Homework 2 Report

Ayhan Okuyan
ayhan.okuyan[at]ug.bilkent.edu.tr

March 27, 2020

Contents

Question 1

This question asks us to work mainly with STAs (Spike Triggered Averages). STA is a concept that is used to explain the non-linearity in a neuron, to describe what the specific neuron is selective for. As data, we are given two variables in .mat format. The variable "counts" is a vector describing the number of spikes that occured in 15.6 ms time bins for a cat LGN cell. Similarly, "stim" is the the stimulus present at the coordinate (x, y) at a time t, where t is the time that corresponds to the bin number. The choice of programming language for this question and assignment is Python. The library imports that are used throughout are given below.

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

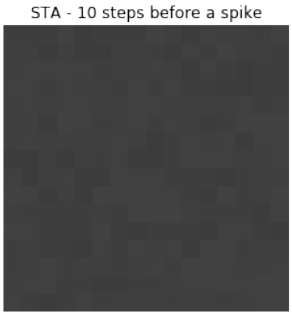
Part A This part of the question requires us to compute the STA (Spike-Triggered Average) images for each of the 10 time steps before each spike and plot them using imagesc with grayscale mapping. Spike-triggered average is a powerful tool in understanding a neuron's response characteristics. STA uses a moving window summation to compute average stimulus before a spike. Since we are using Python and not MATLAB, we have implemented this procedure with imshow function of matplotlib. In order to resolve this, the following code is written.

If spike is detected at a time point $t+10$, then the previous ten images are summed on their respective matrices. We should note that this approach completely avoids the data present in the first 10 time bins. Since there exists 32767 time bins, the first 10 consists of only 0.0305% of the data, which was decided to be negligible given the extra computational cost that should be used to include them. Also we are using this procedure to firmly understand what the neuron is selective for and it wouldn't alter the outcome significantly. The code is provided below.

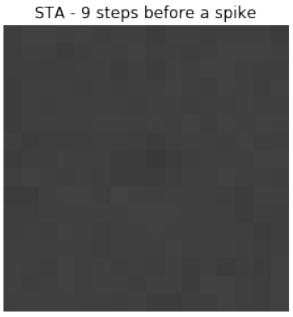
```
sta_images = np.zeros((10,16,16))

for spk in range(len(counts)-10):
    if counts[spk+10]>0:
        for i in range(10):
            sta_images[i,:,:] += stim[:,:,spk+i]*counts[spk+10]
```

Then, we plot the sta_images with respect to its first dimension as 10 separate images. The STA images are presented through Figures 1(a-j). The code used to plot these normalized STA images is provided as follows.

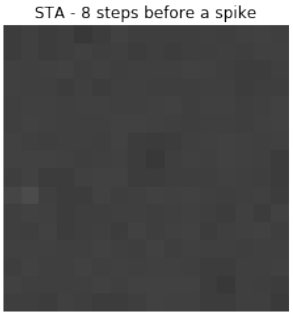
```
min_sta_val = np.min(sta_images)
max_sta_val = np.max(sta_images)

for i in range(sta_images.shape[0]):
    plt.imshow(sta_images[i,:,:], cmap='gray', vmin=min_sta_val,
                                              vmax=max_sta_val)
    plt.title('STA - %d steps before a spike' % (10 -i))
    plt.axis('off')
    plt.show()
```



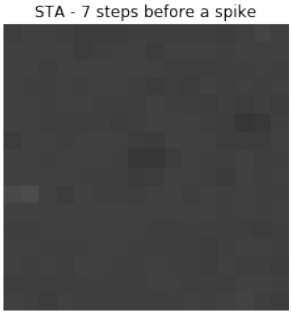
(a) 10 Steps before Spike



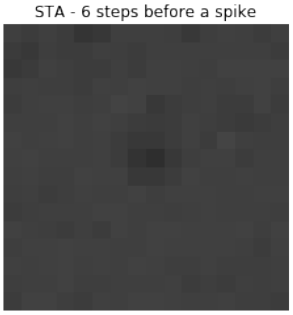
(b) 9 Steps before Spike



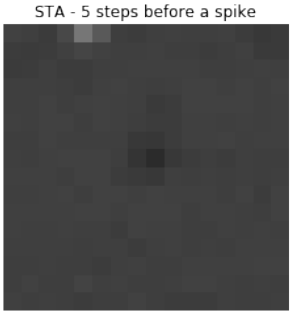
(c) 8 Steps before Spike



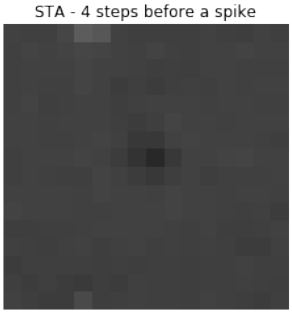
(d) 7 Steps before Spike



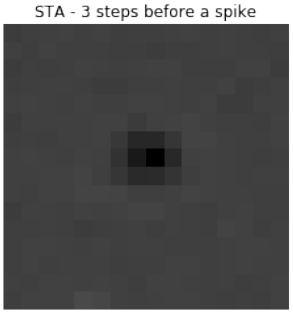
(e) 6 Steps before Spike



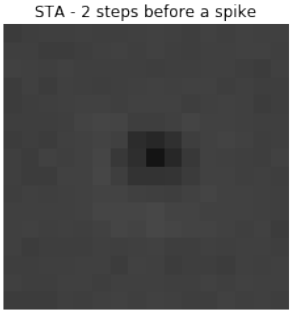
(f) 5 Steps before Spike



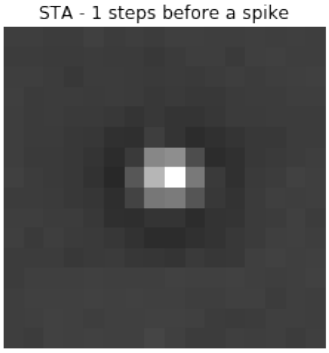
(g) 4 Steps before Spike



(h) 3 Steps before Spike



(i) 2 Steps before Spike

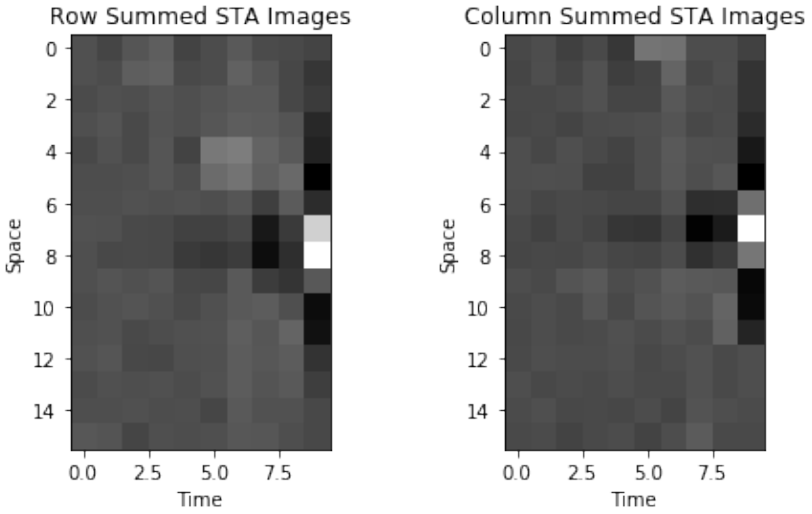


(j) 1 Step before Spike

Figure 1: 10 STA Images for the given cat LGN cell

Usually LGN neurons are sensitive to either direction, orientation or luminosity. All the regions that are highlighted in the STA images are round-shaped and centered in the middle, hence we can observe that the neuron is not sensitive to direction or orientation. We can then conclude that the neuron is sensitive to changes in luminosity. From these images, we observe two sequential conditions for this neuron to fire a spike. The first one seems to be a radial decrease in luminosity in the center area and the second one is a sudden increase in the luminosity again in the same area.

