪fontsize=13)
mpl.show()
```



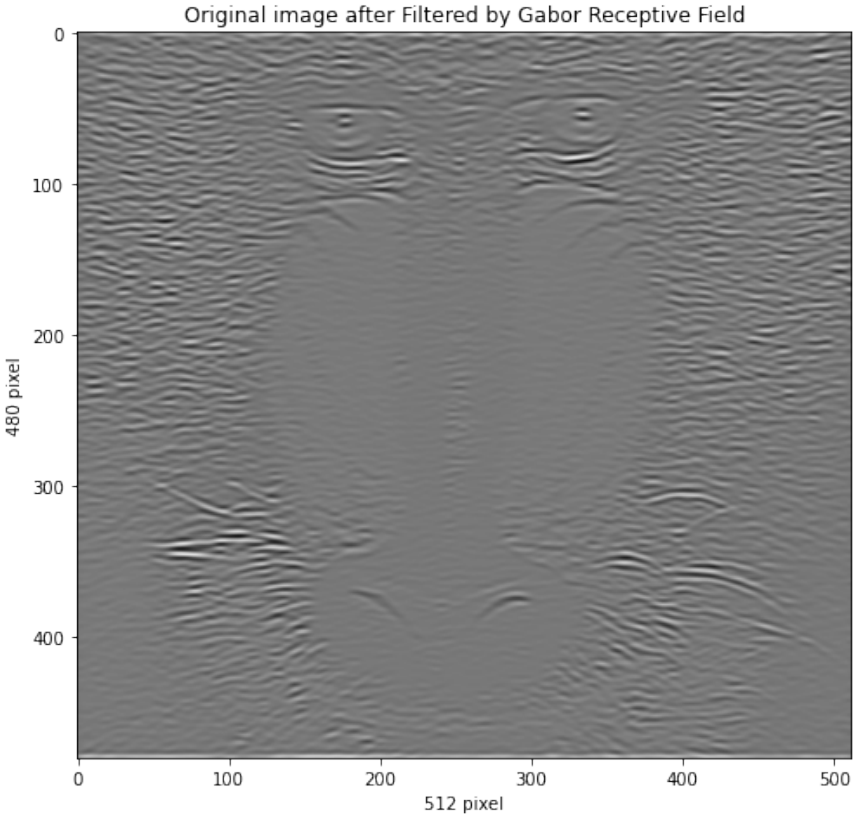


It can clearly be seen from the 2D model that, orientation of the field is parallel to x axis and perpendicular to y axis. Since we have denoted $\theta = \frac{\pi}{2}$, this was the desired output. Also from the 2D model, we can say that at center function has its peak and slowly decays as it goes away from center. It can be seen from the 3D model that function behaves like a decaying sinusoidal with its peak at center.

Part e

Similar to part b, we are asked to construct Gabor_receptive_field on the same given image. The way to represent this V1 cell is to use convolution of the Gabor_receptive_field. Similarly, since the image is grayed out, only one of the R, G or B values will be selected for ease of calculation. Again, scipy's convolve fucntion is used for this convolution. The code and the plot is below:

[11]:
```
#Part E

#Plot for gabor field convoluted image
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
convolved_gabor_image = scipy.signal.convolve(original_image[:, :, 1], gabor_field,␣
 ↪mode='same')
mpl.imshow(convolved_gabor_image, cmap='gray')
mpl.title("Original image after Filtered by Gabor Receptive Field")
mpl.xlabel('512 pixel')
mpl.ylabel('480 pixel')
mpl.show()
```



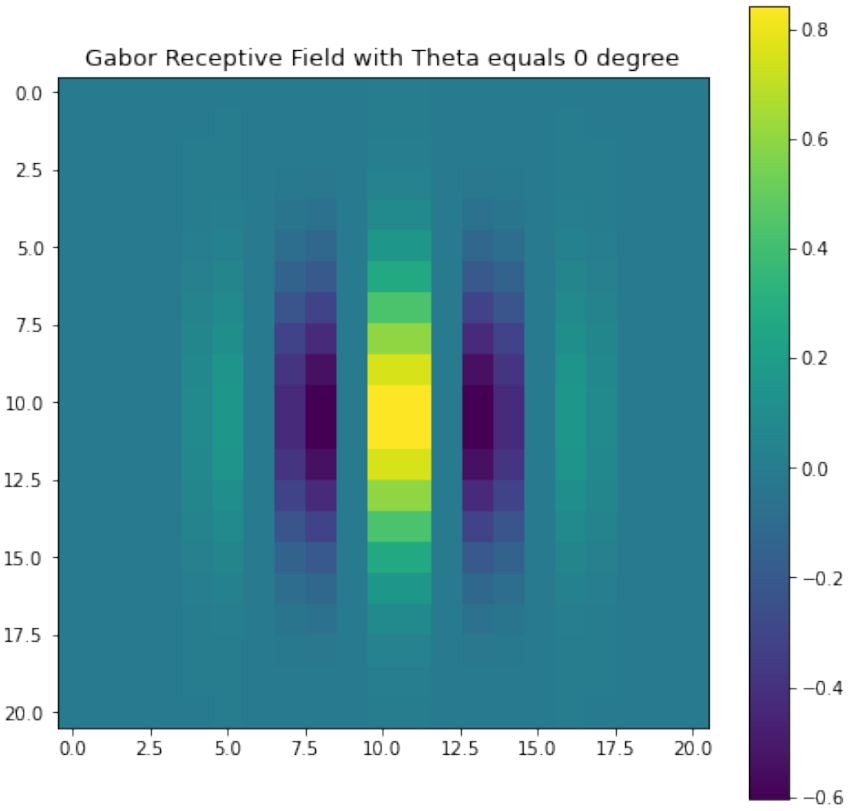
As can be seen from the image, it has ridges at parts which are perpendicular to y and parallel to x. This was expected since we have used $\theta = \frac{\pi}{2}$. The difference between DOG and Gabor is that, DOG detects edges more clearly but Gabor does convolution with respect to orientation so it detect changes for given orientation.

