EEE482 - Computational Neuroscience

Homework Assignment - 2

Ahmet Ali Nuhoğlu

21602149

Ahmet Ali NUHOĞLU
21602149

Homework 1

EEE482

Question 1

In this question, we are provided responses of a LGN cell of a cat as a file. This file contains, counts of the spikes for each 15.6 ms bin and also the images that are shown at corresponding times.

a) Lateral geniculate nucleus (LGN) cells can be selective to the orientation, direction or luminous intensity. In this part,we are expected to calculate the spike triggered average images to determine, what this LGN cell is selective for. To do that we will be calculating the STA images for time steps one to ten, afterwards all found STA images will be displayed.

Those three lines show which libraries will be used in the entire question. **numpy** library is used for computational easiness, **scipy.io** is used for loading the given data for further use in the program and **matplotlib** library is used for data visualization purposes.

```python
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
```

In the following lines of code, all the given data is loaded into a variable named **data**. Since, given data includes both the **counts** and **stim** in the same file, they are parsed into two different variables. Also, their shapes are printed in order to be sure of they are loaded and parsed as they are supposed to be and knowing their shapes will helps us in the future uses of these variables.

```python
data = sio.loadmat('c2p3.mat')
counts = data['counts']
stim = data['stim']
print(np.shape(stim))
print(np.shape(counts))
```

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

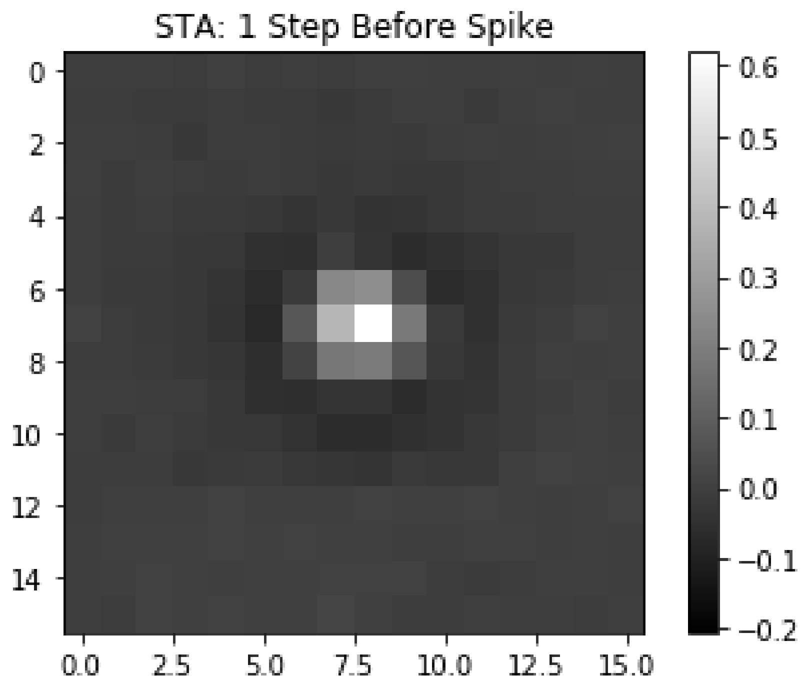
A function to compute the spike triggered averages of given images with respect to the counts of the spikes is defined below. In the function we are taking three parameters as inputs two of them are the data provided to us. Third one was to determine the which time step we are calculating, as it is expected from us to compute each of the ten time steps before each spike and show them all.
In the function we are multiplying the counts of the spikes with its corresponding stim image with respect to which time step is calculating. Therefore, we sum up all these multiplied values and after this process is finished, we divide it to the number of all spikes that are used in the multiplication to get the average.

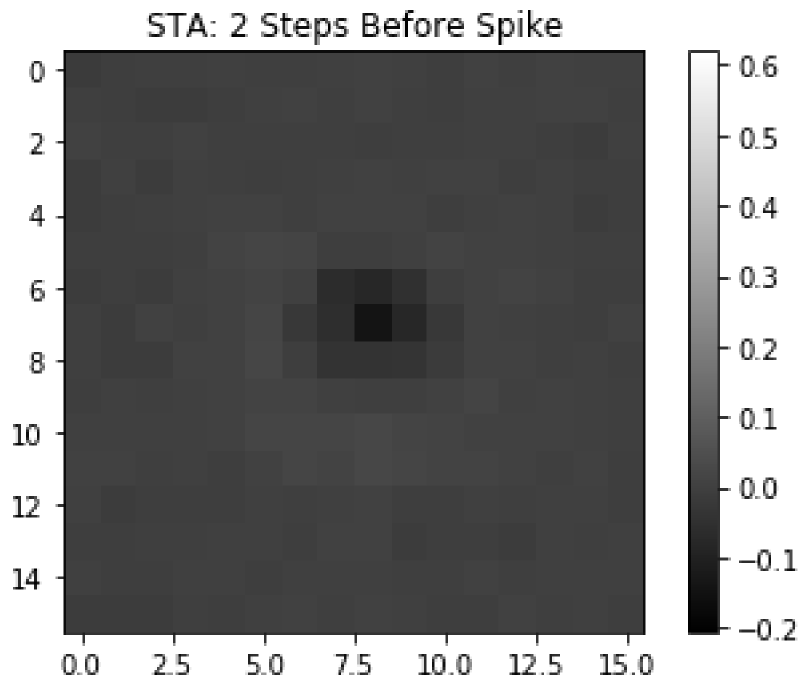
```
def STA(stim, counts, time_s):
    avg = np.zeros((len(stim[0]), len(stim[1])))
    for i in range (0, len(counts) - time_s):
        avg[:,:] += stim[:,:,i] * counts[i + time_s]
    avg /= np.sum(counts[time_s:])
    return avg
```

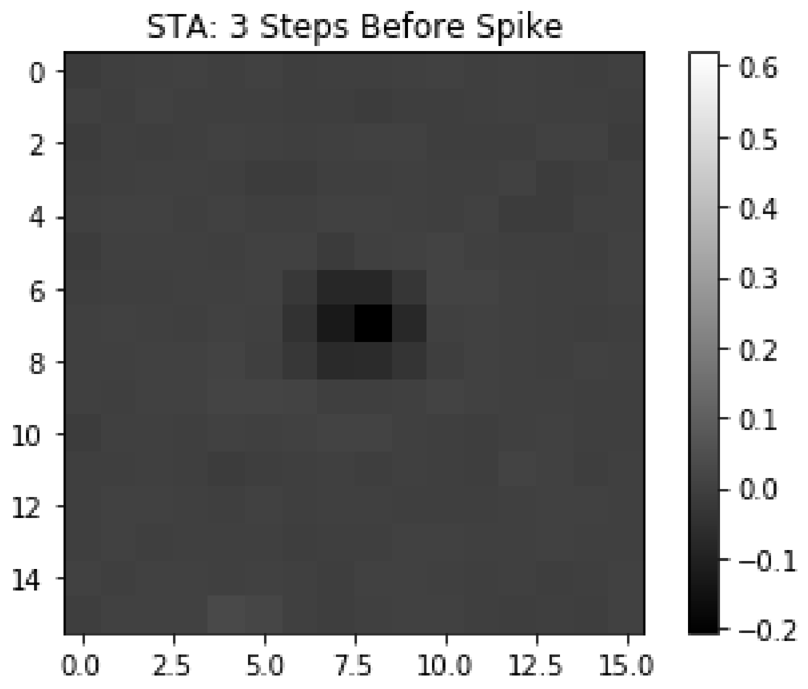
In the following two cells of code, average for each time step is generated using the STA function that we have defined and they are stored. Afterwards, using those stored values of the STA images we display them using the **imshow** function of the **matplotlib.pyplot** library.

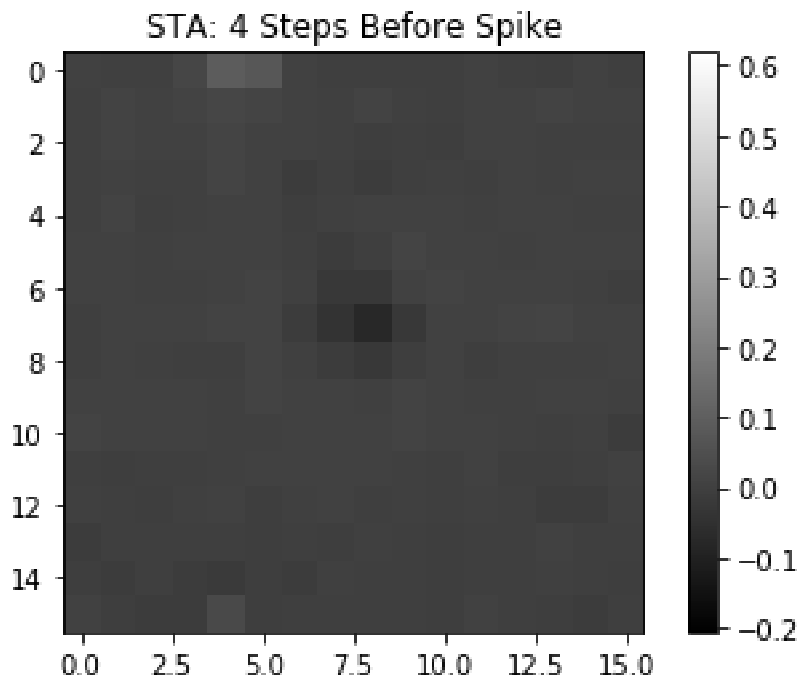
```
averages = np.zeros((len(stim[0]), len(stim[1]), 10))
for i in range(0, 10):
    averages[:,:,i] = STA(stim, counts, i+1)
```

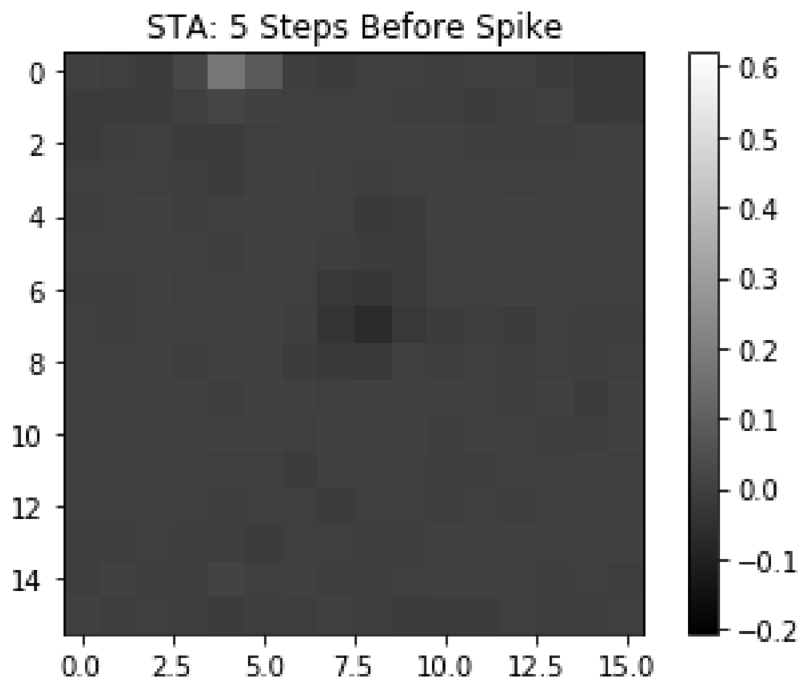
```
for i in range(10):
    plt.figure()
    plt.imshow(averages[:,:, i], cmap='gray', vmin=np.
 ↪min(averages), vmax=np.max(averages))
    plt.colorbar()
    if i == 0:
        plt.title('STA: 1 Step Before Spike')
    else:
        plt.title('STA: %d Steps Before Spike' % (i + 1))
    plt.show(block=False)
```