Part B This part of the question requires us to sum the STA images obtained in the previous part in one spatial direction in order to observe an average change across time axis. These summed images are shown in the Figures 2(a-b).



(a) Row Summed STA Images (b) Column Summed STA Images

Figure 2: STA Images Summed over Spatial Dimensions

The code used to obtain these images are provided below.

```
sta_sum = np.sum(sta_images,axis=1).T
plt.imshow(sta_sum, cmap='gray')
plt.title('Row Summed STA Images')
plt.xlabel('Time')
plt.ylabel('Space')
plt.show()

sta_sum = np.sum(sta_images,axis=2).T
plt.imshow(sta_sum, cmap='gray')
plt.title('Column Summed STA Images')
plt.xlabel('Time')
plt.ylabel('Space')
plt.show()
```

Here, we are easily able to observe that the response of the neuron is independent of the spatial dimensions since the summed images are nearly the same with each

other. It also can be observed mainly that the changes are happening on the pixels 4 to 10, which can be classified as the middle pixels. This observation leads us to the fact that this LGN cell responds to changes in the central pixels.

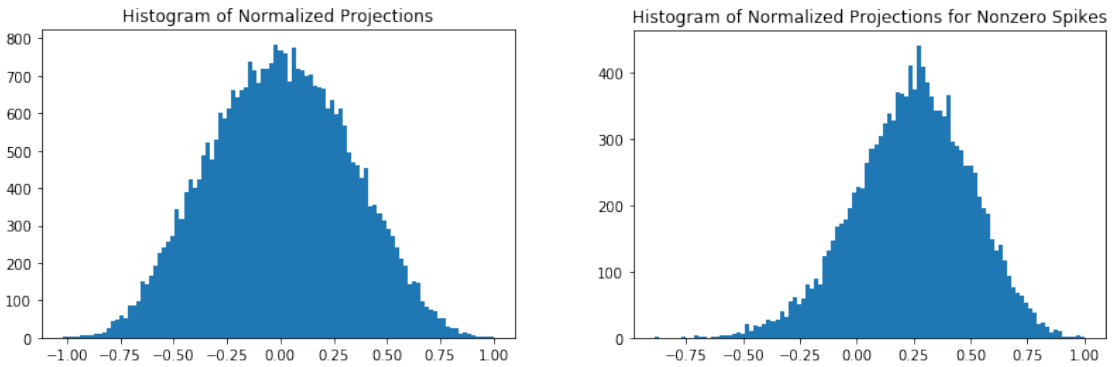
We can say that the matrices given in Figure 2 are not space-time separable since if they were, we should be observe a data where the columns of the matrices would be identical to each other in the same matrix, being time-independent.

Part C Frobenius inner product is an extended form of the commonly-used inner product where this time the inputs of the multiplication are two matrices instead of two vectors, defined as follows.

$$\langle A_{nxm}, B_{nxm} \rangle_F = \sum_i \sum_j A_{i,j} B_{i,j}, \qquad n, m \in \mathbb{Z}^+ \tag{1}$$

The question requires us to use this inner product to project the stimulus onto the STA image one time step prior to the spike occurrence. Then we are asked to create two histograms with normalized values whose maximum reaches one, one for all projections and one for the time bins where a non-zero spike count was observed. The following script is used to do that and the results are presented in Figure 3.

```
projections = []
for spk in range(len(counts)-1):
    projections.append(np.sum(stim[:,:,spk]*sta_images[9,:,:]))
projections /= np.max(projections)

plt.hist(projections, bins=100)
plt.title('Histogram of Normalized Projections')
plt.show()

nonzero_projections = []
for spk in range(len(counts)-1):
    if counts[spk+1]>0:
            nonzero_projections.append(np.sum(stim[:,:,spk]*
                                              sta_images[9,:,:])
                                              )
nonzero_projections /= np.max(nonzero_projections)
plt.hist(nonzero_projections, bins=100)
plt.title('Histogram of Normalized Projections for Nonzero Spikes'
                                    )
plt.show()
```



(a) All Occurrences (b) Nonzero Spike Occurrences

Figure 3: Histograms for Normalized Stimulus Projections

Furthermore, the question requires us to use a bar plot to join the Figures 3-a and 3-b to compare them and comment on whether STA discriminates spike-eliciting stimuli.The code segment and the plot is provided. For visualization purposes, the individual histograms use 100 bins while the joined histogram uses 55, which changes the counts but not the relationship between the histograms.

```
plt.hist([np.asarray(projections),np.asarray(nonzero_projections)]
                                          ,bins=55)
plt.title('Bar Plot Comparisons for All and Nonzero Projections')
plt.show()
```

