hw3.2

November 19, 2019

[1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import math
```

0.0.1 2(i)

[2]:
```
img = cv2.imread("Car.tif", 0)
# img = int(img)
img_pad = np.zeros((512, 512))
p = int((512 - img.shape[0]) / 2)
q = int((512 - img.shape[1]) / 2)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_pad[i + p][j + q] = img[i][j]
```

[3]:
```
f = np.fft.fft2(img_pad)
fshift = np.fft.fftshift(f)
log_magnitude_spectrum = np.log(np.abs(fshift))
```

[4]:
```
uK = [91, 176, 344, 429]
vK = [168, 166, 166, 169]
uK  = uK - np.ones((4, )) * 256
vK = vK - np.ones((4, )) * 256
print(uK, vK)
x_axis = np.linspace(-256,255,512)
y_axis = np.linspace(-256,255,512)
[v,u] = np.meshgrid(x_axis,y_axis)
```

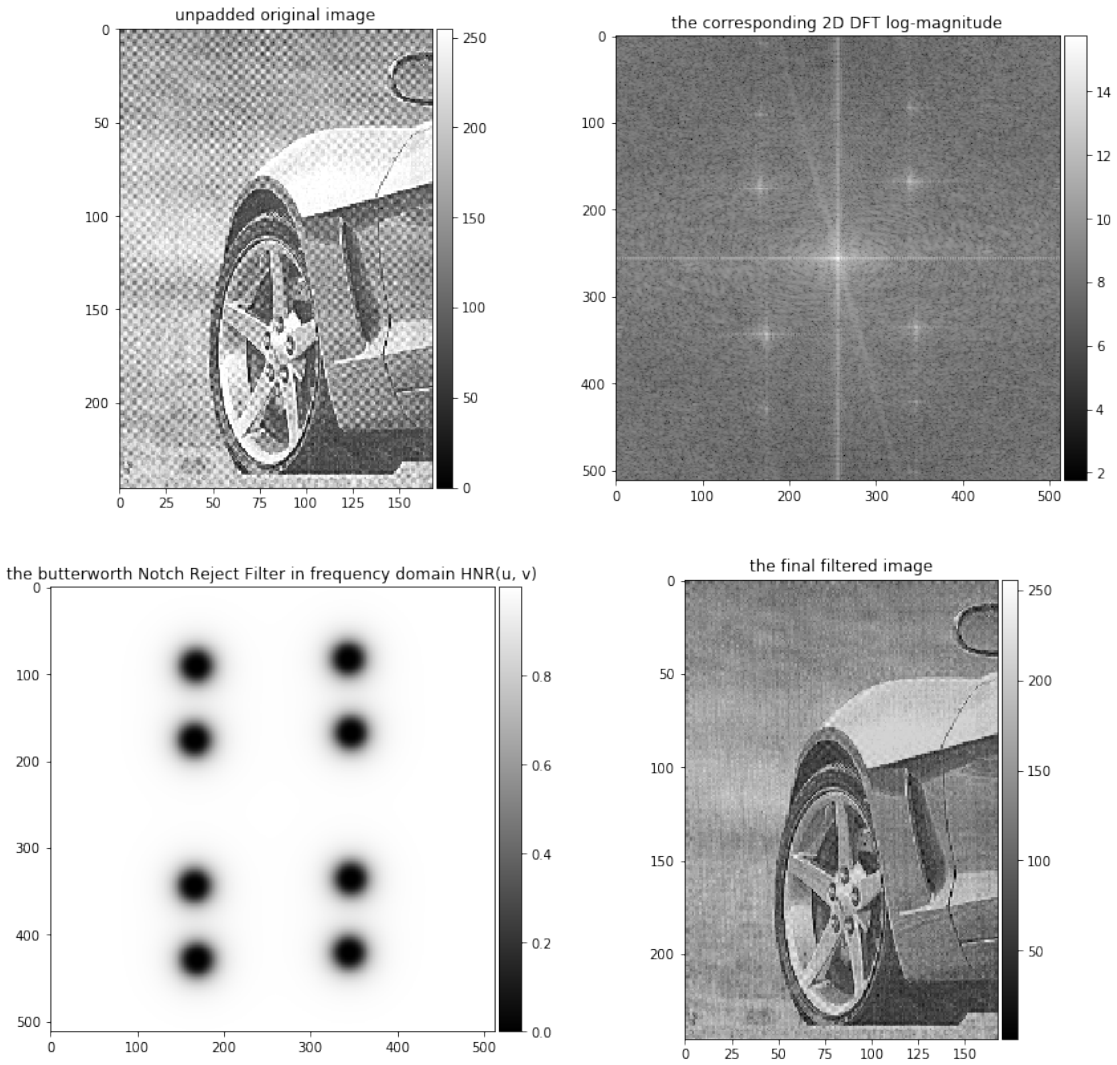
```
[-165.  -80.   88.  173.] [-88. -90. -90. -87.]
```

[5]:
```
img_h = np.zeros(img_pad.shape)
n = 2
d0 = 20
for i in range(img_pad.shape[0]):
```

