CSE 473/573 Fall 2016 Homework Set #2

Prashanth Chandrashekar

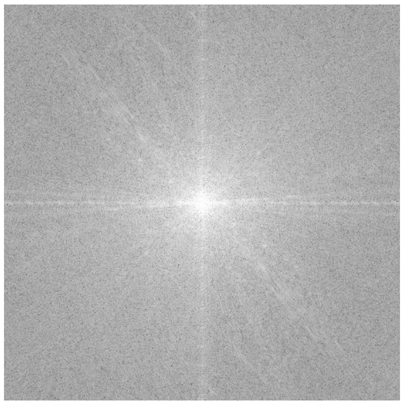
PC74

50207911

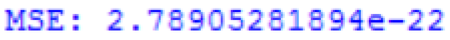
Problem 1

a) I_0:

FFT(I_0):



b) I$_R$:

c) MSE for the 512x512 image: 

```
MSE: 2.78905281894e-22
```

d) The Fourier transform is a discrete function where we are discretely choosing the points and calculating. Due to this we might miss out on some data as we are not continuously sampling and hence the MSE can be non-zero. But in our example, while calculating the DFT, we are taking into account, all the pixels of the original image and hence we may expect the MSE to be zero. Actually, the data loss occurs during the computation; there are so many floating-point numbers which are sort of rounded off beyond certain digits on the left side of the decimal point due to the limitation of the data type. We can try and reduce this MSE by going for a more accurate datatype.

CODE:

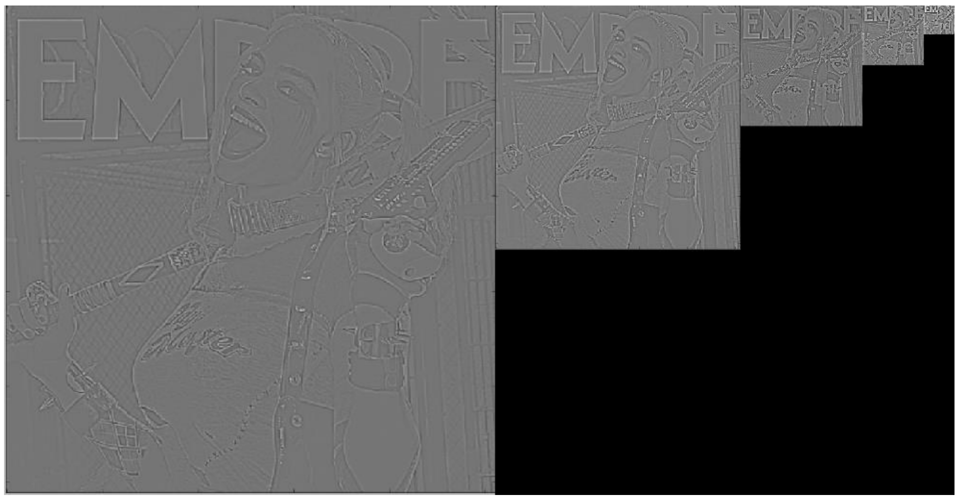
```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def dft_1d(image,k,dft_type):
    P=0.0j

    if dft_type ==0:
        sign = -1
        scale = 1
    else:
        sign = 1
        scale = len(image)

    for a in range(0,len(image)):
        P = P+(image[a]*(np.exp((sign)*(1j)*2*(np.pi)*(float(k*a)/len(image)))))

    P= P/scale
    return P


def dft_2d(image,dft_type):
    #print(image)
    M=len(image)#rows
    N=len(image[0])#collumns
    G=[]
    P=[]

    P=np.zeros(shape=(M,N))*1j
    G=np.zeros(shape=(M,N))*1j


    for k in range(0,M):
        for l in range(0,N):
            P[k][l]=dft_1d(np.transpose(image)[l],k,dft_type)
    print("m,n",len(P),len(P[0]))
    Q=np.transpose(P)

    for k in range(0,N):
        for l in range(0,M):
            G[k][l]=dft_1d(np.transpose(Q)[l],k,dft_type)
    print("m,n",len(G),len(G[0]))
    F=np.transpose(G)

    return F

def getmse(orig,recon):
    mse =0
    M=len(orig)
    N=len(orig[0])
    for k in range(0,M):
        for l in range(0,N):
            mse = (np.abs((orig[k][l]-recon[k][l])))**2
    return mse


print ("start")
img = cv2.imread('C:/Users/presh/Google Drive/MS/CVIP/HW/Hw2/gray_image.jpg')
gimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
M=len(gimg)#rows
N=len(gimg[0])#collumns

dft_img = dft_2d(gimg,0)
idft_img = dft_2d(dft_img,1)
mse=getmse(idft_img,gimg)
print("MSE:",mse)

L=20*np.log(np.absolute(dft_img))

T=np.zeros(shape=(M,N))

T[0:np.floor(M/2),0:np.floor(M/2)]=L[np.floor(M/2):M,np.floor(M/2):M]
T[0:np.floor(M/2),np.floor(M/2):M]=L[np.floor(M/2):M,0:np.floor(M/2)]
T[np.floor(M/2):M,0:np.floor(M/2)]=L[0:np.floor(M/2),np.floor(M/2):M]
T[np.floor(M/2):M,np.floor(M/2):M]=L[0:np.floor(M/2),0:np.floor(M/2)]


cv2.imwrite('dft_out.jpg',np.absolute(dft_img))
cv2.imwrite('scaled.jpg',L)
cv2.imwrite('shifted.jpg',T)
```

```
cv2.imwrite('inv.jpg',np.absolute(idft_img))


plt.subplot(122),plt.imshow(idft_img.astype(np.uint8), cmap = 'gray')
plt.show()

print("done")
```