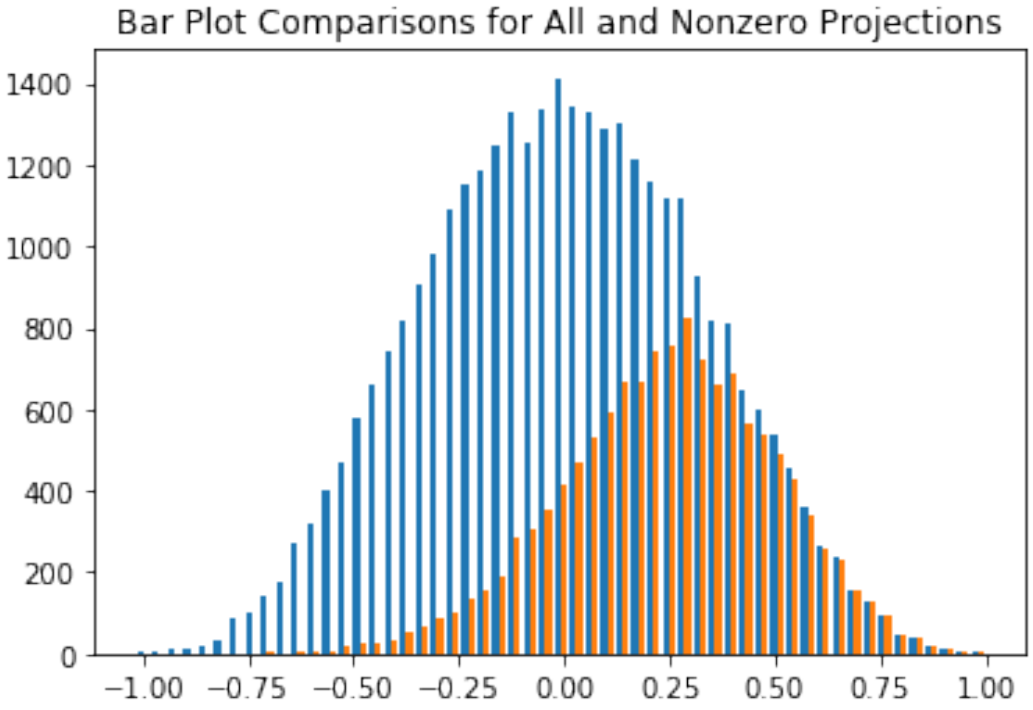


Figure 4: Comparison of the Histograms

In Figure 4, the histogram that is blue refers to the one where we include all of the projections, and the orange one refers to the one in which we only take into consideration the nonzero spike eliciting projections. We can observe two important details from the histogram. The blue one resembles a Gaussian distribution with zero mean, meaning that the projection values are close to zero in most of the cases. However, when we look at the orange histogram, the shape while resembling also a Gaussian, looks like it has a mean around 0.25. This conveys the idea that most of the nonzero spike projections are centered around that value. We can then conclude that the absence of the stimuli that do not elicit any spike, shifts the mean of the distribution significantly. Hence, we can easily discriminate the spike-eliciting stimuli by simply projecting them onto the STA image, which makes STA an important tool.

Question 2

This question requires us to construct two commonly used kernel filters, which are Difference of Gaussians (DoG) and Gabor filters since, these kernels are used in describing the neuron activity in Lateral Geniculate Nuclei (LGN) cells and simple cells in V1 respectively. After constructing the filters, asks several other questions that requires us to convolve the kernels with a given example image, hw2_image.bmp. Which is an RGB grayscale image of a baboon. The question is solved through Python 3 scripting and the used packages are provided below.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D
from PIL import Image
from scipy import signal
```

Part A This part requires us to build an on-center Difference of Gaussians (DoG) center-surround receptive field centered at zero. DoG is defined as given below, which is the difference of two Gaussian filters whose value is defined by $x^2 + y^2$ instead of x^2 and mean given as zero with two separate variances defined as σ_c^2 and σ_s^2 that corresponds to the terms central and surround.

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

To define this function, we use the following code segment.

```
def dif_of_gauss(x, y, std_c, std_s):
    central_gaussian = (0.5/(np.pi*std_c**2))*np.exp(-(x**2+y**2)/
                                                     (2*std_c**2))
    surround_gaussian = (0.5/(np.pi*std_s**2))*np.exp(-(x**2+y**2)
                                                     /(2*std_s**2))
    return central_gaussian - surround_gaussian
```

Then, the question asks us to construct a DoG receptive field with the following paramters.

$$dimensions = (21x21)$$
$$\sigma_c = 2$$
$$\sigma_s = 4$$

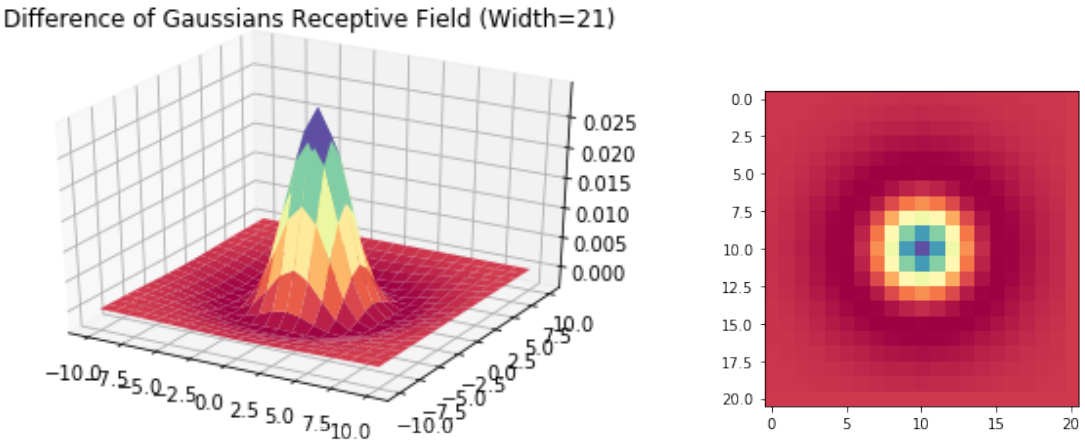
in units of pixels. Hence, we create the following function to generate a receptive field.

```
def dog_receptive_field(std_c, std_s, width=21):
    receptive_field = np.zeros((width,width))
    min_edge = int(-(width-1)/2)
    max_edge = int((width-1)/2)
    x_ind = 0
    y_ind = 0
    for i in range(min_edge,max_edge,1):
```

```
        for j in range(min_edge,max_edge,1):
            receptive_field[x_ind,y_ind] = dif_of_gauss(i,j,std_c,
                                                        std_s)
            x_ind += 1
        y_ind += 1
        x_ind = 0
    return receptive_field
```

To display, we use the plot_surface function of matplotlib and imshow to display the filter in 3D and 2D respectively. The code is as follows and the output is given in Figure 5.

```
receptive_field = dog_receptive_field(2,4)

fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.linspace(-10, 10, 21)
Y = X
X, Y = np.meshgrid(X, Y)
ax.plot_surface(X, Y, receptive_field, cmap='Spectral', edgecolor=
                                        'none')
plt.title("Difference of Gaussians Receptive Field (Width=21)")
plt.show(fig)

plt.imshow(receptive_field, cmap='Spectral')
plt.show()
```



(a) DoG Receptive Field (3D)

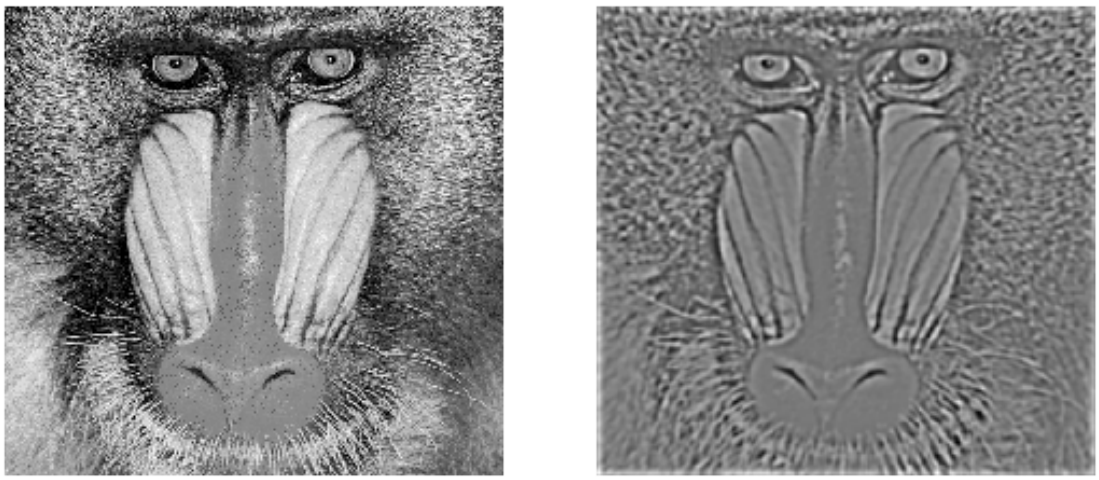
(b) 2D View of DoG Receptive Field

Figure 5: Created DoG Receptive Field

Part B Here, we are suppose there is a separate LGN neuron that is centered at each pixel of the given image. Hence, we compute the response of the neurons and this problem ultimately turns into a two-dimensional convolution problem. Hence, we have used scipy's convolve2d function to convolve the kernel with the image and the results are as follows. Furthermore, since the image is an RGB grayscale image

with three channels identical to each other, we have only used one channel instead of all three.