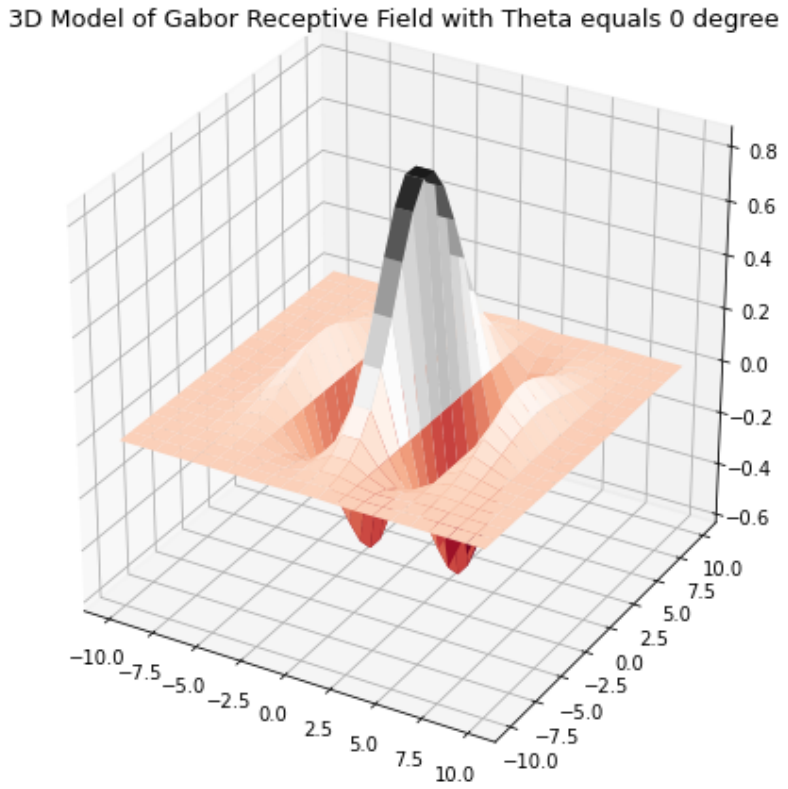
Part f

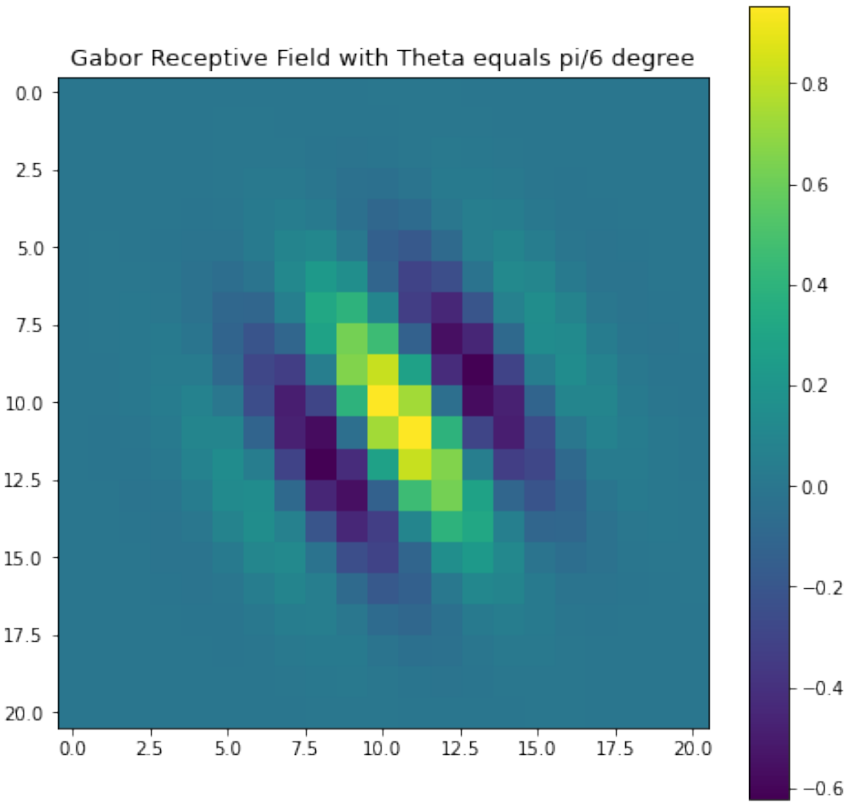
In this part, we try to construct a V1 neuron cell which is summation of Gabor_receptive_fields with different orientations. Those orientations are $\theta = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}$. First, 2D and 3D models for Gabor_receptive_fields with each orientation will be displayed to clearly see the direction of those orientations. Then we will filter the images with Gabor_receptive_fields for each orientation. Also edge detection will be applied to those filtered images. Finally those filtered images and edge detected images will be summed to form summed filtered image and summed edge detected image. The python code for 2D and 3D modeling of Gabor_receptive_fields is below:

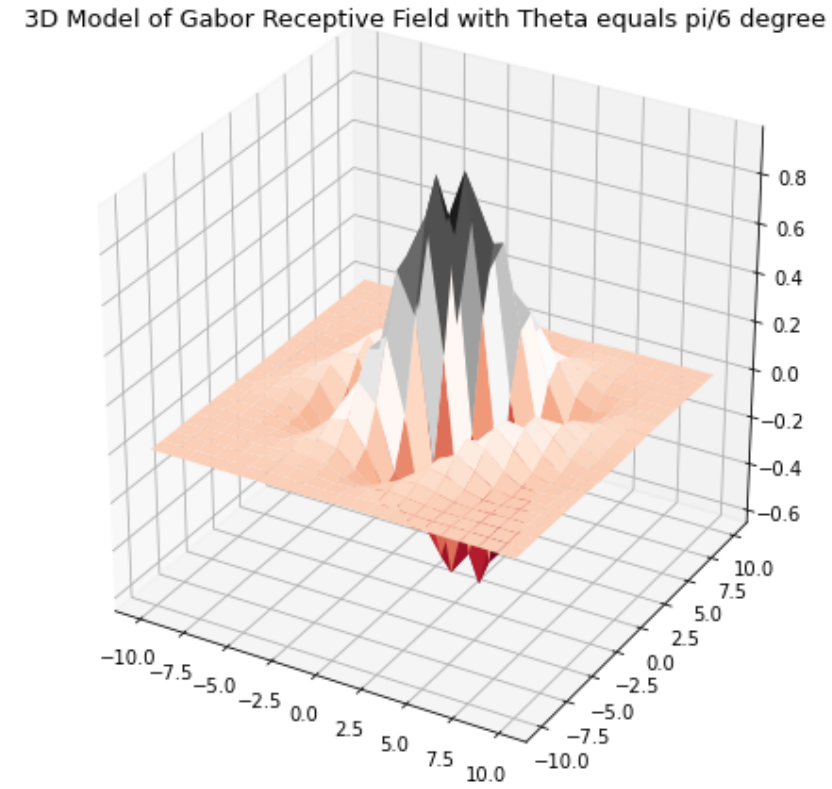
[12]:
```
#Part F

thetas = [[0,0], [math.pi/6,6], [math.pi/3,3], [math.pi/2,2]]
for i,j in thetas:
    # Gabor Receptive fields for different thetas