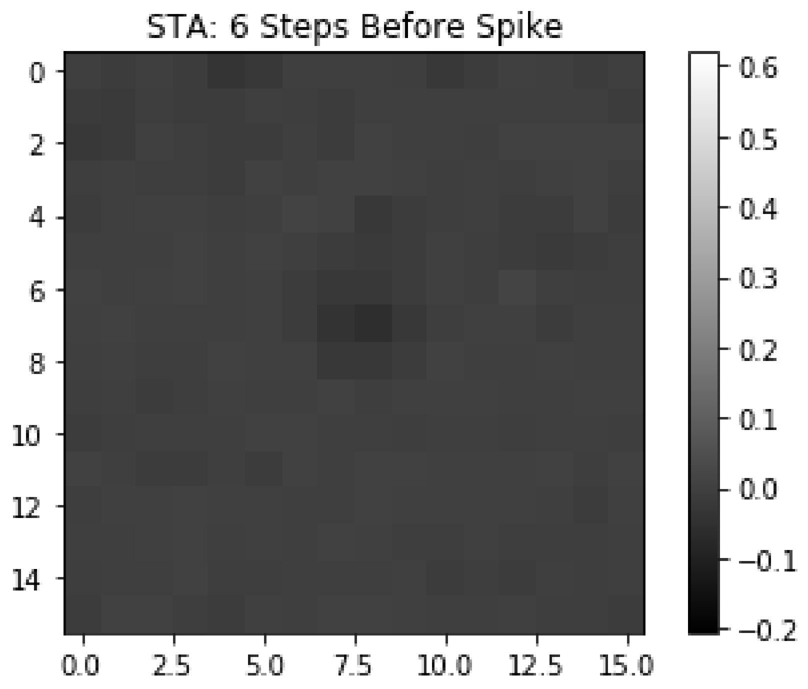


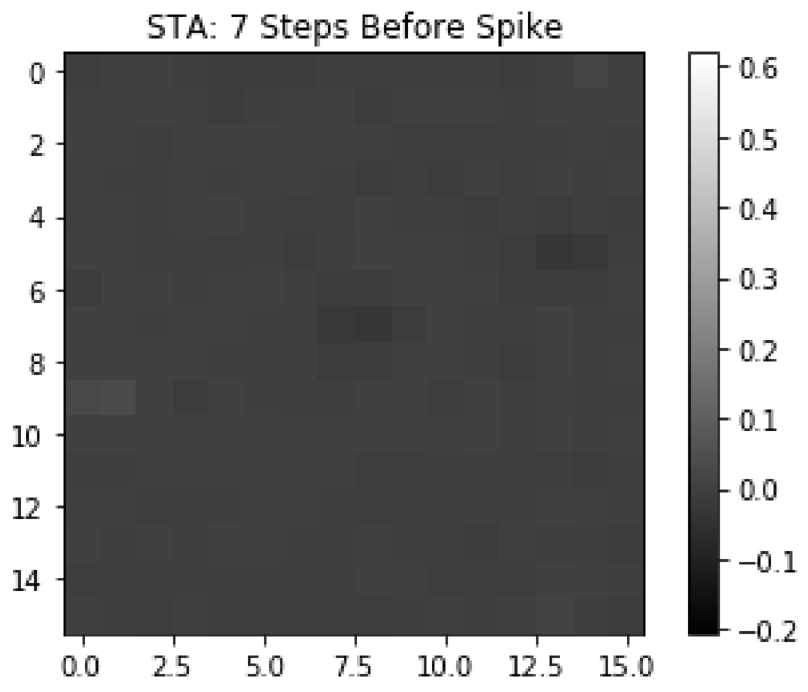

STA: 2 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2

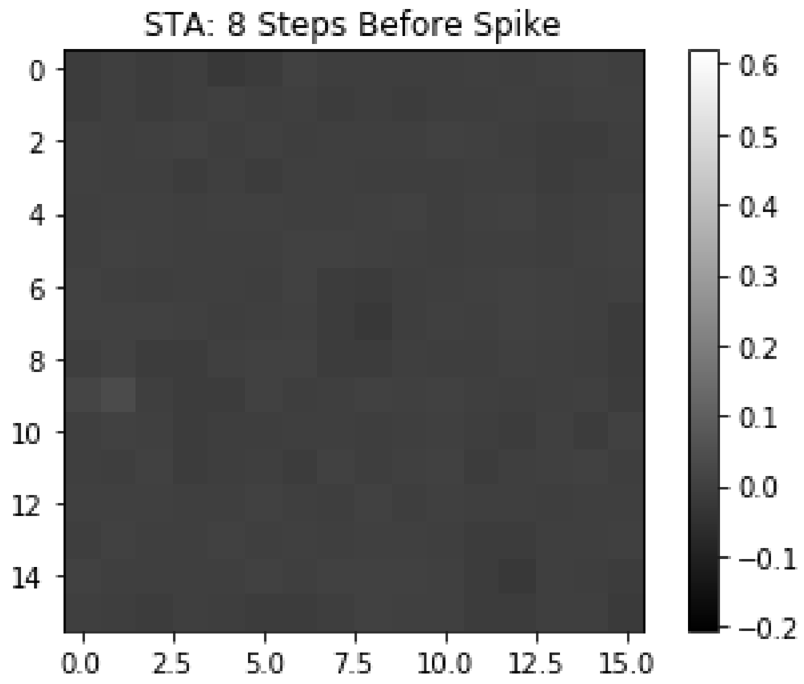

STA: 3 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2

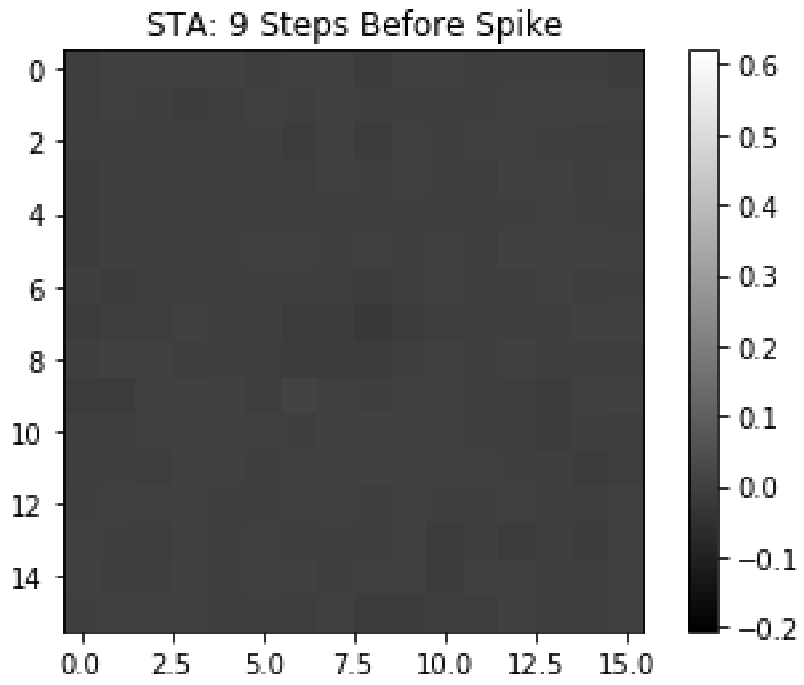

STA: 4 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2

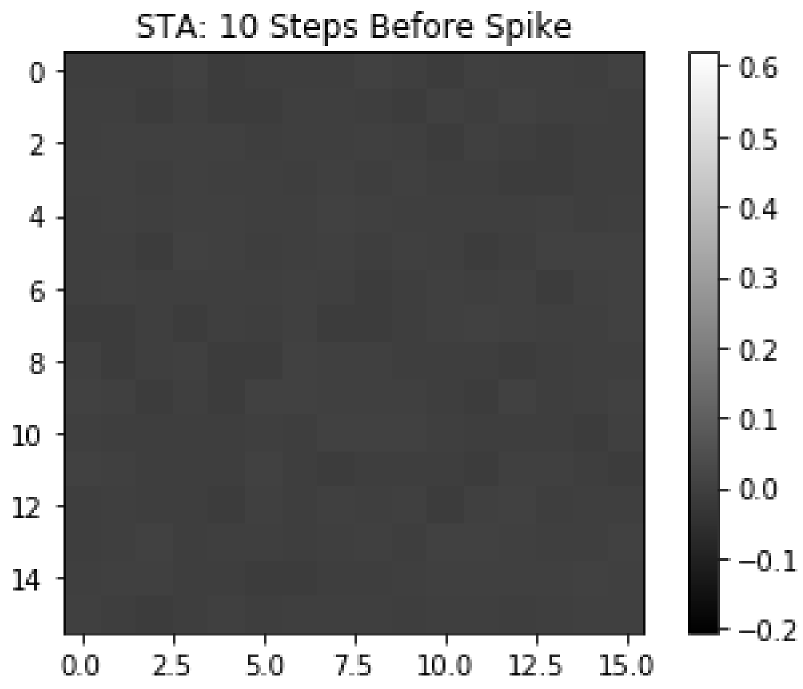

STA: 5 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 6 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 7 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 8 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 9 Steps Before Spike
0
2
4
6
8
10
12
14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2

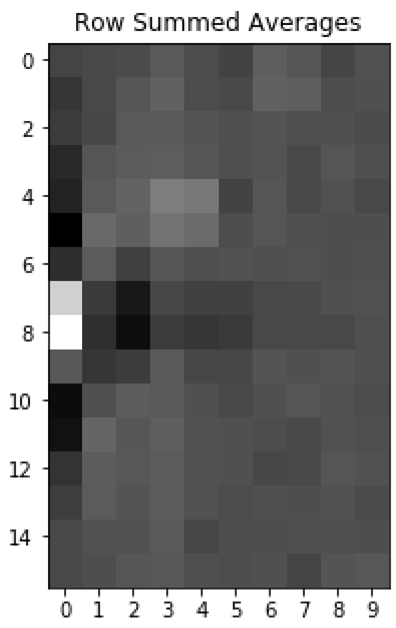


From the STA images obtained, which are displayed above, we can conclude that this LGN cell is not selective for neither orientation nor direction. It seems that, it is selective for the luminous intensity.

b) In this part, it has been asked from to sum the STA images along one of the axes to reduce its dimension by one and obtain an image of shape (16, 10). In this image, dimension with length ten represents the time steps and dimension with length sixteen will represent the axes which is not the one summed along. To do this summation operation **sum** function of the **numpy** library will be used. Since we can denote the axis which we want to sum along to the **sum** function and it will do the rest.