```
img = Image.open('hw2_image.bmp')
img = np.asarray(img)[:,:,0]
print(img.shape)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()

out = signal.convolve2d(img, receptive_field, mode='same')
plt.imshow(out, cmap='gray')
plt.axis('off')
plt.show()
```



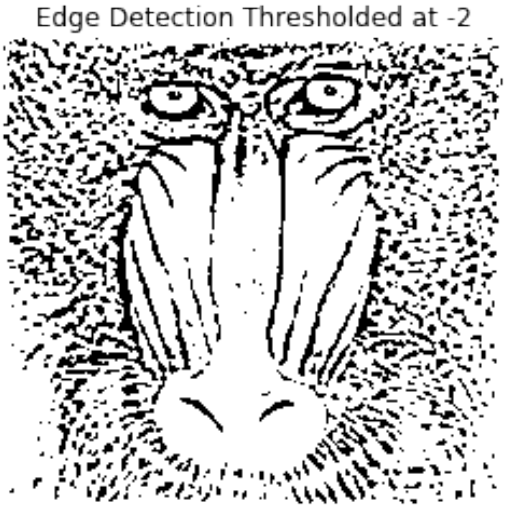
(a) Original Image (b) Response of the Neurons

Figure 6: Original Image and the Neural Activity

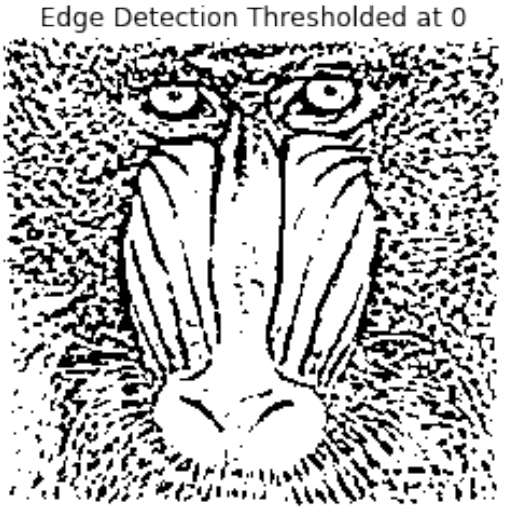
Here, we observe that a center-surround receptive field acts like a spatial filter and can be characterized by a frequency filter transfer function. The filter converges to a band-pass filter where the middle frequencies are attenuated, which enables a technique for edge detection in an image.We see that the parts in the original image where there is a high amount of gradient change between pixels are highlighted.

Part C This part requires us to construct a binary edge detector by thresholding the neural activity that we have covered in the previous part according to trial and error procedure. After observing the maximum and minimum values of the processed image.Here, we have observed four values that can be optimal threshold values being $-2, 0, 2, 5$. Over these values, we have observed that the optimal value of threshold is 0 since the -2 loses some edges while there are parts of the baboon's nose included present in the image with threshold value of 2 which should not occur. For this image at least, zero thresholding is found to be the best option. For simplicity we have only inserted the code that presents the best result.

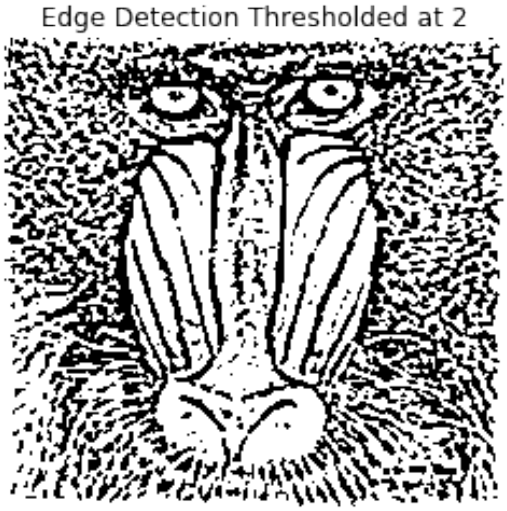
```
out_thr = np.copy(out)
threshold_value = 0
out_thr[out_thr > threshold_value] = 1
out_thr[out_thr <= threshold_value] = 0
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at 0')
plt.axis('off')
plt.show()
```



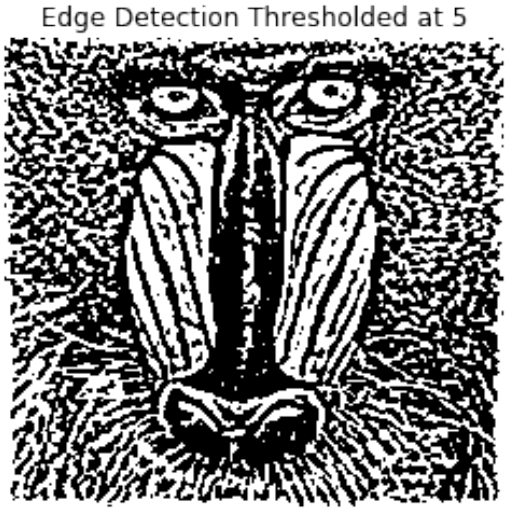
(a) Threshold Value = −2



(b) Threshold Value = 0



(c) Threshold Value = 2



(d) Threshold Value = 5

Figure 7: DoG Filtered Binary Edge Detected Images

Part D This part requires us to create a Gabor receptive field, again on a $21x21$ grid as we have done with the DoG approach. The Gabor function is defined as follows.

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

Here, we observe that the coordinate input of the function is provided with $\vec{x}$. Since, we are dealing with a two dimensional vector space, we can define that vector as $\vec{x} = (x, y)$. Additionally, the equation describes to orthonormal vectors $\vec{k}(\theta)$ and $\vec{k}_\perp(\theta)$, which are two unit vectors one with orientation θ and one being orthogonal to the other. Hence, we can define these two vectors as,

$$\vec{k}(\theta) = (\cos\theta, \sin\theta) \quad (3)$$
$$\vec{k}_\perp(\theta) = (-\sin\theta, \cos\theta) \quad (4)$$

Hence, we can re-express the dot products given in (5) as follows.

$$\vec{k}(\theta)\cdot\vec{x} = x\cos\theta + y\sin\theta \quad (5)$$
$$\vec{k}_\perp(\theta)\cdot\vec{x} = -x\sin\theta + y\cos\theta \quad (6)$$

After understanding the mathematics behind, we define the Gabor function provided below to construct the function.

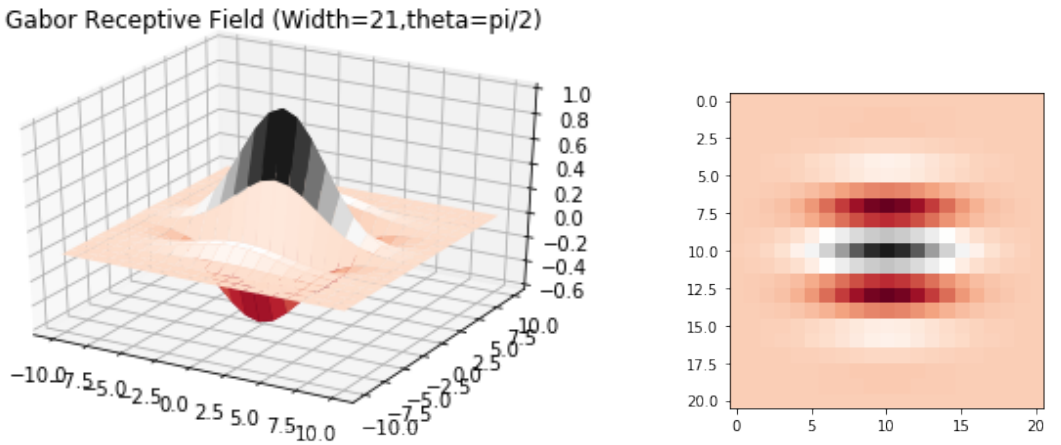
```
def gabor(x, theta, sigma_l, sigma_w, lamda, phi):
    k_theta = np.asarray([np.sin(theta), np.cos(theta)])
    k_orth_theta = np.asarray([np.cos(theta), -np.sin(theta)])
    gaussian_part_l = -(np.dot(k_theta, x))**2/(2*sigma_l**2)
    gaussian_part_w = -(np.dot(k_orth_theta, x))**2/(2*sigma_w**2)
    orientation_part = np.cos((2*np.pi*np.dot(k_orth_theta,x)/
                                        lamda)+phi)
    return np.exp(gaussian_part_l+gaussian_part_w)*
                                        orientation_part
```

Then we construct and visualize the receptive field using the following function with the parameters,

$$\theta = \pi/2$$
$$\sigma_l = \sigma_w = 3$$
$$\lambda = 6$$
$$\phi = 0$$

