Bolin He, A53316428, Nov 19 2019

Table of Contents

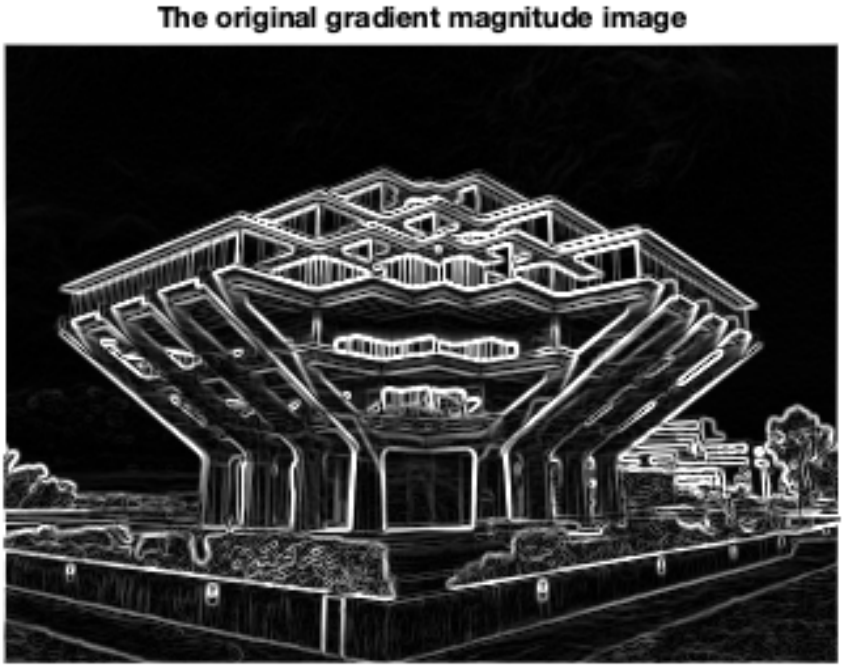
Q1

```
close all;
clear all;
clc;

img = imread('geisel.jpg');
img = rgb2gray(img);
CED(img,70)
fprintf('I choose the thresholding value as 70.')

% function [ ] = CED(img,te)
% % Smoothing
% I = double(img);
% k = 1/159*[2 4 5 4 2;4 9 12 9 4;5 12 15 12 5;4 9 12 9 4; 2 4 5 4 2];
% I_filter = imfilter(I,k);
%
% % Finding Gradients
% kx = [-1 0 1;-2 0 2;-1 0 1];
% ky = [-1 -2 -1;0 0 0;1 2 1];
%
% Gx = imfilter(I_filter,kx);
% Gy = imfilter(I_filter,ky);
% G = sqrt(Gx.^2+Gy.^2);
% G_theta = atan(Gy./Gx);
%
% % Non-maximum Suppression (NMS)
% [row,col] = size(I_filter);
%
% for i = 1:row
%     for j =1:col
%         if G_theta(i,j) <= pi/8 && G_theta(i,j) >=-pi/8
%             G_theta(i,j) = 0;
%         elseif G_theta(i,j) <= 3/8*pi && G_theta(i,j) > pi/8
%             G_theta(i,j) = pi/4;
%         elseif G_theta(i,j) < -pi/8 && G_theta(i,j) >= -3/8*pi
%             G_theta(i,j) = -pi/4;
%         elseif G_theta(i,j) < -3/8*pi && G_theta(i,j) >= -1/2*pi
%             G_theta(i,j) = pi/2;
%         elseif G_theta(i,j) <= 1/2*pi && G_theta(i,j) > -3/8*pi
%             G_theta(i,j) = pi/2;
```