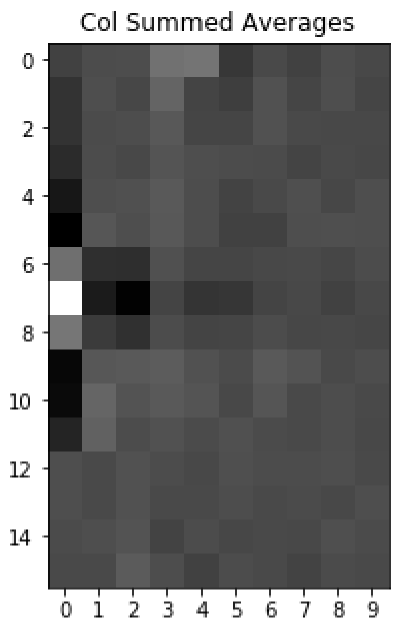
Following code sums along the x-axis and displays the image.

```
row_avgs = np.sum(averages, axis=0)
plt.figure(figsize=(5, 5))
plt.title("Row Summed Averages")
plt.imshow(row_avgs, cmap='gray')
plt.xticks(np.arange(0, 10, step=1))
plt.show(block=False)
```



Code given below can be used to sum along the y-axis and display the resulting image.

```
col_avgs = np.sum(averages, axis=1)
plt.figure(figsize=(5, 5))
plt.title("Col Summed Averages")
plt.imshow(col_avgs, cmap='gray')
plt.xticks(np.arange(0, 10, step=1))
plt.show(block=False)
```



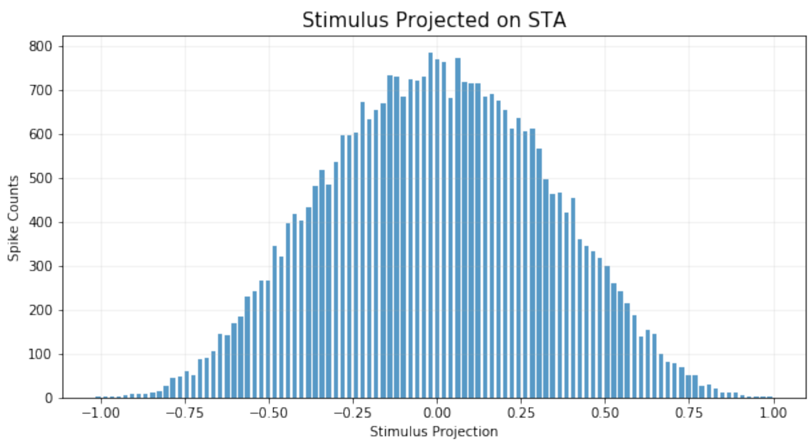
From these obtained images we can conclude that this given LGN cell is again selective for the luminous intensity. Also, it is selective only for the pixels around 7 and 9, we cannot conclude any information about the other pixels, since no change occurs at those pixels. Also, from the time-axis, which has length of ten, we can conclude that it is not possible to make an observation before time step three.

c) As the final part of this question, we are asked to project the stimulus on the STA at a single time step prior to spike. To obtain the projection, we will be using the *Frobenius Inner Product*, which is defined as follows,

$$\begin{aligned}\langle \mathbf{A}, \mathbf{B} \rangle_{\mathrm{F}} =& \overline{A}_{11} B_{11} + \overline{A}_{12} B_{12} + \cdots + \overline{A}_{1m} B_{1m} \\ &+ \overline{A}_{21} B_{21} + \overline{A}_{22} B_{22} + \cdots + \overline{A}_{2m} B_{2m} \\ &\vdots \\ &+ \overline{A}_{n1} B_{n1} + \overline{A}_{n2} B_{n2} + \cdots + \overline{A}_{nm} B_{nm}\end{aligned}$$

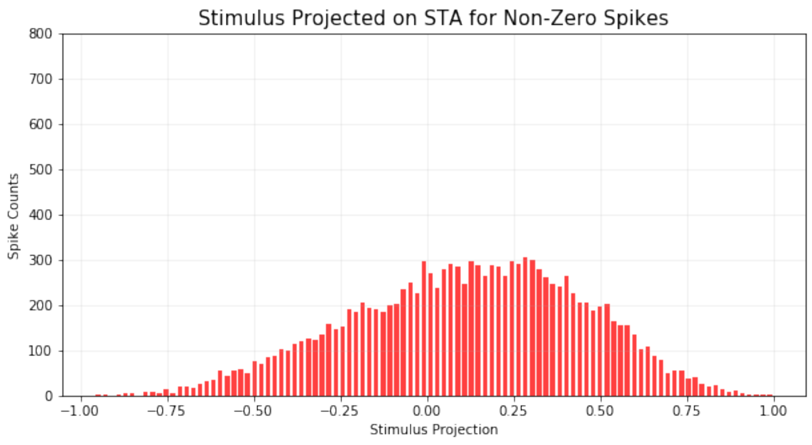
In the following lines of code given below, we have applied the *Frobenius Inner Product*, to stimulus and STA images to obtain the projections of the stimulus on the STA images. Then code to display their projections as histogram follows.

```
stim_proj_sta = np.zeros(len(counts))
for i in range(len(counts)):
    for j in range(len(averages[:,0,0])):
        stim_proj_sta[i] += np.inner(averages[:,j,0], stim[:,j,i])

stim_proj_sta /= np.max(stim_proj_sta)
plt.figure(figsize=(10, 5))
plt.title("Stimulus Projected on STA", fontsize= 15)
plt.ylabel("Spike Counts")
plt.xlabel("Stimulus Projection")
plt.grid(b=1, alpha=.2)
plt.hist(stim_proj_sta, bins=100, alpha=.75, rwidth=.66)
plt.show()
```



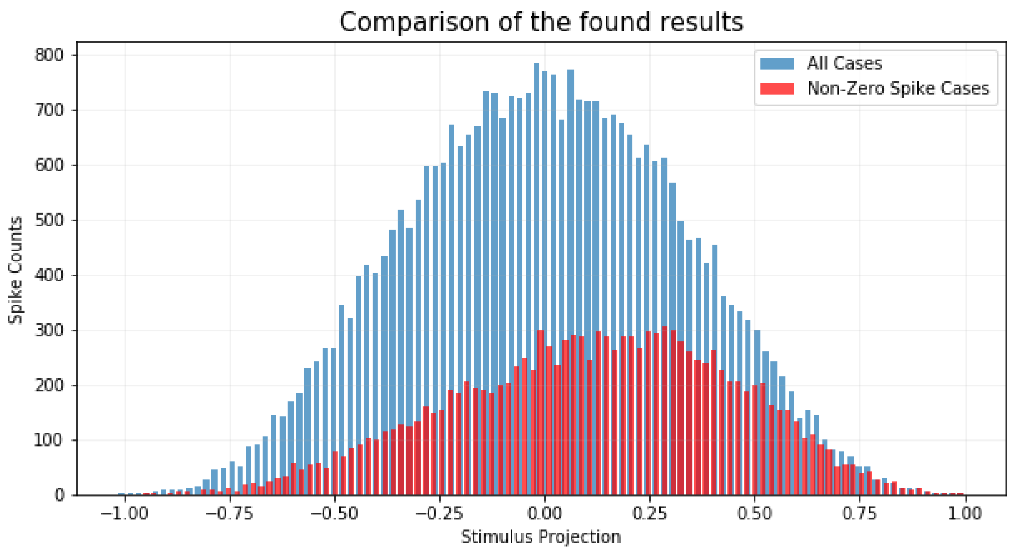
Therefore, we compute the stimulus projections only where we encounter spikes using *Frobenius Inner Product* and we create an other histogram for that. To show both of the histograms in the same graph and compare them we have used one hundred bins for each and we normalize both to one.

```
non_zero_index = []
for i in range (len(counts)):
    if counts[i] != 0:
        non_zero_index.append(i)

non_zero_proj_sta = np.zeros(len(non_zero_index))
for i in range(len(non_zero_proj_sta)):
    for j in range(len(averages[:,0,0])):
        non_zero_proj_sta[i] += np.inner(averages[:,j,0], stim[:
 ↪,j,non_zero_index[i]])

non_zero_proj_sta /= np.max(non_zero_proj_sta)
plt.figure(figsize=(10, 5))
plt.title("Stimulus Projected on STA for Non-Zero Spikes",␣
 ↪fontsize= 15)
plt.ylabel("Spike Counts")
plt.xlabel("Stimulus Projection")
plt.grid(b=1, alpha=.2)
plt.ylim(0, 800)
plt.hist(non_zero_proj_sta, bins=100, alpha=0.75, color='red',␣
 ↪rwidth=.66)
plt.show()
```



Eventually using the following code, we display both of the histograms in the same graph and the corresponding graph is given after the code.

```
plt.figure(figsize=(10, 5))
all_cases = plt.hist(stim_proj_sta, bins=100, label='All Cases',␣
  ↪alpha=.7, rwidth=.66)
non_zeros = plt.hist(non_zero_proj_sta, bins=100, color='red',␣
  ↪label='Non-Zero Spike Cases', alpha=.7, rwidth=.66)
plt.legend()
plt.grid(b=1, alpha=.2)
plt.title('Comparison of the found results', fontsize= 15)
plt.ylabel("Spike Counts")
plt.xlabel("Stimulus Projection")
plt.savefig('my_figure.png')
plt.show()
```



From the comparison of the both graphs we can see that both looks like Gaussian distributions but also it can be observed that they have different means. When we project the stimulus for all cases, we can observe that most of our cases take values zero or close to zero, which means it has a mean of zero. On the other hand when we only project the stimulus if the spike count is not zero our mean shifts to the right. Eventually, we can say that using STA we can discriminate stimulus with spikes from all stimulus by projecting them on to STA.

Question 2

```python
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import scipy.signal as sig
```

These are the imports used through out this question. In addition to the imports from the previous question **mpl_toolkits** and **scipy.signal** are added in order to display 3D plots and take convolution respectively.

a) Initially, it has been asked from us to define a difference-of-gaussian (DOG) receptive field centered at the origin with respect to the given definition of the DOG function.

