Question 1:

In this question, we are given a cat's LGN cells response to two-dimensional visual images. This data is given as spike counts of 15.6 ms bins and the shown images at the corresponding times.

a) Calculate the STA images for each of the 10 time steps before each spike and show them, then comment on spatio-temporal selectivity of the LGN cells.

LGN, Lateral geniculate nucleus cells are a few layers of cells which can show strong directional and orientational selectivity with their combined effort. STA means spike triggered average; It is computed by taking the average of the stimulus at time points where a spike occurs. In this question, we are asked to compute STA for each of the 10 time steps before each spike. Then I will inspect the resultant images to determine the spatio-temporal selectivity of the LGN cells.

To calculate the STA's and the rest of this homework I used python 3.8, python works almost fully on library's and without them it almost doesn't have any functions for the scientific calculations we what we want to do here. Because of this I imported the related packages to do the tasks we are given. I paid attention to not use any library that doesn't already come with the Anaconda install. So, all the libraries in this code already should come with python and they shouldn't require an extra download. I wrote the import part of my code down below. If your python doesn't have the used packages and you have pip, you can install the missing packages with command such as "python -m pip install numpy, python -m pip install scipy or python -m pip install matplotlib."

```
import sys # the primer already has it

import numpy as np # this brings simple matrix operations to python
import matplotlib.pyplot as plt # this brings nice loking plots to python

import scipy.io as sio # this is used to load MATLAB files
from scipy import signal as sig # this brings 2D convolution.
from mpl_toolkits import mplot3d # used for 3D plots
```

Now if we return to the question given in part a. First, I defined a STA function. This function calculates the STA for n time steps before each spike, for a given stimulus and spike count data pair. n is the time step which is given by us. The function does this by looping trough the spike counts in each time bin then multiplies each spike count with the stimulus image n time steps before that time bin and adding it to a sum. Then it sum the total number of spikes in the spike counts data and divides this sum by it to find the average of the stimulus images which cause a spike in n time steps. The code of the STA function is given below.

```
def STA(stim, counts, time_step): # the function to calculate STA for a given
#times step for a stimulus and spike count data pair.
```

```
    total_sum = np.zeros( (np.shape(stim)[0], np.shape(stim)[1]) ) # we will add
#everything spike tringering stimulus to this.

    for i in range (time_step, len(counts)): #looping trough the data set

        total_sum[:,:] += stim[:,:, i - time_step] * counts[i]

    spike_count = np.sum(counts[time_step:]) # finding total spike count
    average = total_sum / spike_count # taking the spike tringing stimules
#average.
    return average
```

Then to start solving this question I first loaded the date given to us for this question in to Python. The data is given in a format which is compatible with MATLAB because of this I needed to use a special function which is used to load (.mat) type files. Then I split the given data to different arrays, these arrays are the stimulus and the spike counts. The code for this is given below.

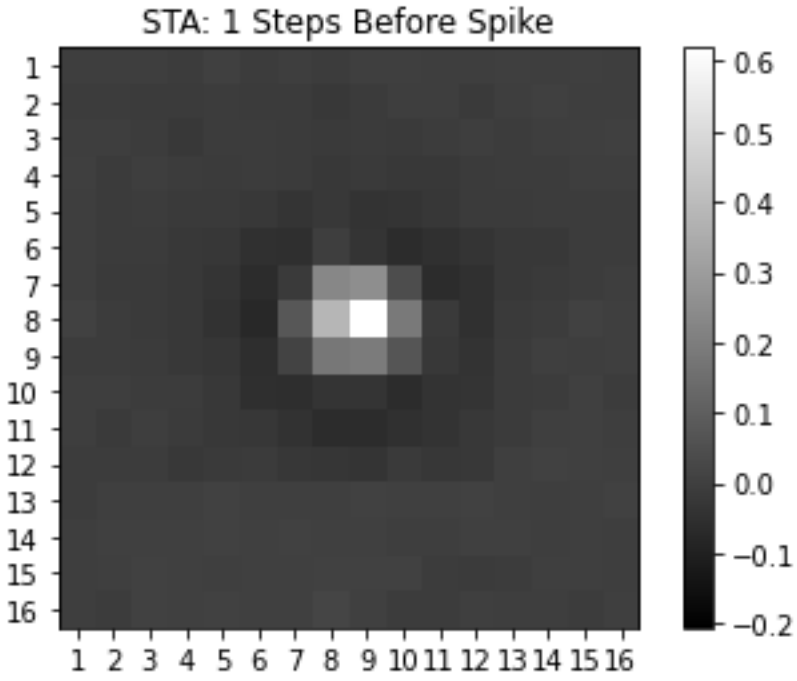
```
    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']
```

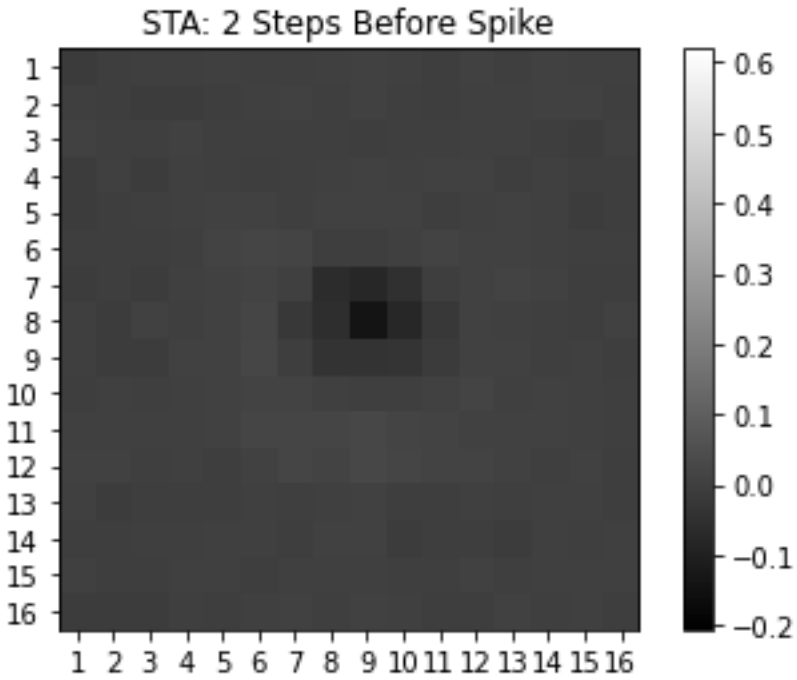
Then I looked at Python variable explorer to see these two arrays' sizes. Counts is (32767, 1) and stim is (16, 16, 32767). This is same as what the homework guideline told. So my loading of the data was successful.

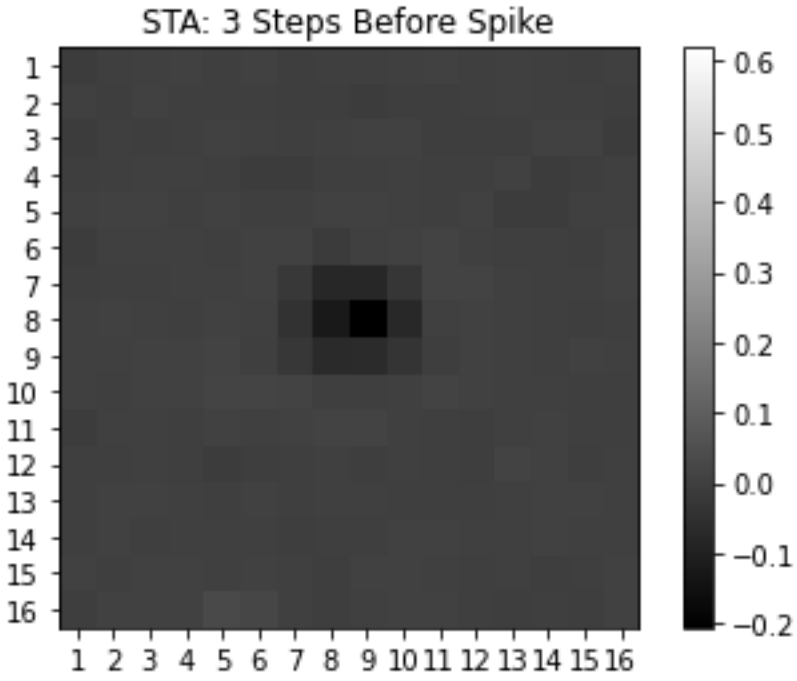
After this I wrote a code which uses my STA function on the given data 10 times with time steps 1 to 10. Then I drew these STA images as grayscale colormaps. The code for this is given below.

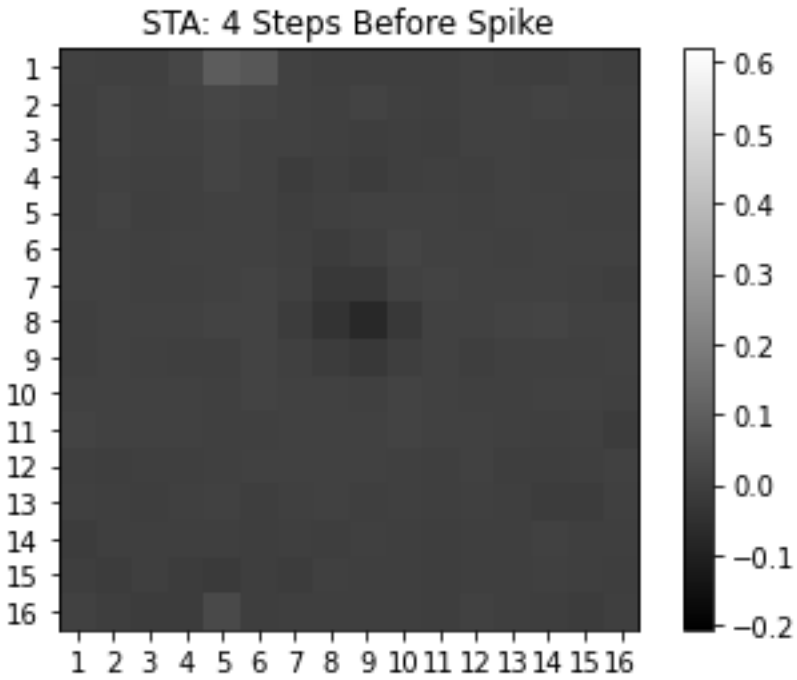
```
    for i in range(10):

        plt.figure()
        plt.imshow( averages[:,:, i] , cmap='gray', vmin=np.min(averages), vmax=n
p.max(averages))
        plt.colorbar()
        plt.title('STA: ' + str(i + 1) + ' Steps Before Spike')
        plt.yticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
        plt.xticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
```

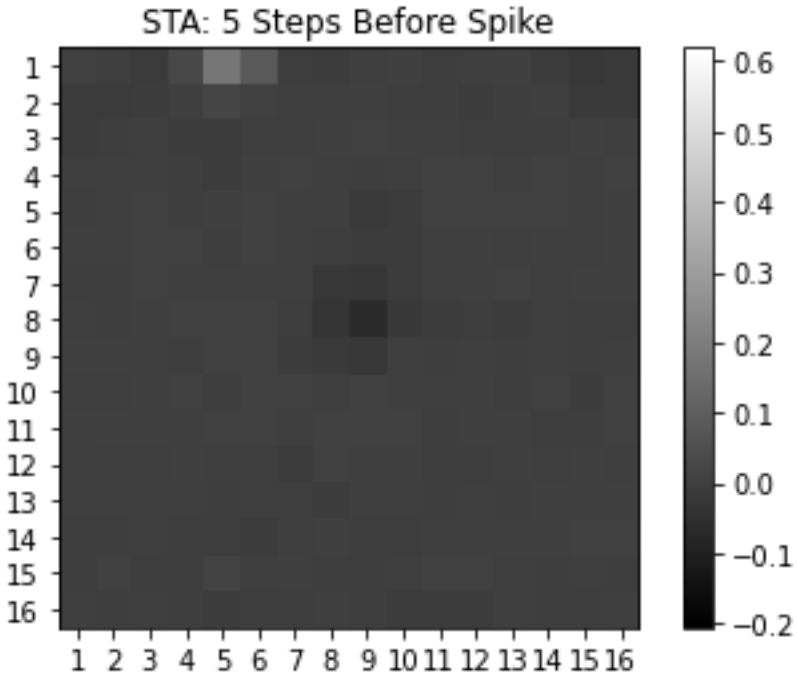
Something important to note in this code is, I specify the "vmin" and "vmax" of these graphs, so the graphs have the identical display windowing. This gives these graphs as its output:

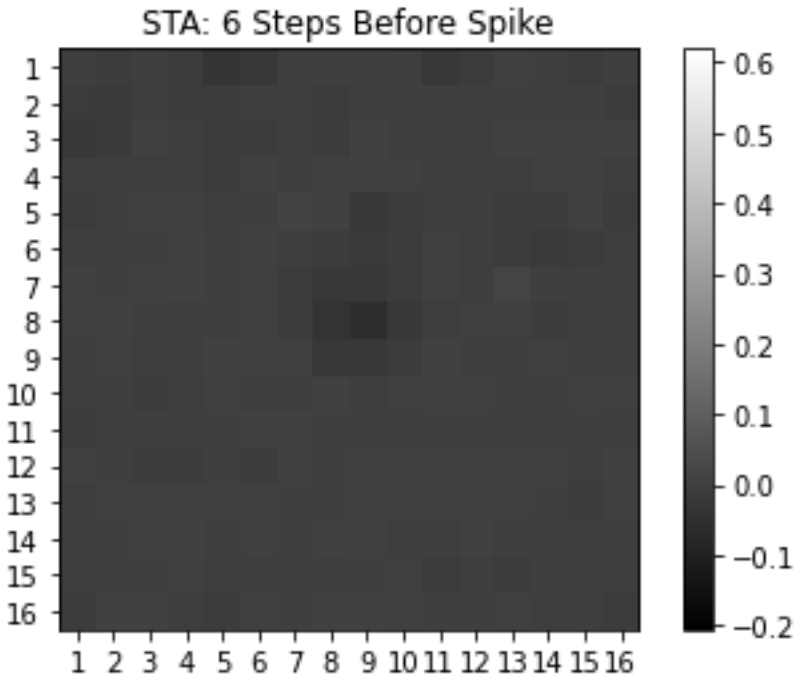

STA: 1 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2

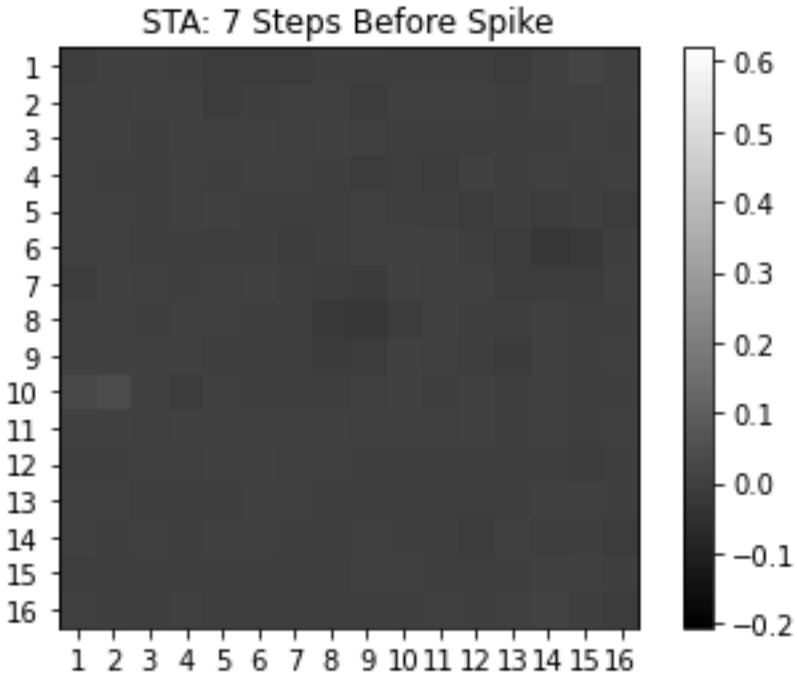

STA: 2 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2

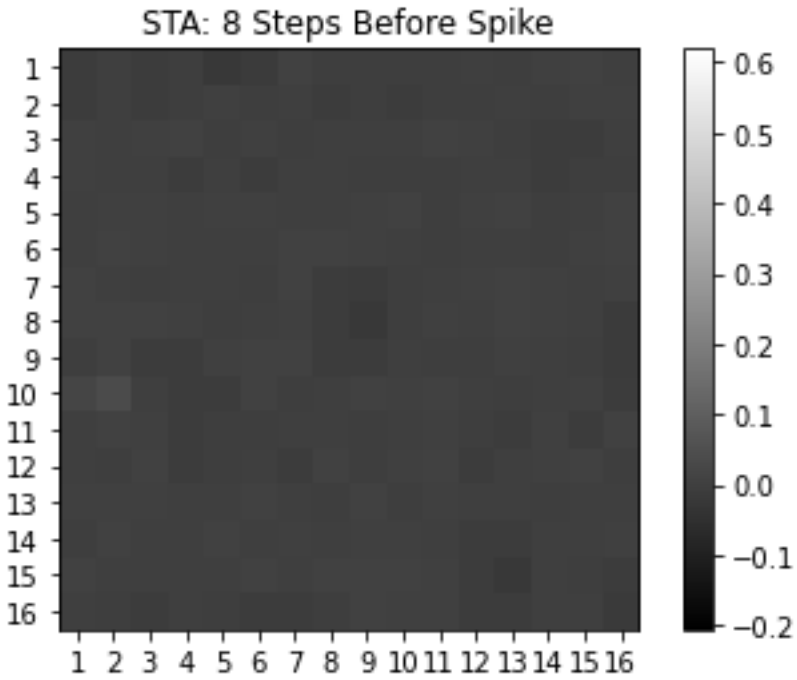

STA: 3 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2

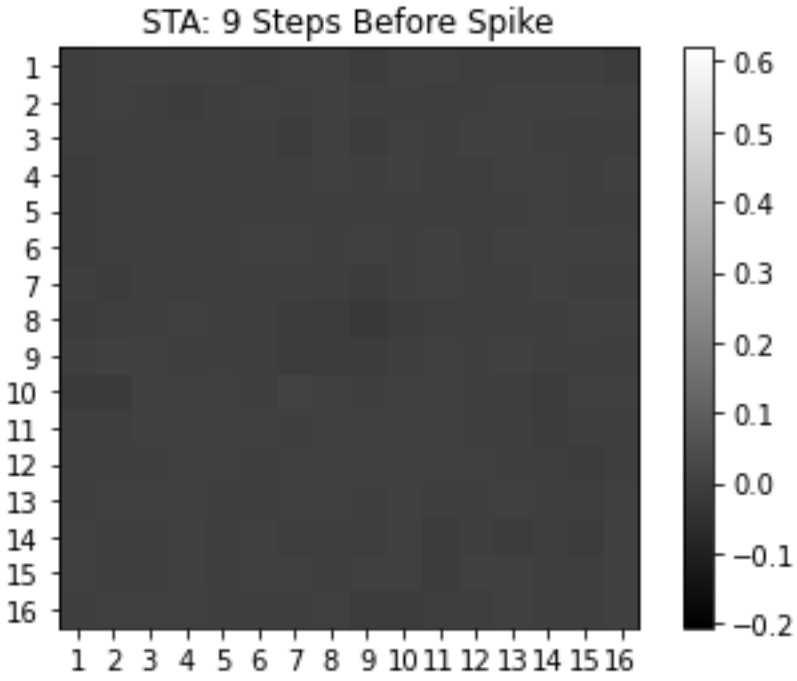

STA: 4 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2

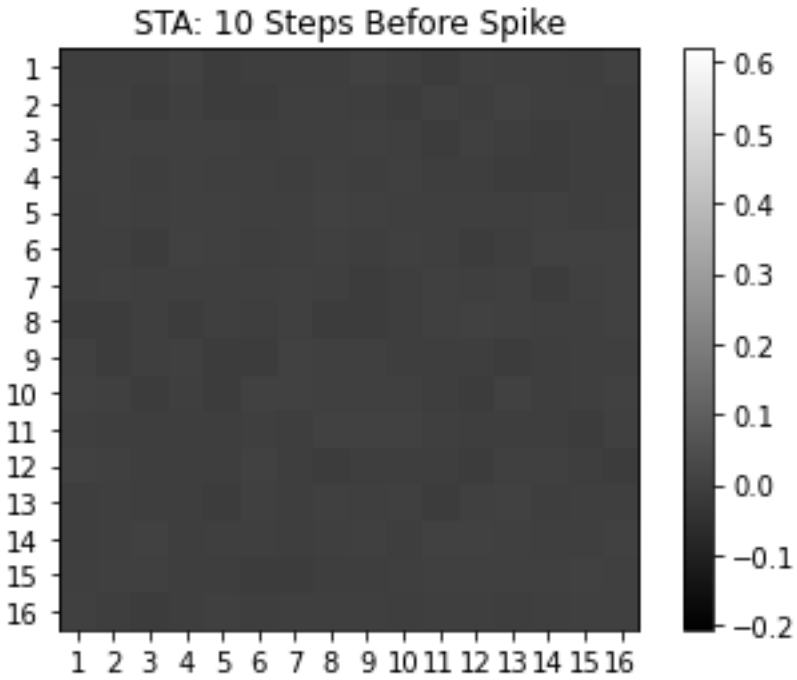

STA: 5 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 6 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2


STA: 7 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2


STA: 8 Steps Before Spike
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0.6 0.5 0.4 0.3 0.2 0.1 0.0 −0.1 −0.2





First I will call the stimulus as it is light intensity because we are never told what it is. The data for the stimulus is made out of 1's and -1's. If we take -1 as a face value light intensity it does not make sense but -1 might represent darkness while 1 represents light and light intensity is the term which makes the most sense in an experiment such as this. From the graphs especially the 1 step before spike, we can say this LGN cell is selective for a bright center spot. It seems like LGN cell wants the area other than the center to have natural light intensity close to zero

point of the given data. Also, if we look at 2, 3 and 4 time steps before STA's, LGN cell also seem to like when the center first gets darker then jumps to brightness value while everywhere has a natural light intensity close to zero point. However, this is an average so LGN cell might not care about the light intensity of the area other then the center and the none center areas light intensity might average to zero while it is taking random values around zero. If we look at the time steps 8, 9 and 10 we can see they have a homogeneous value close to zero. This means these timesteps probably don't affect the spike generation or have a very small effect on it also the natural average brightness of the image is around zero. From this we can confidently say this LGN cell is selective for when the center of the image first darkens then instantly lights up and brightens.

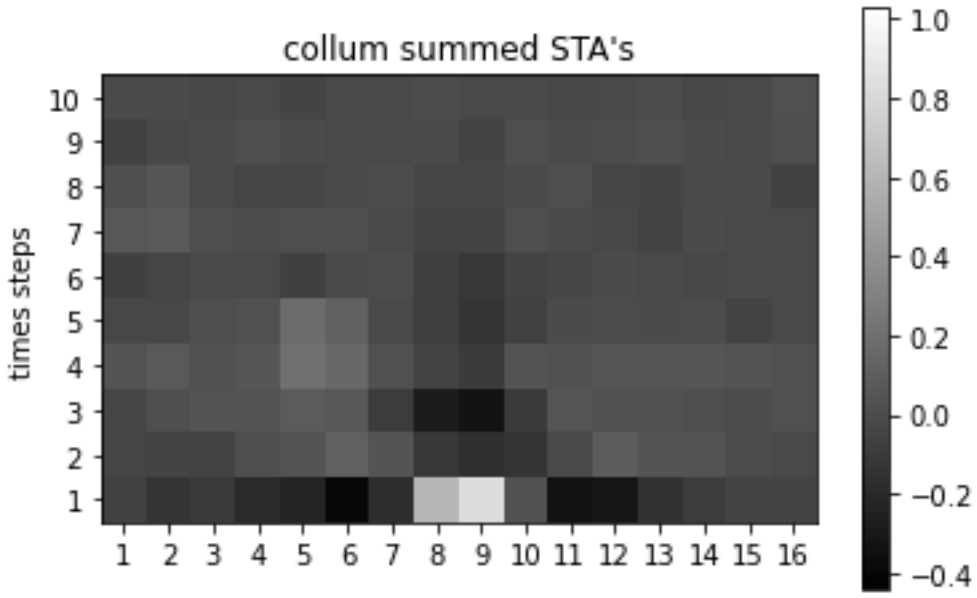
b) Describe the changes in STA images across time. Sum the STA images over one of the spatial dimensions. Describe the temporal selectivity of the LGN cell. Is the matrix space-time separable?

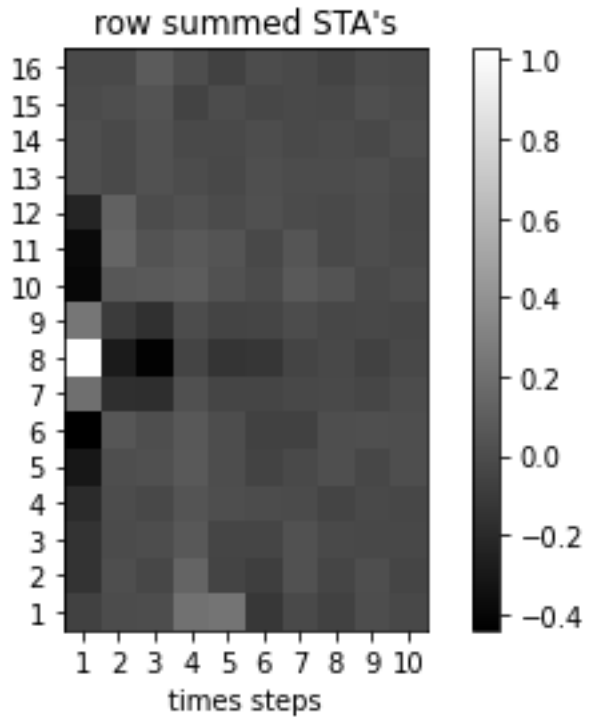
Python's numpy.sum() function is perfect for this question because it can take an axis as an input parameter and only sum values through the axis. As an example if we give it a (3, 4, 5) sized matrix with axis = 0 it will return a (4, 5) sized matrix, if we give it the (3, 4, 5) sized matrix with axis = 1 it will return a (3, 5) sized matrix. I used this function to sum the STA images over both of the axes of the STA images to see it more clearly. Then drew plots of both of these new graphs. The code for this is given below.

```
    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']

    averages = np.zeros( ( np.shape(stim)[0], np.shape(stim)[1] , 10 ) ) # will f
ill this with STA's for each of the 10 timesteps.

    for i in range(10): # calculating the STA's for the 10 time steps.

        averages[:,:, i ] = STA(stim, counts, i + 1 )


    collum_summed_average = np.sum(averages, axis = 0) # the code which sums the
values trough the collums of the matrix

    plt.figure()
    plt.imshow( collum_summed_average.transpose() , cmap='gray' , origin='lower')
 # I drew the transpose here to enphesize how we sum trough the collums of the ma
trix
    plt.colorbar()
```

```
    plt.title("collum summed STA's")
    plt.yticks(np.arange(0, 10, step=1) , np.arange(1, 11, step=1))
    plt.xticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
    plt.ylabel("times steps")


    row_summed_average = np.sum(averages, axis = 1) # the code which sums the val
ues trough the rows of the matrix

    plt.figure()
    plt.imshow( row_summed_average , cmap='gray' , origin='lower')
    plt.colorbar()
    plt.title("row summed STA's")
    plt.xticks(np.arange(0, 10, step=1) , np.arange(1, 11, step=1))
    plt.yticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
    plt.xlabel("times steps")
```

This code and many other codes repeats codes from other part because I wanted each part of the answer to have its own function which did not require other parts function to be run first. Now if we return to this code it gives these output graphs.





If we look at these graphs, we can see this LGN cells first wants a darkness which than gets bright at the center. These graphs show this selectivity because we can see a line which first gets dark (around 3 time steps before) then brightens up (at the 1 time step before point) at the middle of these graphs (columns 8 and 9 on the column summed STA's and row 7, 8, 9 on the row summed STA'S). Because the LGN cell wants a special order of changes (first darkness then brightness), this matrix is not space time separable.

c) Project the stimulus onto the STA image at a single time step prior to the spike. Obtain the projection for each time sample by computing the Frobenius inner product between the stimulus image and the STA image. Create a histogram from all stimulus projections, and another histogram from stimulus projections at time bins where a non-zero spike count was observed. Use identical binning for the two histograms, and normalize each histogram to a maximum of 1. Compare the histograms with a bar plot.

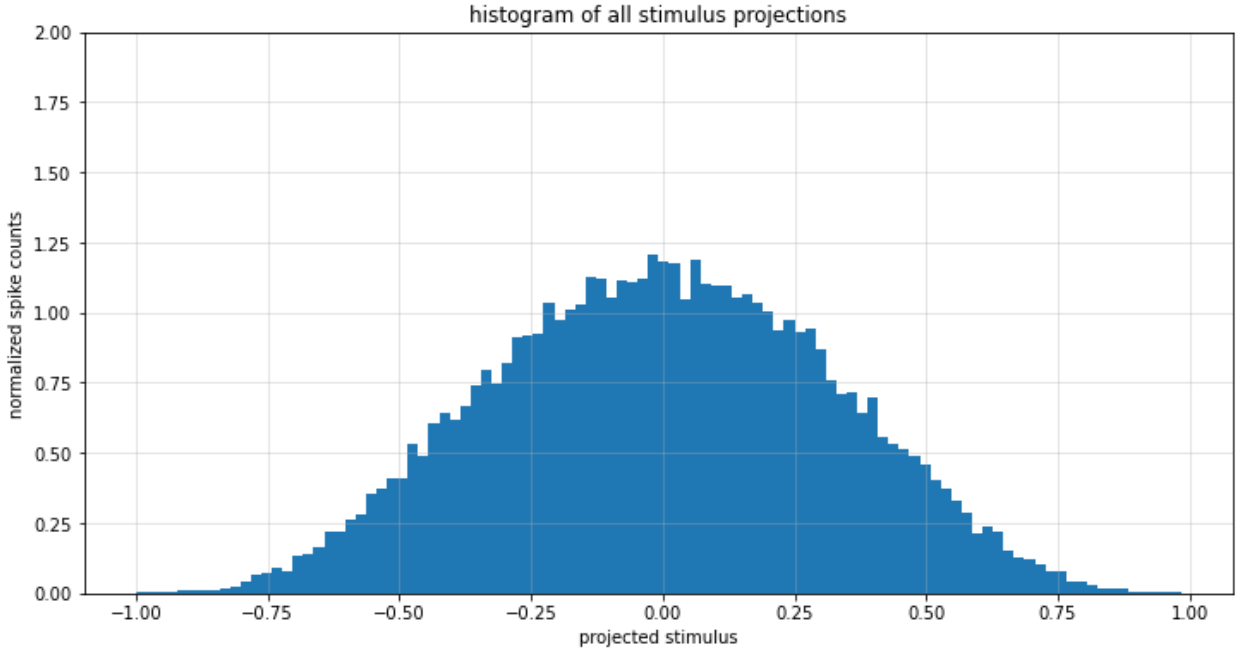
First we are asked to compute the Frobenius inner product between the stimulus and STA image we calculated. Frobenius inner product is shown in equation 1. We are doing it here because normally when using the STA we would open the stimulus matrix as a vector and open the STA as a vector and calculate their dot product to find a single number. How big this number is can be seen as how similar STA and the stimulus are. This opening the matrixes and then taking their dot product is just the same way to explain Frobenius inner product which is shown in equation 1, so we are not doing anything out of the ordinary.

$$\langle A, B \rangle_F = \begin{matrix} A_{11}B_{11} + & A_{12}B_{12} + & \cdots + & A_{1m}B_{1m} \\ + A_{21}B_{21} + & A_{22}B_{22} + & \cdots + & A_{2m}B_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ + A_{n1}B_{n1} + & A_{n2}B_{n2} + & \cdots + & A_{nm}B_{nm} \end{matrix} \quad (1)$$

To compute the Frobenius inner product, I used the inner product function on all the columns of STA and the stimulus then summed their results. I did this because I couldn't find a Frobenius inner product function that didn't require the installation of an external package. Using this, I computed each stimulus's Frobenius inner product with the one step before STA and appended the result to an array. Then I drew its histogram normalized to 1 with 100 bins. The code which does this is given below.

```
    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']

    averages = np.zeros( ( np.shape(stim)[0], np.shape(stim)[1] , 10 ) ) # will
#fill this with STA's for each of the 10 timesteps.

    for i in range(10): # calculating the STA's for the 10 time steps.

        averages[:,:, i ] = STA(stim, counts, i + 1 )


    stimulus_projected_on_STA = np.zeros(len(counts)) # we will fill this with
#the results of each stimuluses projection.
    for i in range(len(counts)):  # we look at every mesurement point
        for j in range( np.shape(averages)[1] ): # this for loop is for the
#Frobenius inner product calculation

            stimulus_projected_on_STA[i] += np.inner(averages[:,j,0],stim[:,j,i])


    stimulus_projected_on_STA = stimulus_projected_on_STA /
np.max(np.abs(stimulus_projected_on_STA)) # making the maximum 1


    plt.figure(figsize=(12, 6)) # this lets me decide how big the plot should be
    plt.title("histogram of all stimulus projections")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which
#increases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(stimulus_projected_on_STA, density = True , bins=100)
```

This prints the graph given below.