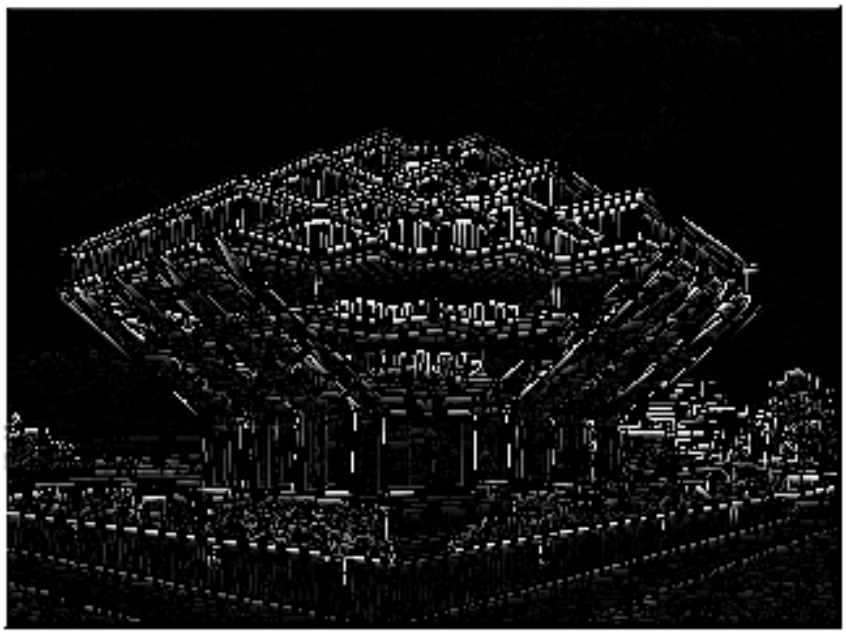
```
%         end
%     end
% end
%
% I_pad = padarray(G,[1,1],0,'both');
% G_theta = padarray(G_theta,[1,1],0,'both');
% [row_pad,col_pad] = size(I_pad);
% I_nms = zeros(row_pad,col_pad);
%
% for i = 2:row_pad-1
%     for j = 2:col_pad-1
%         if G_theta(i,j) == 0
%             if I_pad(i,j+1) > I_pad(i,j) || I_pad(i,j-1) >
 I_pad(i,j)
%                 I_nms(i,j) = 0;
%             else
%                 I_nms(i,j) = I_pad(i,j);
%             end
%             elseif G_theta(i,j) == 1/4*pi
%                 if I_pad(i+1,j-1) > I_pad(i,j) || I_pad(i-1,j+1) >
 I_pad(i,j)
%                     I_pad(i,j) = 0;
%                 else
%                     I_nms(i,j) = I_pad(i,j);
%                 end
%             elseif G_theta(i,j) == -1/4*pi
%                 if I_pad(i-1,j-1) > I_pad(i,j) || I_pad(i+1,j+1) >
 I_pad(i,j)
%                     I_pad(i,j) = 0;
%                 else
%                     I_nms(i,j) = I_pad(i,j);
%                 end
%             elseif G_theta(i,j) == pi/2
%                 if I_pad(i-1,j) > I_pad(i,j) || I_pad(i+1,j) >
 I_pad(i,j)
%                     I_pad(i,j) = 0;
%                 else
%                     I_nms(i,j) = I_pad(i,j);
%                 end
%         end
%     end
% end
%
%
% % Thresholding
% I_edge = zeros(row_pad,col_pad);
% for i = 1:row_pad
%     for j =1:col_pad
%         if I_nms(i,j) < te
%            I_edge(i,j) = 0;
%         else
%            I_edge(i,j) = I_nms(i,j);
%         end
%     end
```

```
% end
%
%
% G = imshow(uint8(G));
% title('The original gradient magnitude image')
% figure
% I_nms = imshow(uint8(I_nms));
% title('The image after NMS')
% figure
% I_edge = imshow(uint8(I_edge));
% title('The image after thresholding')
% end
```

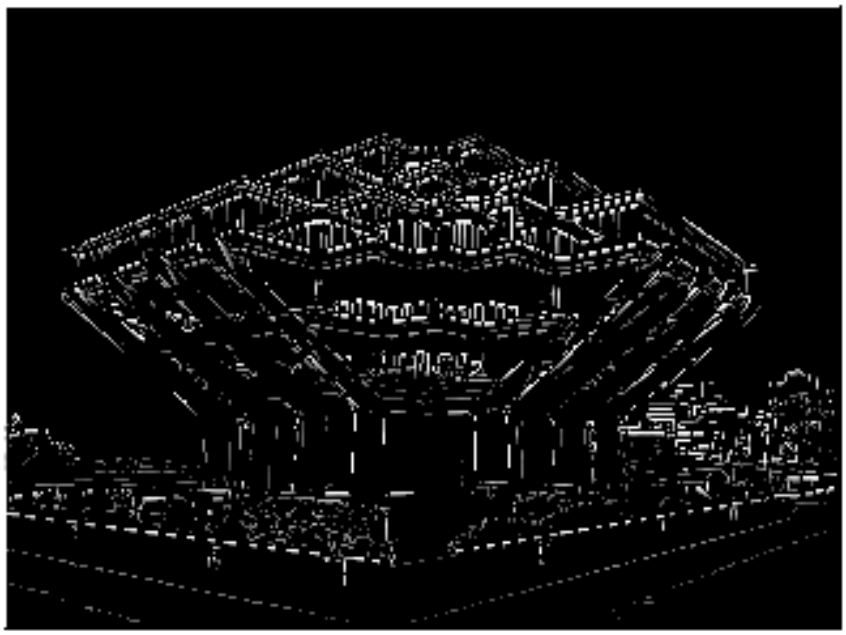
I choose the thresholding value as 70.