```
theta = np.pi/2
sigma_l = sigma_w = 3
lamda = 6
phi = 0
gabor_rec_field = gabor_receptive_field(sigma_l, sigma_w, theta,
                                        lamda, phi)
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.linspace(-10, 10, 21)
Y = X
X, Y = np.meshgrid(X, Y)
ax.plot_surface(X, Y, gabor_rec_field, cmap='RdGy', edgecolor='
                                        none')
plt.title("Gabor Receptive Field (Width=21,theta=pi/2)")
plt.show(fig)

plt.imshow(gabor_rec_field, cmap='RdGy')
plt.show()
```



(a) Gabor Receptive Field (3D)

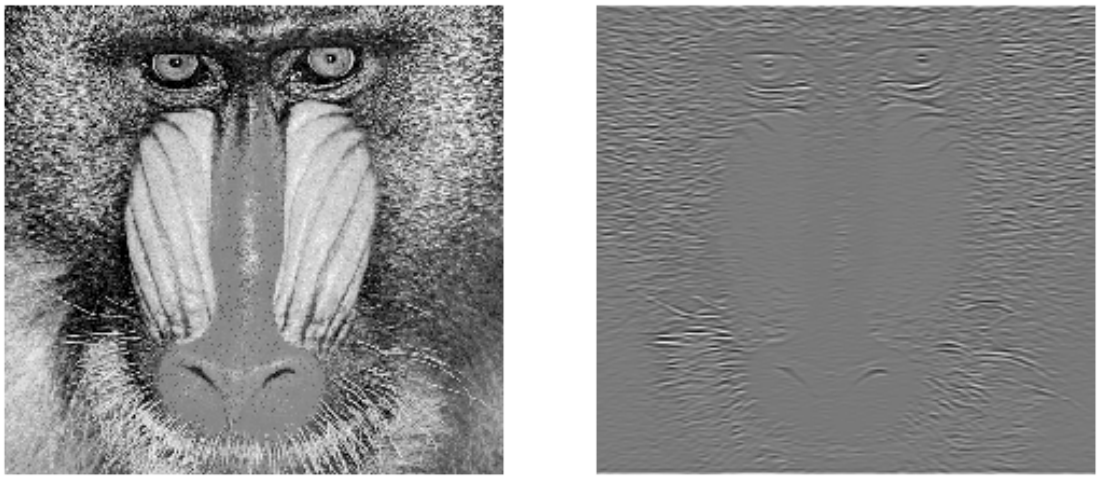
(b) 2D View of Gabor Receptive Field

Figure 8: Created Gabor Receptive Field

After observing the generated receptive field, we can try to comprehend how Gabor filter functions better. Again we see a relative change of values through the center of the kernel similar to the DoG receptive field. However, we see that the shape is more linear instead of circular and there exists one central peak and two minima. Also, we can observe that the theta is used to choose orientation which was $\pi/2$ in our case. Furthermore, there is a gradient decrease in the magnitudes of the hills, which we believe is a result of the cosine factor. Here, we understand that the receptive field is selective for orientation.

Part E This part consists of the same procedure that we ahve gone through in Part B, where we suppose that there exists Gabor receprive fields centered on each pixel of the image, which will be a convolution operation, hence we use the same function as in Part B and the result is given below.