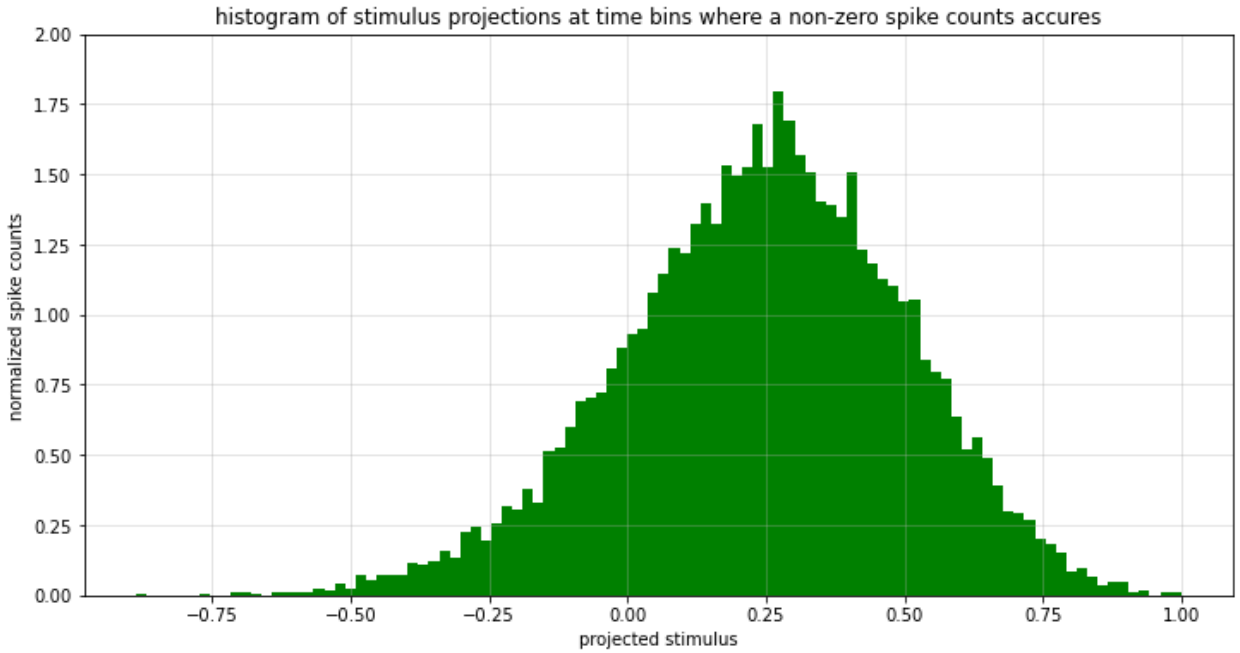


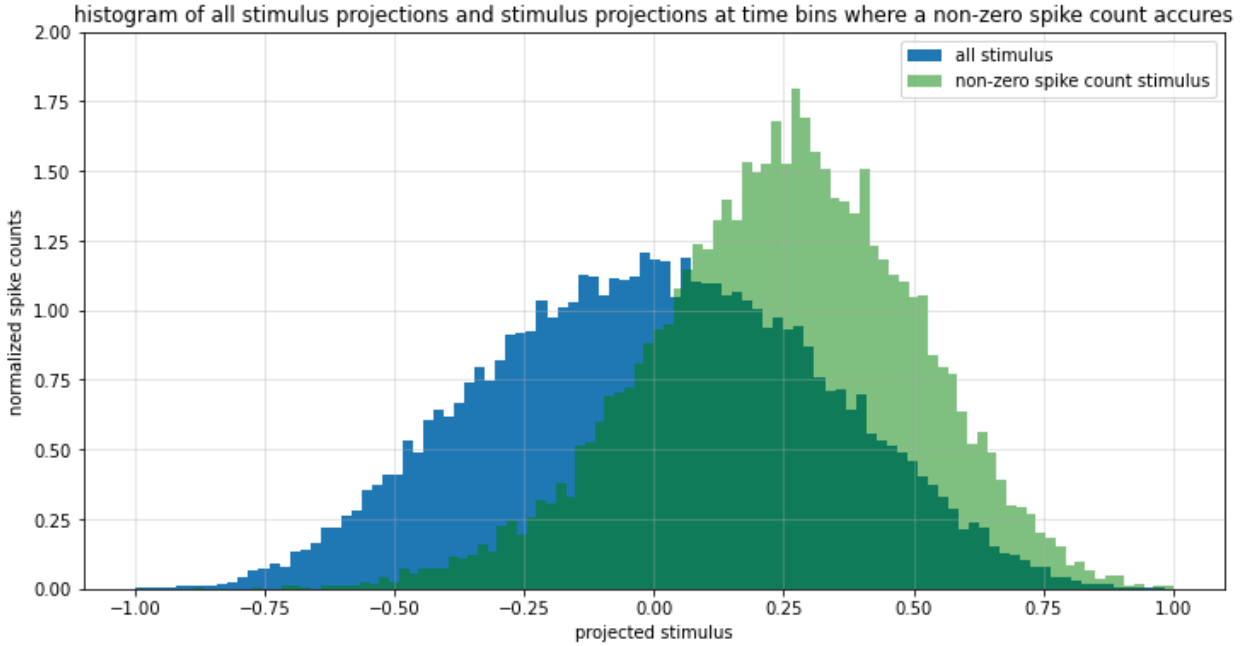
Then I did the same computation but this time I also cheeked one-time step after every stimulus and only appended the ones which had a non-zero spike count. Then I drew its histogram normalized to 1 with 100 bins. Then I draw both histograms on the same plot to compare them. The code which does these is given below.

```
    nonzero_stimulus_projected_on_STA = [] # this is an empth list we will
#convert it to an array once it reaches its full size.
    for i in range(1 , len(counts)): # we look at every mesurement point

        if 0 != counts[i]: # look if there are any spikes

            inner_product = 0 #we will use this to calculate the inner product

            for j in range( np.shape(averages)[1] ): # this for loop is for the
#Frobenius inner product calculation

                inner_product += np.inner(averages[:,j,0], stim[:,j,i - 1])


            nonzero_stimulus_projected_on_STA.append(inner_product) # we append
#the summed total to the list
```

```
    nonzero_stimulus_projected_on_STA = np.array(
nonzero_stimulus_projected_on_STA) # converting our list to an array

    nonzero_stimulus_projected_on_STA = nonzero_stimulus_projected_on_STA /
np.max(np.abs(nonzero_stimulus_projected_on_STA)) # making the maximum 1


    plt.figure(figsize=(12, 6))  # this lets me decide how big the plot should be
    plt.title("histogram of stimulus projections at time bins where a non-
zero spike counts accures")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which
#increases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(nonzero_stimulus_projected_on_STA, bins=100, color = "green", densit
y = True )

    print("The STA doesn’t discrimination spike-
eliciting stimuli very significantly because only around 80% of the Frobenius inn
er products’ result is bigger than 0. Still, we can say the STA discriminates spi
ke-eliciting stimuli but not very significantly.")

    plt.figure(figsize=(12, 6))  # this lets me decide how big the plot should be
    plt.title("histogram of all stimulus projections and stimulus projections at
time bins where a non-zero spike count accures")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which
#increases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(stimulus_projected_on_STA, bins=100, label="all stimulus", density =
 True )
    plt.hist(nonzero_stimulus_projected_on_STA, bins=100, color = "green", alpha
= 0.5, label="non-zero spike count stimulus", density = True )
    plt.legend(loc='upper right')
    plt.show() # this privents the plot from closing themselfs at the end
```

This code gives the output graphs shown below.





If we look at the last graph, we can see the mean of the all stimulus' projection to the STA is 0 or very close to zero. Also, the mean of the non-zero spike count stimulus' projection to the STA is around 0.25 which is bigger than 0. These are the 2 main conditions for saying an STA discriminates spike-eliciting stimuli. Also something to note, the variance of non-zero spike count stimulus' projection to the STA is lower than the variance of the all stimulus' projection to the STA. This means non-zero spike count stimulus' projection to the STA is more consistent in the values it takes. But the STA doesn't discrimination spike-eliciting stimuli very significantly

because only around 80% of the Frobenius inner products' results are bigger than 0. Still, we can say the STA discriminates spike-eliciting stimuli but not very significantly.

Question 2:

a) Construct an on-center difference-of-gaussians (DOG) center-surround receptive field centered at 0. Sample this receptive field as a 21x21 matrix, with a central Gaussian width of $\sigma_c = 2$ pixels and a surround Gaussian width of $\sigma_s = 4$ pixels. Display the generated receptive field.

The DOG receptive fields equation is given in equation 2 down below.

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

I first created a DOG function which returns the value of the DOG function shown in equation 2 for given σ_c and σ_s values at a given x, y point. It takes σ_c and σ_s values because I will use this function in part c with different, σ_c and σ_s values. The DOG function is given below.

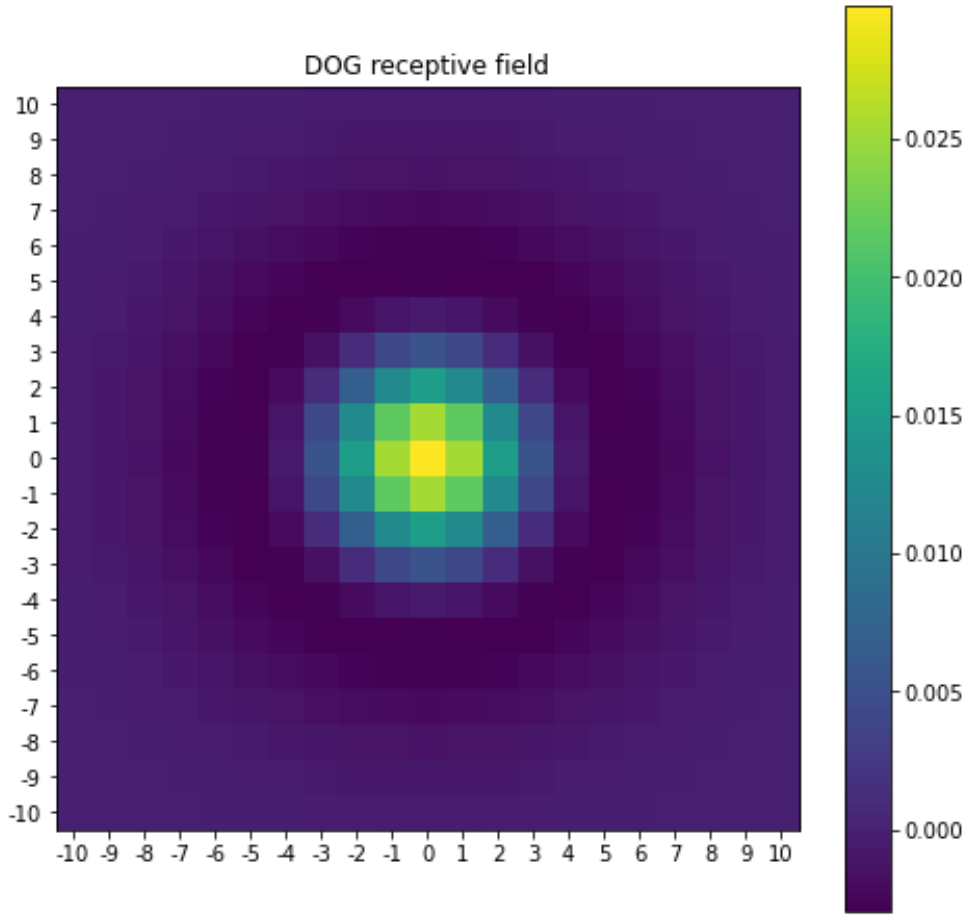
```
def DOG (x, y, sigma_s, sigma_c): # the DOG function given in the HW guideline

    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))
```

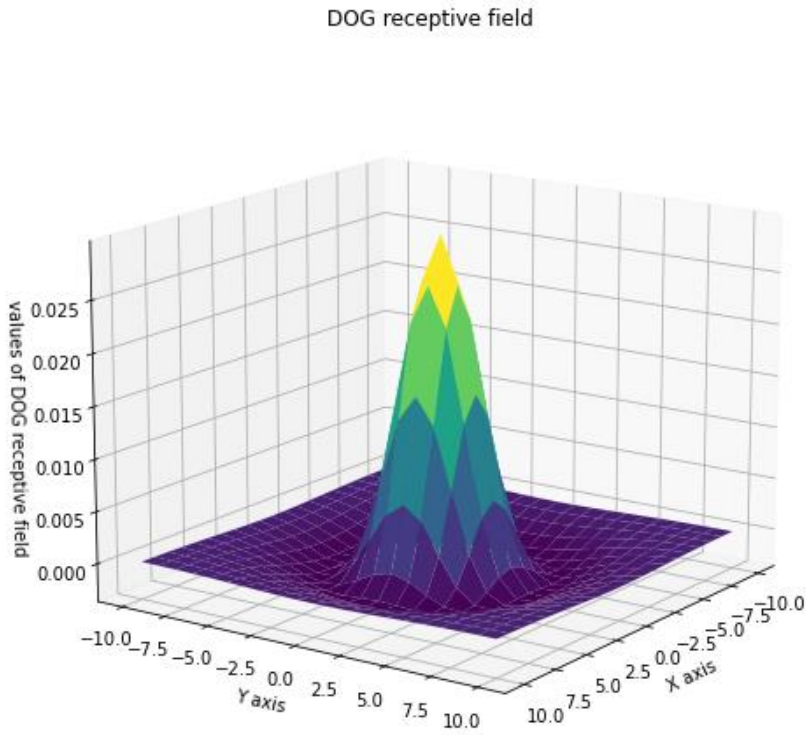
Then I wrote a code which constructs the on-center 21x21 matrix DOG field with central Gaussian width of $\sigma_c = 2$ pixels and a surround Gaussian width of $\sigma_s = 4$ pixels centered at 0. Then I drew its graph as a heat map and a 3D plot. The code for this is given below.

```
    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , 4 , 2)


    plt.figure(figsize=(8, 8)) # code for showing the DOG filter
    plt.imshow( DOG_receptive_field , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
```

```
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("DOG receptive field")

    plt.figure(figsize=(8 , 8)) # code for showing the DOG filter in 3D
    ax = plt.axes(projection='3d')
    A = B = np.linspace(-10, 10, 21)
    X, Y = np.meshgrid(A, B)
    ax.plot_surface(X, Y, DOG_receptive_field, rstride=1, cstride=1,
cmap='viridis', edgecolor='none')
    ax.set_title('DOG receptive field')
    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('values of DOG receptive field')
    ax.view_init(elev= 18, azim=35) # where we see the 3D plot from
```

This gives the output of:



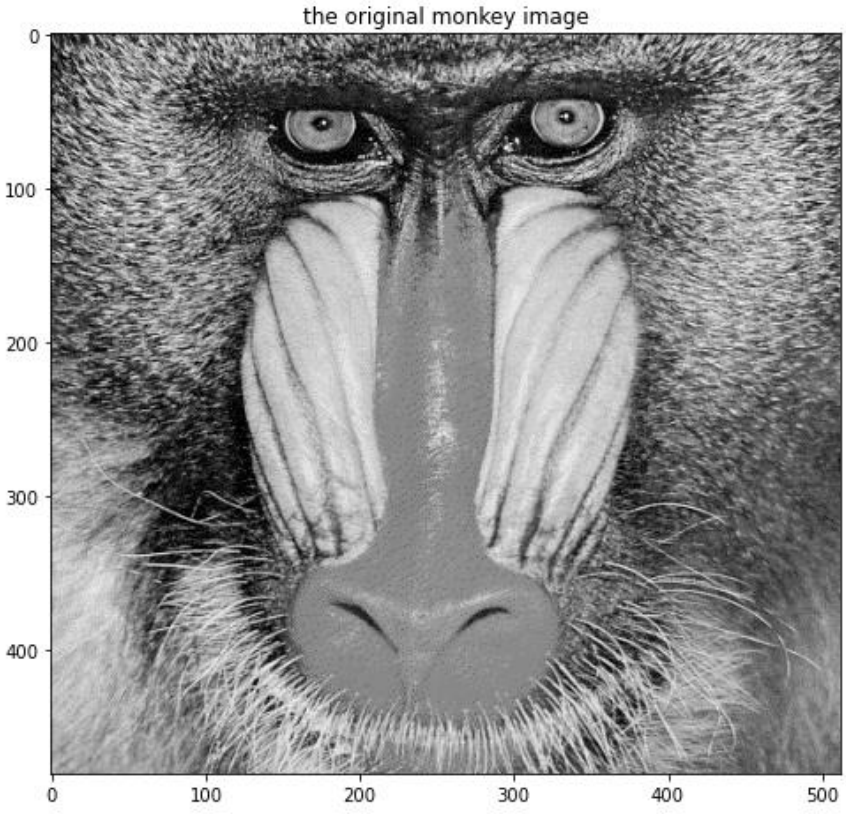


b) Neurons in lateral geniculate nuclei (LGN) have DOG receptive fields. Suppose that there is a separate LGN neuron with a receptive field centered on each pixel in the image. Compute the responses of each neuron to the given image.

I first loaded the given image into Python and displayed it to see how it looks and compare it to the filtering in the next stages. The code for this is given below.

```
    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in
#to Python

    plt.figure(figsize=(8, 8)) # code for showing the DOG filter
    plt.imshow( monkey_img )
    plt.title("the original monkey image")
```

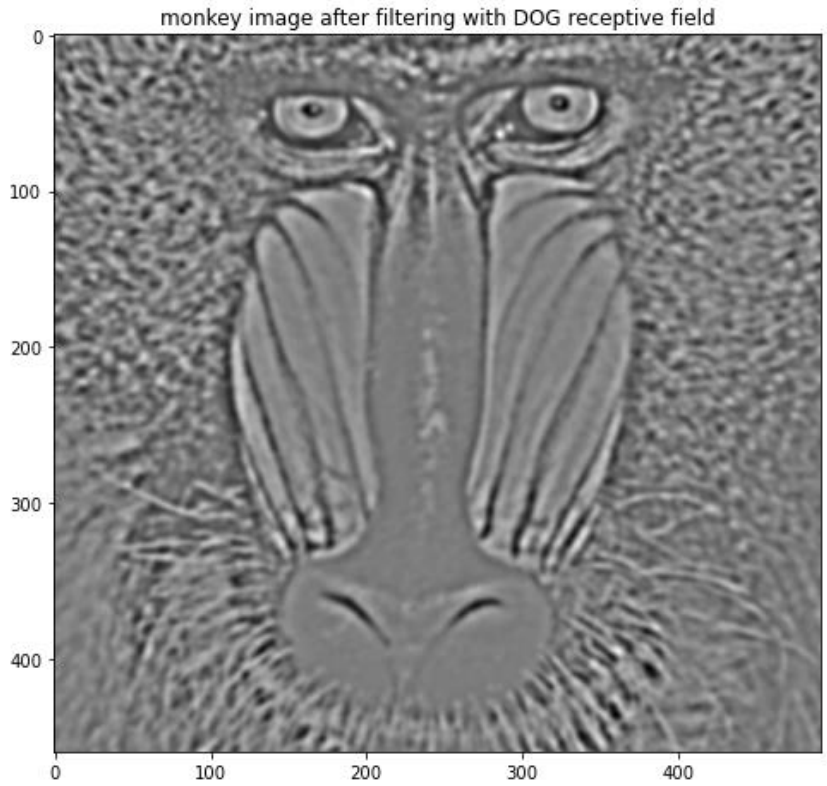
the original image this code prints is given below.



The image is given as an RGB with red green and blue values equal to each other. I will only take its red value and draw my plots as grey scales to have the same effect with less data. Having a DOG receptive fields at each pixel is same as convolving the image with the DOG receptive field. Python has a very convenient function for 3D convolution, we can also give it the mode= “valid” parameter to make the output matrix only come from convolutions which do not really on zero padding. This means we won’t be getting the edges which has visual artifacts as the zero padding is the cause of the artifacts. The code which does this then draws the resulted filtered image is given below.

```
    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , 4 , 2)

    neural_response_img = sig.convolve(monkey_img[:, :, 0], DOG_receptive_field ,
mode="valid") # Puting a DOG filter on every singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered monkey image
    plt.imshow( neural_response_img , cmap = "gray")
    plt.title("monkey image after filtering with DOG receptive field")
```

this code gives the output.



Basically, speaking difference-of-gaussians (DOG) is an edge detecting filter on the image and it makes the edges on the image more prominent.

c) Build an edge detector by thresholding the neural activity image. Tune the parameters of the DOG receptive fields and the threshold to optimize the edge detector's performance.

To create a edge detecting function I first created a thresholding filter function which takes an image (in 2D array format) and a threshold value. It sets every value in the given image, above the given threshold value to 1 and rest of the value in the image to 0 and returns the image, like it is described in the homework guideline. The code for the thresholding filter function is given below. (46)

```
def thresholding_filter(threshold, img): # treash holding filter funtion, this
#will be re-used in part f
    for i in range(np.shape(img)[0]): # we loop on each pixel of the image to do
#a treashold cheak on all of them
        for j in range(np.shape(img)[1]):

            if img[i , j] >= threshold:
```

```
                img[i , j] = 1

            else:

                img[i , j] = 0

    return img
```

This function will be re-used in part f of this question.