The image after NMS



The image after thresholding

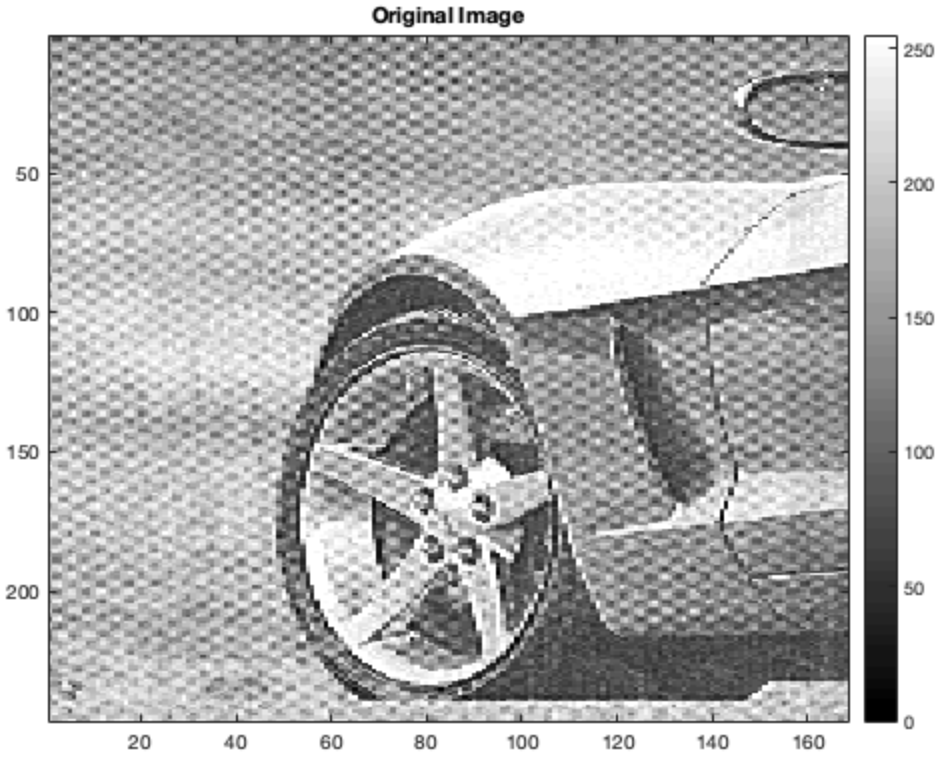


Q2

```
close all;
clear all;
clc;

%%%%% Image Car %%%%%
car = imread('Car.tif');
imagesc(car);
colormap(gray);
colorbar;
title('Original Image');

car = double(car);
car_pad = padarray(car,[133,172],0,'both');
imFFT = fft2(car_pad);
imFFT = fftshift(imFFT);
figure
imagesc(-256:255,-256:255,log(abs(imFFT)));
colorbar;
title('Image after FFT')
xlabel('u');
ylabel('v');

% Notch Filter
D0 = 30;
n = 4;
Position =[-90,-167;-90,-82;-82,87;-82,173];
uk = Position(:,1);
vk = Position(:,2);
H_NR = 1;
[u,v] = meshgrid(-256:255);
for k = 1:4
    Dk1=sqrt(((u-uk(k)).^2)+((v-vk(k)).^2));
    Dk2=sqrt(((u+uk(k)).^2)+((v+vk(k)).^2));
    H_NR=H_NR.*(1./(1+(D0./Dk1).^(2*n))).*(1./(1+(D0./Dk2).^(2*n)));
end
figure
imagesc(-256:255,-256:255,abs(H_NR));
colorbar;
colormap(gray);
title('Butterworth Notch Reject Filter');
xlabel('u');
ylabel('v');

% Apply Filter
figure
imagesc(-256:255,-256:255,log(abs(imFFT.*H_NR)));
title('Image after FFT with Notch Filter');
colorbar;
xlabel('u');
ylabel('v');
```

```
imfilter = uint8(ifft2(ifftshift(imFFT.*H_NR)));
figure
imagesc(imfilter(134:379,173:340))
colormap(gray);
title('Filtered Image')
colorbar;

%%%%% Image Street %%%%%
street = imread('Street.png');
figure;
imagesc(street);
colorbar;
colormap(gray);
title('Original Image');

street = double(street);
street_pad = padarray(street,[180,153],0,'post');
imFFT2 = fftshift(fft2(street_pad));
figure;
imagesc(-256:255,-256:255,log(abs(imFFT2)));
colorbar;
xlabel('u');
ylabel('v');

% Notch Filter
D0 = 20;
n = 3;
Position2 =[0,164;-168,0];
uk = Position2(:,1);
vk = Position2(:,2);
H_NR2 = 1;
[u,v] = meshgrid(-256:255);
for k = 1:2
    Dk1=sqrt(((u-uk(k)).^2)+((v-vk(k)).^2));
    Dk2=sqrt(((u+uk(k)).^2)+((v+vk(k)).^2));
    H_NR2=H_NR2.*(1./(1+(D0./Dk1).^(2*n))).*(1./(1+(D0./Dk2).^(2*n)));
end
figure
imagesc(-256:255,-256:255,abs(H_NR2));
colorbar;
colormap(gray);
title('Butterworth Notch Reject Filter');
xlabel('u');
ylabel('v');

% Apply Filter
figure
imagesc(-256:255,-256:255,log(abs(imFFT2.*H_NR2)));
title('Image after FFT with Notch Filter');
colorbar;
xlabel('u');
ylabel('v');

imfilter2 = uint8(ifft2(ifftshift(imFFT2.*H_NR2)));
```

```
figure
imagesc(imfilter2(1:332,1:359))
colormap(gray);
title('Filtered Image')
colorbar;

%
Para1 = {'n';'D0';'u1';'v1';'u2';'v2';'u3';'v3';'u4';'v4'};
Para2 = {'n';'D0';'u1';'v1';'u2';'v2'};
num1 = [4;30;-90;-167;-90;-82;-82;87;-82;173];
num2 = [3;20;0;164;-168;0];
Parameter_in_2i = table(categorical(Para1),num1)
Parameter_in_2ii = table(categorical(Para2),num2)


Parameter_in_2i =

  10×2 table

    Var1    num1
    ____    ____

     n         4
     D0       30
     u1      -90
     v1     -167
     u2      -90
     v2      -82
     u3      -82
     v3       87
     u4      -82
     v4      173


Parameter_in_2ii =

  6×2 table

    Var1    num2
    ____    ____

     n         3
     D0       20
     u1        0
     v1      164
     u2     -168
     v2        0
```