    gabor_field = Gabor_receptive_field(21, 21, 3, 3, 6, i, 0)

    # Plot for all Gabor Receptive Fields
    figure += 1
    mpl.figure(figure)
    mpl.figure(figsize=(8,8))
    mpl.title('Gabor Receptive Field with Theta equals %s degree' % (('pi/%d' %j)␣
 ↪if j != 0 else '0'),fontsize=13)
    mpl.imshow(gabor_field)
```

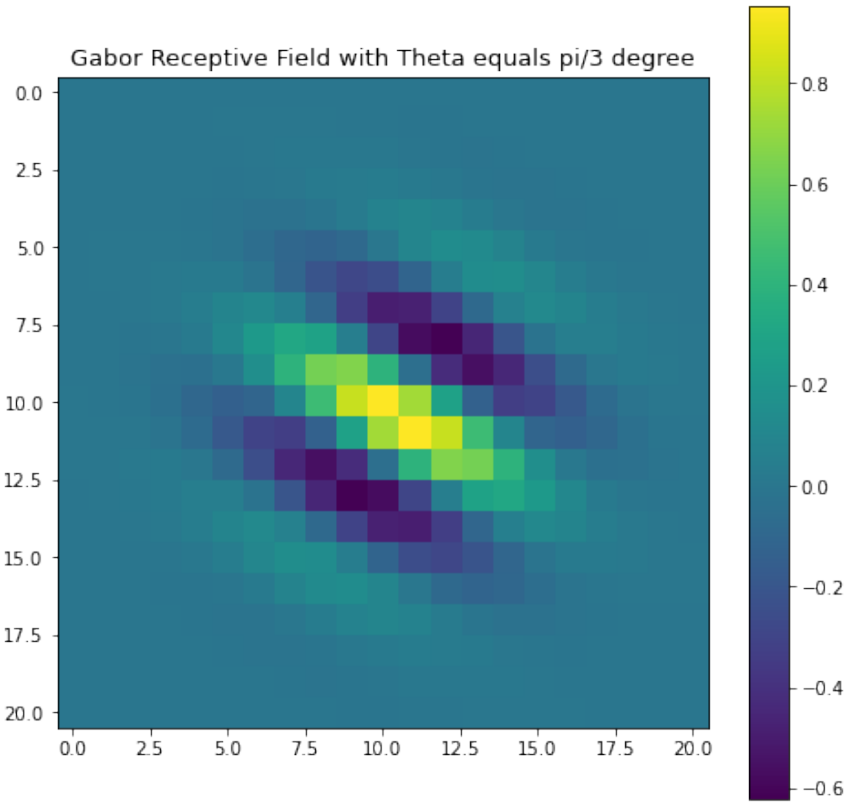
```
    mpl.colorbar()
    mpl.show()

    # 3D plot for all Gabor Receptive Fields
    figure += 1
    mpl.figure(figure)
    mpl.figure(figsize=(8,8))
    X = Y = np.linspace(-10, 10, 21)
    X, Y = np.meshgrid(X, Y)
    mpl.axes(projection='3d').plot_surface(X, Y, gabor_field, cmap='RdGy',␣
↪edgecolor='none')
    mpl.title('3D Model of Gabor Receptive Field with Theta equals %s degree' %␣
↪(('pi/%d' %j) if j != 0 else '0'), fontsize=13)
    mpl.show()
```

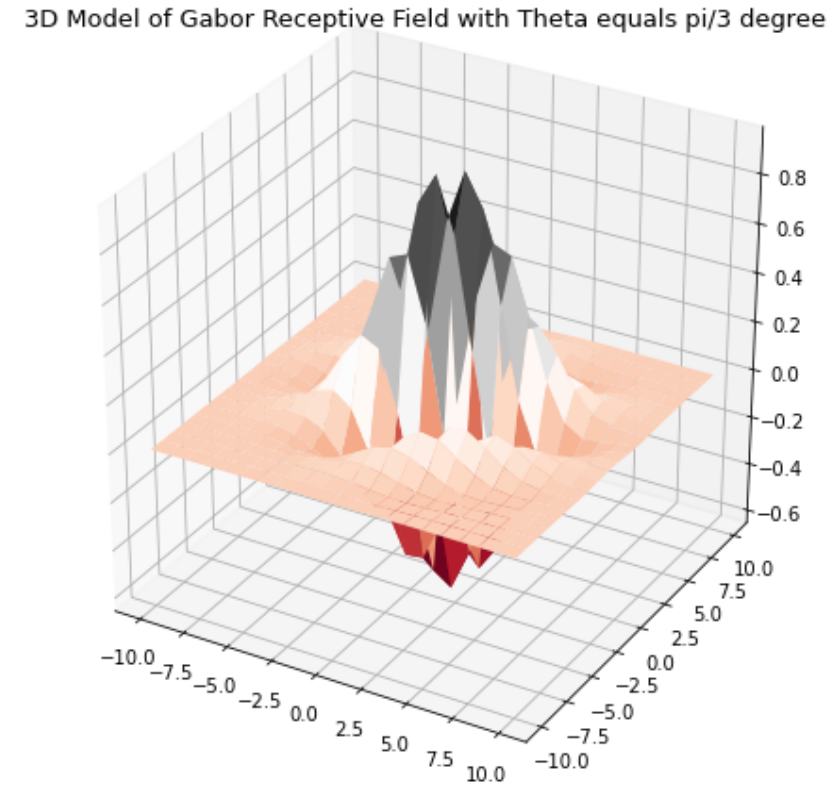


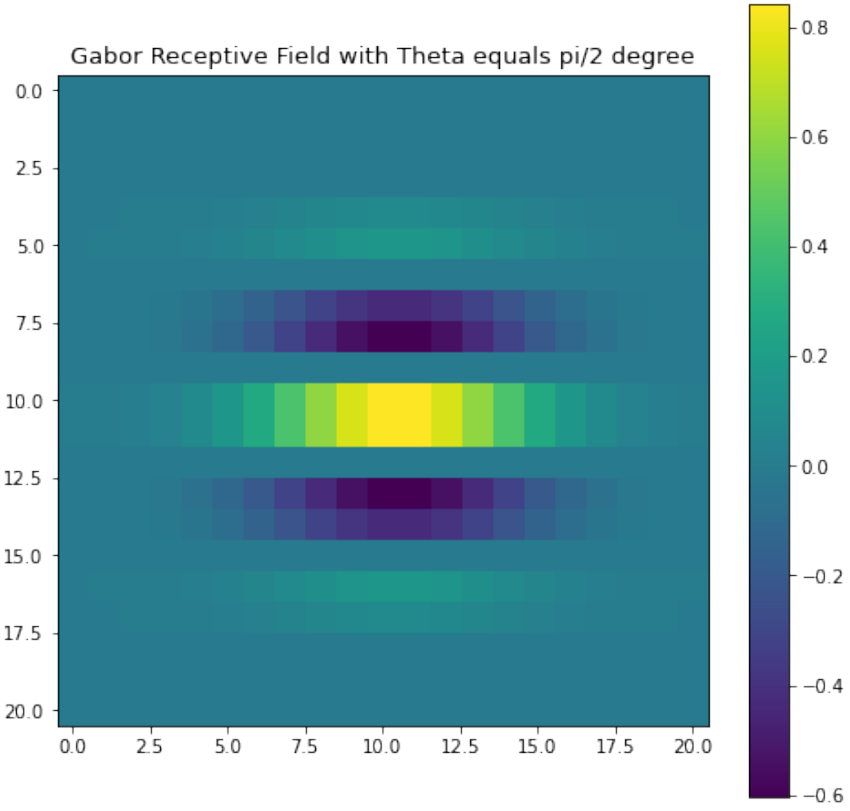

3D Model of Gabor Receptive Field with Theta equals 0 degree