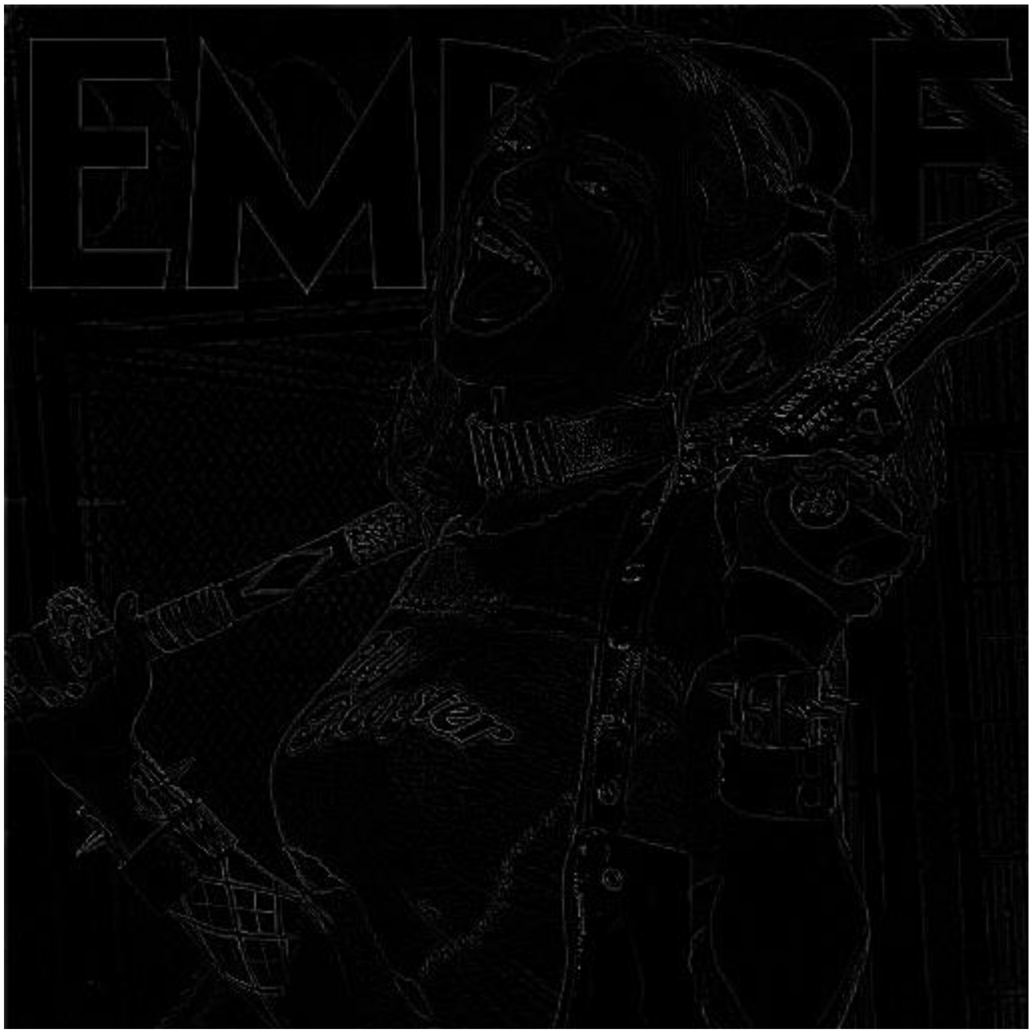
Problem 2

a) I_0:

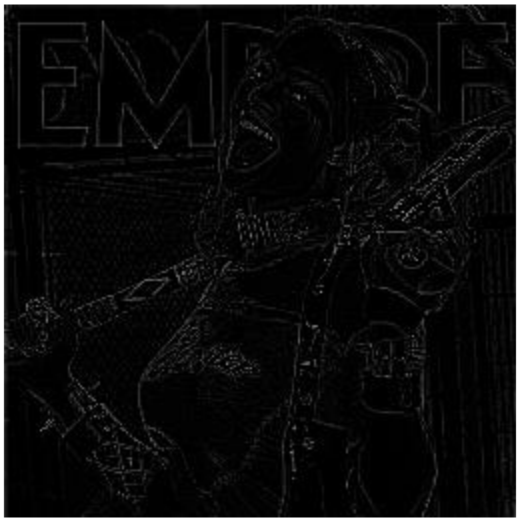
Laplacian pyramid:



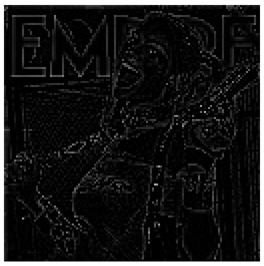
L1(I_0):



L2(I_0):



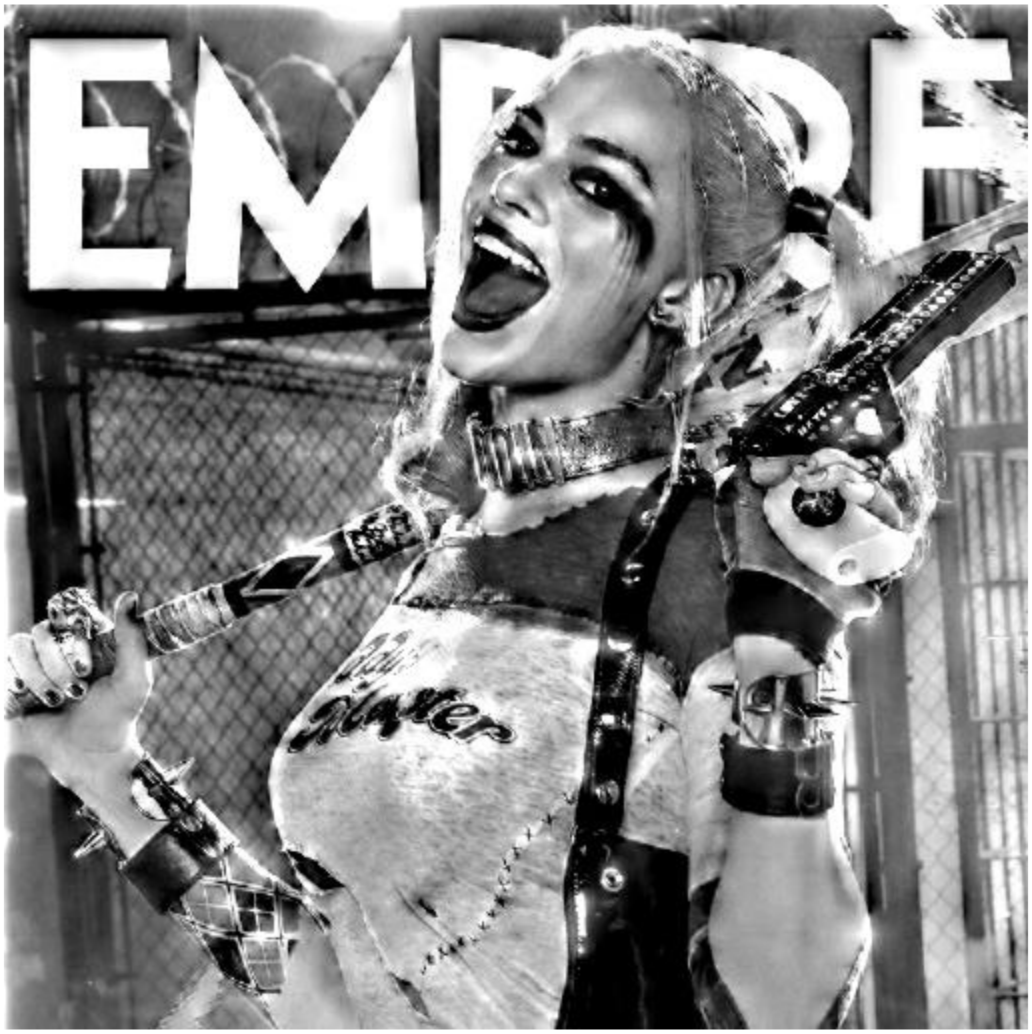
L3(I_0):



L4(I_0):

$L5(I_0)$:

b) I_R:



c) MSE:

```
MSE: 196.0
```

d) In this problem, there is a good chance of getting non-zero MSE, as this process involves upsampling, where we scale up a lower resolution image into a higher resolution one. We may employ various interpolation techniques to predict the values of the new pixels created in the reconstructed image, but this value may not always be correct hence resulting in a difference when compared to the original image.

CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def convolve_1d(image,k,dft_type):
    P=0.0j

    if dft_type ==0:
        sign = -1
        scale = 1
    else:
        sign = 1
        scale = len(image)

    for a in range(0,len(image)):
        P = P+(image[a]*(np.exp((sign)*(1j)*2*(np.pi)*(float(k*a)/len(image)))))

    P= P/scale
    return P


def convolve_2d(image,kernel):
    #print(image)
    M=len(image)#rows
    N=len(image[0])#collumns
    a=len(kernel)
    pad_size=int(np.floor(a/2))




    padded_image=np.lib.pad(image, (pad_size,pad_size), 'constant', constant_values=(0))

    conv_img = np.zeros(shape=(M,N))
    temp_mat = np.zeros(shape=(a,a))


    for k in range(0,M):
        for l in range(0,N):
            temp_mat=padded_image[k:k+a,l:l+a]*kernel
            conv_img[k,l]=sum(sum(temp_mat))

    return conv_img
```

```
def getMSE(orig,recon):
    mse =0
    M=len(orig)
    N=len(orig[0])
    for k in range(0,M):
        for l in range(0,N):
            mse = (np.abs((orig[k][l]-recon[k][l])))**2
    return mse

def upsample(image):
    M=len(image)#rows
    N=len(image[0])#collumns
    upimage = np.zeros(shape=(M*2,N*2))

    for i in range(0,M):
        for j in range(0,N):
            upimage[i*2,j*2] = image[i,j]

    #cv2.imwrite('big_image1.jpg',upimage)

    #averager = ([1,1,1],[1,1,1],[1,1,1])/9
    #pad_size=int(np.floor(len(averager)/2))

    #padded_image=np.lib.pad(upimage, (pad_size,pad_size), 'constant',
constant_values=(0))
    #temp_mat = np.zeros(shape=(len(averager),len(averager))

    #for k in range(0,M):
        #for l in range(0,N):
            #temp_mat=padded_image[(2*k)+1:(2*k)+1+a,(2*l)+1:(2*l)+1+a]*kernel
            #upimage[(2*k)+1,(2*l)+1]=sum(sum(temp_mat))
    for k in range(0,M):
        for l in range(0,2*N):
            if k==M-1:
                upimage[(2*k)+1,l]=upimage[(2*k),l]
            else:
                upimage[(2*k)+1,l]=(upimage[(2*k),l]+upimage[(2*k)+2,l])/2

    #cv2.imwrite('big_image2.jpg',upimage)

    for l in range(0,N):
        for k in range(0,2*M):
            if l==N-1:
                upimage[k,(2*l)+1]=upimage[k,(2*l)]
            else:
                upimage[k,(2*l)+1]=(upimage[k,(2*l)]+upimage[k,(2*l)+2])/2

    #cv2.imwrite('big_image3.jpg',upimage)

    return upimage

print ("start")
img = cv2.imread('C:/Users/presh/Google Drive/MS/CVIP/HW/Hw2/gray_image.jpg')
gimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
M=len(gimg)#rows
```

```
N=len(gimg[0])#collumns

#big_image=upsample(gimg)

GM=[]
LM=[]
IM=[]
recon=[]

gausian=([0.0625,0.125,0.0625],[0.125,0.25,0.125],[0.0625,0.125,0.0625])
dup=gimg
#blur=convolve_2d(gimg,gausian)

for i in range(0,5):

    blur=convolve_2d(dup,gausian)
    blur = blur[::2,::2]
    blur=upsample(blur)

    GM.append(blur)
    IM.append(dup)
    laplace=(dup-blur)
    plt.subplot(122),plt.imshow(laplace, cmap = 'gray')
    plt.show()
    LM.append(laplace)
    filenameL="laplacian_%s.jpg"%(i)
    filenameI="orig_imag_%s.jpg"%(i)
    filenameG="gausian_%s.jpg"%(i)

    cv2.imwrite(filenameL,laplace)
    cv2.imwrite(filenameG,blur)
    cv2.imwrite(filenameI,dup)
    dup = dup[::2,::2]

gaus_dup=GM[4]

for i in range(0,5):


    recon=LM[4-i]+gaus_dup
    filenameR="reconst_%s.jpg"%(i)
    cv2.imwrite(filenameR,recon)
    gaus_dup=upsample(recon)

mse=getmse(gimg,recon)


print("MSE:",mse)


print("done")
```