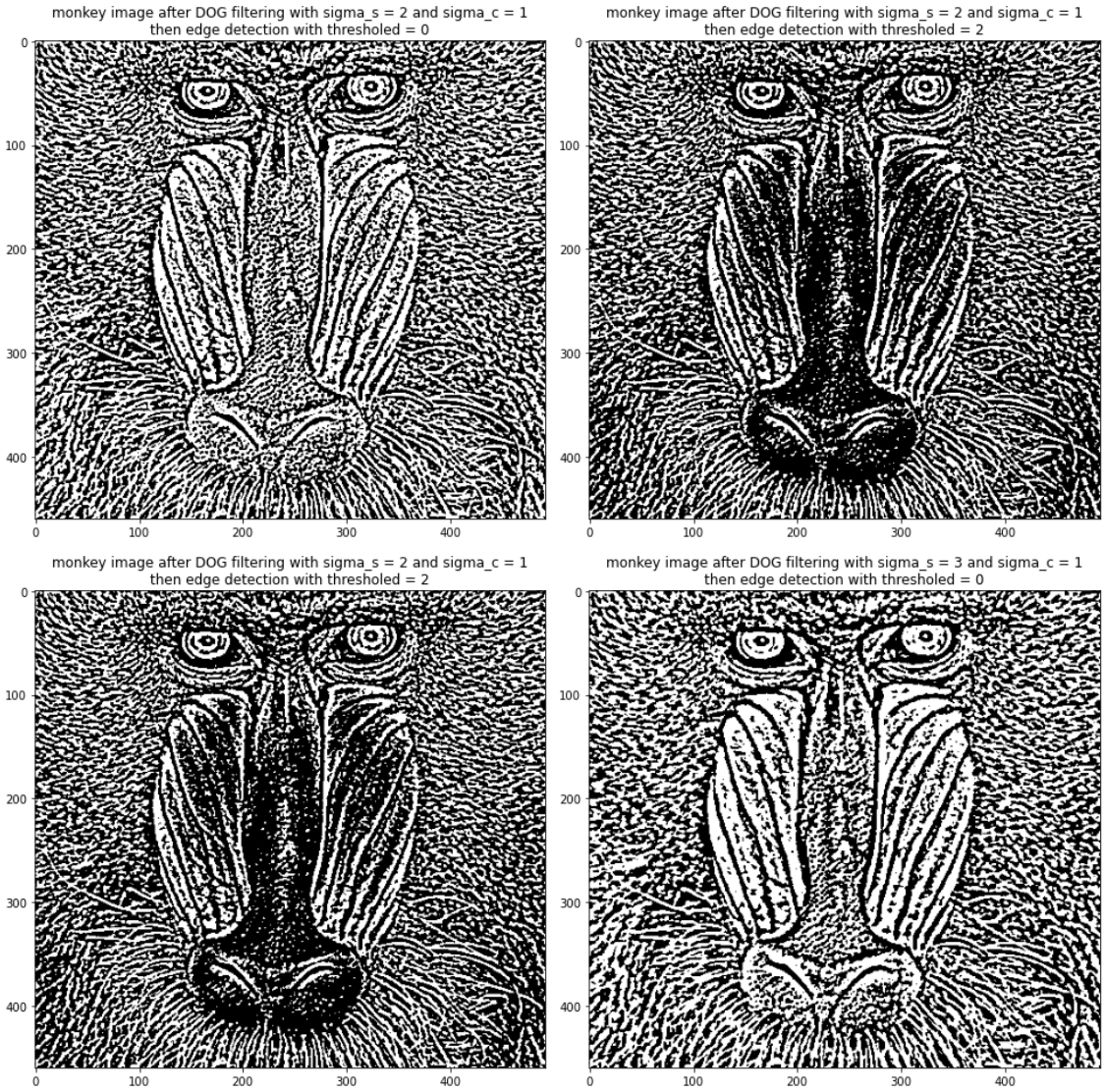
Then I wanted to test a lot of different σ_c, σ_s and threshold value combinations to find a combination which had good edge detecting performance. To make the code for it shorter and cleaner I created a function which takes an image (in 2D array format), σ_c, σ_s and threshold value, then creates a DOG receptive field for the given σ_c and σ_s values. Then computes the response of neurons with this DOG filter. Then puts the neurons response through the thresholding filter given above with the given threshold value. Then finally it plots the output of the thresholding filter with the appropriate title. This function is given below.

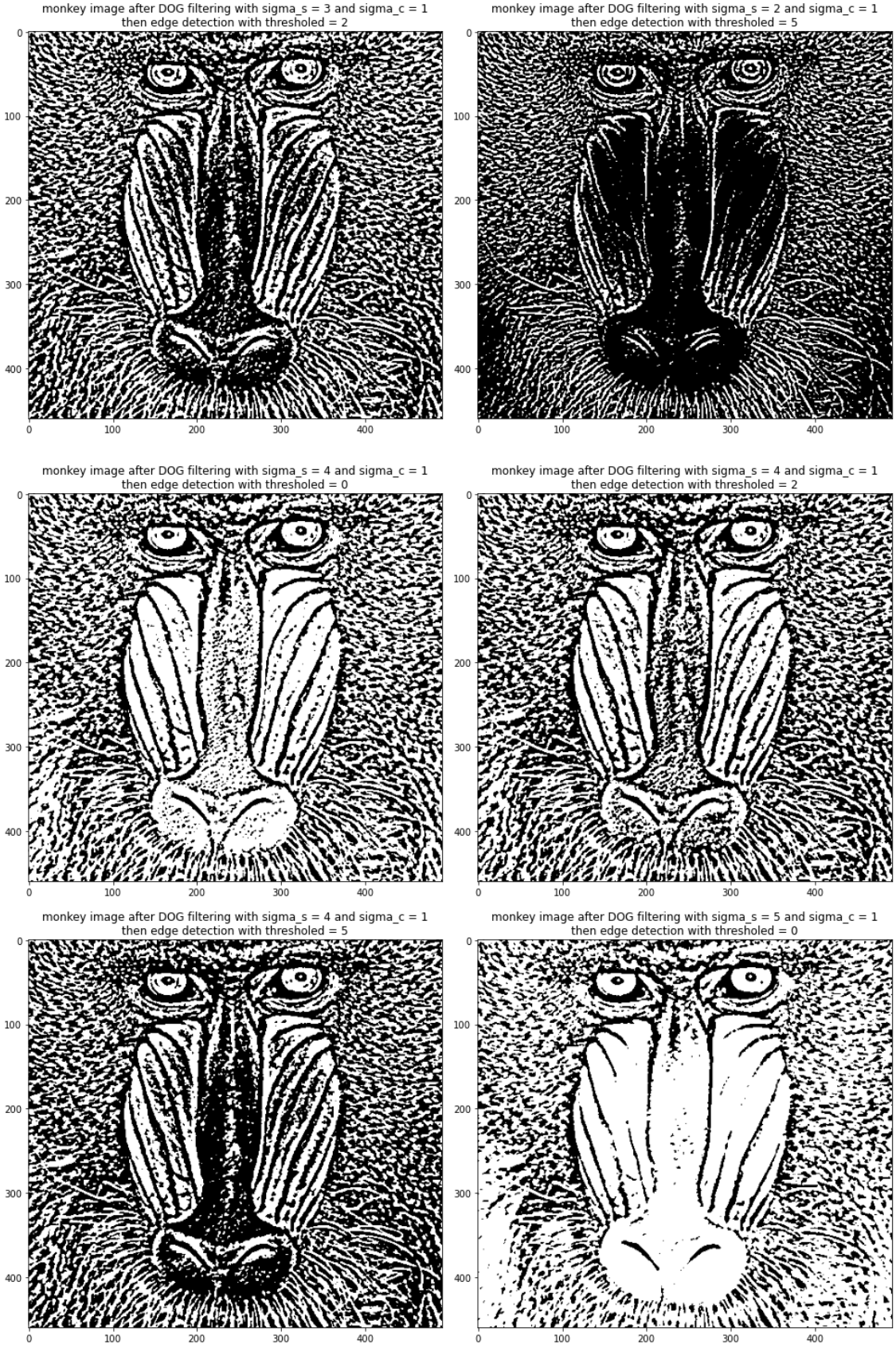
```
def draw_thresholded_dog_filtered_img(sigma_s, sigma_c, threshold, img):

    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , sigma_s , sigma_c)


    neural_response_img = sig.convolve(img[:, :, 0], DOG_receptive_field , mode="
valid") # Puting the DOG filter on every singal pixel.

    filtered_img = thresholding_filter(threshold, neural_response_img)

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered edge detected
#monkey image
    plt.imshow(filtered_img , cmap = "gray")
    plt.title("monkey image after DOG filtering with sigma_s = " + str(sigma_s) +
 " and sigma_c = " + str(sigma_c) + "\n then edge detection with thresholed = " +
str(threshold))
```

Then I wrote a code to loop trough the above function for σ_c and σ_s values between 1 and 5 always choosing σ_c smaller then σ_s with the threshold values 0, 2, and 5. The code for this is given below.

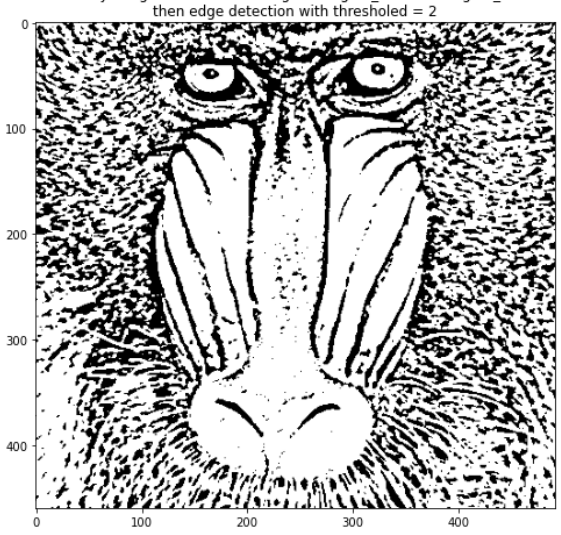
```
monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in to
#Python
    for sigma_c in range(1, 5): # code of testing difrent DOF parameter,
#threshold combinations a
        for sigma_s in range(sigma_c + 1 , 6):
            for threshold in (0 , 1 , 2):

                draw_thresholded_dog_filtered_img(sigma_s, sigma_c, threshold, mo
nkey_img)

    print("\nI belive the best one is sigma_s = 4, sigma_c = 1 and threshold = 0"
)
```

this gives the output of.

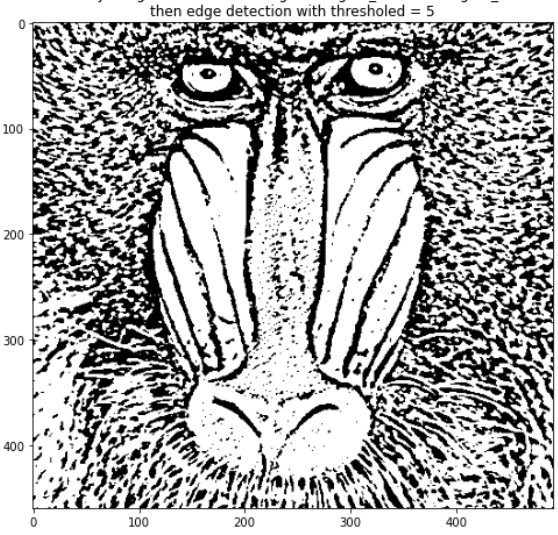



monkey image after DOG filtering with sigma_s = 3 and sigma_c = 1
then edge detection with thresholed = 2
monkey image after DOG filtering with sigma_s = 2 and sigma_c = 1
then edge detection with thresholed = 5
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 1
then edge detection with thresholed = 0
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 1
then edge detection with thresholed = 2
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 1
then edge detection with thresholed = 5
monkey image after DOG filtering with sigma_s = 5 and sigma_c = 1
then edge detection with thresholed = 0
0
100
200
300
400

monkey image after DOG filtering with sigma_s = 5 and sigma_c = 1
then edge detection with thresholed = 2

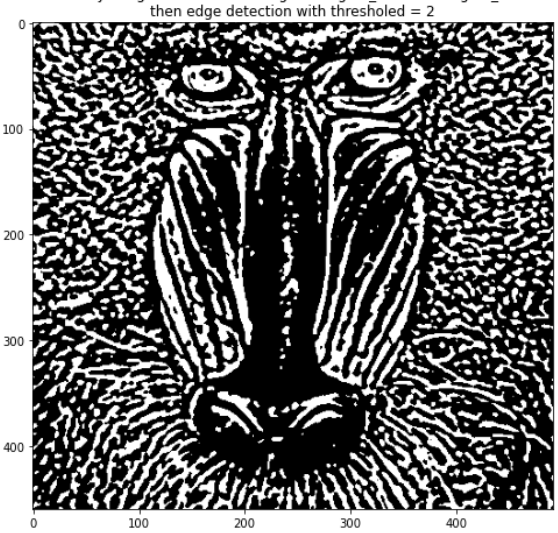


monkey image after DOG filtering with sigma_s = 5 and sigma_c = 1
then edge detection with thresholed = 5

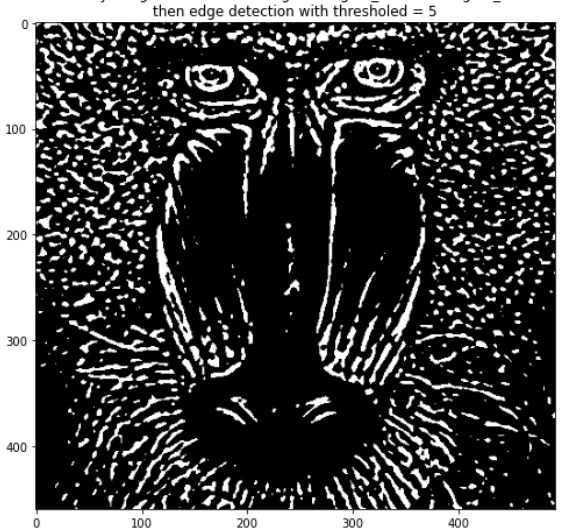


monkey image after DOG filtering with sigma_s = 3 and sigma_c = 2
then edge detection with thresholed = 0

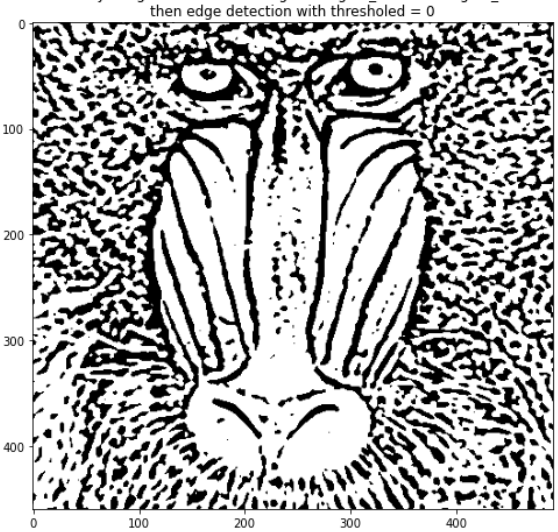
monkey image after DOG filtering with sigma_s = 3 and sigma_c = 2
then edge detection with thresholed = 2

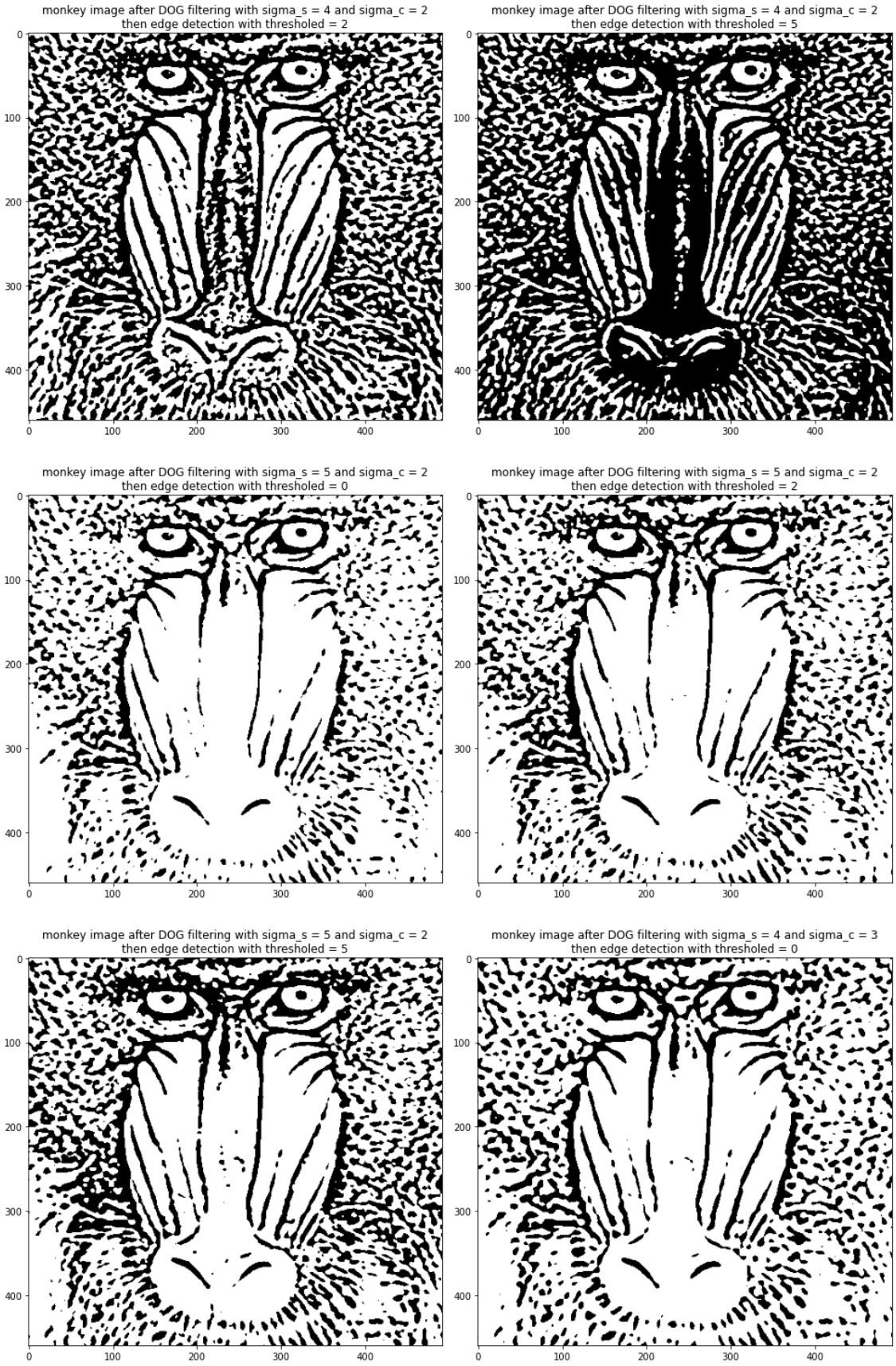


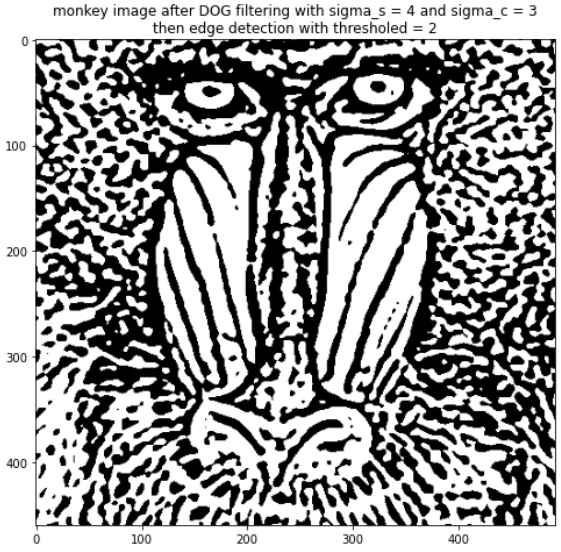
monkey image after DOG filtering with sigma_s = 3 and sigma_c = 2
then edge detection with thresholed = 5