```
out = signal.convolve2d(img, gabor_rec_field, mode='same')
plt.imshow(out, cmap='gray')
plt.axis('off')
plt.show()
```



(a) Original Image (b) Response of the Neurons

Figure 9: Original Image and the Neural Activity

In order to better understand what this image is representative of, we use the same thresholding techniques that we have applied in Part C to clearly observe the outcome in Figure 9-b.

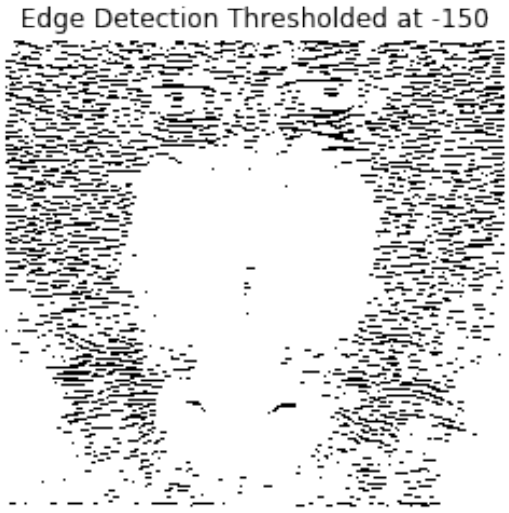


Figure 10: Gabor Filtered Image Thresholded at -150

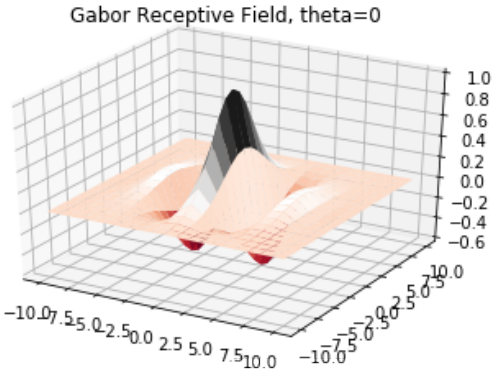
From Figure 10, we observe the dominance of the lines that are parallel to the x-axis. Hence, we can say that the Gabor filter does not by itself form a procedure enough to apply edge detection like DoG kernel, however we clearly see that the filter selects the lines that are horizontal in the image, which we believe corresponds to the θ angle if the starting axis is taken as the y-axis.

Part F This parts asks us to construct four Gabor filters with sepearte θ orientations given as $\theta = [0, \pi/6, \pi/3, \pi/2]$ and then sum the response images on top of each other to again build an edge detector and comment on the quality. We first created the receptive fields and visualized them in both 2-D and 3-D as we have done in previous parts and then found all the responses and summed them up in a single for loop which is provided below.

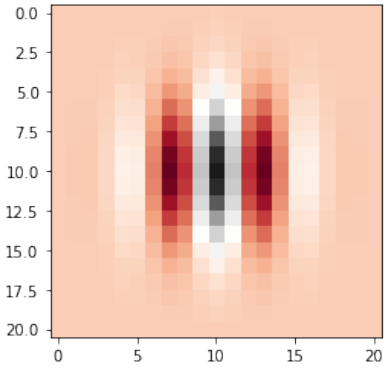
```
theta_list = [0, np.pi/6, np.pi/3, np.pi/2]
theta_list_str = ['0', 'pi/6', 'pi/3', 'pi/2']
sigma_l = sigma_w = 3
lamda = 6
phi = 0

ind = 0
out_f = np.zeros(out.shape)
for el in theta_list:
    gabor_temp = gabor_receptive_field(sigma_l, sigma_w, el, lamda
                                       , phi)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X = np.linspace(-10, 10, 21)
    Y = X
    X, Y = np.meshgrid(X, Y)
    ax.plot_surface(X, Y, gabor_temp, cmap='RdGy', edgecolor='none
                                       ')
    plt.title("Gabor Receptive Field, theta=%s" % theta_list_str[
                                       ind])
    ind += 1
    plt.show(fig)

    plt.imshow(gabor_temp, cmap='RdGy')
    plt.show()

    out_temp = signal.convolve2d(img, gabor_temp, mode='same')
    plt.imshow(out_temp, cmap='gray')
    plt.axis('off')
    plt.show()
    out_f += out_temp