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

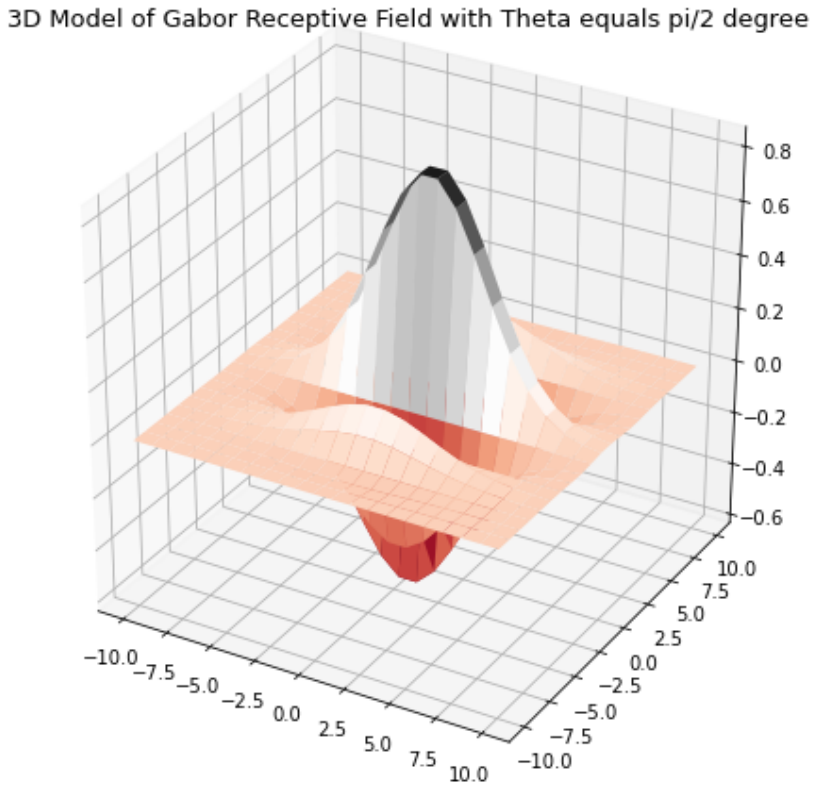

Gabor Receptive Field with Theta equals pi/6 degree
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


3D Model of Gabor Receptive Field with Theta equals pi/6 degree
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Gabor Receptive Field with Theta equals pi/3 degree
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


3D Model of Gabor Receptive Field with Theta equals pi/3 degree
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Gabor Receptive Field with Theta equals pi/2 degree
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

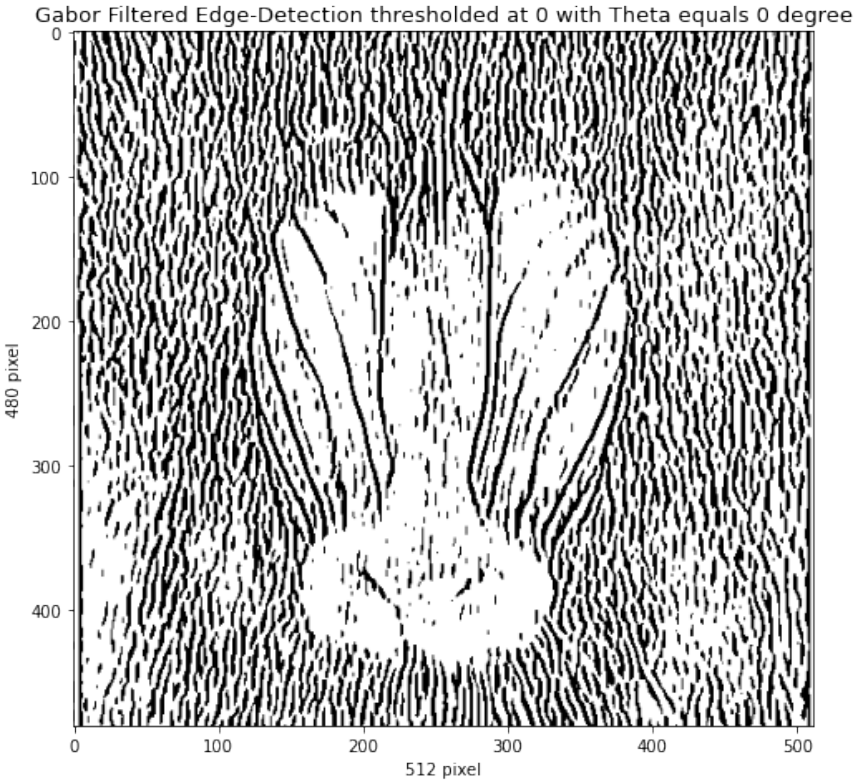


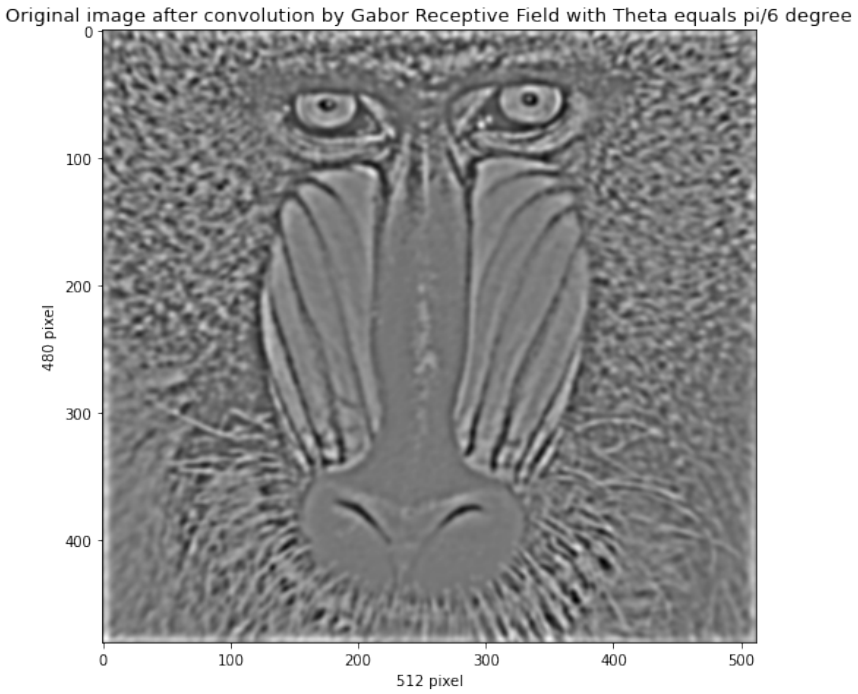
We can see the change in direction with change in θ values. Now each of these Gabor_receptive_fields will be used to filter the image and then edge detect the filtered image. Python code for that is below:

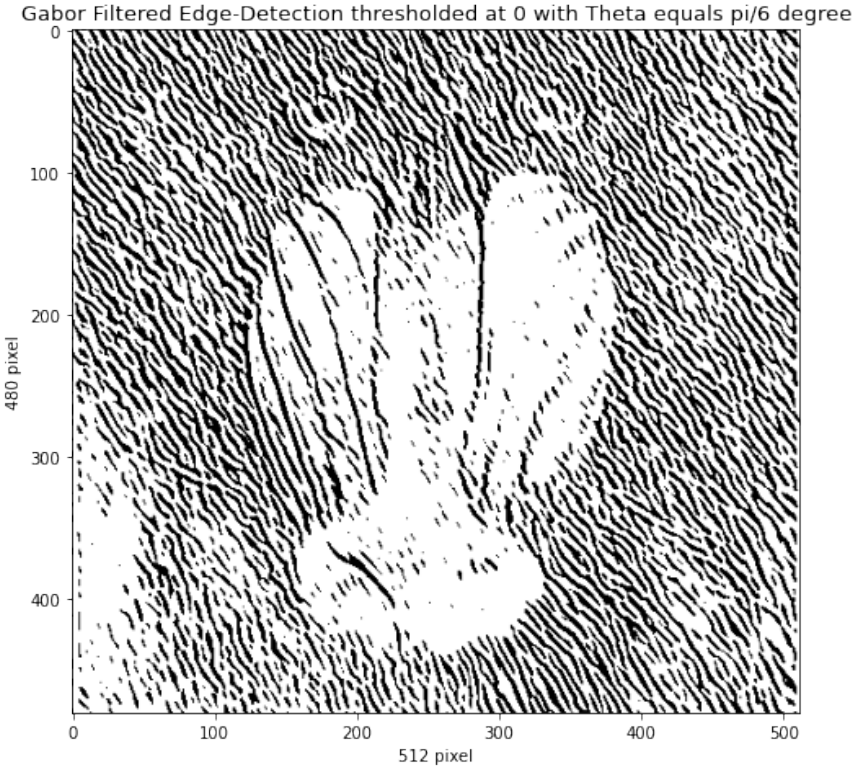
