2019702002
Nivedita Rufus

Digital Image Processing
Assignment 3

—

INTRODUCTION

Q1)

1. **Implement 1D Fast Fourier Transform (Recursive Formulation).**

$$y^k = F_{even\ k} + \omega^k F_{odd\ k}$$
$$y^{k+n/2} = F_{even\ k} - \omega^k F_{odd\ k}$$

Steps:

- Compute the fourier transform for the even and odd indices.
- In accordance with the algorithm stated above, compute the fourier transform of the entire sequence.

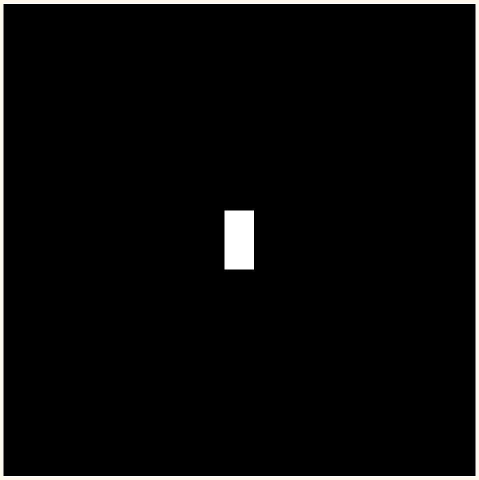
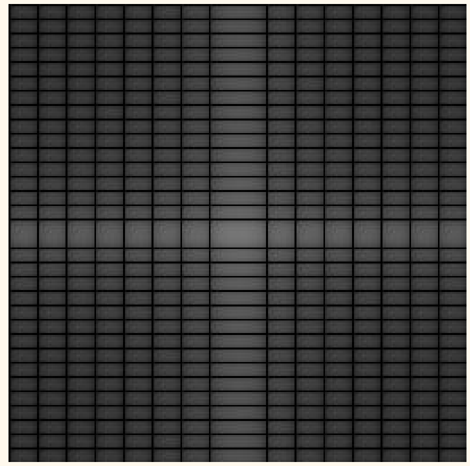
```
def Fast_fourier(x):
      N = x.shape[0]
      if N ==1:
            return x[0]
      else:
            X_even = Fast_fourier(x[::2])
            X_odd = Fast_fourier(x[1::2])
            factor = np.exp(-2*complex(0,1) * np.pi * np.arange(N) / N)
            return np.concatenate([X_even + factor[:int(N / 2)] * X_odd,
                                   X_even + factor[int(N / 2):] * X_odd])
```

2. **Use it to implement 2D FFT and display the result on suitable images of your choice.**

Steps:

- Compute 1D fft for rows using the above function
- Compute 1D fft for columns obtained from step 1

```
def Fast_fourier(x):
      N = x.shape[0]
      if N ==1:
            return x[0]
      else:
            X_even = Fast_fourier(x[::2])
            X_odd = Fast_fourier(x[1::2])
            factor = np.exp(-2*complex(0,1) * np.pi * np.arange(N) / N)
            return np.concatenate([X_even + factor[:int(N / 2)] * X_odd,
                                   X_even + factor[int(N / 2):] * X_odd])
def get_fourier_2D(x):
      y = np.zeros(x.shape)
      for i in range(x.shape[0]):
            y[i] = Fast_fourier(x[:,i])
      z = np.zeros(x.shape)
      for i in range(x.shape[1]):
            z[i] = Fast_fourier(y[:,i])
      return z
```

Input	Output
	

Q2)

1. **Implement the Ideal, Butterworth and Gaussian Low Pass Filters and apply them on lena.jpg.**

Ideal LPF:

$$H(u,v) = \begin{cases} 1 \text{ if } D(u,v) \leq D_0 \\ 0 \text{ if } D(u,v) > D_0 \end{cases}$$

Steps:

- Perform appropriate padding on the image.
- Compute the fourier transform of the image
- Design the filter H according to the function given above.
- Do an element wise multiplication of the fourier transformed image and the filter
- Compute the inverse fourier transform to get the result

```
def ideal_lpf(x,D0 = 50):
      H = np.zeros(x.shape)
      for i in range(H.shape[0]):
            for j in range(H.shape[1]):
                  y = np.sqrt((i-(H.shape[1])/2)**2 +
(j-(H.shape[0])/2)**2)
                  if y<=D0:
                        H[i,j] = 1
      return H
```

Input	Output
	

Butterworth LPF:

$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$$

Steps:

- Perform appropriate padding on the image.
- Compute the fourier transform of the image
- Design the filter H according to the function given above.
- Do an element wise multiplication of the fourier transformed image and the filter
- Compute the inverse fourier transform to get the result

```
def butterworth_lpf(x,D0 = 50,n=1):
      H = np.zeros(x.shape)
      for i in range(H.shape[0]):
            for j in range(H.shape[1]):
                  y = np.sqrt((i-(H.shape[1])/2)**2 +(j-(H.shape[0])/2)**2)
                  H[i,j] = 1/(1 + (y/D0))**(2*n)
      return H
```

Input	Output
	

Gaussian filter:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

Steps:

- Perform appropriate padding on the image.
- Compute the fourier transform of the image
- Design the filter H according to the function given above.
- Do an element wise multiplication of the fourier transformed image and the filter
- Compute the inverse fourier transform to get the result

```
def gaussian(x,D0 = 50):
      H = np.zeros(x.shape)
      for i in range(H.shape[0]):
            for j in range(H.shape[1]):
                  y = np.sqrt((i-(H.shape[1])/2)**2 +(j-(H.shape[0])/2)**2)
                  H[i,j] = math.exp(-((y**2)/(2*(D0)**2)))
      return H
```

Input	Output
	

2. **Using lena.jpg, apply the Gaussian low pass filter with two different values of σ. Compute the difference of the two outputs and display it. Report your observations.**

Steps:

- Follow the same procedure for obtaining the gaussian filtered image for different values of σ
- Subtract the images to see the difference.

Input:

Output: $\sigma = 20$	**Output : $\sigma = 60$**
	

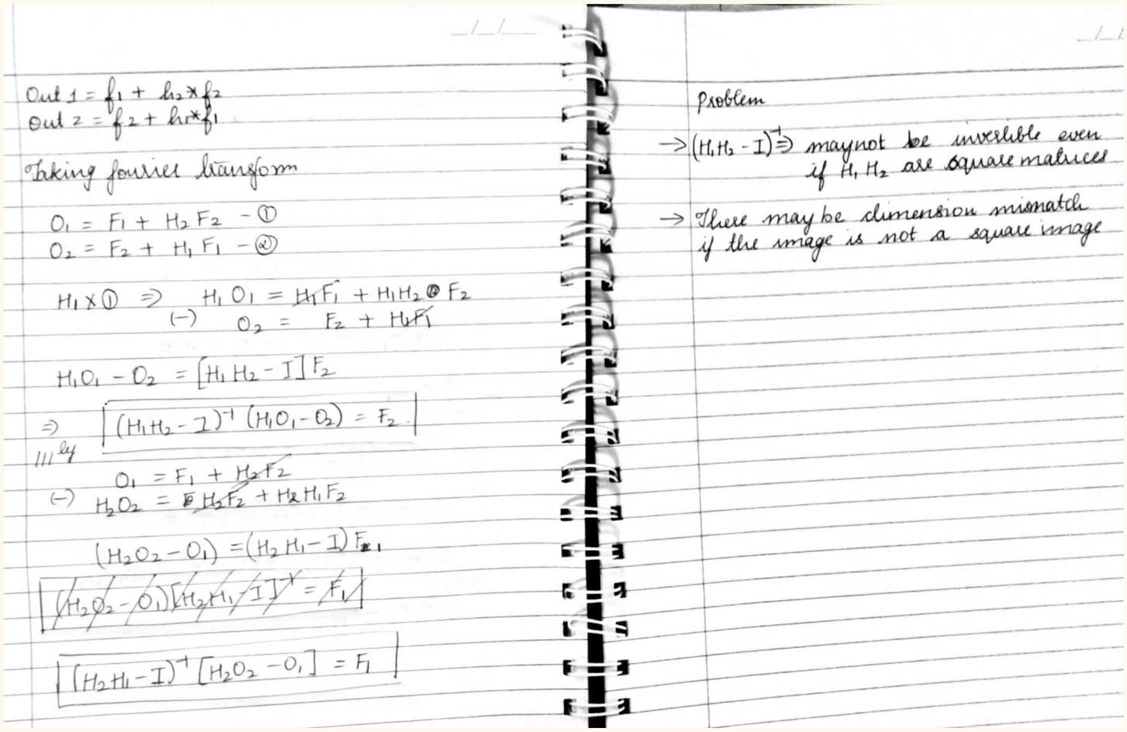
Difference:

Observations:

It can be observed that the higher the value of sigma produces less smoothing in comparison to lesser sigma value, the edges are blurred over the higher value of the variance of the gaussian.

Q3)

1. **Say you are travelling in a bus for a city tour in Paris and you want to capture the scene outside. Thankfully the Bus is stationary. Unfortunately, you can't open the window and the window acts as a semi-reflecting surface and the image contains reflections from inside the bus :(. But Hey, you got a camera which can focus on the outside scene by blurring the reflection off the window. This can be written as out1 = f1 + h2 * f2 where h2 is the blurring filter applied on f2. The second image is taken focusing the window surface, blurring the outside scene. This can be written as out2 = h1 * f1 + f2 where h1 is the blurring filter applied on f1. You are given two images out1 and out2. Assuming you know h1 and h2, how would you find f1 and f2. Do you see any issues with the formula derived?**



Out 1 = $f_1 + h_2 * f_2$
Out 2 = $f_2 + h_1 * f_1$

Taking fourier transform

$O_1 = F_1 + H_2 F_2$ — ①
$O_2 = F_2 + H_1 F_1$ — ②

$H_1 \times ① \Rightarrow H_1 O_1 = H_1 F_1 + H_1 H_2 F_2$
$(-)\quad O_2 = F_2 + H_1 F_1$

$H_1 O_1 - O_2 = [H_1 H_2 - I] F_2$

$\Rightarrow \boxed{(H_1 H_2 - I)^{-1} (H_1 O_1 - O_2) = F_2}$

|||ly

$O_1 = F_1 + H_2 F_2$
$(-)\quad H_2 O_2 = H_2 F_2 + H_2 H_1 F_2$

$(H_2 O_2 - O_1) = (H_2 H_1 - I) F_1$

$\cancel{(H_2 O_2 - O_1)[H_2 H_1 - I]^{-1} = F_1}$

$\boxed{(H_2 H_1 - I)^{-1} [H_2 O_2 - O_1] = F_1}$

Problem

→ $(H_1 H_2 - I)^{-1} \Rightarrow$ may not be invertible even if H_1, H_2 are square matrices

→ There may be dimension mismatch if the image is not a square image

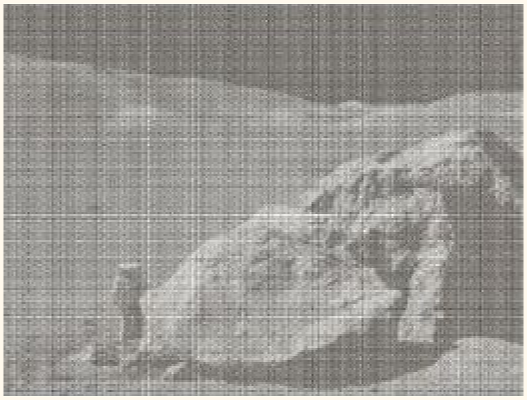
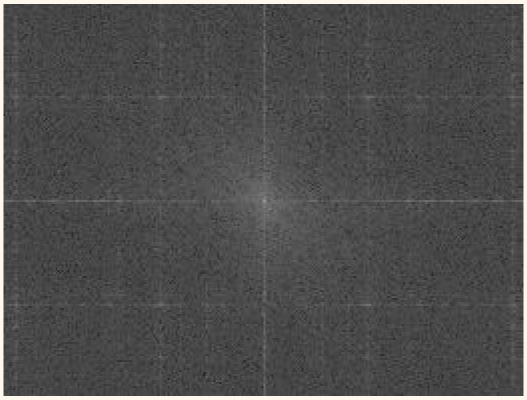
Q4)