```
    for j in range(img_pad.shape[1]):
        h = 1
        uc = u[i][j]
        vc = v[i][j]
        for k in range(4):
            dk = np.sqrt(np.square(uc - uK[k]) + np.square(vc - vK[k]))
            d_k = np.sqrt(np.square(uc + uK[k]) + np.square(vc + vK[k]))
            if (dk == 0 or d_k == 0):
                h *= 0
            else:
                h *= 1/(1 + math.pow(d0 / dk, 2 * n)) * 1/(1 + math.pow(d0 /␣
↪d_k, 2 * n))
        img_h[i][j] = h
```

[6]:
```
img_no_shift = np.fft.ifftshift(fshift * img_h)
img_back = np.abs(np.fft.ifft2(img_no_shift))
img_back -= img_back.min()
img_back = img_back[p : img_back.shape[0] - p, q : img_back.shape[1] - q] * 256␣
↪/ img_back.max()

f = plt.figure(figsize=(14,14))
f_ax1 = f.add_subplot(221)
f_ax2 = f.add_subplot(222)
f_ax3 = f.add_subplot(223)
f_ax4 = f.add_subplot(224)

img1_1 = f_ax1.imshow(img, cmap = 'gray')
f_ax1.title.set_text("unpadded original image")
divider = make_axes_locatable(f_ax1)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_1, cax, orientation='vertical')

img1_2 = f_ax2.imshow(log_magnitude_spectrum, cmap = 'gray')
f_ax2.title.set_text("the corresponding 2D DFT log-magnitude")
divider = make_axes_locatable(f_ax2)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_2, cax, orientation='vertical')

img2_1 = f_ax3.imshow(img_h, cmap = 'gray')
f_ax3.title.set_text("the butterworth Notch Reject Filter in frequency domain␣
↪HNR(u, v)")
divider = make_axes_locatable(f_ax3)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_1, cax, orientation='vertical')

img2_2 = f_ax4.imshow(img_back, cmap = 'gray')
f_ax4.title.set_text("the final filtered image")
```

```
divider = make_axes_locatable(f_ax4)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_2, cax, orientation='vertical')
plt.savefig("result_car.jpg")
plt.show()
```



(i) The 10 parameters I choose here are shown as below:

$$n = 2$$

$$D_0 = 20$$

$$u_1 = 91 - 256 = -165, v_1 = 168 - 256 = -88$$

$$u_2 = 176 - 256 = -80, v_2 = 166 - 256 = -90$$

$$u_3 = 344 - 256 = 88, v_3 = 166 - 256 = -90$$

$$u_4 = 429 - 256 = 173, v_4 = 169 - 256 = -87$$

0.0.2 2(ii)

[7]:
```
img = cv2.imread("Street.png", 0)
# img = int(img)
img_pad = np.zeros((512, 512))
p = int((512 - img.shape[0]) / 2)
q = int((512 - img.shape[1]) / 2)
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        img_pad[i + p][j + q] = img[i][j]
```

[8]:
```
f = np.fft.fft2(img_pad)
fshift = np.fft.fftshift(f)
log_magnitude_spectrum = np.log(np.abs(fshift))
```

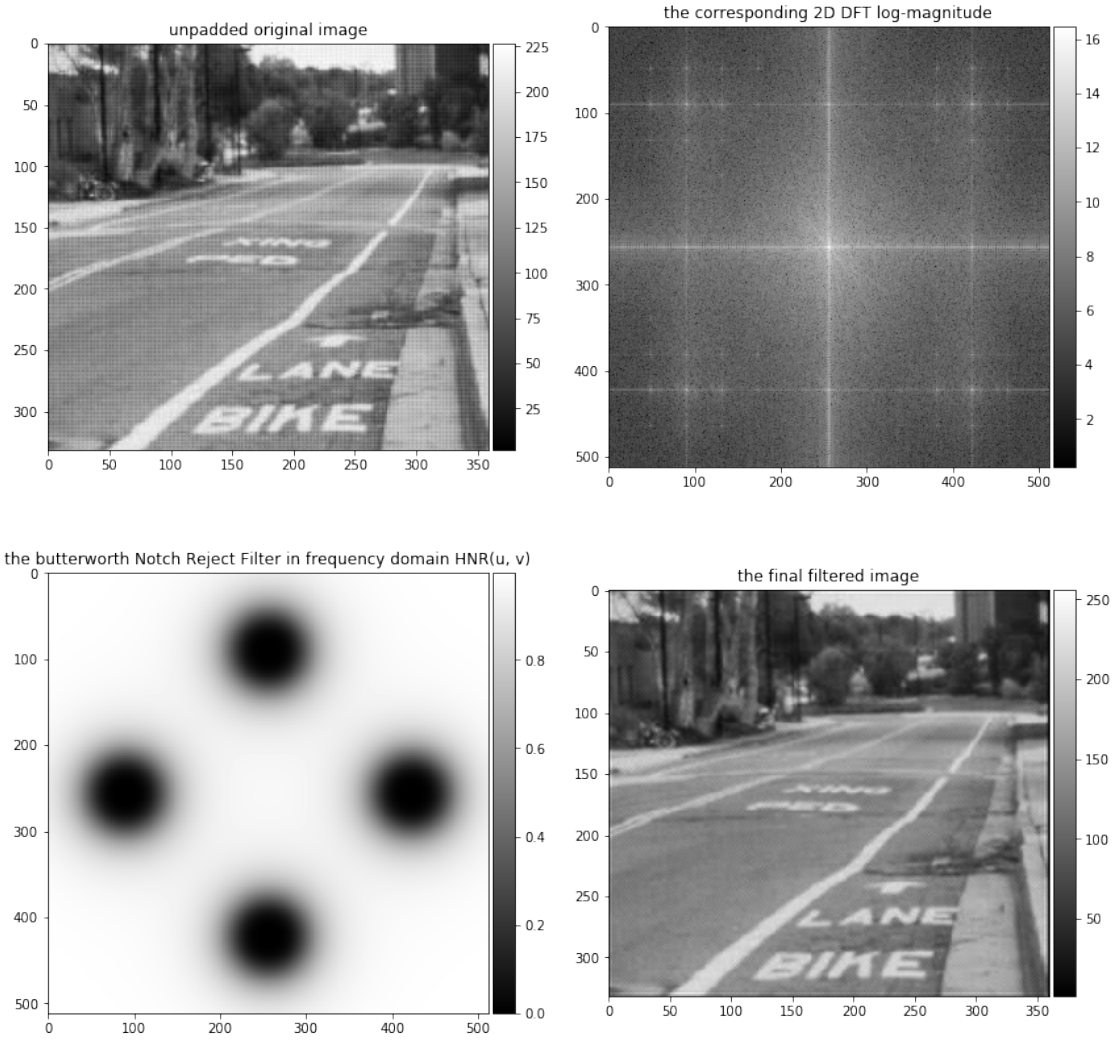
[9]:
```
uK = [256, 90]
vK = [90, 256]
uK  = uK - np.ones((2, )) * 256
vK = vK - np.ones((2, )) * 256
print(uK, vK)
x_axis = np.linspace(-256,255,512)
y_axis = np.linspace(-256,255,512)
[v,u] = np.meshgrid(x_axis,y_axis)
```