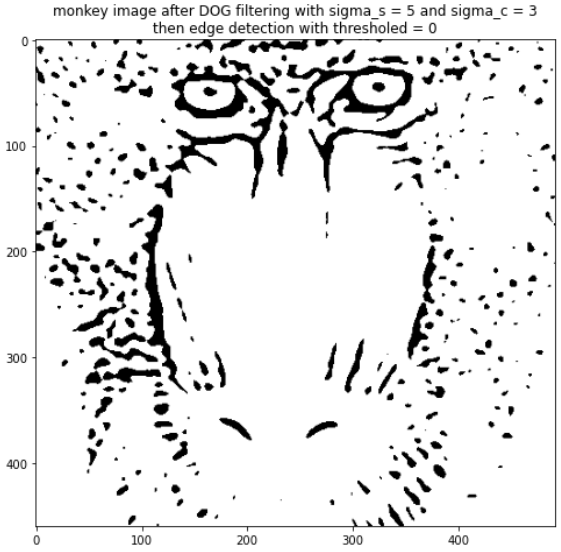
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 2
then edge detection with thresholed = 0

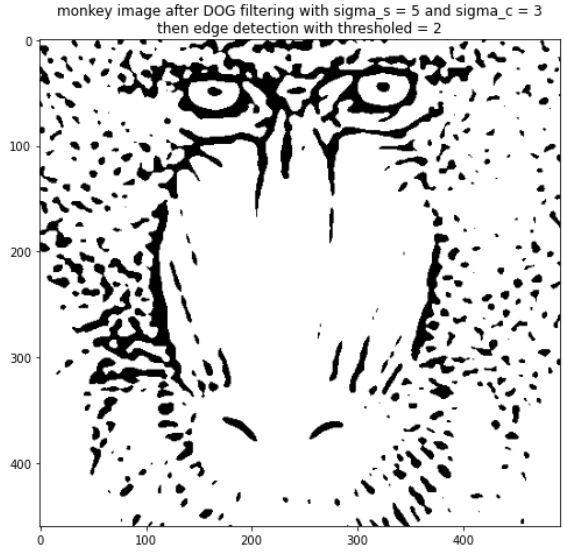


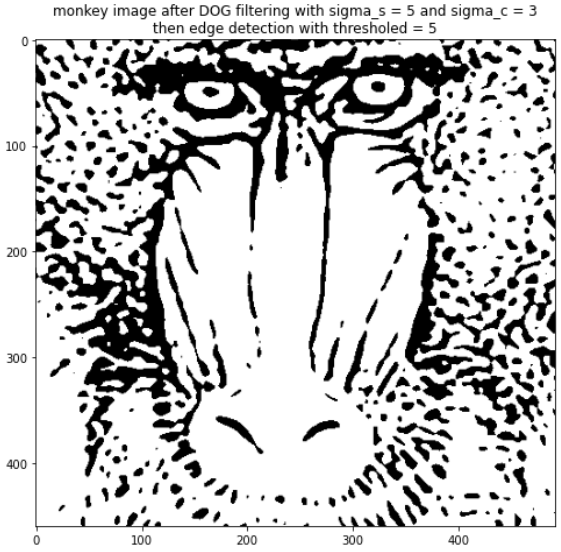

monkey image after DOG filtering with sigma_s = 4 and sigma_c = 2
then edge detection with thresholed = 2
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 2
then edge detection with thresholed = 5
monkey image after DOG filtering with sigma_s = 5 and sigma_c = 2
then edge detection with thresholed = 0
monkey image after DOG filtering with sigma_s = 5 and sigma_c = 2
then edge detection with thresholed = 2
monkey image after DOG filtering with sigma_s = 5 and sigma_c = 2
then edge detection with thresholed = 5
monkey image after DOG filtering with sigma_s = 4 and sigma_c = 3
then edge detection with thresholed = 0


monkey image after DOG filtering with sigma_s = 4 and sigma_c = 3
then edge detection with thresholed = 2

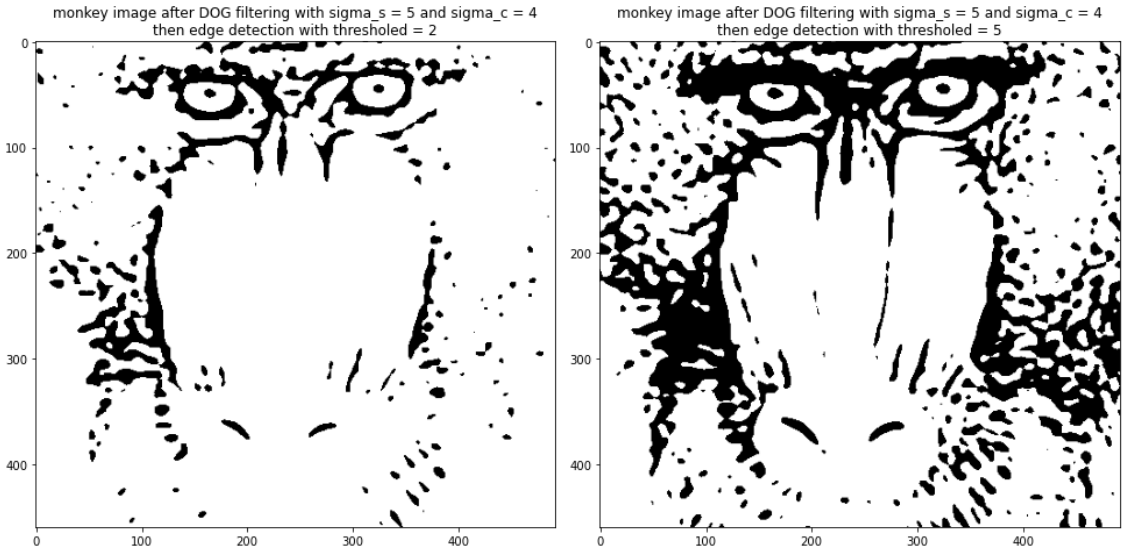

monkey image after DOG filtering with sigma_s = 4 and sigma_c = 3
then edge detection with thresholed = 5


monkey image after DOG filtering with sigma_s = 5 and sigma_c = 3
then edge detection with thresholed = 0


monkey image after DOG filtering with sigma_s = 5 and sigma_c = 3
then edge detection with thresholed = 2


monkey image after DOG filtering with sigma_s = 5 and sigma_c = 3
then edge detection with thresholed = 5


monkey image after DOG filtering with sigma_s = 5 and sigma_c = 4
then edge detection with thresholed = 0



After examining these images, I found the $\sigma_c = 1$, $\sigma_s = 4$, threshold = 0 and $\sigma_c = 1$, $\sigma_s = 5$, threshold = 5 have the best edge detecting performance. $\sigma_c = 1$, $\sigma_s = 4$, threshold = 0 is the combinates were the edges around the long hair around the nose of the monkey is most distinguishable but it has a lot black spots on its nose and cheeks because the filter picks up the change in brightness' on them as edges. and $\sigma_c = 1$, $\sigma_s = 5$, threshold = 5 is another combination where even the edges around the long hair around the nose of the monkey is distinguishable also it has way less un-wanted edge detection on the nose and asespecially on the cheeks. If I had to choose the best, I would pick $\sigma_c = 1$, $\sigma_s = 5$, threshold = 5 because even it misses some of the edges it has one of the best un-wanted edge detection suppression.

d) Construct a Gabor receptive field on the same 21x21 pixel grid. Start with assumption that $\theta = \pi/2, \sigma_l = \sigma_w = 3$ pixels, $\lambda = 6$ pixels, and $\varphi = 0$. Display the generated receptive field.

The gabor receptive fields equation is given in equation 3 down below.

$$D(x,y) = e^{-\frac{(\hat{k}_\theta \cdot \vec{x})^2}{2\sigma_l^2} - \frac{(\hat{k}_{\perp\theta} \cdot \vec{x})^2}{2\sigma_w^2}} \cos\left(2\pi \frac{\hat{k}_{\perp\theta} \cdot \vec{x}}{\lambda} + \phi\right) \quad (3)$$

Here $\hat{k}_\theta$ is a unit vector with the orientation θ, Here $\hat{k}_{\perp\theta}$ is a unit vector orthogonal to $\hat{k}_\theta$ and $\theta, \sigma_l, \sigma_w, \lambda \; and \; \phi$ are parameters which are already given to us in the homework guideline.

I first created a gabor function which returns the value of the gabor function shown in equation 3 for θ value at a given x, y point. It only takes θ value and not $\sigma_l, \sigma_w, \lambda \; and \; \phi$ because the homework guideline only tells us to change θ, so $\sigma_l, \sigma_w, \lambda \; and \; \phi$ values are taken as constants in the function. The gabor function is given below.

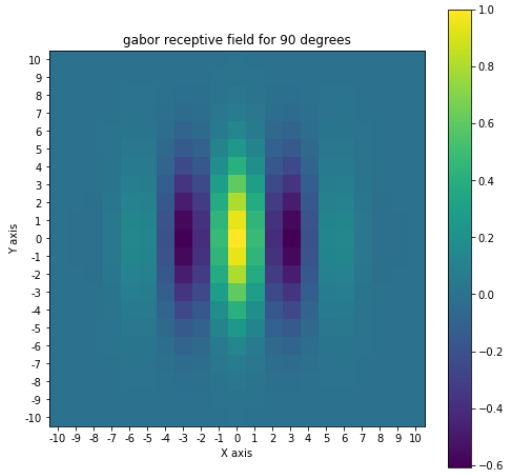
```
def gabor(x, theta): # the gabor function given in the HW guideline
    sigma_l = 3 # this time most of the internal paremeters are taken as a
#constant because we are not asked to change them.
    sigma_w = 3
    lambda_ = 6
    phi = 0
    k_theta = np.array([np.cos(theta), np.sin(theta)])
    k_theta_ort = np.array([-np.sin(theta), np.cos(theta)])

    D = np.exp( - (k_theta.dot(x))**2) / (2*(sigma_l**2))- (k_theta_ort.dot(x)**2
) / (2*(sigma_w**2))) * np.cos(2 * np.pi * k_theta_ort.dot(x) / lambda_ + phi)

    return D
```

Then I wrote a code which constructs the on-center 21x21 matrix gabor field with $\theta = \pi/2$, $\sigma_l = \sigma_w = 3$ pixels, $\lambda = 6$ pixels, and $\varphi = 0$ centered at 0. Then I drew its graph as a heat map and a 3D plot. The code for this is given below.

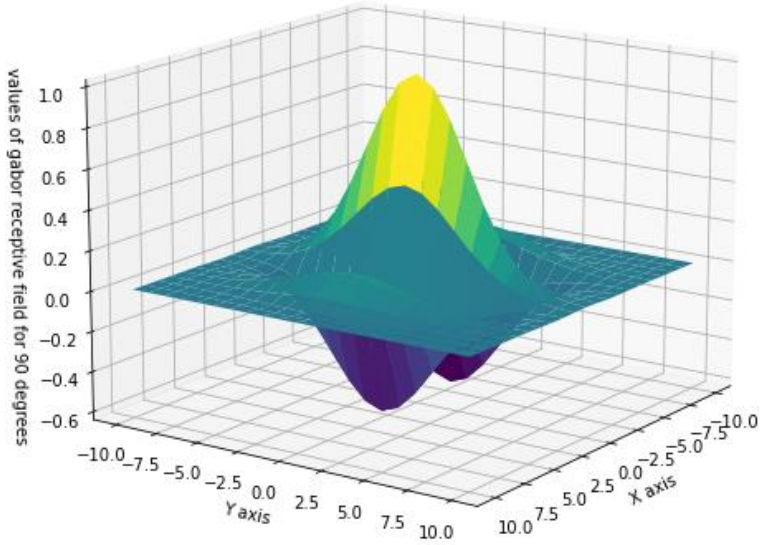
```
    gabor_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field[i + 10 , j + 10] = gabor(np.array([j , i]) ,
np.pi/2 )

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 90 degreas")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8 , 8)) # code for showing the gabor filter in 3D
    ax = plt.axes(projection='3d')
    A = B = np.linspace(-10, 10, 21)
    X, Y = np.meshgrid(A, B)
    ax.plot_surface(X, Y, gabor_receptive_field, rstride=1, cstride=1,
cmap='viridis', edgecolor='none')
    ax.set_title("gabor receptive field for 90 degreas");
    ax.set_xlabel('X axis')
```

```
    ax.set_ylabel('Y axis')
    ax.set_zlabel("values of gabor receptive field for 90 degreas")
    ax.view_init(elev=18, azim=35) # where we see the 3D plot from
```

This gives the output of.



gabor receptive field for 90 degrees



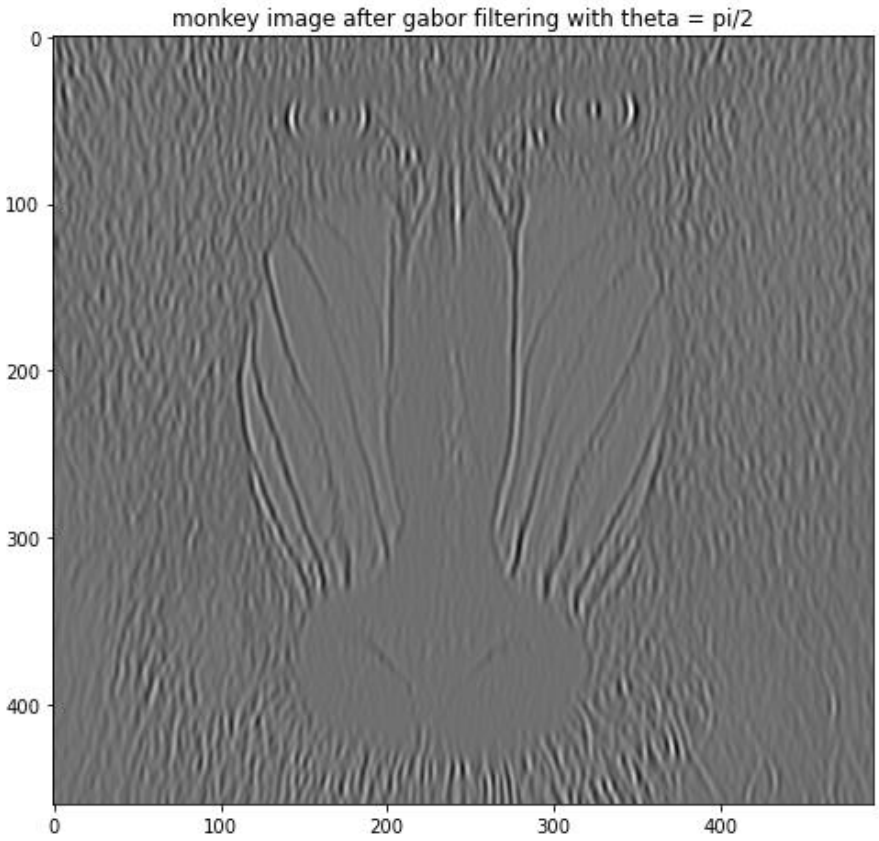
From these plots we can say gabor filter is an edge detection filter for an edge with orientation θ. It does this by making the edges more prominent.

e) Simple cells in V1 have Gabor receptive fields. Suppose that there is a separate V1 neuron with a receptive field centered on each pixel in the image. Compute the responses of each neuron to the given image. Place the neural responses topographically according to the centers of their receptive fields, and display the neural activity. What is the function of this Gabor filter?

What is asked from us in this part is very similar to part b, so what we will do to solve is also very similar to part b. Again, having a gabor receptive fields at each pixel is same as convolving the image with the gabor receptive field. Again we will be using the very convenient function Python has for 3D convolution, we can also give it the mode= “valid” parameter to make the output matrix only come from convolutions which do not really on zero padding. This means we won’t be getting the edges which has visual artifacts as the zero padding is the cause of the artifacts. The code which does this then draws the resulted filtered image is given below.

```
    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in
#to Python


    gabor_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field[i + 10 , j + 10] = gabor(np.array([j , i]) ,
np.pi/2 )


    neural_response_img = sig.convolve(monkey_img[:, :, 0], gabor_receptive_field
 , mode="valid") # Puting a DOG filter on every singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered edge detected
#monkey image
    plt.imshow(neural_response_img , cmap = "gray")
    plt.title("monkey image after gabon filtering with theta = pi/2")
```

This code gives the output.



From this image we can see gabor filter detects edges with an orientation close to θ degrees and it makes these edges more prominent. This effect is easiest to see around the eyes of the monkey because we can clearly see its right and left edges but the top and bottom edges are almost non-existent.

f) Construct 4 Gabors with $\theta = 0, \pi/6, \pi/3, \pi/2$. Compute combined neural responses to the given image, by summing the outputs of the individual receptive fields (for different θ). Does the edge detection performance look better in this case? What can you do with these 4 Gabors to further improve the performance?

I used the gabor function defined in the previous part to create gabor receptive field with $\theta = 0, \pi/6, \pi/3, \pi/2$. The I convolved each filter with the given monkey image and draw the plots of the filters and the filtered monkey images one by one. The code for this is given below.