[13]:
```
gabor_image_combined = np.empty(np.shape(convolved_gabor_image))
edged_gabor_image_combined = np.empty(np.shape(convolved_gabor_image))
for i,j in thetas:

    # For convolution
    gabor_field = Gabor_receptive_field(21, 21, 3, 3, 6, i, 0)
    # Convolution plotting
    figure += 1
    mpl.figure(figure)
    mpl.figure(figsize=(8,8))
    convolved_gabor_image = scipy.signal.convolve(original_image[:, :, 1],␣
↪gabor_field, mode='same')
    mpl.imshow(convolved_image, cmap='gray')
    mpl.title('Original image after convolution by Gabor Receptive Field with Theta␣
↪equals %s degree' % (('pi/%d' %j) if j != 0 else '0') , fontsize=13)
    mpl.xlabel('512 pixel')
    mpl.ylabel('480 pixel')
    mpl.show()

    # Sum up all the images from convolution
    gabor_image_combined += convolved_gabor_image

    # For Edge Detection
    edged_gabor_image = edge_detect(convolved_gabor_image,0)
```

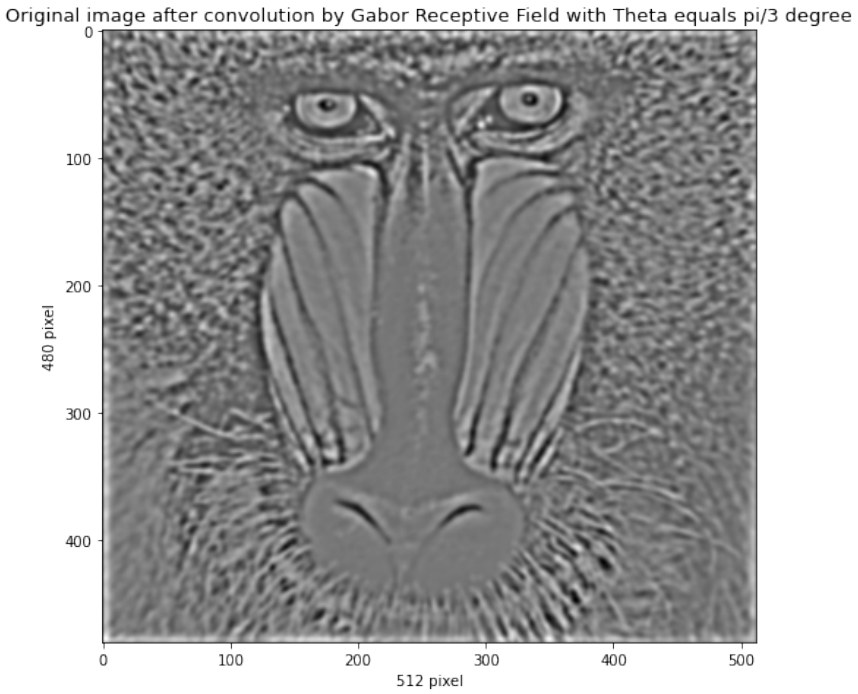
```