```
[   0. -166.] [-166.    0.]
```

[10]:
```
img_h = np.zeros(img_pad.shape)
n = 2
d0 = 50
for i in range(img_pad.shape[0]):
    for j in range(img_pad.shape[1]):
        h = 1
        uc = u[i][j]
        vc = v[i][j]
        for k in range(2):
            dk = np.sqrt(np.square(uc - uK[k]) + np.square(vc - vK[k]))
            d_k = np.sqrt(np.square(uc + uK[k]) + np.square(vc + vK[k]))
            if (dk == 0 or d_k == 0):
                h *= 0
            else:
                h *= 1/(1 + math.pow(d0 / dk, 2 * n)) * 1/(1 + math.pow(d0 /␣
↪d_k, 2 * n))
        img_h[i][j] = h
```

[11]:
```
img_no_shift = np.fft.ifftshift(fshift * img_h)
img_back = np.abs(np.fft.ifft2(img_no_shift))
img_back -= img_back.min()
```

```
img_back = img_back[p : img_back.shape[0] - p, q : img_back.shape[1] - q] * 256␣
↪/ img_back.max()

f = plt.figure(figsize=(14,14))
f_ax1 = f.add_subplot(221)
f_ax2 = f.add_subplot(222)
f_ax3 = f.add_subplot(223)
f_ax4 = f.add_subplot(224)

img1_1 = f_ax1.imshow(img, cmap = 'gray')
f_ax1.title.set_text("unpadded original image")
divider = make_axes_locatable(f_ax1)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_1, cax, orientation='vertical')

img1_2 = f_ax2.imshow(log_magnitude_spectrum, cmap = 'gray')
f_ax2.title.set_text("the corresponding 2D DFT log-magnitude")
divider = make_axes_locatable(f_ax2)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img1_2, cax, orientation='vertical')

img2_1 = f_ax3.imshow(img_h, cmap = 'gray')
f_ax3.title.set_text("the butterworth Notch Reject Filter in frequency domain␣
↪HNR(u, v)")
divider = make_axes_locatable(f_ax3)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_1, cax, orientation='vertical')

img2_2 = f_ax4.imshow(img_back, cmap = 'gray')
f_ax4.title.set_text("the final filtered image")
divider = make_axes_locatable(f_ax4)
cax = divider.append_axes('right', size='5%', pad=0.05)
plt.colorbar(img2_2, cax, orientation='vertical')
plt.savefig("result_street.jpg")
plt.show()
```



(ii) The 6 parameters I choose here are shown as below:

$$n = 2$$

$$D_0 = 50$$

$$u_1 = 256 - 256 = 0, v_1 = 90 - 256 = -166$$

$$u_2 = 90 - 256 = -166, v_2 = 256 - 256 = 0$$