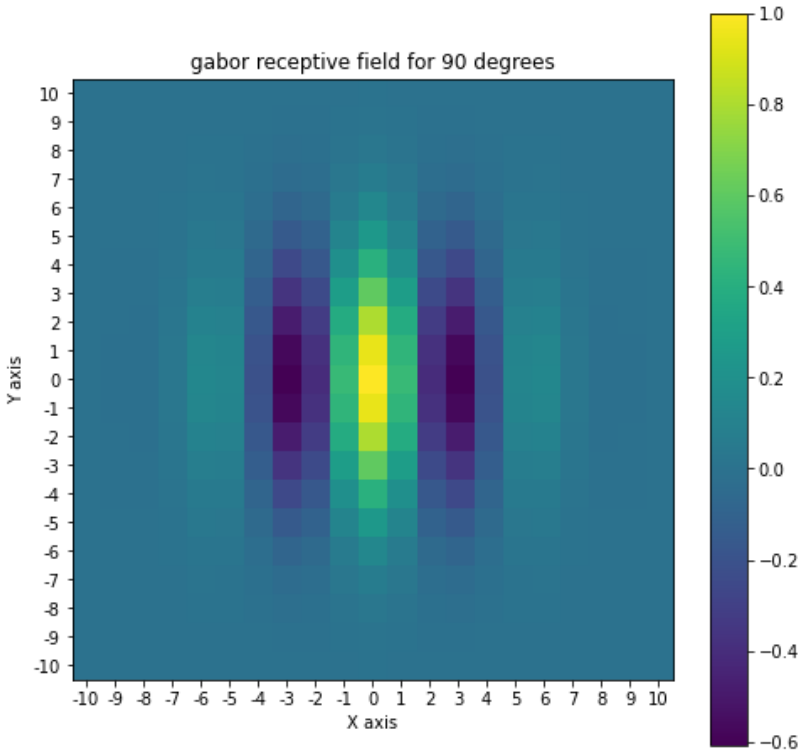
```
    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in t
o Python

    gabor_receptive_field_pi_over_2 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):
```

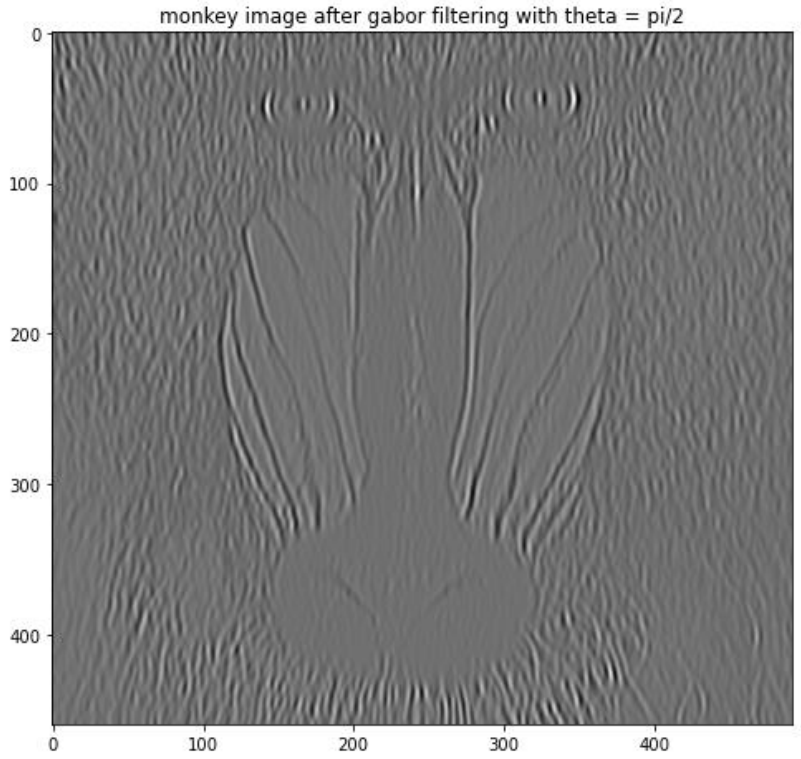
```
            gabor_receptive_field_pi_over_2[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/2 )

    neural_response_img_pi_over_2 = sig.convolve(monkey_img[:, :, 0],
#gabor_receptive_field_pi_over_2 , mode="valid") # Puting a gabor filter on every
# singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_2 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 90 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_2 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/2")


    gabor_receptive_field_pi_over_3 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_pi_over_3[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/3 )

    neural_response_img_pi_over_3 = sig.convolve(monkey_img[:, :, 0],
gabor_receptive_field_pi_over_3 , mode="valid") # Puting a gabor filter on every
#singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_3 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 60 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_3 , cmap = "gray")
```

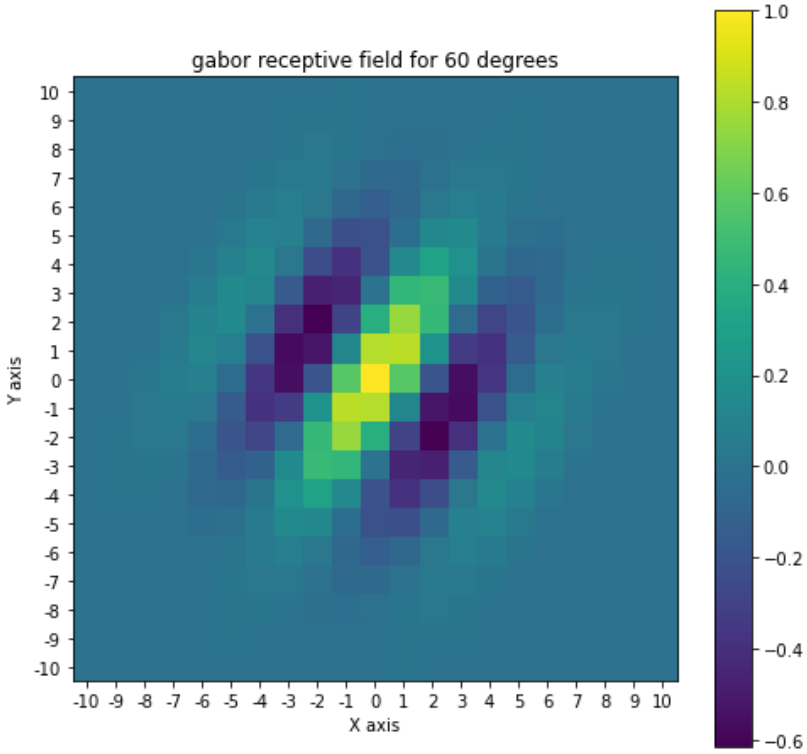
```
    plt.title("monkey image after gabor filtering with theta = pi/3")


    gabor_receptive_field_pi_over_6 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_pi_over_6[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/6 )

    neural_response_img_pi_over_6 = sig.convolve(monkey_img[:, :, 0],
gabor_receptive_field_pi_over_6 , mode="valid") # Puting a gabor filter on every
#singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_6 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 30 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_6 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/6")


    gabor_receptive_field_0 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_0[i + 10 , j + 10] = gabor(np.array([j , i]) ,
0 )

    neural_response_img_0 = sig.convolve(monkey_img[:, :, 0],
#gabor_receptive_field_0 , mode="valid") # Puting a gabor filter on every singal
#pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_0 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
```

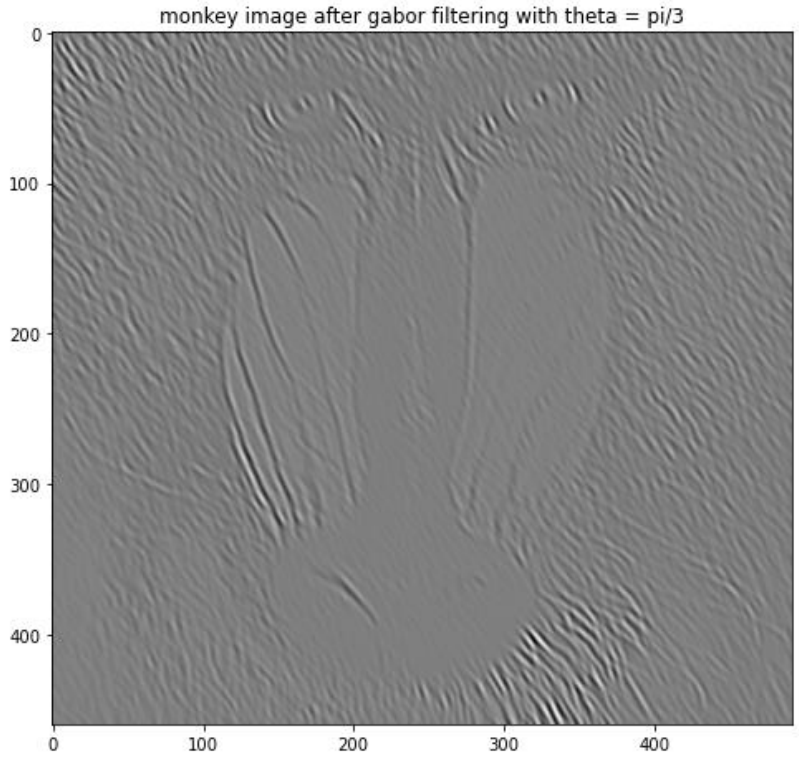
```
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 0 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_0 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 0")
```

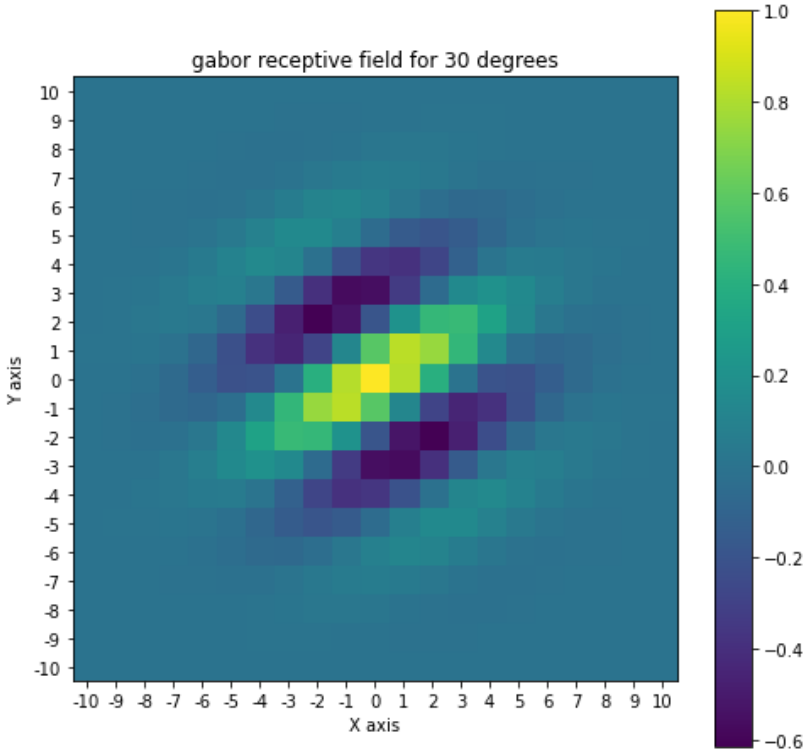
This code gives the output.

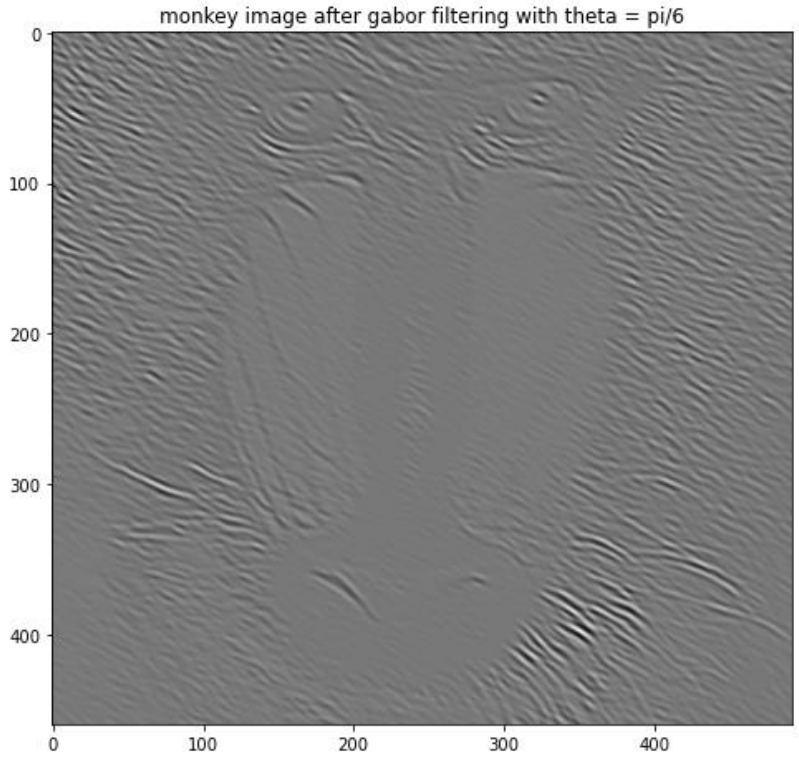


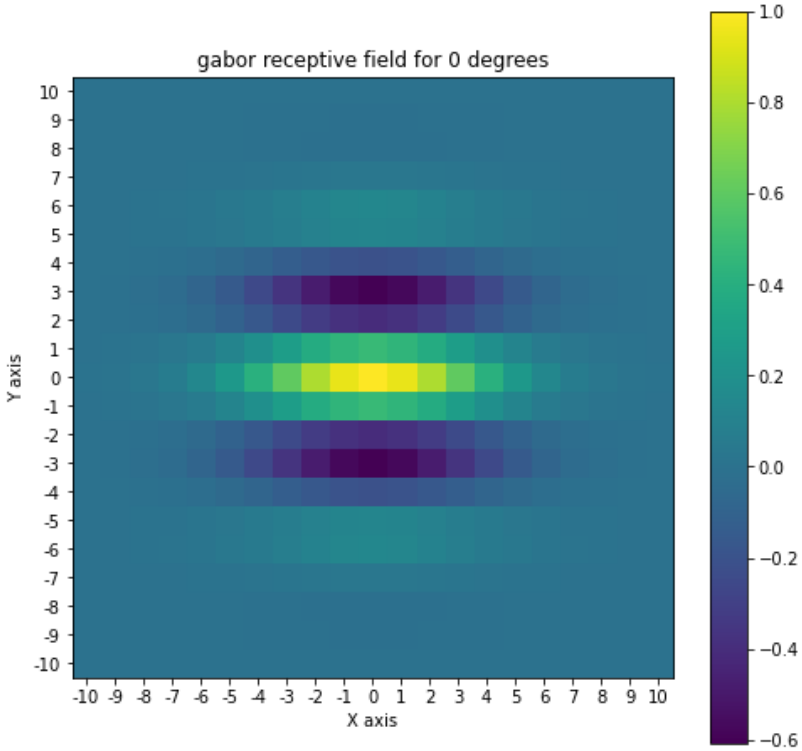

monkey image after gabor filtering with theta = pi/2
0
100
200
300
400
0
100
200
300
400

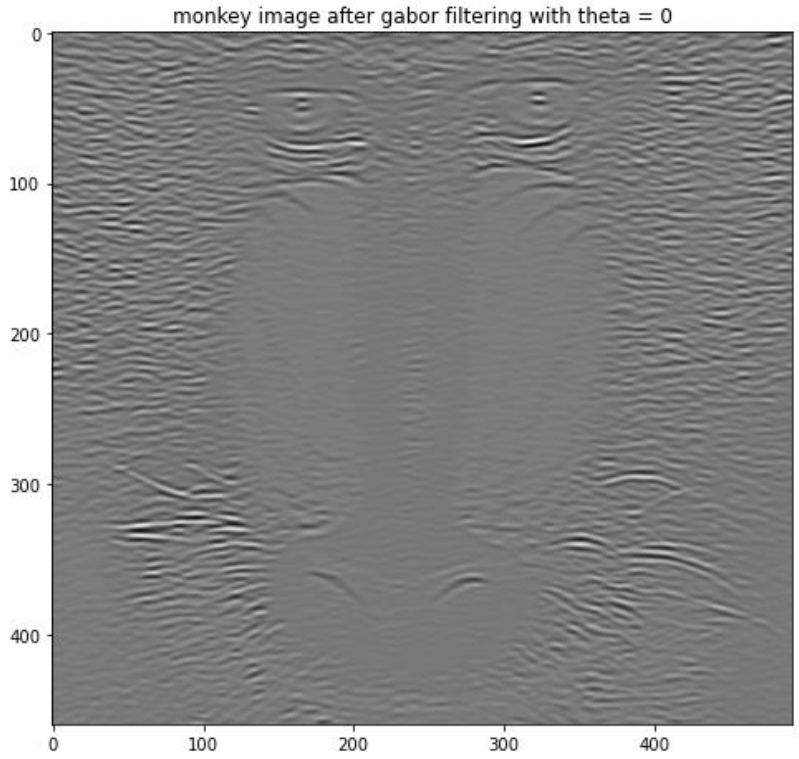

gabor receptive field for 60 degrees
Y axis
X axis
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


monkey image after gabor filtering with theta = pi/3
0
100
200
300
400
0
100
200
300
400


gabor receptive field for 30 degrees
Y axis
X axis
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


monkey image after gabor filtering with theta = pi/6
0
100
200
300
400
0
100
200
300
400


gabor receptive field for 0 degrees
Y axis
X axis
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

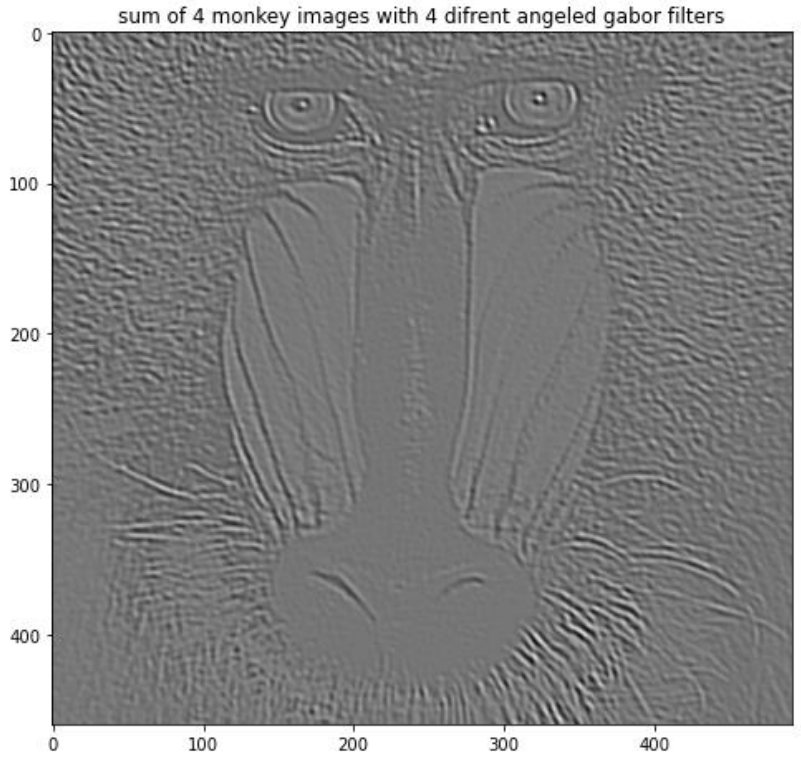


These images show the gabor receptive fields effect of detecting edges which have an orientation close to θ. Something to note is the 30 degree and 60 degree might seem like they don't have the correct orientation but they are actually correct. These are plots of matrixes' so the (0, 0) point is on the upper left corner, this means y increases as we move down on the plot instead of up because of this the gabor filters are upside down on the plot. So 30 degrees looks like -30 degrees and 60 degrees looks like – 60 degrees. This means all the plots and the gabor receptive fields are correct.