Original Image
50
100
150
200
20
40
60
80
100
120
140
160
250
200
150
100
50
0

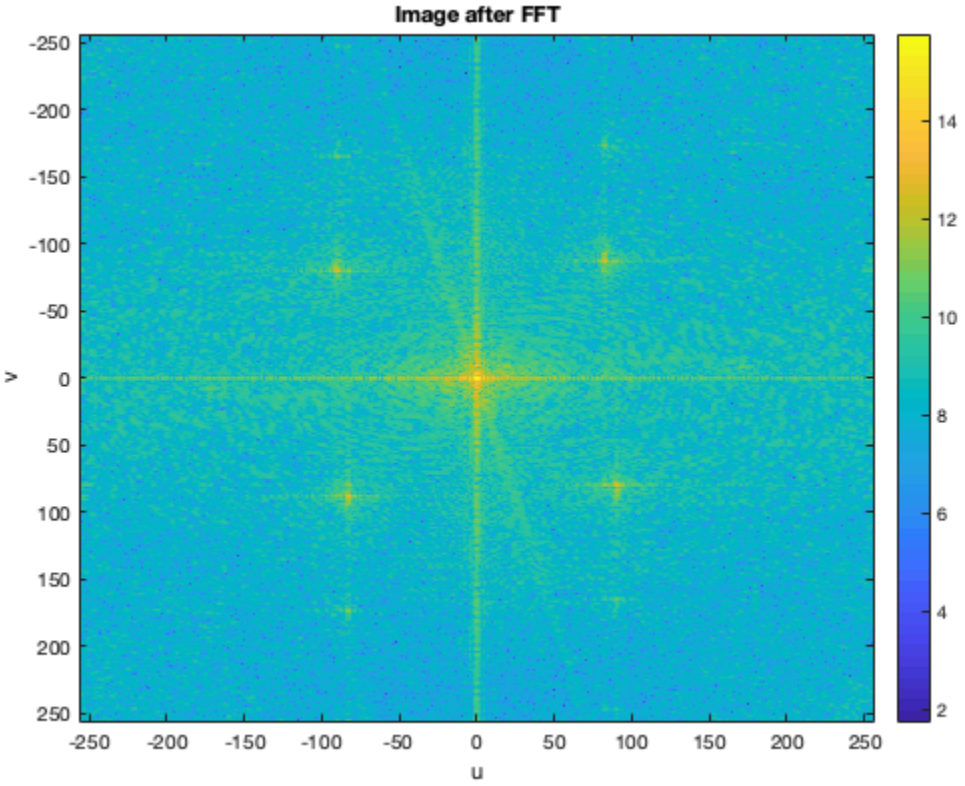
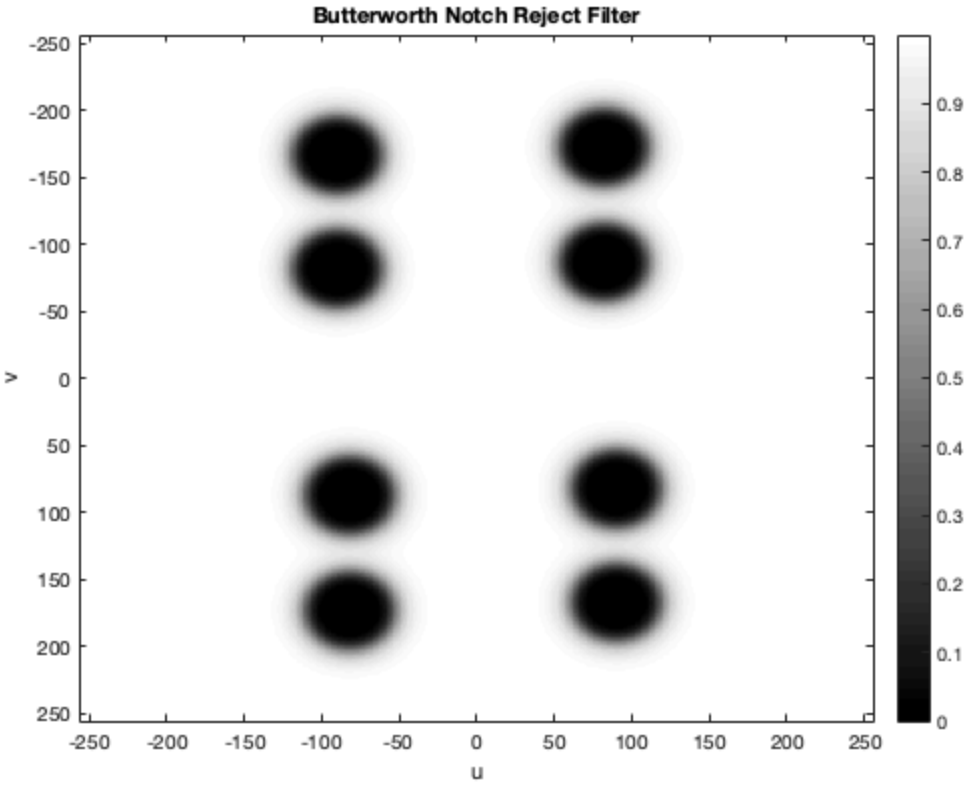


Image after FFT
-250
-200
-150
-100
-50
0
50
100
150
200
250
v
u
14
12
10
8
6
4
2