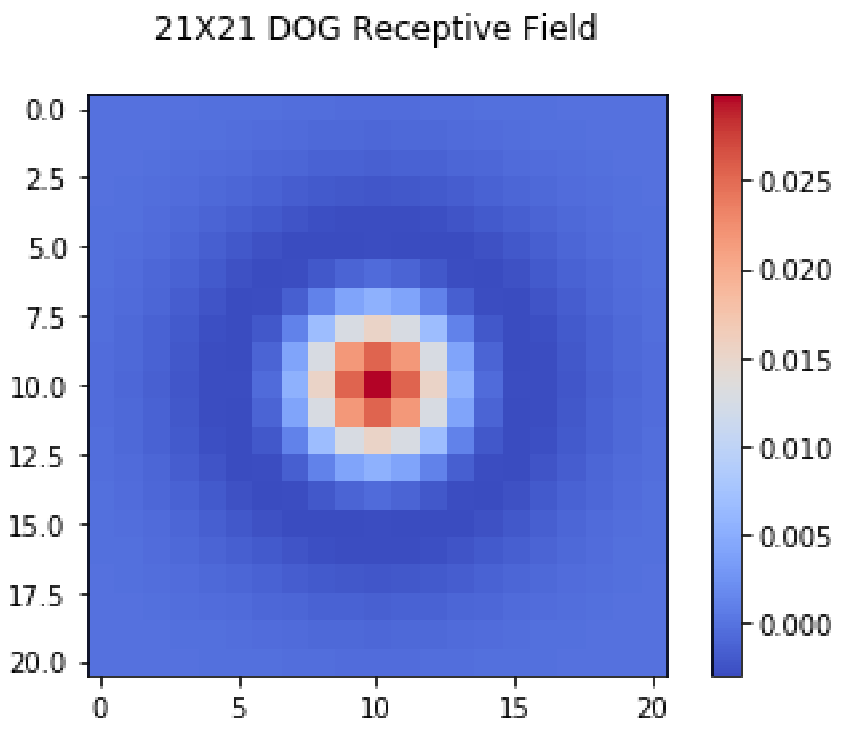
$$D(x,y) = \frac{1}{2\pi\sigma_c^2}e^{-(x^2+y^2)/2\sigma_c^2} - \frac{1}{2\pi\sigma_s^2}e^{-(x^2+y^2)/2\sigma_s^2}$$

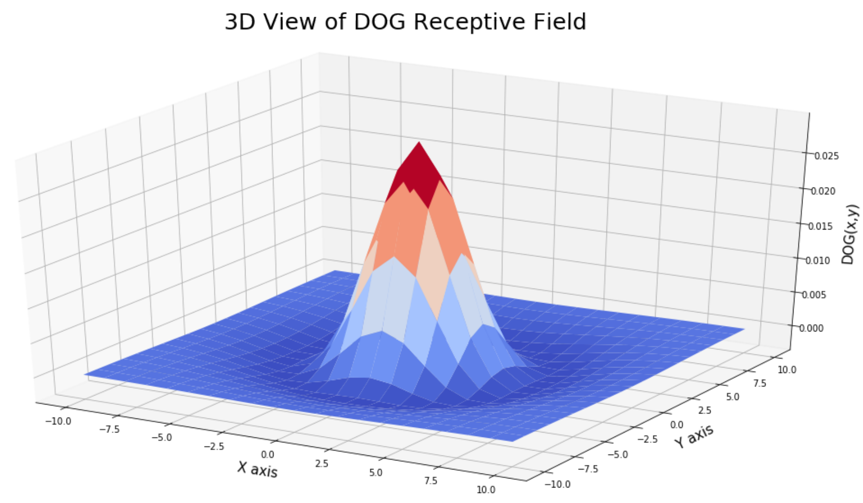
Afterwards, it is expected from us to sample this receptive field as a 21x21 matrix with the given parameters, $\sigma_c = 2$ pixels and $\sigma_s = 4$ pixels. In the code block given below, we have defined a DOG function with respect to the given DOG in the assignment.

```python
def DOG (x, y):
    sigma_c = 2
    sigma_s = 4
    return 1/(2*np.pi*sigma_c**2)*np.exp(-(x**2+y**2)/
  ↪(2*sigma_c**2))-1/(2*np.pi*sigma_s**2)*np.exp(-(x**2+y**2)/
  ↪(2*sigma_s**2))
```

Using the DOG function that we have defined according to the given equation, we have created a DOG center-surround receptive field and sampled it as a 21x21 matrix as expected. Then, that matrix is shown as a 2D image from the top view and also 3D view of that sample is displayed using the code following the sampling code, in the code cell given below.

```
dog_receptive_field = np.zeros((21, 21))
for i in range(-10, 11):
    for j in range(-10, 11):
        dog_receptive_field[10+i][10+j] = DOG(i,j)

plt.title('21X21 DOG Receptive Field\n')
plt.imshow(dog_receptive_field, cmap='coolwarm')
plt.colorbar()
plt.show()

plt.figure(figsize=(20, 10))
X = Y = np.linspace(-10, 10, 21)
X, Y = np.meshgrid(X, Y)
ax = plt.axes(projection='3d')
ax.set_title('3D View of DOG Receptive Field\n', fontsize=25);
ax.set_xlabel("\nX axis", fontsize= 15)
ax.set_ylabel("\nY axis", fontsize= 15)
ax.set_zlabel("\nDOG(x,y)", fontsize= 15)
ax.plot_surface(X, Y, dog_receptive_field, cmap='coolwarm',␣
 ↪edgecolor='none')
plt.show(block=False)
```

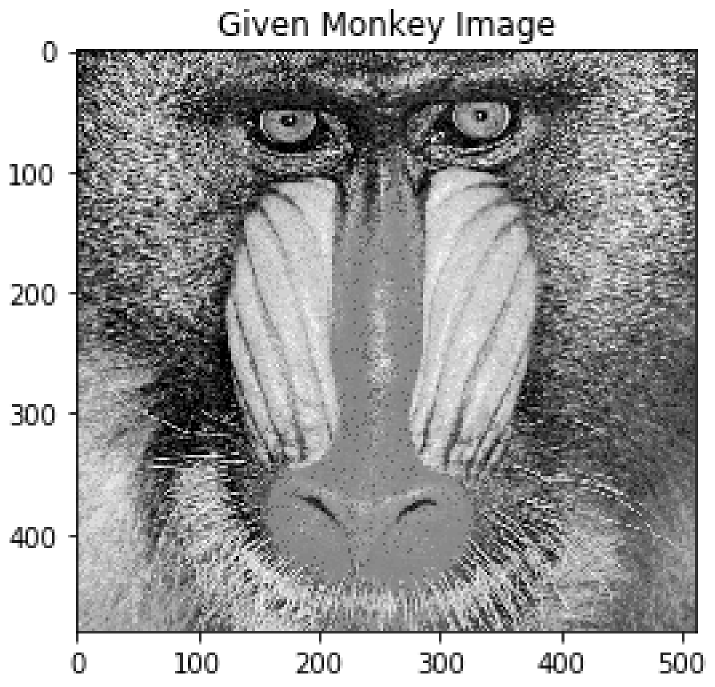


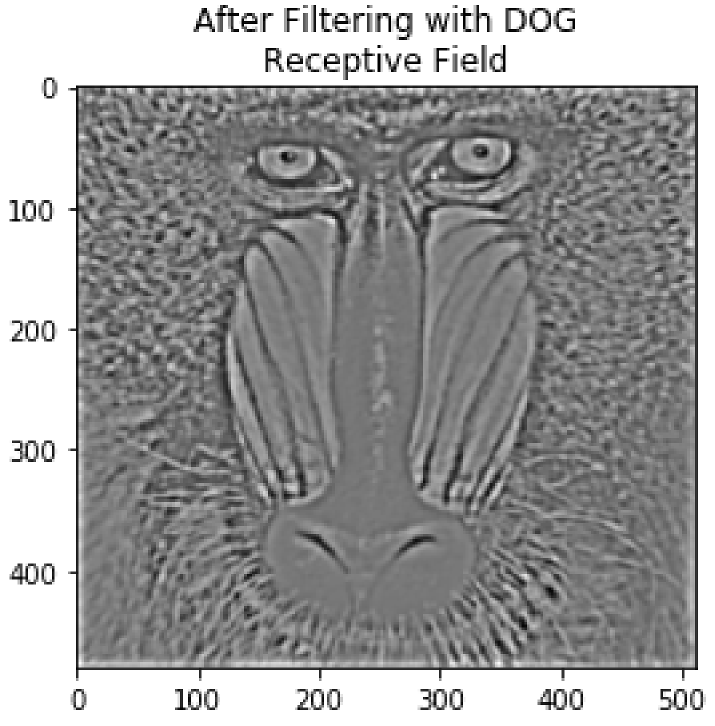


b) We are provided that neurons in LGN have DOG receptive fields and we should suppose that there is a separate LGN neuron with a receptive field centered on each pixel in the image. With respect to this given definition of LGN neuron, we can simply convolute the DOG receptive field, which we have found in the previous question, and the provided image, for computing the response of that LGN cell.

Operation defined above is conducted using the code given below. For easing the comparison, provided image is displayed initially. Therefore, given image is convolute with the DOG receptive field pixel by pixel and the resulting image is displayed.

```
plt.figure()
monkey = plt.imread('hw2_image.bmp')
plt.imshow(monkey)
plt.title("Given Monkey Image")
plt.show()
plt.figure()
conv_img = sig.convolve(monkey[:, :, 0], dog_receptive_field,␣
 ↪mode='same')
plt.imshow(conv_img, cmap='gray')
plt.title("After Filtering with DOG\nReceptive Field")
plt.show()
```