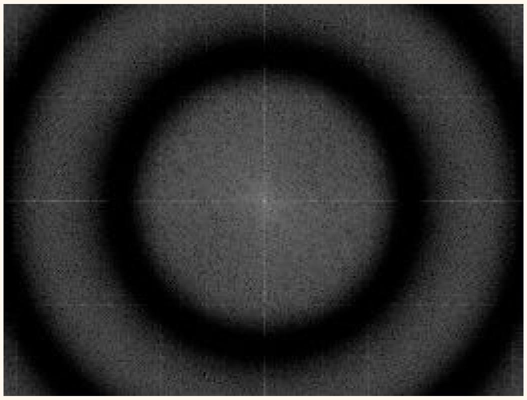
1. **Denoise the given image land.png and explain your process.**

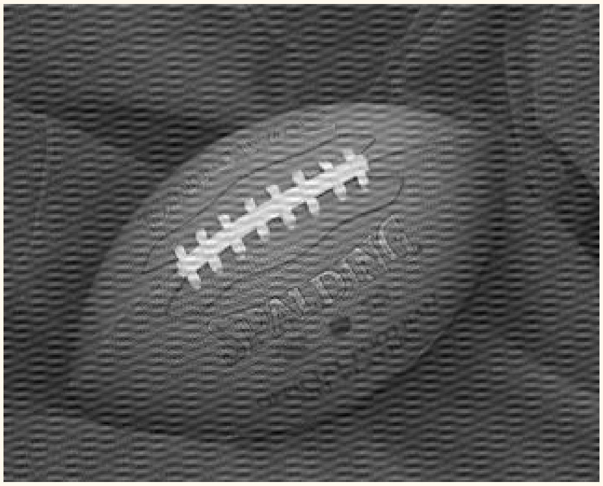
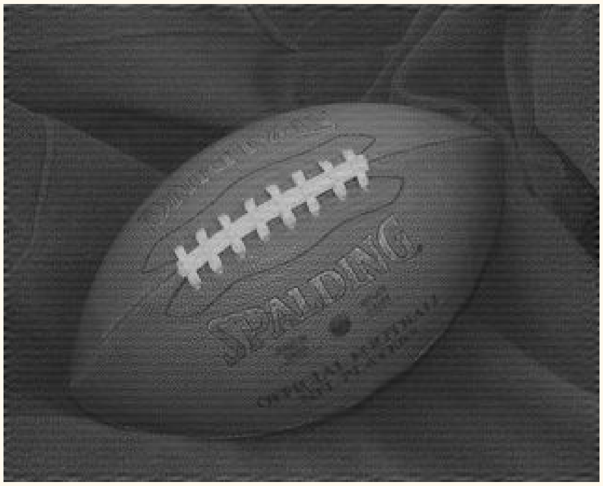
Steps:

- Compute the fourier transform of the image
- Investigate the location of peaks
- Design an appropriate band reject filter to remove the unwanted frequencies.

```
def butterworth_BR(x,D0 = 73,w=50,n=3):
      H = np.ones(x.shape)
      for i in range(H.shape[0]):
            for j in range(H.shape[1]):
                  D = np.sqrt((i-(H.shape[0])/2)**2 +
(j-(H.shape[1])/2)**2)
                  y = ((D*w)/(D**2 - D0**2))**(2*n)
                  H[i,j] = 1/(1 + y)

      cv2.imwrite('filter.jpg',20*np.log(abs(H)+1))
      return H
```