Butterworth Notch Reject Filter
v
u
-250
-200
-150
-100
-50
0
50
100
150
200
250
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

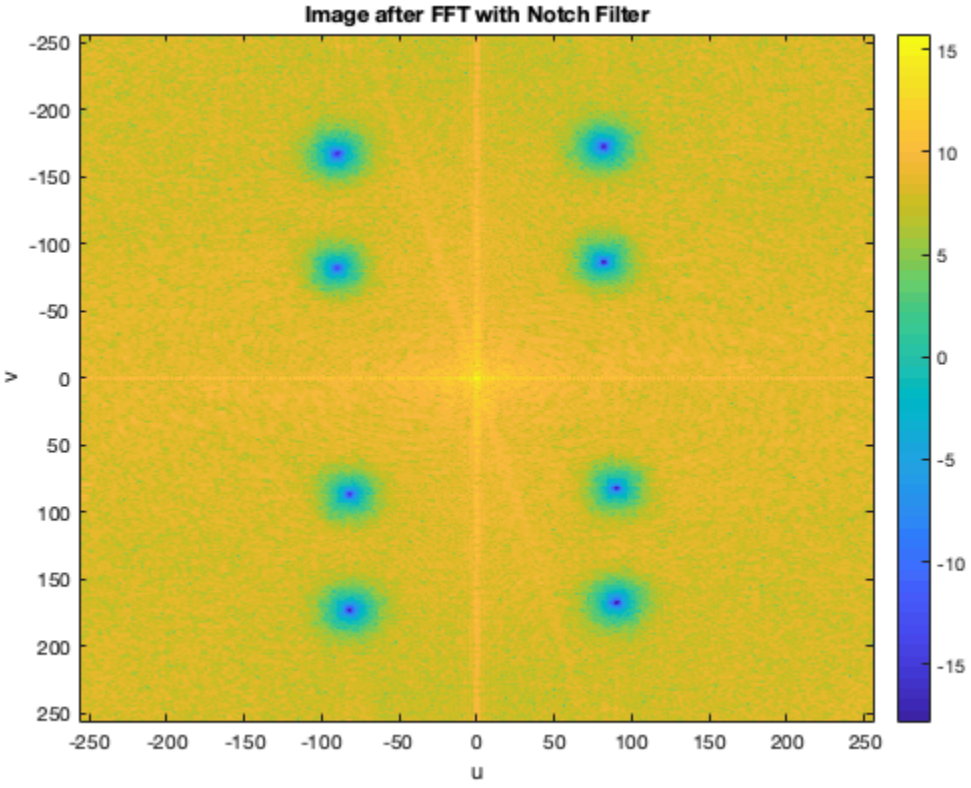