plt.imshow(out_f, cmap='gray')
plt.axis('off')
plt.show()
```

The receptive fields in both their 3-D and 2-D form are given below.

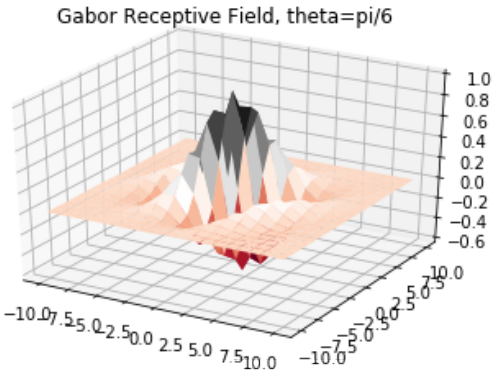
After creating the fields, we find their neural responses separately and sum them to find the combined responses in Figures 12 and 13.



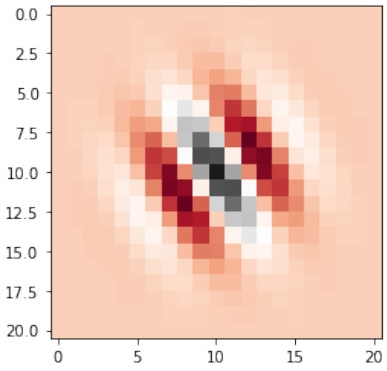
(a) Gabor Receptive Field (3D), $\theta = 0$



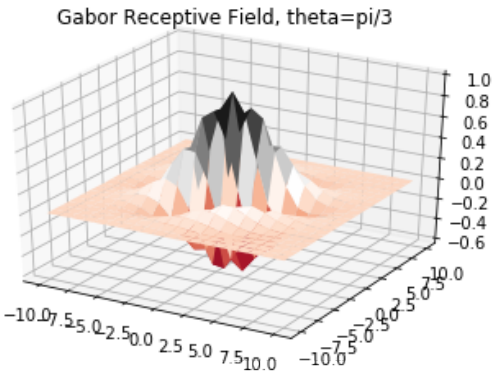
(b) 2D View of Gabor Receptive Field, $\theta = 0$



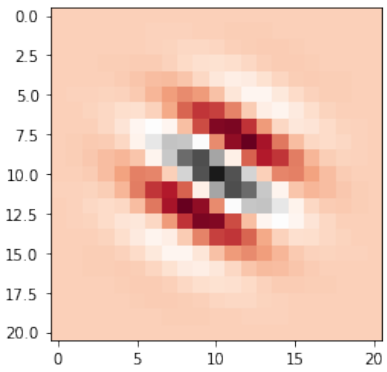
(a) Gabor Receptive Field (3D), $\theta = 30$



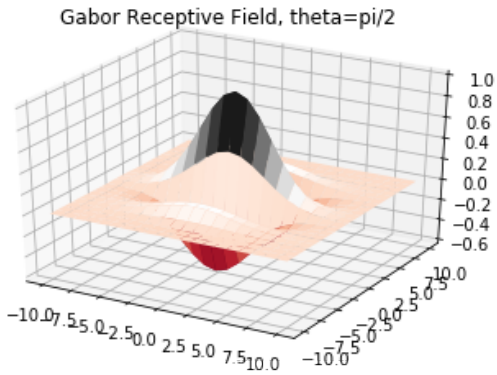
(b) 2D View of Gabor Receptive Field, $\theta = 30$



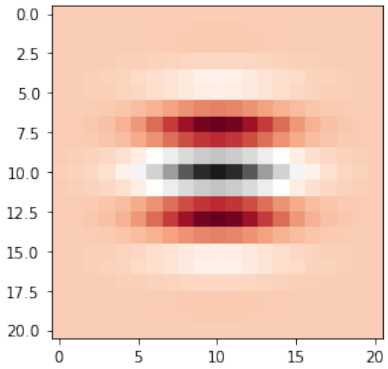
(a) Gabor Receptive Field (3D), $\theta = 60$



(b) 2D View of Gabor Receptive Field , $\theta = 60$

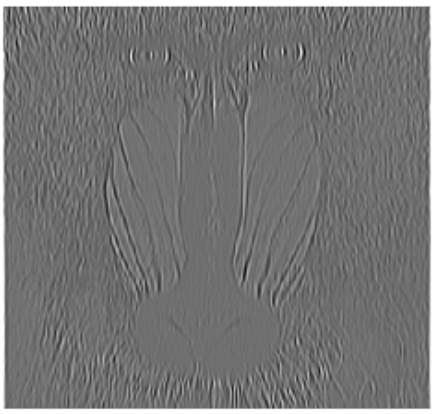


(a) Gabor Receptive Field (3D), $\theta = 90$



(b) 2D View of Gabor Receptive Field , $\theta = 90$

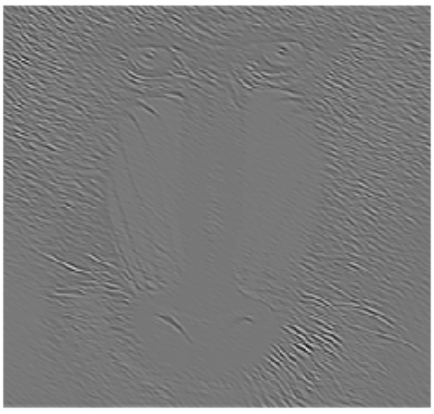
Figure 11: Created Gabor Receptive Fields for $\theta = [0, \pi/6, \pi/3, \pi/2]$



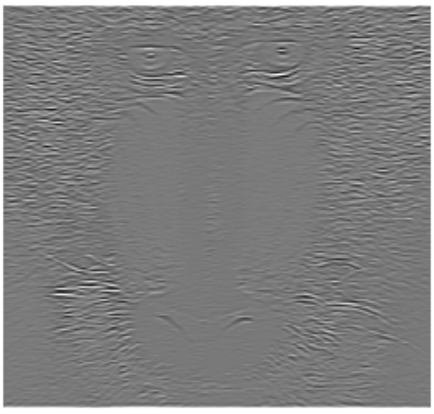
(a) $\theta = 0$

(b) $\theta = \pi/6$

Figure 12: Neural Responses of Separate Gabor Filters for $\theta = [0, \pi/6]$



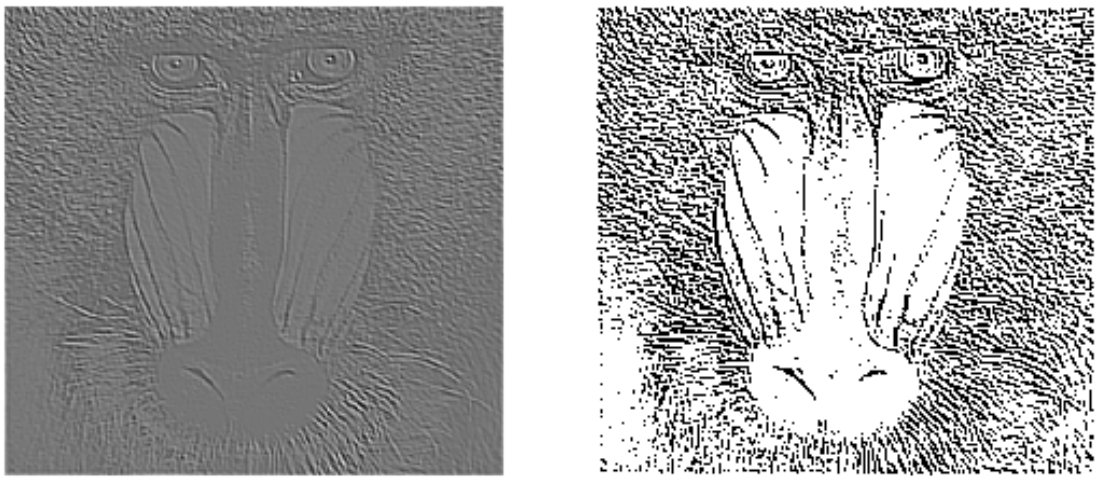
(a) $\theta = \pi/3$



(b) $\theta = \pi/2$

Figure 13: Neural Responses of Separate Gabor Filters for $\theta = [\pi/3, \pi/2]$

Parallel to what is expected, the responses look the same in terms of texture, however their orientations are different from each other since the value of θ is different in each filter which is the deciding factor for orientation. After successfully recovering the responses, we linearly combine them by adding them on top of each other. Then in order to observe the edge detection performance, we threshold this image. The results are given in Figure 14a-b.



(a) Summation of Outputs

(b) Thresholded Version

Figure 14: Summation of Individual Neural Responses and the Thresholded Binary Image

The edge detection performance looks better in this case when compared with both the individual Gabor responses and the DoG filtering. We can say that even though the Gabor filter is selective for one orientation, it is more specific in that orientation than the DoG response and the ensemble of many Gabor filters results in a high-quality edge detection performance. The DoG filtering's edge detection performance looks crude when compared with this one. The ensemble model is a representation of a Complex V1 cell in the sense that it requires the linear combination of outputs of individual Simple V1 cells with different orientation selectivity features. In order to improve performance, we can use the phase angle ϕ to alter the filters in frequency domain to make them off-phase with each other. We can tune or train weight parameters to linearly combine the outputs and not take them all as 1 as we have done. We can also use separate thresholds for individual responses and then sum them up.

Python Code

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as io
from mpl_toolkits.mplot3d import axes3d, Axes3D
from PIL import Image
from scipy import signal

question = sys.argv[1]

def ayhan_okuyan_21601531_hw2(question):
    if question == '1' :
        print('Question 1')

        data = io.loadmat('c2p3.mat')
        counts = data['counts']
        stim = data['stim']
        #print(counts.shape)
        #print(stim.shape)

        print('Part A')
        sta_images = np.zeros((10, 16, 16))

        for spk in range(len(counts) - 10):
            if counts[spk + 10] > 0:
                for i in range(10):
                    sta_images[i, :, :] += stim[:, :, spk + i] * counts[
                                                        spk + 10]

        min_sta_val = np.min(sta_images)
        max_sta_val = np.max(sta_images)

        for i in range(sta_images.shape[0]):
            plt.imshow(sta_images[i, :, :], cmap='gray', vmin=
                                                min_sta_val, vmax=
                                                max_sta_val)
            plt.title('STA - %d steps before a spike' % (10 - (i)))
            plt.axis('off')
            plt.show()

        print('Part B')
        #print(sta_images.shape)
        sta_sum = np.sum(sta_images, axis=1).T
        plt.imshow(sta_sum, cmap='gray')
        plt.title('Row Summed STA Images')
        plt.xlabel('Time')
        plt.ylabel('Space')
        plt.show()

        sta_sum = np.sum(sta_images, axis=2).T
        plt.imshow(sta_sum, cmap='gray')
        plt.title('Column Summed STA Images')
```

```
        plt.xlabel('Time')
        plt.ylabel('Space')
        plt.show()

        print('Part C')
        projections = []

        for spk in range(len(counts) - 1):
            projections.append(np.sum(stim[:, :, spk] * sta_images[9, :,
                                                                   :]))
        projections /= np.max(projections)

        plt.hist(projections, bins=100)
        plt.title('Histogram of Normalized Projections')
        plt.show()

        nonzero_projections = []

        for spk in range(len(counts) - 1):
            if counts[spk + 1] > 0:
                nonzero_projections.append(np.sum(stim[:, :, spk] *
                                                  sta_images[9, :, :])
                                                  )
        nonzero_projections /= np.max(nonzero_projections)
        plt.hist(nonzero_projections, bins=100)
        plt.title('Histogram of Normalized Projections for Nonzero
                                                  Spikes')
        plt.show()

        plt.hist([np.asarray(projections), np.asarray(
                                                  nonzero_projections)], bins=
                                                  55)
        plt.title('Bar Plot Comparisons for All and Nonzero Projections'
                                                  )
        plt.show()

    elif question == '2' :
        print('Question 2')

        print('Part A')