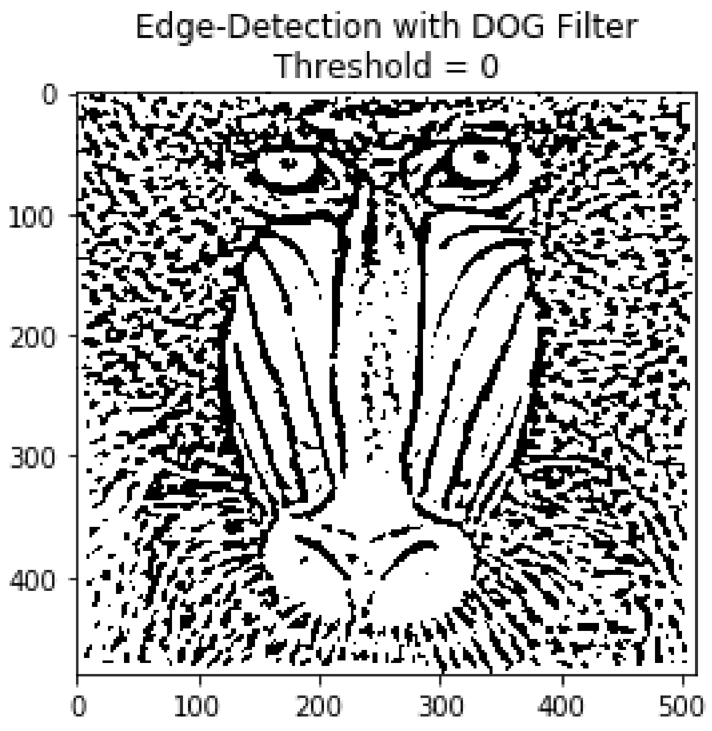


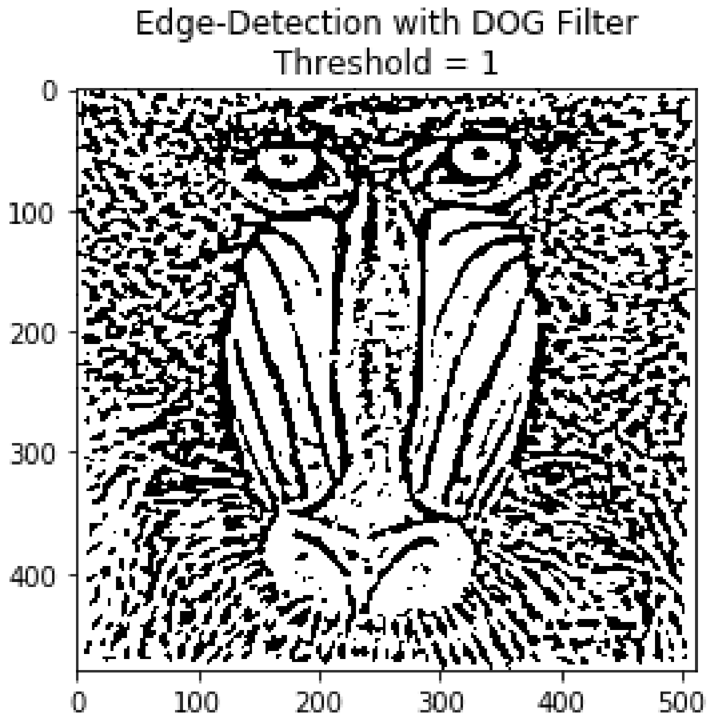
Comparing the two images that has been displayed, we can see that our neurons in LGN cells are responsible of detecting the edges that we see. After applying the DOG filter, edges in the image became more obvious.

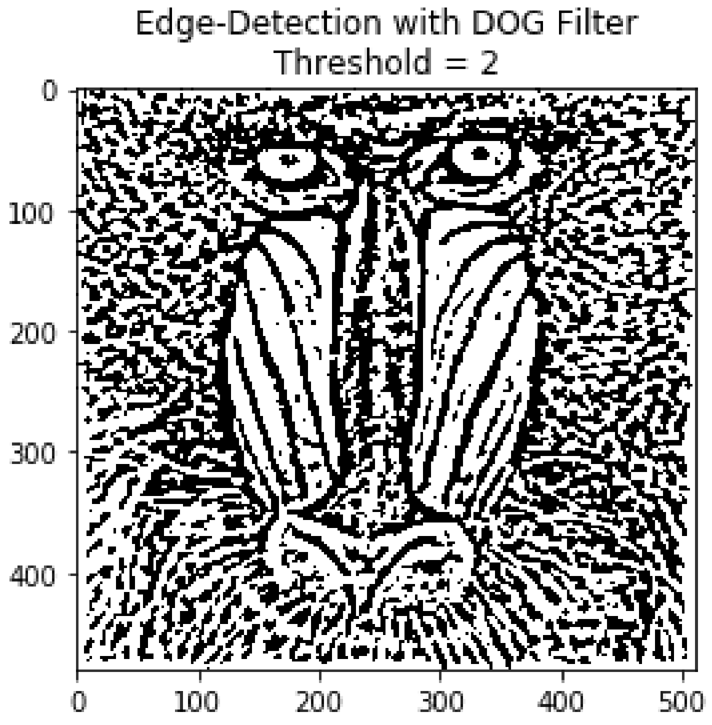
c) In this part, for making our observations easier, it has been asked from us to detect the edges of the image after applying the DOG filter to the image, by image thresholding method. Main purpose of image thresholding is to partition a image into foreground and background.

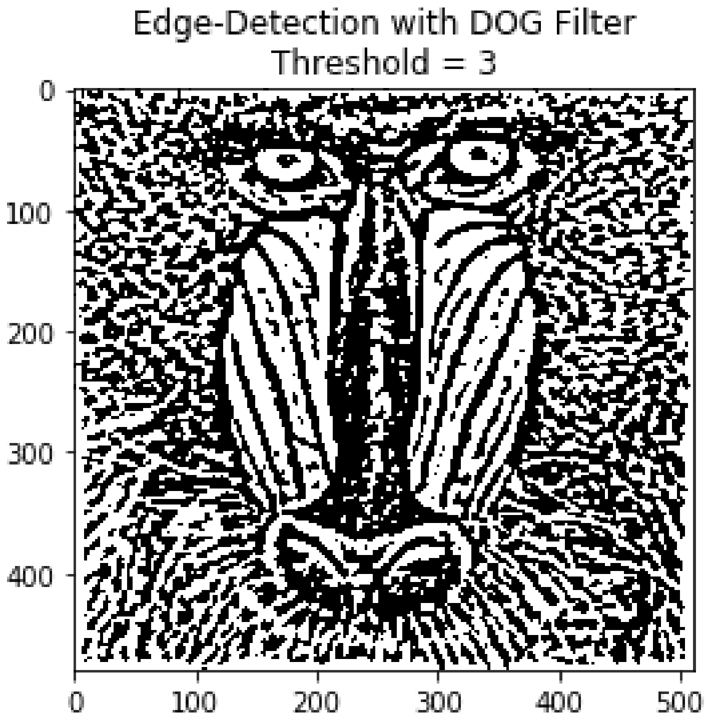
For partitioning our image, threshold algorithm is written and applied as it is described in the given assignment. To determine the value of the threshold, different values will be taken as threshold value and the results will be examined in order to determine the best threshold value.

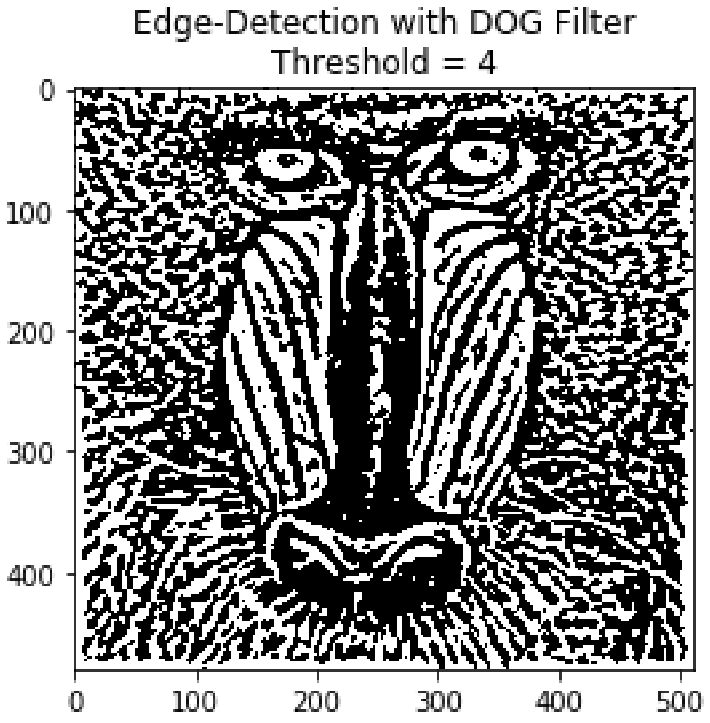
```
def detect_edge(img, threshold):
    result_img = np.zeros(np.shape(img))
    for i in range(np.shape(result_img)[0]):
        for j in range(np.shape(result_img)[1]):
            if img[i,j] >= threshold:
                result_img[i,j] = 1
            else:
                result_img[i,j] = 0
    return result_img

for i in range(5):
    plt.figure()
    plt.title("Edge-Detection with DOG Filter\nThreshold = %d" %i)
    plt.imshow(detect_edge(conv_img, i), cmap='gray')
```




Edge-Detection with DOG Filter
Threshold = 1
0
100
200
300
400
0
100
200
300
400
500


Edge-Detection with DOG Filter
Threshold = 2
0
100
200
300
400
0
100
200
300
400
500


Edge-Detection with DOG Filter
Threshold = 3
0
100
200
300
400
0
100
200
300
400
500


Edge-Detection with DOG Filter
Threshold = 4
0
100
200
300
400
0
100
200
300
400
500

Among the displayed images, images with threshold value 0 and 1 seems to give better results for edge detection. Also, it can be said that as the threshold value increases, performance of image thresholding gets worse for our case. So, we can choose threshold value as 1 and conclude that it is an optimal choice for edge detection.

d) Similar to **Part A**, we are provided equation of Gabor and we are expected to build a sample receptive field as a 21x21 matrix. Provided equation is as follows

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

in the given equation, $\vec{k}(\theta)$ is given as a unit vector with orientation θ. $k(\vec{\theta})$ can be written as $\langle\cos(\theta), \sin(\theta)\rangle$. Therefore, we know that for a vector to be orthogonal to another, their dot product must be 0. Respectively we can define $\vec{k}_\perp(\theta)$ as $\langle-\sin(\theta), \cos(\theta)\rangle$ With respect to the given equation and orthogonal form of $\vec{k}$, gabor function is defined in the following code block.