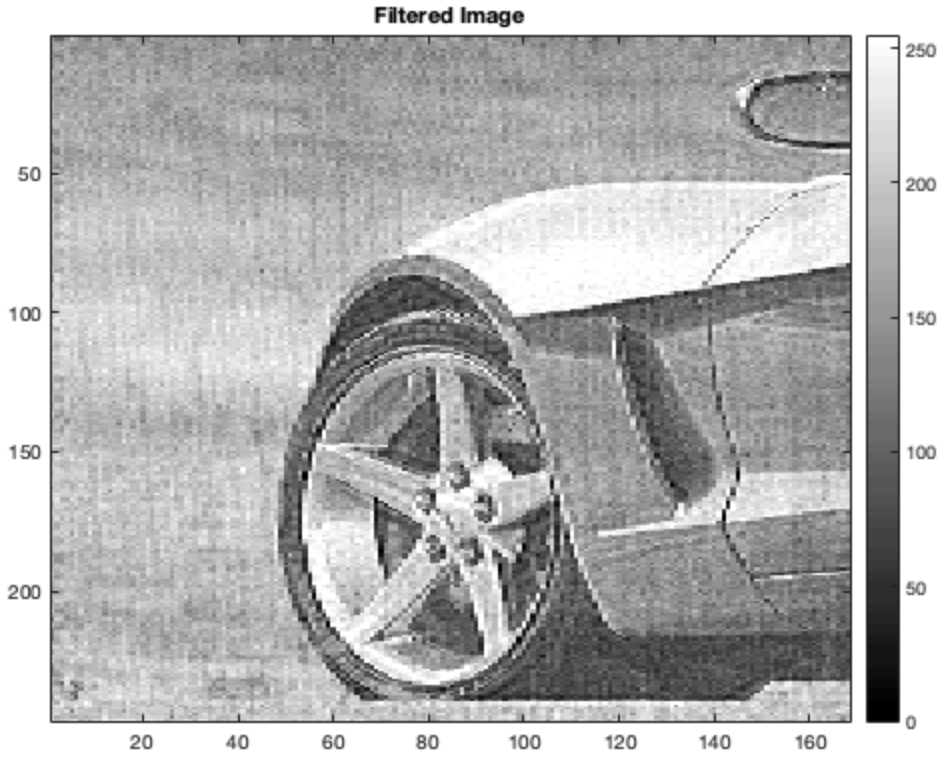
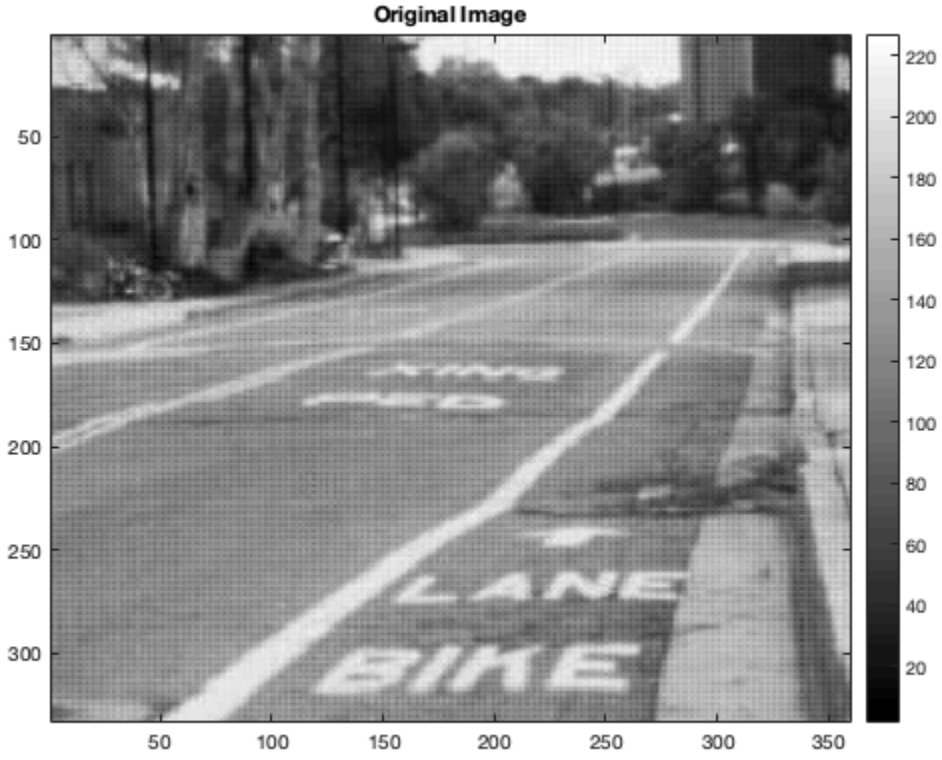
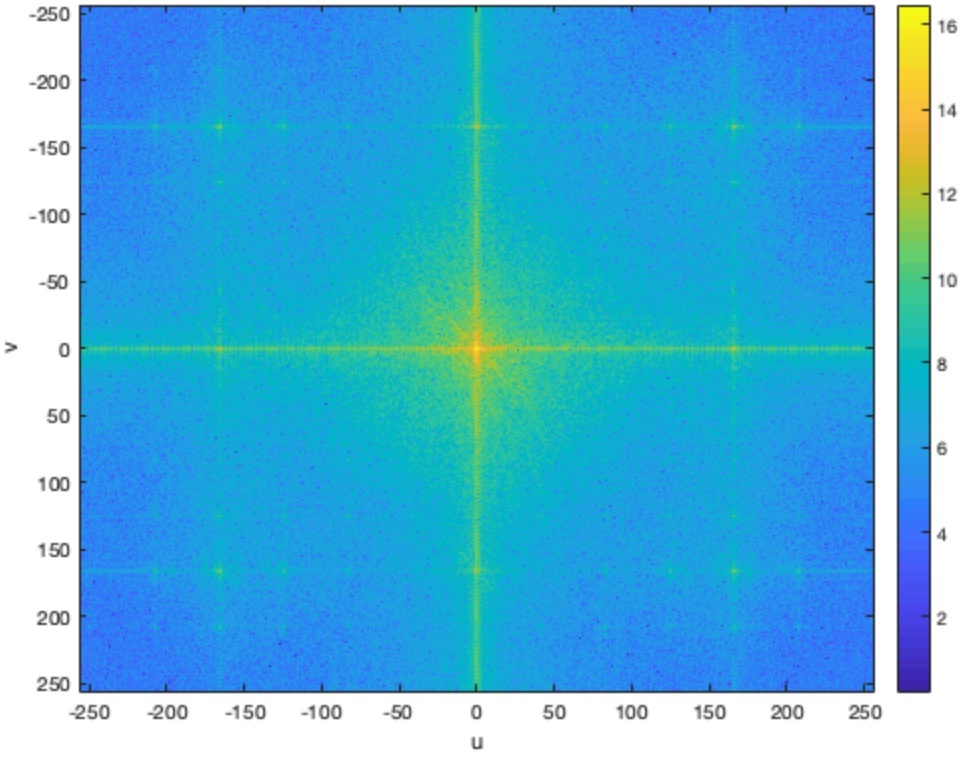
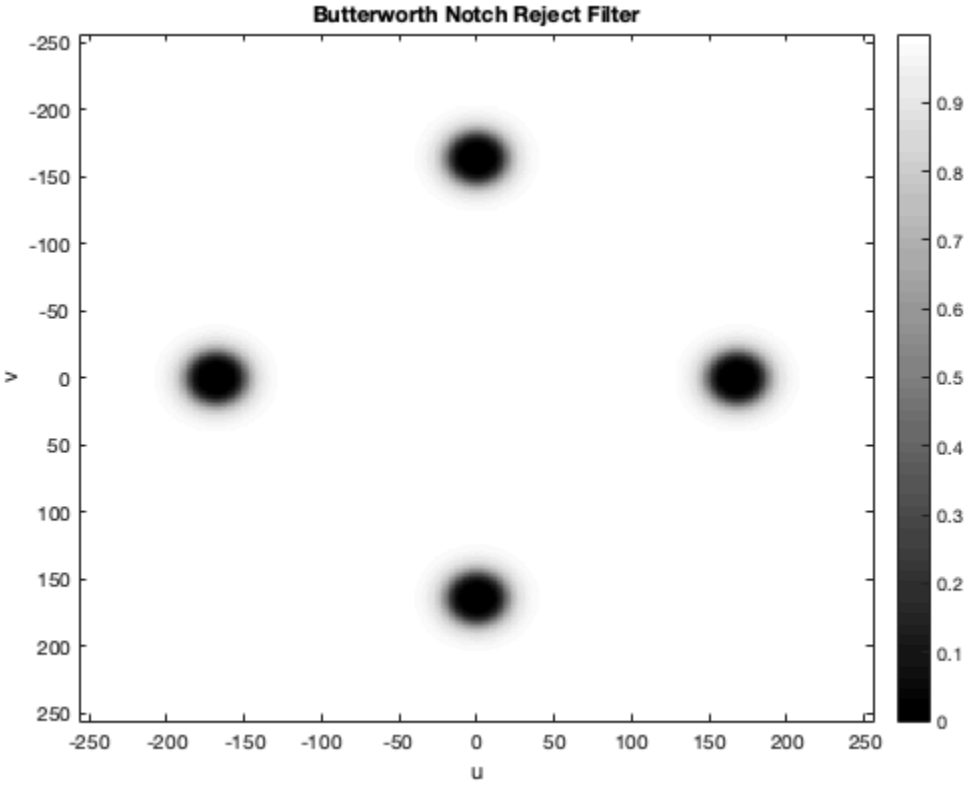