Then I summed these filtered images and draw the sums plot. The code for this is given below.

```
    gabor_receptive_field_sum = neural_response_img_pi_over_2 +
neural_response_img_pi_over_3 + neural_response_img_pi_over_6 +
neural_response_img_0

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered
#monkey images
    plt.imshow(gabor_receptive_field_sum , cmap = "gray")
    plt.title("sum of 4 monkey images with 4 difrent angeled gabor filters")
```

This code gives the output of.

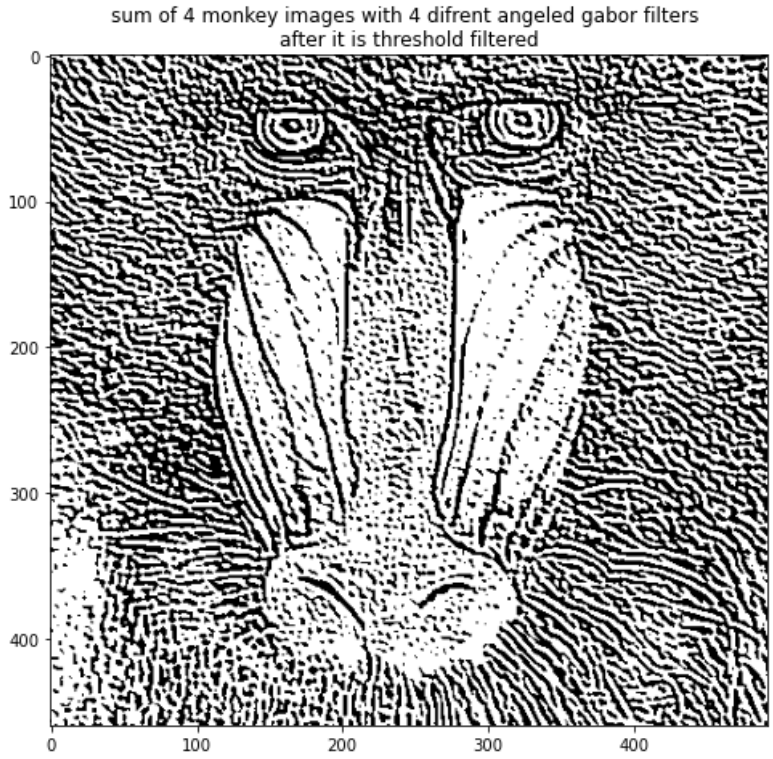


This looks better then any of the gabor filters alone because it has more edges with difrent orientations are visible and it is more detailed then the result of the DOG filter from part b but the edges are not as prominent as they were in the DOG filter.

To increase its edge detection performance, first I applied a thresholding filter to the sum of the gabor filters. The code for this test is given below.

```
    gabor_receptive_field_sum_after_TH = thresholding_filter(150 ,
gabor_receptive_field_sum) # threshold filtering the result

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered
#monkey images after the sum is threshold filtered
    plt.imshow(gabor_receptive_field_sum_after_TH , cmap = "gray")
    plt.title("sum of 4 monkey images with 4 difrent angeled gabor filters \n
after it is threshold filtered")
```

The threshold values in this code and the tests after this were found with a lot of trial and error but I did not add the trials to the report to not make the report extremely long. This code gives the output of.



This image has better egde detection performance compared to just the sum of the 4 gabor filtered images. This show thresholding is once again a good method to increase edge detection performance. But if we can increase the number of gobar filter we can use, this can be further improved.

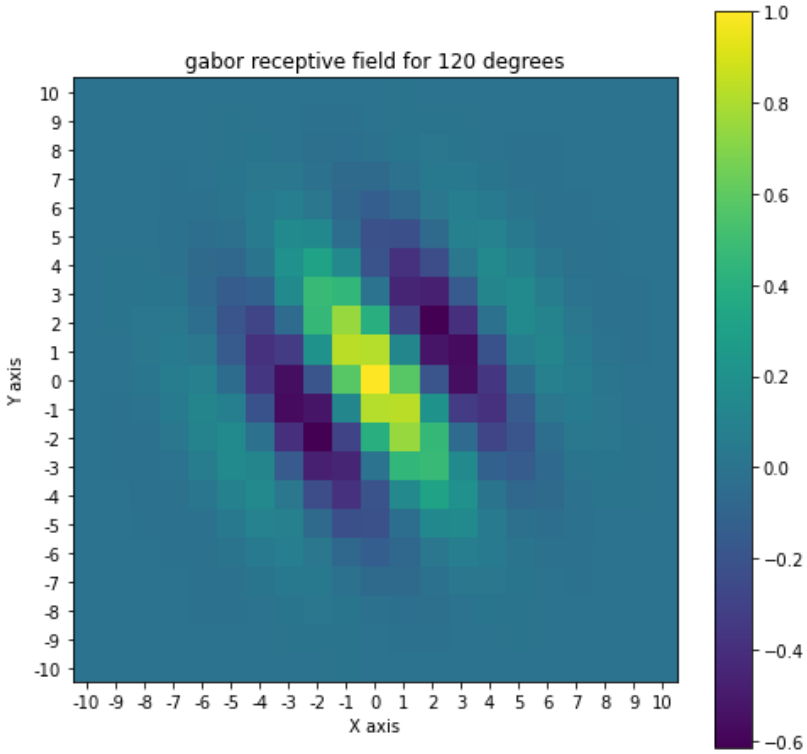
After further inspecting this image, I realized the -30 and -60 degree edges are not very clear in the filtered image. This can be best seen in in the right cheek of the monkey, it is not as clear as the left cheek because of the orientation of the edges in it. This is not surprising because we did not gabor filtered those orientations and we did not add them to the sum too. To fix this and improve the edge detection performance I created gabor receptive fields for -30 and -60 degrees, then convolved them with the original image and add them to the original 4 gabor filtered images. The applied a thresh hold to the sum. The code which does this while showing the result of the intermediate steps and the final result is given below.

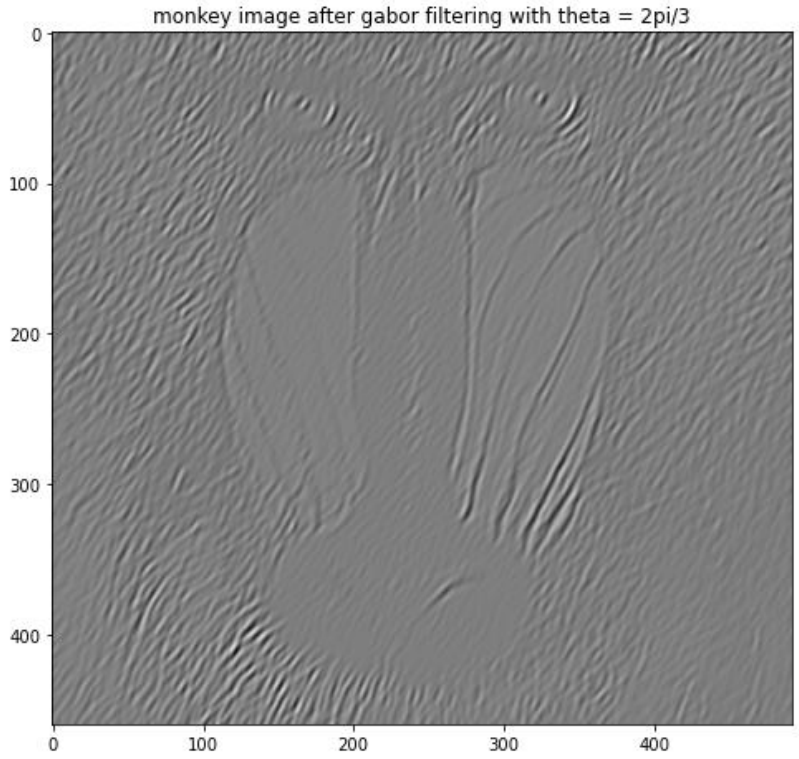
```
    gabor_receptive_field_2pi_over_3 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_2pi_over_3[i + 10 , j + 10] =
gabor(np.array([j , i]) , 2*np.pi/3 )
```

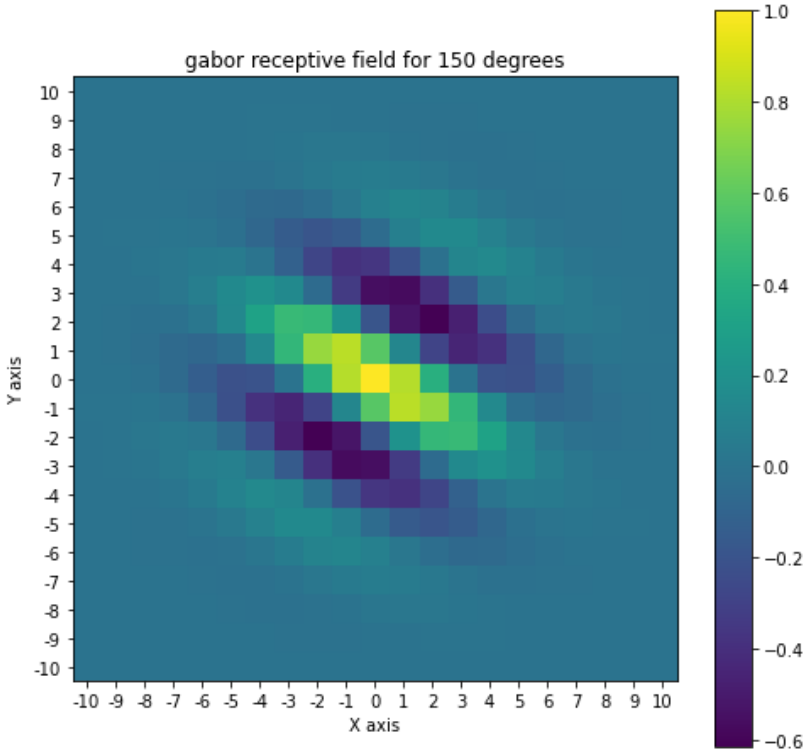
```
    neural_response_img_2pi_over_3 = sig.convolve(monkey_img[:, :, 0],
gabor_receptive_field_2pi_over_3 , mode="valid") # Puting a gabor filter on every
# singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_2pi_over_3 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 120 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_2pi_over_3 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 2pi/3")


    gabor_receptive_field_5pi_over_6 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_5pi_over_6[i + 10 , j + 10] =
gabor(np.array([j , i]) , 5*np.pi/6 )

    neural_response_img_5pi_over_6 = sig.convolve(monkey_img[:, :, 0],
gabor_receptive_field_5pi_over_6 , mode="valid") # Puting a gabor filter on every
# singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_5pi_over_6 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 150 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_5pi_over_6 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 5pi/6")
```

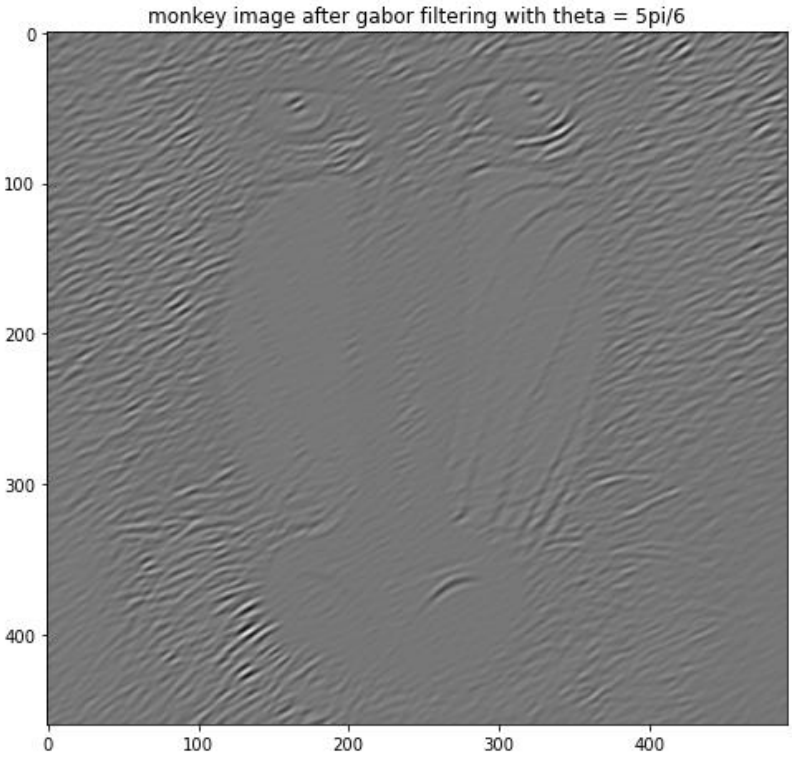
```
    gabor_receptive_field_sum_2 = neural_response_img_pi_over_2 +
neural_response_img_pi_over_3 + neural_response_img_pi_over_6 +
neural_response_img_0 + neural_response_img_2pi_over_3 +
neural_response_img_5pi_over_6

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered
#monkey images
    plt.imshow(gabor_receptive_field_sum_2 , cmap = "gray")
    plt.title("sum of 6 monkey images with 6 difrent angeled gabor filters")

    # these are test done to improve the results

    gabor_receptive_field_sum_after_TH_2 = thresholding_filter(250 ,
gabor_receptive_field_sum_2) # threshold filtering the result

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered
#monkey images after the sum is threshold filtered
    plt.imshow(gabor_receptive_field_sum_after_TH_2 , cmap = "gray")
    plt.title("sum of 6 monkey images with 6 difrent angeled gabor filters \nafte
r it is threshold filtered")
```

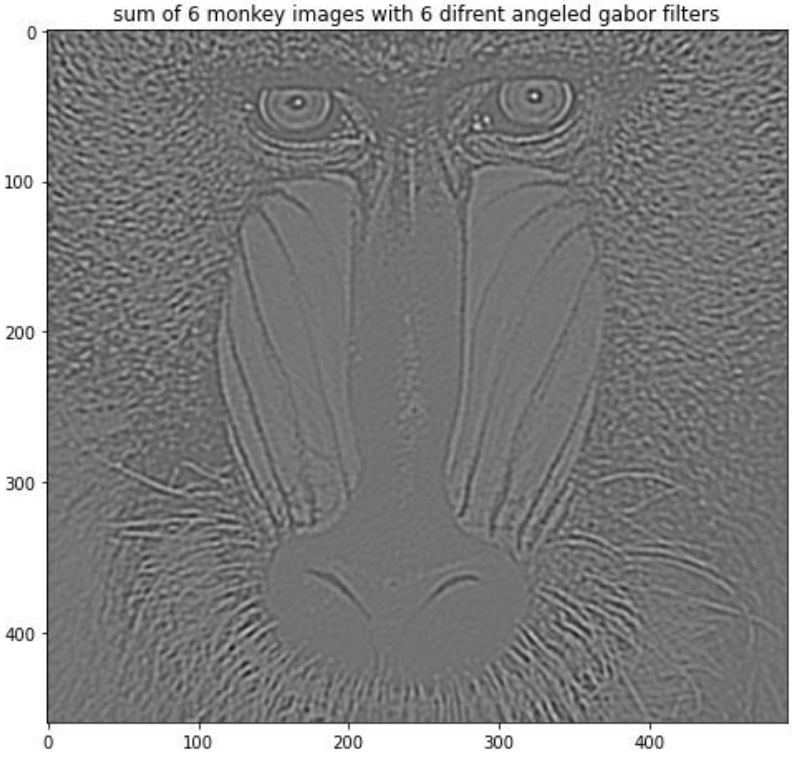
this code gives the output.

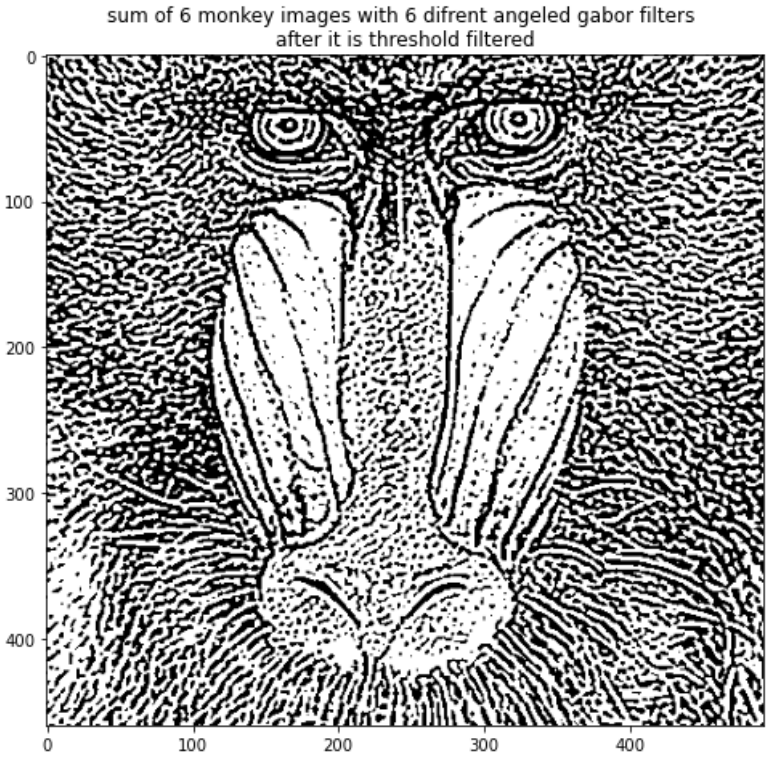



monkey image after gabor filtering with theta = 2pi/3
0
100
200
300
400
0
100
200
300
400


gabor receptive field for 150 degrees
Y axis
X axis
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


monkey image after gabor filtering with theta = 5pi/6
0
100
200
300
400
0
100
200
300
400


sum of 6 monkey images with 6 difrent angeled gabor filters
0
100
200
300
400
0
100
200
300
400



If we look at the final image the missing edges on the right cheek are restored, so its edge detection performance increased. The edges on this image look very clear and this is probably the best edge detection performance achieved.