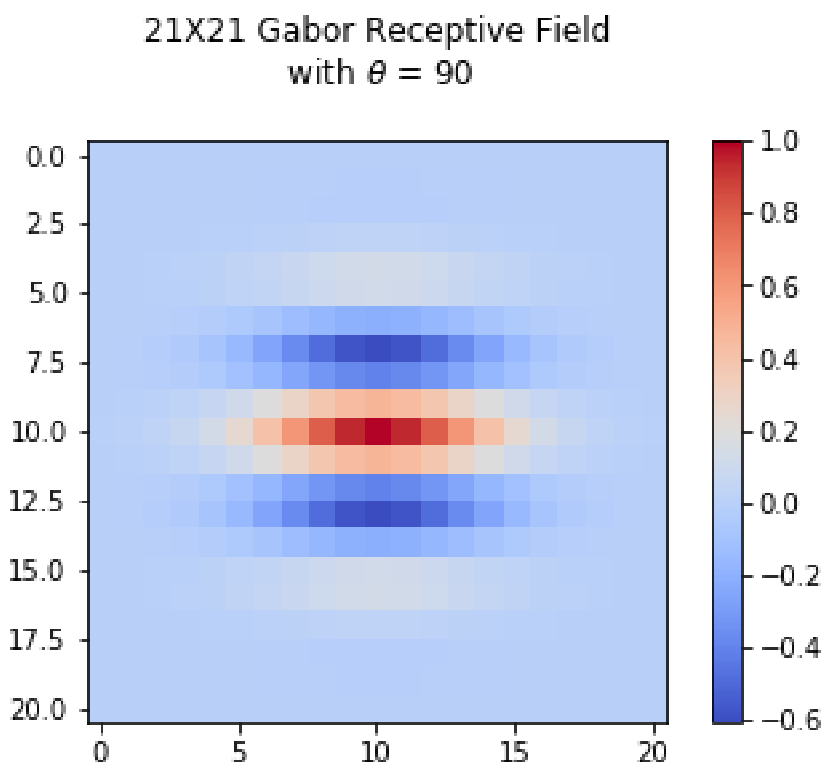
```python
def gabor(x, th):
    theta = th
    sigma_l = sigma_w = 3
    lmbd = 6
    phi = 0
    k_theta = np.array([np.cos(theta), np.sin(theta)])
    k_orthogonal = np.array([-np.sin(theta), np.cos(theta)])
    k_dot_x = k_theta.dot(x)
    ko_dot_x = k_orthogonal.dot(x)
    return np.exp(-(k_dot_x**2)/(2*(sigma_l**2))-(ko_dot_x**2)/
 ↪(2*(sigma_w**2)))*np.cos(2*np.pi*ko_dot_x/lmbd+phi)
```

In the code below, similar to defining a DOG receptive field, a gabor receptive field is defined.

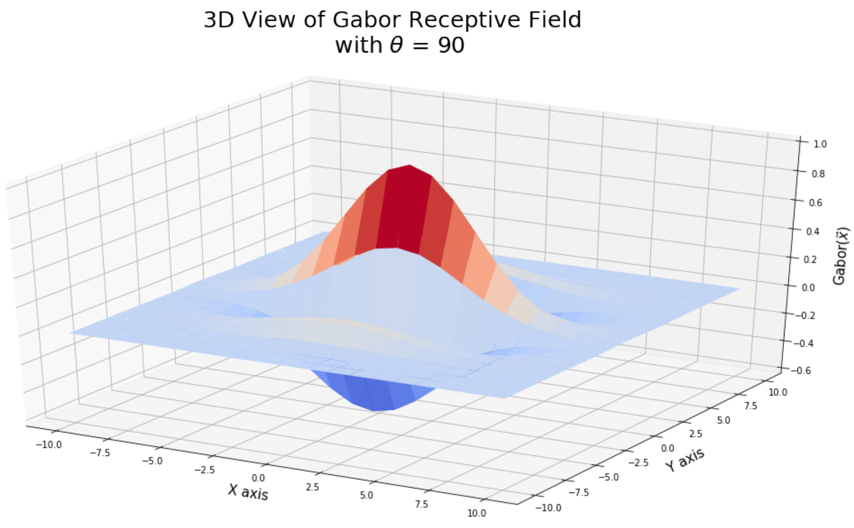
```python
gabor_receptive_field_90 = np.zeros((21, 21))
for i in range(-10, 11):
    for j in range(-10, 11):
        gabor_receptive_field_90[10+i][10+j] = gabor(np.
 ↪array([i,j]), np.pi/2)
```

Using the codes below, Gabor receptive field from top view and also 3D view of it from some angle is displayed respectively.

```
plt.title('21X21 Gabor Receptive Field\n' r' with $\theta$ = 90'␣
    ↪'\n')
plt.imshow(gabor_receptive_field_90, cmap='coolwarm')
plt.colorbar()
plt.show()
```

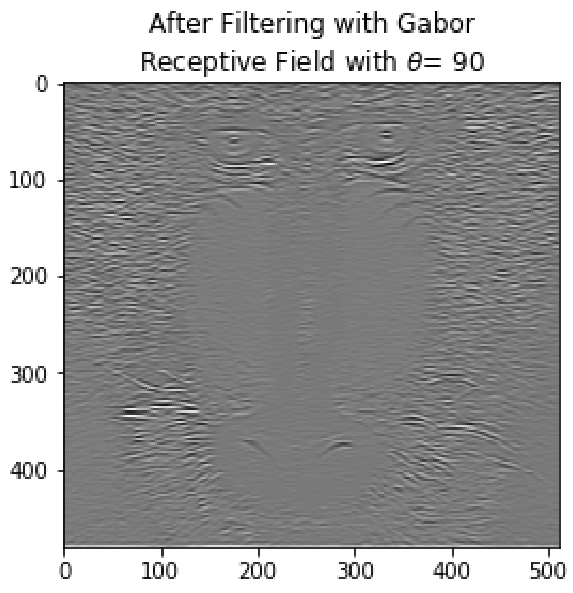


```
plt.figure(figsize=(20, 10))
X = Y = np.linspace(-10, 10, 21)
X, Y = np.meshgrid(X, Y)
ax = plt.axes(projection='3d')
ax.set_title('3D View of Gabor Receptive Field \n' r' with␣
    ↪$\theta$ = 90' '\n', fontsize=25);
ax.set_xlabel("\nX axis", fontsize= 15)
ax.set_ylabel("\nY axis", fontsize= 15)
ax.set_zlabel("\n"r"Gabor($\vec{x}$)", fontsize= 15)
ax.plot_surface(X, Y, gabor_receptive_field_90, cmap='coolwarm',␣
    ↪edgecolor='none')
plt.show(block=False)
```



e) This time cells in V1 have Gabor receptive fields and we assume that seperate V1 neurons are assigned to each pixel in the image. As we did in the **Part B**, we will convolve Gabor receptive field and the given image in order to obtain the output as an image.

```
gabor_conv_90 = sig.convolve(monkey[:, :, 0],␣
 ↪gabor_receptive_field_90, mode='same')
plt.imshow(gabor_conv_90, cmap='gray')
plt.title("After Filtering with Gabor\nReceptive Field with "␣
 ↪r"$\theta$= 90")
plt.show()
```



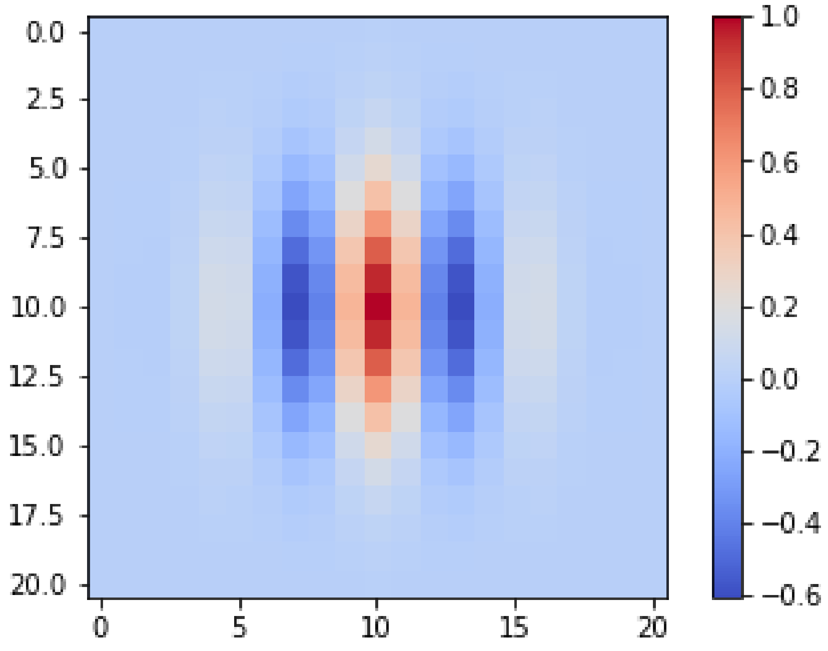
From the image we have obtained, we can see that Gabor receptive is not only detecting the edges, it is detecting the edges with some direction or orientation.

f) Finallly, we are asked to construct three more gabors with $\theta = 0, \pi/6$ and $\pi/3$. Then provided image will be convoluted with each one of them separately and all of their results will be summed up to obtain all edges detected image.

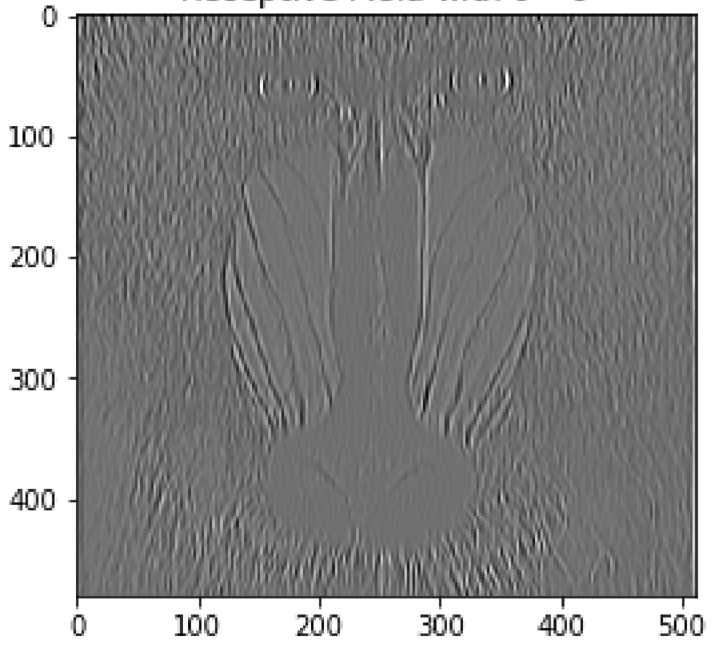
```python
gabor_receptive_field_0 = np.zeros((21, 21))
for i in range(-10, 11):
    for j in range(-10, 11):
        gabor_receptive_field_0[10+i][10+j] = gabor(np.
 ↪array([i,j]), 0)

gabor_receptive_field_30 = np.zeros((21, 21))
for i in range(-10, 11):
    for j in range(-10, 11):
        gabor_receptive_field_30[10+i][10+j] = gabor(np.
 ↪array([i,j]), np.pi/6)

gabor_receptive_field_60 = np.zeros((21, 21))
for i in range(-10, 11):
    for j in range(-10, 11):
        gabor_receptive_field_60[10+i][10+j] = gabor(np.
 ↪array([i,j]), np.pi/3)
```

```python
plt.title('21X21 Gabor Receptive Field\n' r' with $\theta$ = 0'␣
 ↪'\n')
plt.imshow(gabor_receptive_field_0, cmap='coolwarm')
plt.colorbar()
plt.show()

gabor_conv_0 = sig.convolve(monkey[:, :, 0],␣
 ↪gabor_receptive_field_0, mode='same')
plt.imshow(gabor_conv_0, cmap='gray')
plt.title("After Filtering with Gabor\nReceptive Field with "␣
 ↪r"$\theta$= 0")
plt.show()
```