Image after FFT with Notch Filter
v
u
-250
-200
-150
-100
-50
0
50
100
150
200
250
15
10
5
0
-5
-10
-15


Filtered Image
50
100
150
200
20
40
60
80
100
120
140
160
250
200
150
100
50
0


Original Image
50
100
150
200
250
300
50
100
150
200
250
300
350
LANE
BIKE
220
200
180
160
140
120
100
80
60
40
20


-250
-200
-150
-100
-50
0
50
100
150
200
250
v
u
16
14
12
10
8
6
4
2


Butterworth Notch Reject Filter
-250
-200
-150
-100
-50
0
50
100
150
200
250
v
u
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

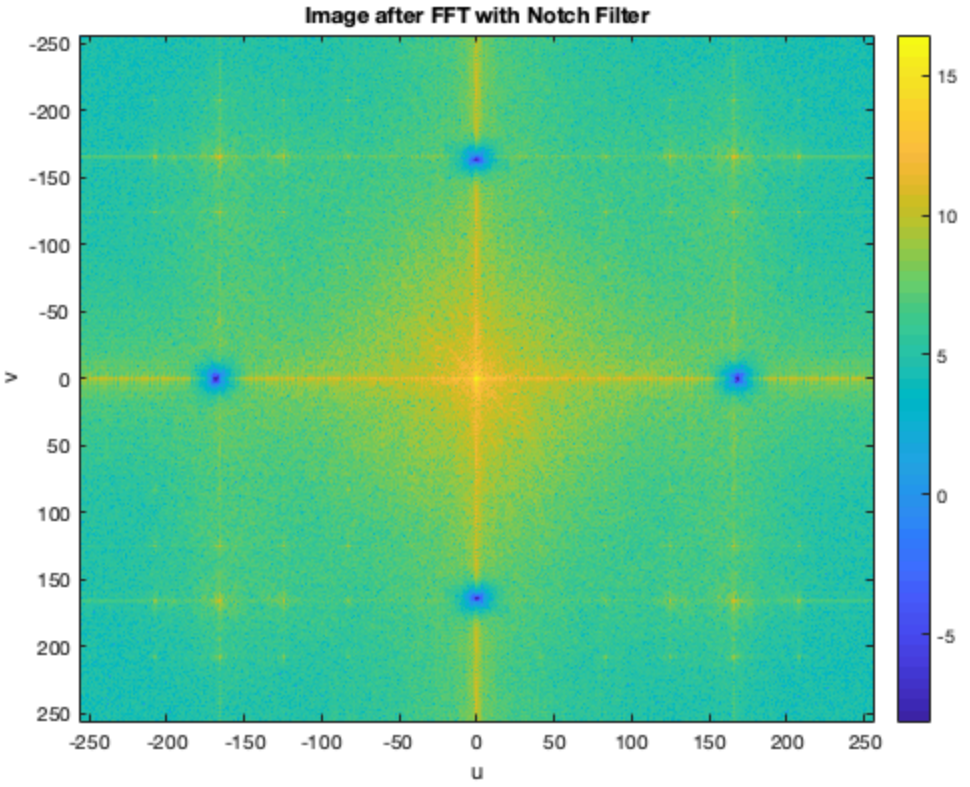
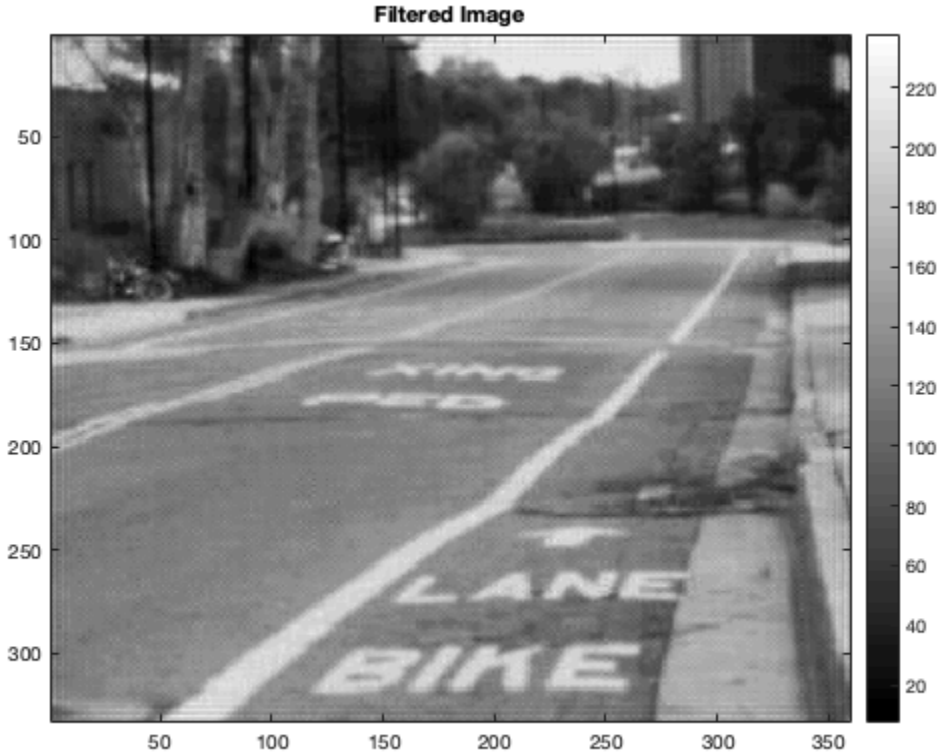