CODE:

As a formality I am adding all of my code as it is at the end of the report:

```
# -*- coding: utf-8 -*-

import sys # the primer already has it

import numpy as np # this brings simple matrix operations to python
import matplotlib.pyplot as plt # this brings nice loking plots to python

import scipy.io as sio # this is used to load MATLAB files
from scipy import signal as sig # this brings 2D convolution.
from mpl_toolkits import mplot3d # used for 3D plots

"""
If you want to run this code in a IDE you need to commend out the line below and
write the line
```

```
question = '1'
then run the code for question 1 and write the line
question = '2'
then run the code for question 2
"""

question = sys.argv[1]

def Batu_Arda_Düzgün_21802633_hw2(question):
    if question == '1' :
        print("Answer of question 1.) \n")
        question1_part_a()
        question1_part_b()
        question1_part_c()

    elif question == '2' :
        print("Answer of question 2.) \n")
        question2_part_a()
        question2_part_b()
        question2_part_c()
        question2_part_d()
        question2_part_e()
        question2_part_f()


def question1_part_a():

    print("\nAnswer of question 1 part a: \n")

    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']

    averages = np.zeros( ( np.shape(stim)[0], np.shape(stim)[1] , 10 ) ) # will f
ill this with STA's for each of the 10 timesteps.

    for i in range(10): # calculating the STA's for the 10 time steps.

        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=n
p.max(averages))
```

```
        plt.colorbar()
        plt.title('STA: ' + str(i + 1) + ' Steps Before Spike')
        plt.yticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
        plt.xticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))

    print("This LGN cell is selective for when the center of the image first dark
ens then instantly lights up and brightens. ")




def STA(stim, counts, time_step): # the function to calculate STA for a given tim
es step for a stimulus and spike count data pair.
    total_sum = np.zeros( (np.shape(stim)[0], np.shape(stim)[1]) ) # we will add
everything spike tringering stimulus to this.

    for i in range (time_step, len(counts)): #looping trough the data set

        total_sum[:,:] += stim[:,:, i - time_step] * counts[i]

    spike_count = np.sum(counts[time_step:]) # finding total spike count
    average = total_sum / spike_count # taking the spike tringing stimules averag
e.
    return average



def question1_part_b():

    print("\nAnswer of question 1 part b: \n")

    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']

    averages = np.zeros( ( np.shape(stim)[0], np.shape(stim)[1] , 10 ) ) # will f
ill this with STA's for each of the 10 timesteps.

    for i in range(10): # calculating the STA's for the 10 time steps.

        averages[:,:, i ] = STA(stim, counts, i + 1 )


    collum_summed_average = np.sum(averages, axis = 0) # the code which sums the
values trough the collums of the matrix
```

```
    row_summed_average = np.sum(averages, axis = 1) # the code which sums the val
ues trough the rows of the matrix

    plt.figure()
    plt.imshow( collum_summed_average.transpose() , cmap='gray' , origin='lower',
 vmin=np.min(row_summed_average), vmax=np.max(row_summed_average)) # I drew the t
ranspose here to enphesize how we sum trough the collums of the matrix
    plt.colorbar()
    plt.title("collum summed STA's")
    plt.yticks(np.arange(0, 10, step=1) , np.arange(1, 11, step=1))
    plt.xticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
    plt.ylabel("times steps")


    plt.figure()
    plt.imshow( row_summed_average , cmap='gray' , origin='lower', vmin=np.min(ro
w_summed_average), vmax=np.max(row_summed_average))
    plt.colorbar()
    plt.title("row summed STA's")
    plt.xticks(np.arange(0, 10, step=1) , np.arange(1, 11, step=1))
    plt.yticks(np.arange(0, 16, step=1) , np.arange(1, 17, step=1))
    plt.xlabel("times steps")

    print("Because the LGN cell wants a special order of changes (first darkness
then brightness), this matrix is not space time separable.")

def question1_part_c():

    print("\nAnswer of question 1 part c: \n")

    data = sio.loadmat('c2p3.mat') # loading the data given to us
    counts = data['counts']
    stim = data['stim']

    averages = np.zeros( ( np.shape(stim)[0], np.shape(stim)[1] , 10 ) ) # will f
ill this with STA's for each of the 10 timesteps.

    for i in range(10): # calculating the STA's for the 10 time steps.

        averages[:,:, i ] = STA(stim, counts, i + 1 )


    stimulus_projected_on_STA = np.zeros(len(counts)) # we will fill this with th
e results of each stimuluses projection.
```

```
    for i in range(len(counts)):  # we look at every mesurement point
        for j in range( np.shape(averages)[1] ): # this for loop is for the Frobe
nius inner product calculation

            stimulus_projected_on_STA[i] += np.inner(averages[:,j,0], stim[:,j,i]
)

    stimulus_projected_on_STA = stimulus_projected_on_STA / np.max(np.abs(stimulu
s_projected_on_STA)) # making the maximum 1

    plt.figure(figsize=(12, 6)) # this lets me decide how big the plot should be
    plt.title("histogram of all stimulus projections")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which incre
ases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(stimulus_projected_on_STA, density = True , bins=100)


    nonzero_stimulus_projected_on_STA = [] # this is an empth list we will conver
t it to an array once it reaches its full size.
    for i in range(1 , len(counts)): # we look at every mesurement point

        if 0 != counts[i]: # look if there are any spikes

            inner_product = 0 #we will use this to calculate the inner product

            for j in range( np.shape(averages)[1] ): # this for loop is for the F
robenius inner product calculation

                inner_product += np.inner(averages[:,j,0], stim[:,j,i - 1])


            nonzero_stimulus_projected_on_STA.append(inner_product) # we append t
he summed total to the list

    nonzero_stimulus_projected_on_STA = np.array(nonzero_stimulus_projected_on_ST
A) # converting our list to an array

    nonzero_stimulus_projected_on_STA = nonzero_stimulus_projected_on_STA / np.ma
x(np.abs(nonzero_stimulus_projected_on_STA)) # making the maximum 1

    plt.figure(figsize=(12, 6))  # this lets me decide how big the plot should be
```

```
    plt.title("histogram of stimulus projections at time bins where a non-
zero spike counts accures")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which incre
ases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(nonzero_stimulus_projected_on_STA, bins=100, color = "green", densit
y = True )

    print("The STA doesn’t discrimination spike-
eliciting stimuli very significantly because only around 80% of the Frobenius inn
er products’ result is bigger than 0. Still, we can say the STA discriminates spi
ke-eliciting stimuli but not very significantly.")

    plt.figure(figsize=(12, 6))  # this lets me decide how big the plot should be
    plt.title("histogram of all stimulus projections and stimulus projections at
time bins where a non-zero spike count accures")
    plt.ylabel("normalized spike counts")
    plt.xlabel("projected stimulus")
    plt.grid( alpha=.4) # this is the line which creats the grid line which incre
ases the redability of the plot
    plt.ylim(0, 2) # this is here so all the graphs have the same y axis'
    plt.hist(stimulus_projected_on_STA, bins=100, label="all stimulus", density =
 True )
    plt.hist(nonzero_stimulus_projected_on_STA, bins=100, color = "green", alpha
= 0.5, label="non-zero spike count stimulus", density = True )
    plt.legend(loc='upper right')
    plt.show() # this privents the plot from closing themselfs at the end



def question2_part_a():

    print("\nAnswer of question 2 part a: \n")

    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , 4 , 2)


    plt.figure(figsize=(8, 8)) # code for showing the DOG filter
```

```
    plt.imshow( DOG_receptive_field , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("DOG receptive field")

    plt.figure(figsize=(8 , 8)) # code for showing the DOG filter in 3D
    ax = plt.axes(projection='3d')
    A = B = np.linspace(-10, 10, 21)
    X, Y = np.meshgrid(A, B)
    ax.plot_surface(X, Y, DOG_receptive_field, rstride=1, cstride=1, cmap='viridi
s', edgecolor='none')
    ax.set_title('DOG receptive field')
    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('values of DOG receptive field')
    ax.view_init(elev= 15, azim=35) # where we see the 3D plot from

    print("DOG receptive field is a edge decetion filter")

def DOG (x, y, sigma_s, sigma_c): # the DOG function given in the HW guideline

    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))



def question2_part_b():

    print("\nAnswer of question 2 part b: \n")

    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in t
o Python

    plt.figure(figsize=(8, 8)) # code for showing the DOG filter
    plt.imshow( monkey_img )
    plt.title("the original monkey image")


    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , 4 , 2)
```

```
    neural_response_img = sig.convolve(monkey_img[:, :, 0], DOG_receptive_field ,
 mode="valid") # Puting a DOG filter on every singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered monkey image
    plt.imshow( neural_response_img , cmap = "gray")
    plt.title("monkey image after filtering with DOG receptive field")

    print("Basically, speaking difference-of-
gaussians (DOG) is an edge detecting filter on the image and it makes the edges o
n the image more prominent. ")

def question2_part_c():

    print("\nAnswer of question 2 part c: \n")

    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in t
o Python

    for sigma_c in range(1, 5): # code of testing difrent DOF parameter, threshol
d combinations a
        for sigma_s in range(sigma_c + 1 , 6):
            for threshold in (0 , 2 , 5):

                draw_thresholded_dog_filtered_img(sigma_s, sigma_c, threshold, mo
nkey_img)


    print("\nI belive the best ones are sigma_s = 4, sigma_c = 1, threshold = 0 a
nd sigma_s = 5, sigma_c = 1, threshold = 5 but the one with treash hold 5 is slig
htly better")


def thresholding_filter(threshold, img): # treash holding filter funtion, this wi
ll be re-used in part f

    for i in range(np.shape(img)[0]): # we loop on each pixel of the image to do
a treashold cheak on all of them
        for j in range(np.shape(img)[1]):

            if img[i , j] >= threshold:

                img[i , j] = 1
```

```
            else:

                img[i , j] = 0

    return img


def draw_thresholded_dog_filtered_img(sigma_s, sigma_c, threshold, img):

    DOG_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 DOG field
        for j in range(-10, 11):

            DOG_receptive_field[i + 10 , j + 10] = DOG(j , i , sigma_s , sigma_c)


    neural_response_img = sig.convolve(img[:, :, 0], DOG_receptive_field , mode="
valid") # Puting the DOG filter on every singal pixel.

    filtered_img = thresholding_filter(threshold, neural_response_img)

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered edge detected
monkey image
    plt.imshow(filtered_img , cmap = "gray")
    plt.title("monkey image after DOG filtering with sigma_s = " + str(sigma_s) +
 " and sigma_c = " + str(sigma_c) + "\nthen edge detection with thresholed = " +
str(threshold))



def question2_part_d():

    print("\nAnswer of question 2 part d: \n")

    gabor_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field[i + 10 , j + 10] = gabor(np.array([j , i]) , np
.pi/2 )

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field , origin='lower')
```

```
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 90 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8 , 8)) # code for showing the gabor filter in 3D
    ax = plt.axes(projection='3d')
    A = B = np.linspace(-10, 10, 21)
    X, Y = np.meshgrid(A, B)
    ax.plot_surface(X, Y, gabor_receptive_field, rstride=1, cstride=1, cmap='viri
dis', edgecolor='none')
    ax.set_title("gabor receptive field for 90 degrees");
    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel("values of gabor receptive field for 90 degrees")
    ax.view_init(elev=15, azim=35) # where we see the 3D plot from

    print("From this image we can see gabor filter detects edges with an orientat
ion close to θ degrees and it makes these edges more prominent. ")



def gabor(x, theta): # the gabor function given in the HW guideline
    sigma_l = 3 # this time most of the internal paremeters are taken as a consta
nt because we are not asked to change them.
    sigma_w = 3
    lambda_ = 6
    phi = 0
    k_theta = np.array([np.cos(theta), np.sin(theta)])
    k_theta_ort = np.array([-np.sin(theta), np.cos(theta)])

    D = np.exp( -
( (k_theta.dot(x))**2) / (2*(sigma_l**2))- (k_theta_ort.dot(x)**2) / (2*(sigma_w*
*2))) * np.cos(2 * np.pi * k_theta_ort.dot(x) / lambda_ + phi)

    return D



def question2_part_e():

    print("\nAnswer of question 2 part e: \n")
```

```
    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in t
o Python


    gabor_receptive_field = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field[i + 10 , j + 10] = gabor(np.array([j , i]) , np
.pi/2 )


    neural_response_img = sig.convolve(monkey_img[:, :, 0], gabor_receptive_field
 , mode="valid") # Puting a DOG filter on every singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the DOG filtered edge detected
monkey image
    plt.imshow(neural_response_img , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/2")


    print("From this image we can see gabor filter detects edges with an orientat
ion close to θ degrees and it makes these edges more prominent.")



def question2_part_f():

    print("\nAnswer of question 2 part f: \n")

    monkey_img = plt.imread('hw2_image.bmp') # loading the image given to us in t
o Python

    gabor_receptive_field_pi_over_2 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_pi_over_2[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/2 )

    neural_response_img_pi_over_2 = sig.convolve(monkey_img[:, :, 0], gabor_recep
tive_field_pi_over_2 , mode="valid") # Puting a gabor filter on every singal pixe
l.
```

```
    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_2 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 90 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_2 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/2")


    gabor_receptive_field_pi_over_3 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_pi_over_3[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/3 )

    neural_response_img_pi_over_3 = sig.convolve(monkey_img[:, :, 0], gabor_recep
tive_field_pi_over_3 , mode="valid") # Puting a gabor filter on every singal pixe
l.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_3 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 60 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_3 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/3")


    gabor_receptive_field_pi_over_6 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):
```

```
            gabor_receptive_field_pi_over_6[i + 10 , j + 10] = gabor(np.array([j
, i]) , np.pi/6 )

    neural_response_img_pi_over_6 = sig.convolve(monkey_img[:, :, 0], gabor_recep
tive_field_pi_over_6 , mode="valid") # Puting a gabor filter on every singal pixe
l.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_pi_over_6 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 30 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_pi_over_6 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = pi/6")


    gabor_receptive_field_0 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_0[i + 10 , j + 10] = gabor(np.array([j , i]) ,
0 )

    neural_response_img_0 = sig.convolve(monkey_img[:, :, 0], gabor_receptive_fie
ld_0 , mode="valid") # Puting a gabor filter on every singal pixel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_0 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 0 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_0 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 0")
```

```
    gabor_receptive_field_sum = neural_response_img_pi_over_2 + neural_response_i
mg_pi_over_3 + neural_response_img_pi_over_6 + neural_response_img_0

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered m
onkey images
    plt.imshow(gabor_receptive_field_sum , cmap = "gray")
    plt.title("sum of 4 monkey images with 4 difrent angeled gabor filters")

    # these are test done to improve the results

    gabor_receptive_field_sum_after_TH = thresholding_filter(150 , gabor_receptiv
e_field_sum) # threshold filtering the result

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered m
onkey images after the sum is threshold filtered
    plt.imshow(gabor_receptive_field_sum_after_TH , cmap = "gray")
    plt.title("sum of 4 monkey images with 4 difrent angeled gabor filters \nafte
r it is threshold filtered")


    print("threshold can be further improved by adding the missing 120 degrees an
d 150 degrees edges. So, I will add them.")


    gabor_receptive_field_2pi_over_3 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_2pi_over_3[i + 10 , j + 10] = gabor(np.array([j
 , i]) , 2*np.pi/3 )

    neural_response_img_2pi_over_3 = sig.convolve(monkey_img[:, :, 0], gabor_rece
ptive_field_2pi_over_3 , mode="valid") # Puting a gabor filter on every singal pi
xel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_2pi_over_3 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 120 degrees")
    plt.ylabel("Y axis")
```

```
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_2pi_over_3 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 2pi/3")


    gabor_receptive_field_5pi_over_6 = np.zeros((21 , 21)) # we will file this in

    for i in range(-10, 11): # creation of the 21 21 gabor field
        for j in range(-10, 11):

            gabor_receptive_field_5pi_over_6[i + 10 , j + 10] = gabor(np.array([j
 , i]) , 5*np.pi/6 )

    neural_response_img_5pi_over_6 = sig.convolve(monkey_img[:, :, 0], gabor_rece
ptive_field_5pi_over_6 , mode="valid") # Puting a gabor filter on every singal pi
xel.

    plt.figure(figsize=(8, 8)) # code for showing the gabor filter
    plt.imshow( gabor_receptive_field_5pi_over_6 , origin='lower')
    plt.colorbar()
    plt.xticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.yticks(np.arange(0, 21, step=1) , np.arange(-10, 11, step=1))
    plt.title("gabor receptive field for 150 degrees")
    plt.ylabel("Y axis")
    plt.xlabel("X axis")

    plt.figure(figsize=(8, 8)) # code for showing the gabor filtered monkey image
    plt.imshow(neural_response_img_5pi_over_6 , cmap = "gray")
    plt.title("monkey image after gabor filtering with theta = 5pi/6")


    gabor_receptive_field_sum_2 = neural_response_img_pi_over_2 + neural_response
_img_pi_over_3 + neural_response_img_pi_over_6 + neural_response_img_0 + neural_r
esponse_img_2pi_over_3 + neural_response_img_5pi_over_6

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered m
onkey images
    plt.imshow(gabor_receptive_field_sum_2 , cmap = "gray")
    plt.title("sum of 6 monkey images with 6 difrent angeled gabor filters")

    # these are test done to improve the results
```

```
    gabor_receptive_field_sum_after_TH_2 = thresholding_filter(250 , gabor_recept
ive_field_sum_2) # threshold filtering the result

    plt.figure(figsize=(8, 8)) # code for showing the sum of the gabor filtered m
onkey images after the sum is threshold filtered
    plt.imshow(gabor_receptive_field_sum_after_TH_2 , cmap = "gray")
    plt.title("sum of 6 monkey images with 6 difrent angeled gabor filters \nafte
r it is threshold filtered")
    plt.show() # this privents the plot from closing themselfs at the end



Batu_Arda_Düzgün_21802633_hw2(question) # this is the only line which runs so it
is very important.
```