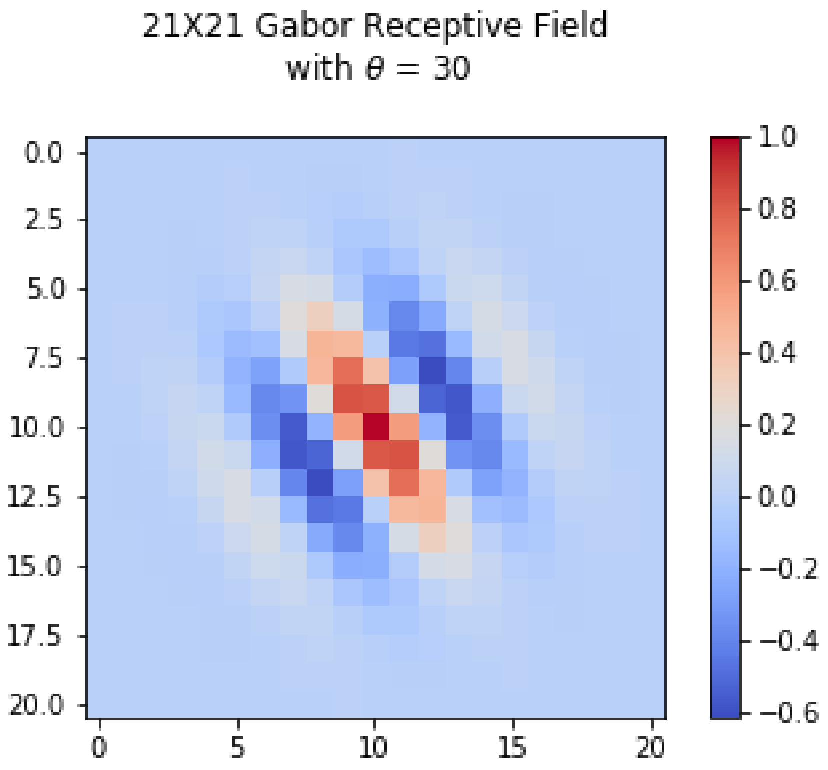
21X21 Gabor Receptive Field
with θ = 0

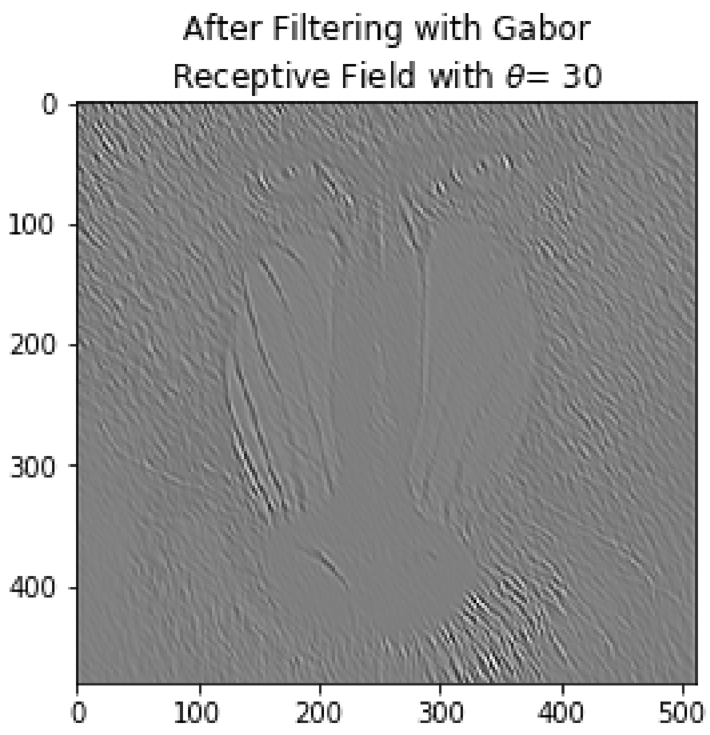


After Filtering with Gabor
Receptive Field with θ= 0

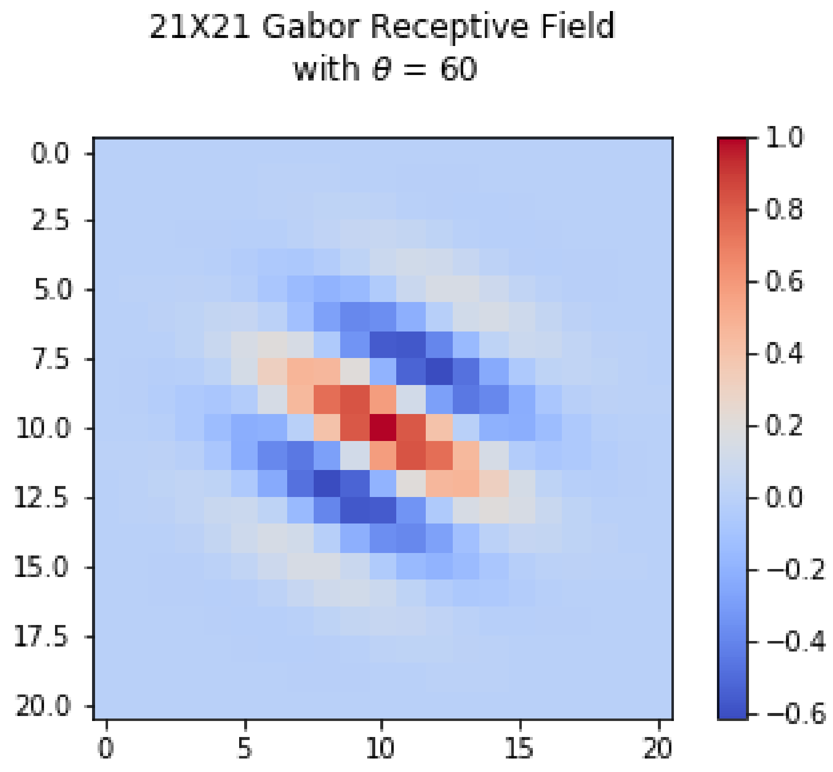


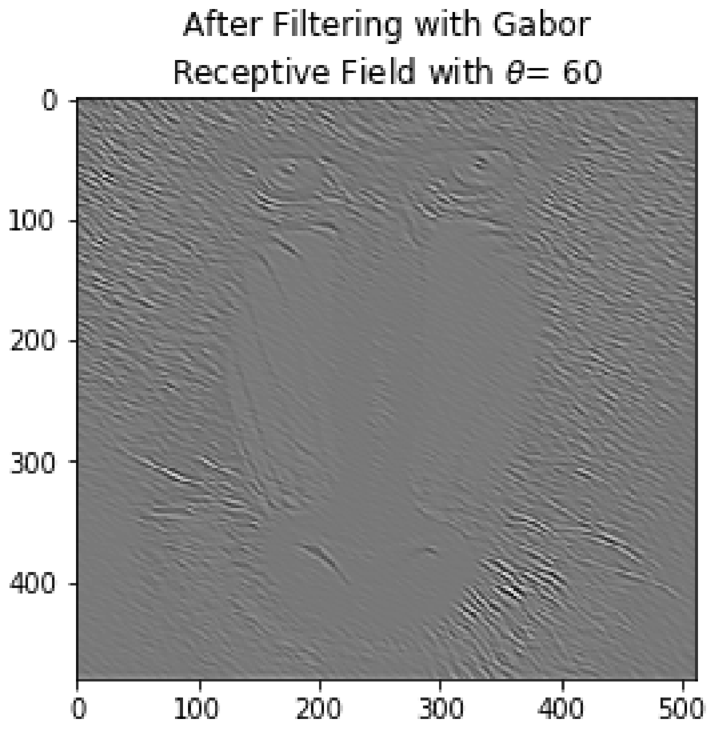
```
plt.title('21X21 Gabor Receptive Field\n' r' with $\theta$ = 90'␣
  ↪'\n')
plt.imshow(gabor_receptive_field_30, cmap='coolwarm')
plt.colorbar()
plt.show()

gabor_conv_30 = sig.convolve(monkey[:, :, 0],␣
  ↪gabor_receptive_field_30, mode='same')
plt.imshow(gabor_conv_30, cmap='gray')
plt.title("After Filtering with Gabor\nReceptive Field with "␣
  ↪r"$\theta$= 30")
plt.show()
```



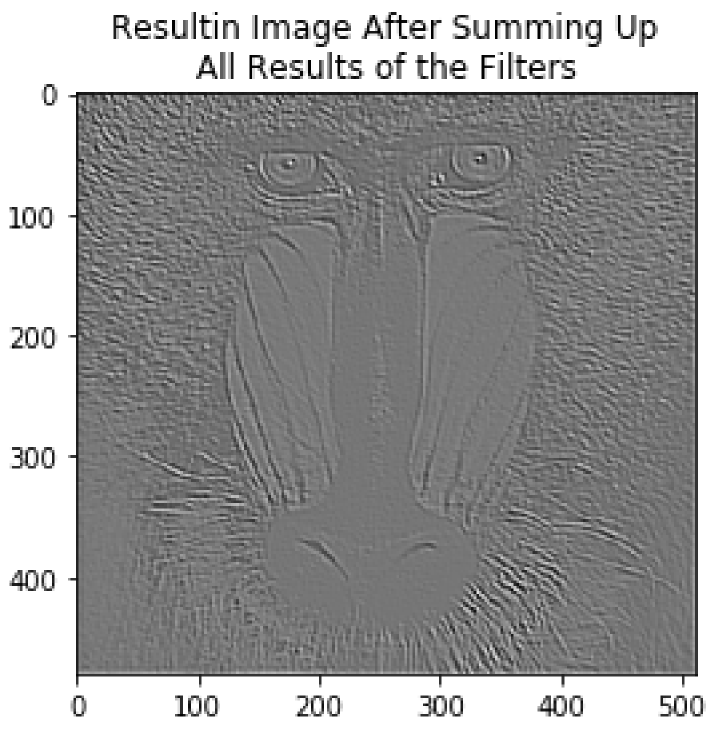


```
plt.title('21X21 Gabor Receptive Field\n' r' with $\theta$ = 90'␣
  ↪'\n')
plt.imshow(gabor_receptive_field_60, cmap='coolwarm')
plt.colorbar()
plt.show()

gabor_conv_60 = sig.convolve(monkey[:, :, 0],␣
  ↪gabor_receptive_field_60, mode='same')
plt.imshow(gabor_conv_60, cmap='gray')
plt.title("After Filtering with Gabor\nReceptive Field with "␣
  ↪r"$\theta$= 60")
plt.show()
```


21X21 Gabor Receptive Field
with θ = 60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
5
10
15
20
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