Image after FFT with Notch Filter
v
u
-250
-200
-150
-100
-50
0
50
100
150
200
250
15
10
5
0
-5


Filtered Image
50
100
150
200
250
300
350
LANE
BIKE
220
200
180
160
140
120
100
80
60
40
20

Q3 PyTorch tutorial and questions

(ii) 50000 images are packed into 12500 batches for training.

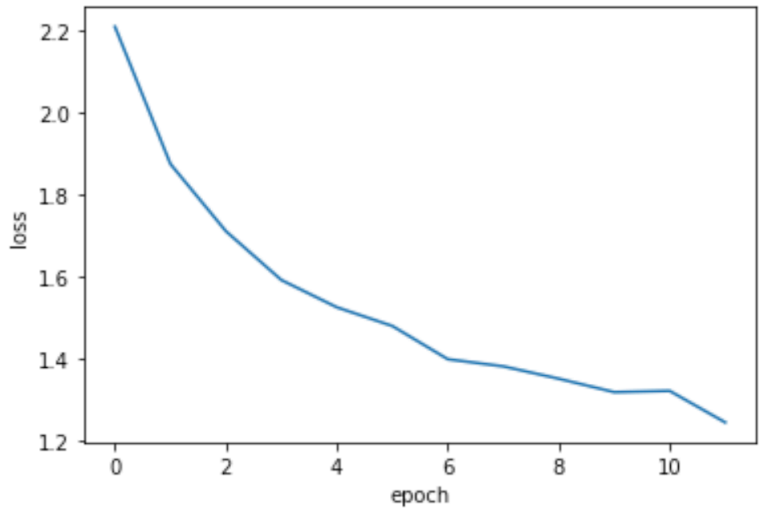
(iii) Yes, we normalize the images. The output of torchvision datasets are PILImage images of range [0, 1]. In our example, we transform them to Tensors of normalized range [-1, 1].

(iv)

```
[1,  2000] loss: 2.209
[1,  4000] loss: 1.874
[1,  6000] loss: 1.711
[1,  8000] loss: 1.592
[1, 10000] loss: 1.525
[1, 12000] loss: 1.480
[2,  2000] loss: 1.399
[2,  4000] loss: 1.381
[2,  6000] loss: 1.351
[2,  8000] loss: 1.319
[2, 10000] loss: 1.322
[2, 12000] loss: 1.245
Finished Training
```

```
import matplotlib.pyplot as plt
plt.plot(lossf)
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
Text(0, 0.5, 'loss')
```



(v)

```
class_names = trainset.classes
```

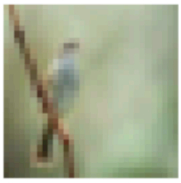
```
def visualize_model(model, num_images=4):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(trainloader,0):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: {}'.format([class_names[j]]))
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

```
visualize_model(net)
```

predicted: ['airplane']

predicted: ['automobile']

predicted: ['bird']



predicted: ['cat']

(vi)

```
dataiter = iter(trainloader)
input_image, label = dataiter.next()
data = net.conv1(input_image).cpu().data.numpy()
data = data[0];
```

```
kernel_num = data.shape[0]

fig, axes = plt.subplots(ncols=kernel_num, figsize=(2*kernel_num, 2))

for col in range(kernel_num):
    axes[col].imshow(data[col, :, :])
plt.show()
```