Input 	**Fourier transform** 

Output 	**Fourier transform** 

Input	Output
	

Q5)

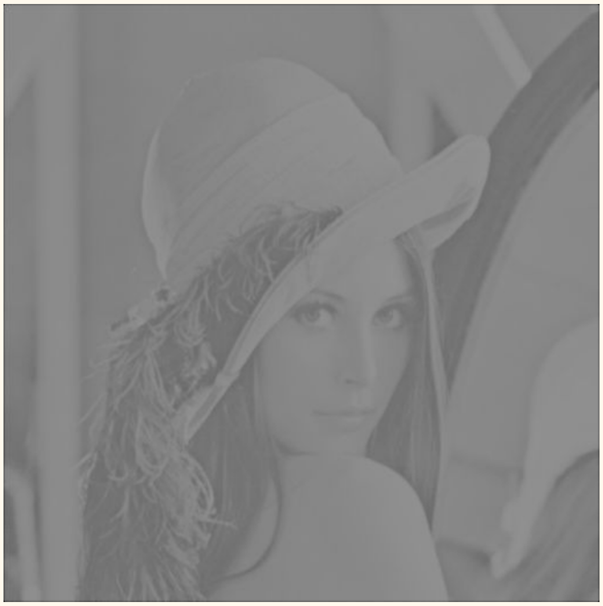
1. **Find the equivalent filter H(u, v) in the frequency domain for the following spatial filter and show results of applying this filter on an image of your choice (in the frequency domain). Is H(u, v) a low-pass filter or a high-pass filter? Show it mathematically.**

Steps:

- Perform appropriate padding on the image.
- Compute the fourier transform of the image
- Compute the fourier transform of the given mask and perform proper padding
- Multiply the image to obtain the corresponding frequency spectrum of the frequency filtered image.

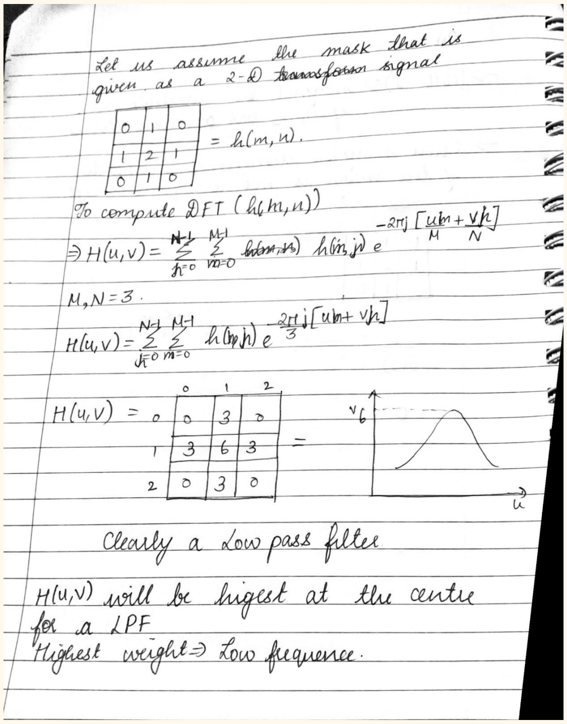
```
def filter(img):
      H = np.array([[0,1,0],[1,2,1],[0,1,0]])
      sz = (img.shape[0] - H.shape[0], img.shape[1] -H.shape[1])  # total
      H= np.pad(H, (((sz[0]+1)//2, sz[0]//2), ((sz[1]+1)//2,
sz[1]//2)),'constant')
      return H
```

```
def padding(im,kernel_row = 3, kernel_col = 3):
      image_row, image_col = im.shape
      pad_height = int((kernel_row - 1) / 2)
      pad_width = int((kernel_col - 1) / 2)
      padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2
* pad_width)))
      padded_image[pad_height:padded_image.shape[0] - pad_height,
pad_width:padded_image.shape[1] - pad_width] = im
      return padded_image

im = cv2.imread('lena.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im = padding(im)

im_fft = np.fft.fft2(im)
im_fft = np.fft.fftshift(im_fft)
cv2.imwrite("input.jpg",np.fft.ifft2(np.fft.fftshift(im_fft)).astype("uint8
"))

H = filter(im_fft)
cv2.imwrite("filter.jpg",10*np.log(abs(H)+1))
o = np.real(np.fft.fftshift(np.fft.ifft2(im_fft *
np.fft.fftshift(np.fft.fft2(H)))))+np.imag(np.fft.fftshift(np.fft.ifft2((im
_fft) * np.fft.fftshift(np.fft.fft2(H)))))
cv2.imwrite("offt.jpg",20*np.log(abs(o)+1))
cv2.imwrite("output.jpg",abs(o))
```