        receptive_field = dog_receptive_field(2, 4)

        fig = plt.figure()
        ax = fig.gca(projection='3d')
        X = np.linspace(-10, 10, 21)
        Y = X
        X, Y = np.meshgrid(X, Y)
        ax.plot_surface(X, Y, receptive_field, cmap='Spectral',
                                                  edgecolor='none')
        plt.title("Difference of Gaussians Receptive Field (Width=21)")
        plt.show(fig)

        plt.imshow(receptive_field, cmap='Spectral')
        plt.show()
```

```
print('Part B')
img = Image.open('hw2_image.bmp')
img = np.asarray(img)[:, :, 0]
#print(img.shape)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()

out = signal.convolve2d(img, receptive_field, mode='same')
plt.imshow(out, cmap='gray')
plt.axis('off')
plt.show()

print('Part C')
out_thr = np.copy(out)
#print(np.min(out_thr), np.max(out_thr))
threshold_value = -2
out_thr[out_thr > threshold_value] = 1
out_thr[out_thr <= threshold_value] = 0
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at -2')
plt.axis('off')
plt.show()

out_thr = np.copy(out)
threshold_value = 0
out_thr[out_thr > threshold_value] = 1
out_thr[out_thr <= threshold_value] = 0
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at 0')
plt.axis('off')
plt.show()

out_thr = np.copy(out)
threshold_value = 2
out_thr[out_thr <= threshold_value] = 0
out_thr[out_thr > threshold_value] = 1
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at 2')
plt.axis('off')
plt.show()

out_thr = np.copy(out)
threshold_value = 5
out_thr[out_thr <= threshold_value] = 0
out_thr[out_thr > threshold_value] = 1
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at 5')
plt.axis('off')
plt.show()

print('Part D')
theta = np.pi / 2
sigma_l = sigma_w = 3
lamda = 6
```

```
phi = 0
gabor_rec_field = gabor_receptive_field(sigma_l, sigma_w, theta,
                                        lamda, phi)

fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.linspace(-10, 10, 21)
Y = X
X, Y = np.meshgrid(X, Y)
ax.plot_surface(X, Y, gabor_rec_field, cmap='RdGy', edgecolor='
                                       none')
plt.title("Gabor Receptive Field (Width=21,theta=pi/2)")
plt.show(fig)

plt.imshow(gabor_rec_field, cmap='RdGy')
plt.show()

print('Part E')
out = signal.convolve2d(img, gabor_rec_field, mode='same')
plt.imshow(out, cmap='gray')
plt.axis('off')
plt.show()

out_thr = np.copy(out)
#print(np.min(out_thr), np.max(out_thr))
threshold_value = 0
out_thr[out_thr > threshold_value] = 1
out_thr[out_thr <= threshold_value] = 0
plt.imshow(out_thr, cmap='gray')
plt.title('Edge Detection Thresholded at -150')
plt.axis('off')
plt.show()

print('Part F')
theta_list = [0, np.pi / 6, np.pi / 3, np.pi / 2]
theta_list_str = ['0', 'pi/6', 'pi/3', 'pi/2']
sigma_l = sigma_w = 3
lamda = 6
phi = 0

ind = 0
out_f = np.zeros(out.shape)
for el in theta_list:
    gabor_temp = gabor_receptive_field(sigma_l, sigma_w, el,
                                       lamda, phi)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X = np.linspace(-10, 10, 21)
    Y = X
    X, Y = np.meshgrid(X, Y)
    ax.plot_surface(X, Y, gabor_temp, cmap='RdGy', edgecolor='
                                       none')
    plt.title("Gabor Receptive Field, theta=%s" % theta_list_str
                                       [ind])
```

```
            ind += 1
            plt.show(fig)

            plt.imshow(gabor_temp, cmap='RdGy')
            plt.show()

            out_temp = signal.convolve2d(img, gabor_temp, mode='same')
            plt.imshow(out_temp, cmap='gray')
            plt.axis('off')
            plt.show()
            out_f += out_temp

        plt.imshow(out_f, cmap='gray')
        plt.axis('off')
        plt.show()

        out_ff = np.copy(out_f)
        #print(np.max(out_ff))
        #print(np.min(out_ff))
        thr_val = 0
        out_ff[out_ff <= thr_val] = 0
        out_ff[out_ff > thr_val] = 1
        plt.imshow(out_ff, cmap='gray')
        plt.axis('off')
        plt.show()

#Functions for Question 2
def dif_of_gauss(x, y, std_c, std_s):
    central_gaussian = (0.5/(np.pi*std_c**2))*np.exp(-(x**2+y**2)/(2*
                                        std_c**2))
    surround_gaussian = (0.5/(np.pi*std_s**2))*np.exp(-(x**2+y**2)/(2*
                                        std_s**2))
    return central_gaussian - surround_gaussian


def dog_receptive_field(std_c, std_s, width=21):
    receptive_field = np.zeros((width, width))
    min_edge = int(-(width - 1) / 2)
    max_edge = int((width - 1) / 2)
    x_ind = 0
    y_ind = 0
    for i in range(min_edge, max_edge, 1):
        for j in range(min_edge, max_edge, 1):
            receptive_field[x_ind, y_ind] = dif_of_gauss(i, j, std_c,
                                                std_s)
            x_ind += 1
        y_ind += 1
        x_ind = 0
    return receptive_field

def gabor(x, theta, sigma_l, sigma_w, lamda, phi):
    k_theta = np.asarray([np.sin(theta), np.cos(theta)])
    k_orth_theta = np.asarray([np.cos(theta), -np.sin(theta)])
    gaussian_part_l = -(np.dot(k_theta, x))**2/(2*sigma_l**2)
    gaussian_part_w = -(np.dot(k_orth_theta, x))**2/(2*sigma_w**2)
```

```
    orientation_part = np.cos((2*np.pi*np.dot(k_orth_theta,x)/lamda)+phi
                                                                    )
    return np.exp(gaussian_part_l+gaussian_part_w)*orientation_part


def gabor_receptive_field(sigma_l, sigma_w, theta, lamda, phi, width=21)
                                                                    :
    receptive_field = np.zeros((width, width))
    min_edge = int(-(width - 1) / 2)
    max_edge = int((width - 1) / 2)
    x_ind = 0
    y_ind = 0
    for i in range(min_edge, max_edge, 1):
        for j in range(min_edge, max_edge, 1):
            receptive_field[x_ind, y_ind] = gabor(np.asarray([i, j]),
                                                  theta, sigma_l, sigma_w,
                                                   lamda, phi)
            x_ind += 1
        y_ind += 1
        x_ind = 0
    return receptive_field

ayhan_okuyan_21601531_hw2(question)
```