    # Edge Detection Plotting
    figure += 1
    mpl.figure(figure)
    mpl.figure(figsize=(8,8))
    mpl.imshow(edged_gabor_image, cmap='gray')
    mpl.title('Gabor Filtered Edge-Detection thresholded at 0 with Theta equals %s␣
 ↪degree' % (('pi/%d' %j) if j != 0 else '0') , fontsize=13)
    mpl.xlabel('512 pixel')
    mpl.ylabel('480 pixel')
    mpl.show()

    # Sum up all the images from Edge Detection
    edged_gabor_image_combined += edged_gabor_image

# Plot summed up convolved images
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
convolved_image = scipy.signal.convolve(original_image[:, :, 1], gabor_field,␣
 ↪mode='same')
mpl.imshow(convolved_image, cmap='gray')
mpl.title('Original image after convolution by Gabor Receptive Field ([0, math.pi/
 ↪6, math.pi/3, math.pi/2] all summed)', fontsize=13)
mpl.xlabel('512 pixel')
mpl.ylabel('480 pixel')
mpl.show()

# Plot summed up edge detected images
figure += 1
mpl.figure(figure)
mpl.figure(figsize=(8,8))
mpl.imshow(edged_gabor_image_combined, cmap='gray')
mpl.title('Gabor Filtered Edge-Detection thresholded at 0 (For Theta [0, math.pi/6,␣
 ↪math.pi/3, math.pi/2] all summed))' , fontsize=13)
mpl.xlabel('512 pixel')
mpl.ylabel('480 pixel')
mpl.show()
```

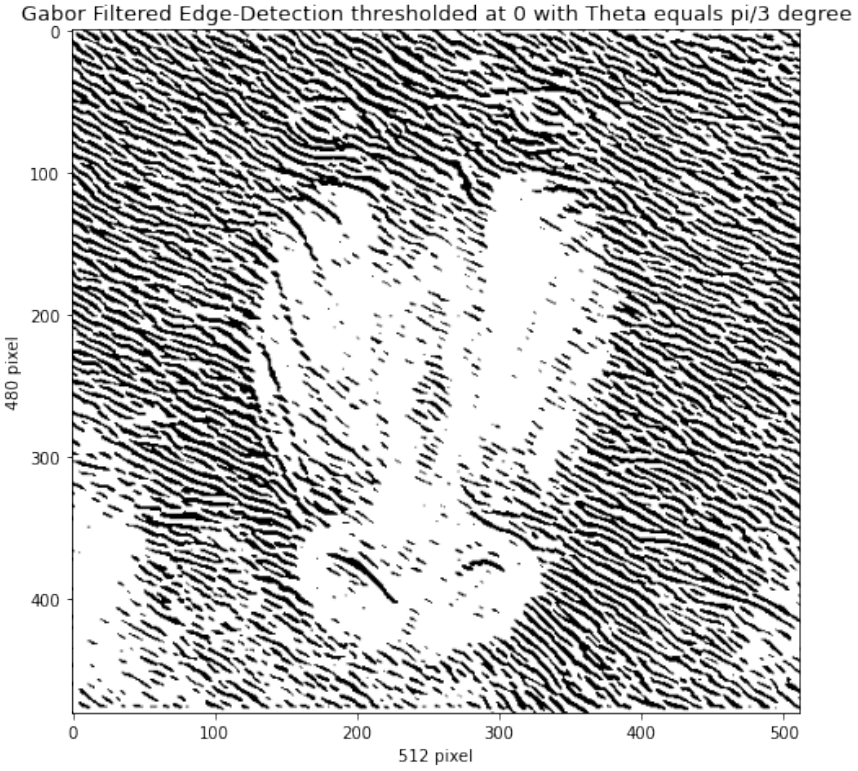

Original image after convolution by Gabor Receptive Field with Theta equals 0 degree
480 pixel
512 pixel
0
100
200
300
400
500

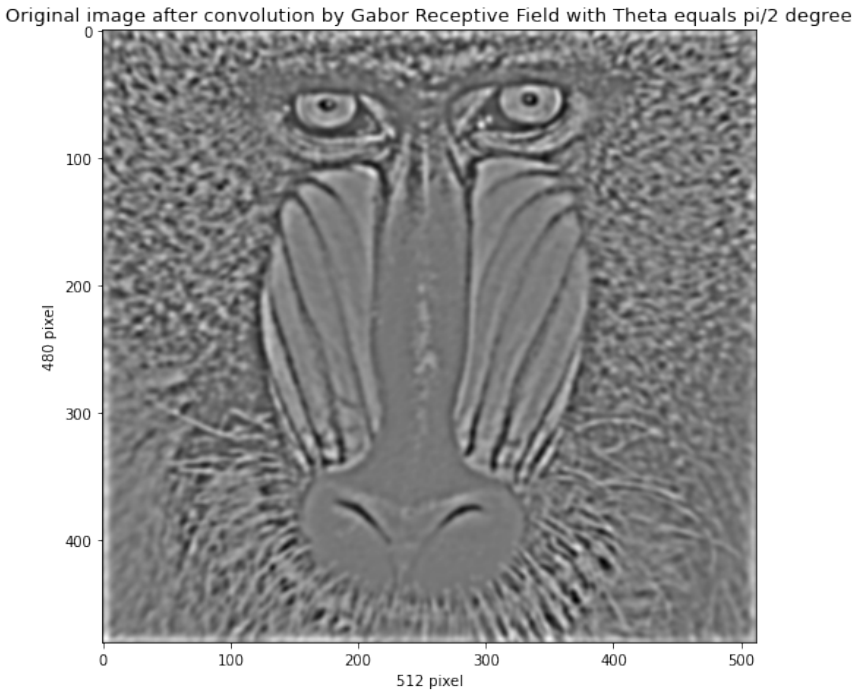

Gabor Filtered Edge-Detection thresholded at 0 with Theta equals 0 degree
480 pixel
512 pixel
0
100
200
300
400
500

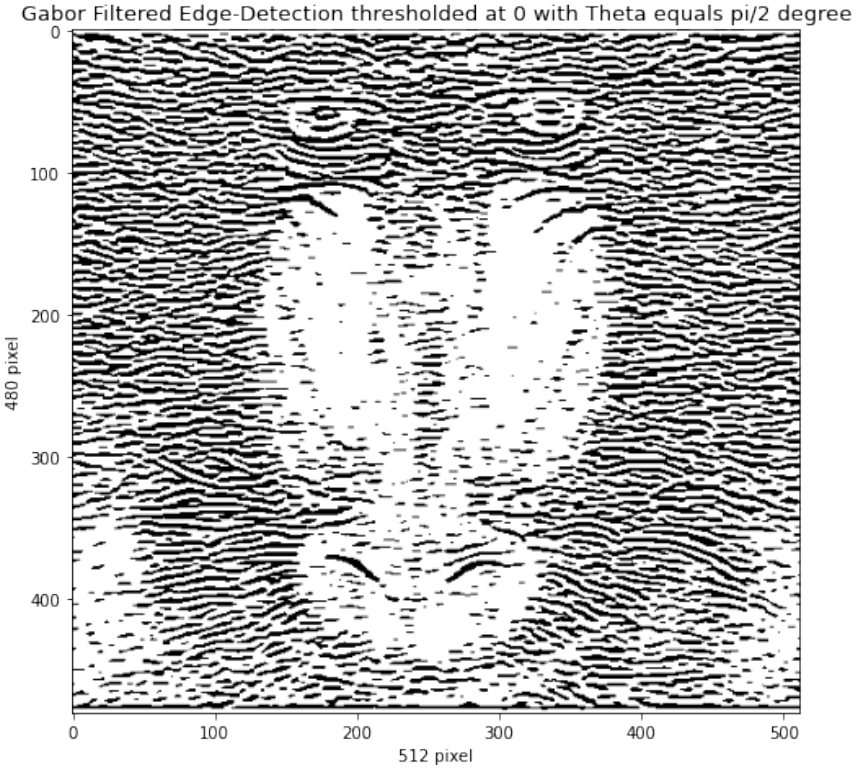

Original image after convolution by Gabor Receptive Field with Theta equals pi/6 degree
480 pixel
512 pixel
0
100
200
300
400
500


Gabor Filtered Edge-Detection thresholded at 0 with Theta equals pi/6 degree
480 pixel
512 pixel
0
100
200
300
400
500

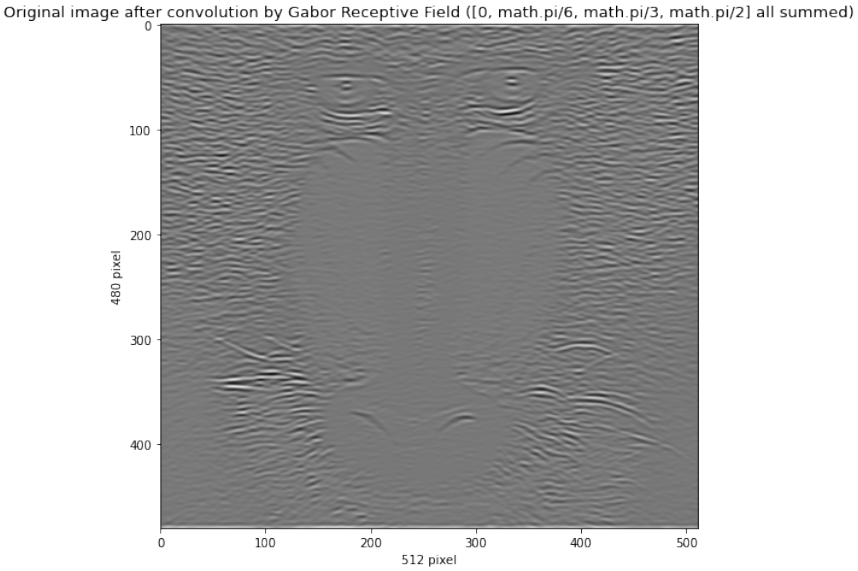

Original image after convolution by Gabor Receptive Field with Theta equals pi/3 degree