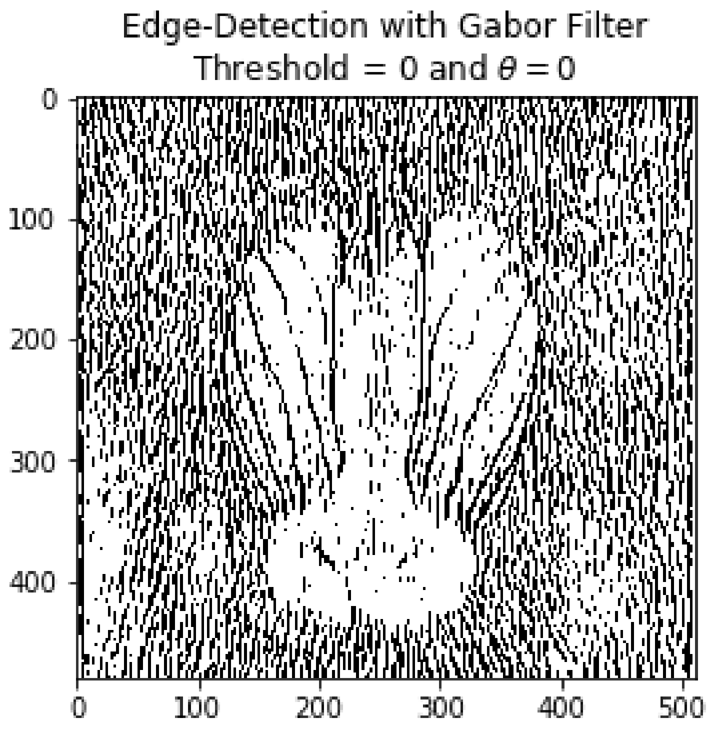

After Filtering with Gabor
Receptive Field with θ= 60
0
100
200
300
400
0
100
200
300
400
500

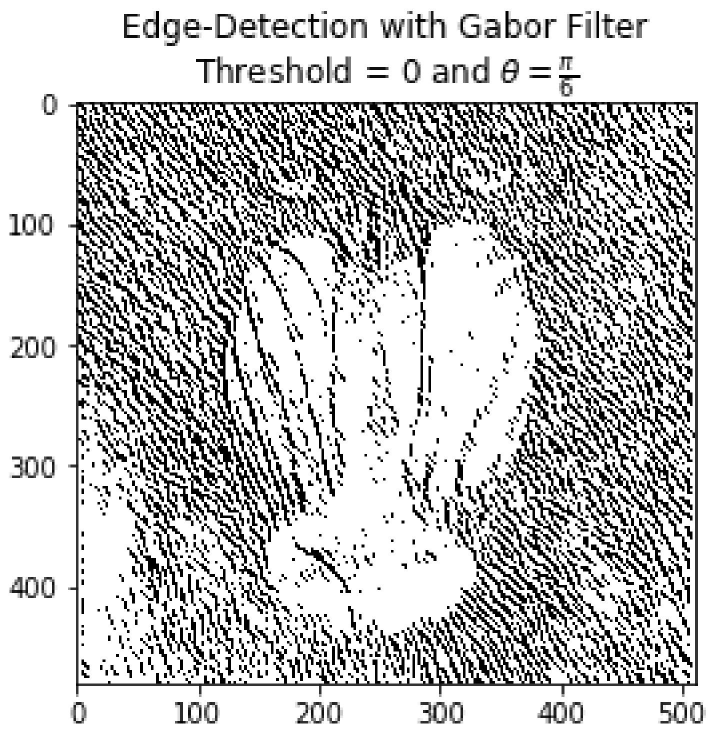
```
summed_image = gabor_conv_0 + gabor_conv_30 + gabor_conv_60 +␣
  ↪gabor_conv_90
plt.imshow(summed_image, cmap='gray')
plt.title("Resultin Image After Summing Up\nAll Results of the␣
  ↪Filters")
plt.show()
```

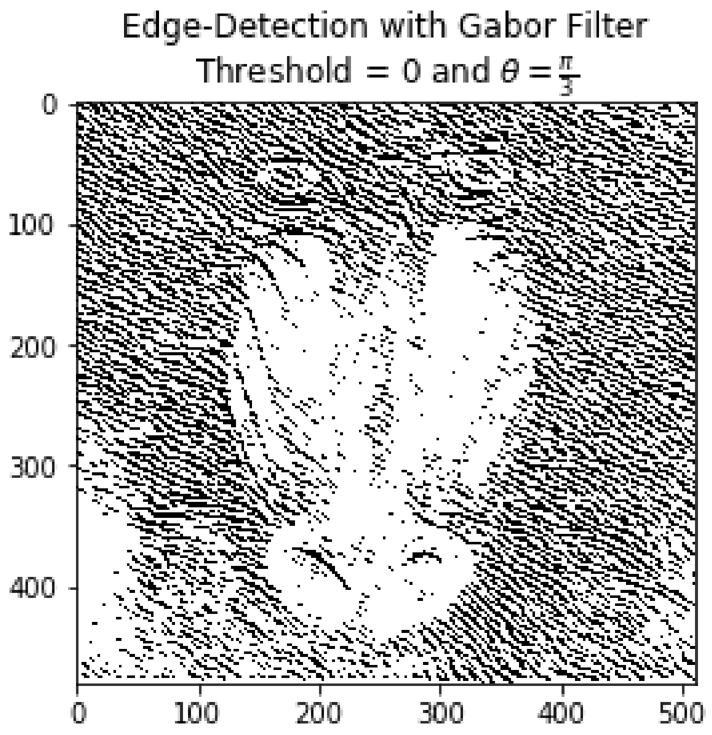


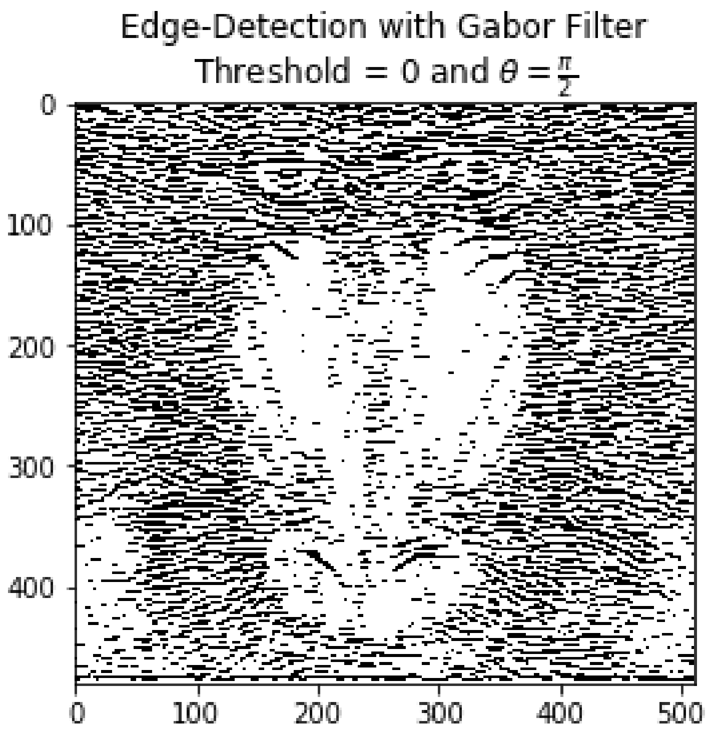
```
gabor_edge_0 = detect_edge(gabor_conv_0, 0)
plt.figure()
plt.title("Edge-Detection with Gabor Filter\nThreshold = 0 and "␣
  ↪r"$\theta = 0$")
plt.imshow(gabor_edge_0, cmap='gray')

gabor_edge_30 = detect_edge(gabor_conv_30, 0)
plt.figure()
plt.title("Edge-Detection with Gabor Filter\nThreshold = 0 and "␣
  ↪r"$\theta = \frac{\pi}{6}$")
plt.imshow(gabor_edge_30, cmap='gray')

gabor_edge_60 = detect_edge(gabor_conv_60, 0)
```

```
plt.figure()
plt.title("Edge-Detection with Gabor Filter\nThreshold = 0 and "␣
  ↪r"$\theta = \frac{\pi}{3}$")
plt.imshow(gabor_edge_60, cmap='gray')

gabor_edge_90 = detect_edge(gabor_conv_90, 0)
plt.figure()
plt.title("Edge-Detection with Gabor Filter\nThreshold = 0 and "␣
  ↪r"$\theta = \frac{\pi}{2}$")
plt.imshow(gabor_edge_90, cmap='gray')
plt.show(block=False)
```

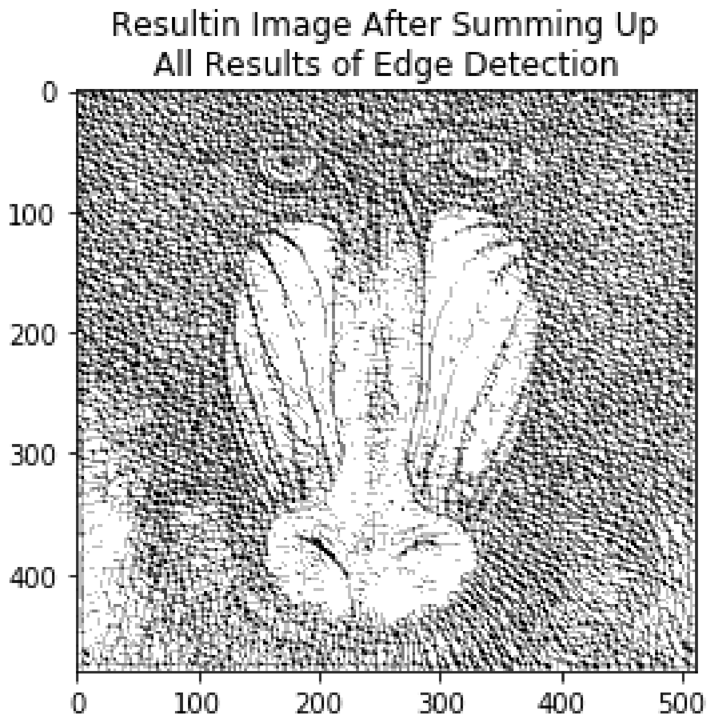



Edge-Detection with Gabor Filter
Threshold = 0 and $\theta = \frac{\pi}{6}$
0
100
200
300
400
500


Edge-Detection with Gabor Filter
Threshold = 0 and $\theta = \frac{\pi}{3}$
0
100
200
300
400
500



```
edge_summed_img = gabor_edge_0 + gabor_edge_30 + gabor_edge_60 +␣
  ↪gabor_edge_90
plt.imshow(edge_summed_img, cmap='gray')
plt.title("Resulting Image After Summing Up\nAll Results of Edge␣
  ↪Detection")
plt.show()
```


Resultin Image After Summing Up
All Results of Edge Detection
0
100
200
300
400
0
100
200
300
400
500