480 pixel
512 pixel
0
100
200
300
400
500

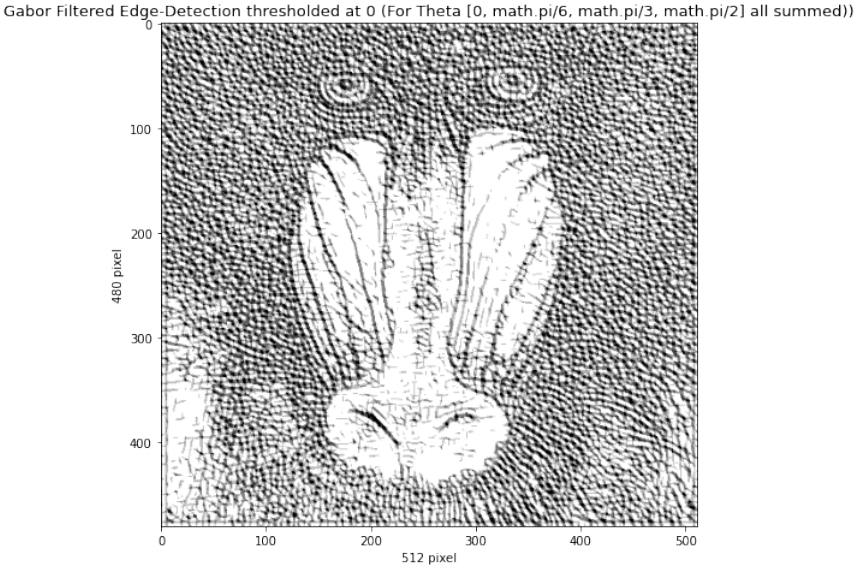

Gabor Filtered Edge-Detection thresholded at 0 with Theta equals pi/3 degree
480 pixel
512 pixel
0
100
200
300
400
500





We can see that for both filtered images and edge detected images, the filtering ridges are parallel to orientation direction.





For both filtered images and edge detected images, summed over all orientations version is clearly performs better. This is due to fact that the edges are well emphasized in summed version since we have summed over 4 orientation. To make the image even more clear, one can include more orientations in summation.

References

[1] G. Bebis, Advances in visual computing: 6th international symposium, ISVC 2010, Las Vegas, NV, USA, November 29 - December 1, 2010: proceedings. Berlin: Springer, 2010.

[2] Gabor Filters. [Online]. Available: http://www.cs.utah.edu/ arul/report/node13.html. [Accessed: 27-Mar-2020].