Input	Output
	

Explanation :

Low pass filter



Let us assume the mask that is given as a 2-D ~~transform~~ signal

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} = h(m,n).$$

To compute DFT $(h(m,n))$

$$\Rightarrow H(u,v) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \cancel{h(m,n)}\, h(m,n)\, e^{-2\pi j \left[\frac{um}{M} + \frac{vn}{N}\right]}$$

$M, N = 3.$

$$H(u,v) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} h(m,n)\, e^{-\frac{2\pi}{3} j [um + vn]}$$

$$H(u,v) = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 0 & 3 & 0 \\ 1 & 3 & 6 & 3 \\ 2 & 0 & 3 & 0 \end{array} =$$

Clearly a Low pass filter

H(u,v) will be higest at the centre for a LPF

Highest weight ⇒ Low frequence.

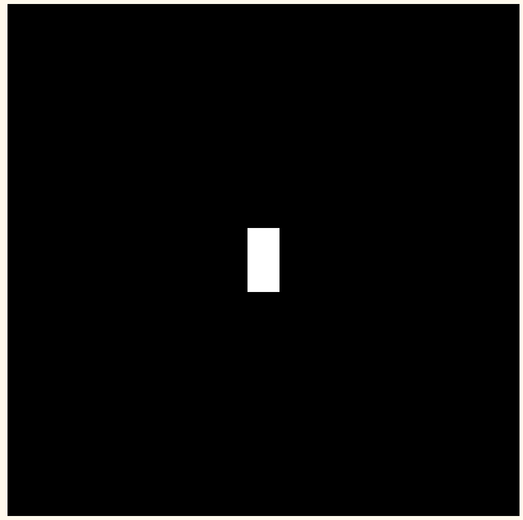
Q6)

1. **Pick images f and h of different dimensions, each not necessarily square, and verify the convolution theorem (DF T [f ∗ h] = F z H z , where F z and H z are the 2D-DFT of the images f z, h z , with f z , h z being the images f and h,with appropriate zero-padding).**

Steps:

- Compute fourier transform of the images
- Multiply the fourier transform of the images.
- Compute the convolution of the two images.
- Verify the difference by subtracting the images.

```
f = cv2.imread('lena.jpg')
h = cv2.imread('rectangle.jpg')
f = cv2.cvtColor(f, cv2.COLOR_BGR2GRAY)
h = cv2.cvtColor(h, cv2.COLOR_BGR2GRAY)
M = f.shape[0] + h.shape[0] - 1
N = f.shape[1] + h.shape[1] - 1
conv = convolve2d(f, h.astype(float))
print("fdg")
res = ifft2(ifftshift(fftshift(fft2(f, s=(M, N))) * fftshift(fft2(h, s=(M,
N)))))
res = abs(res)
print(h.shape)
print(conv.shape)
cv2.imwrite("idft.jpg",10*np.log(res+1))
cv2.imwrite("conv.jpg",10*np.log(conv+1))
```

Input (f)	Input (h)
	

Output (convolution)	Output (using fft)
	

2. **In the above question, find the time required to compute the convolution directly (using conv2) and using the DFT (find F e , H e after zero-padding, multiply point-wise, and take inverse DFT). Use matlab functions (tic, toc, cputime) for**

calculating the time required for your operations. What are your observations for various different dimensions of f, h ?

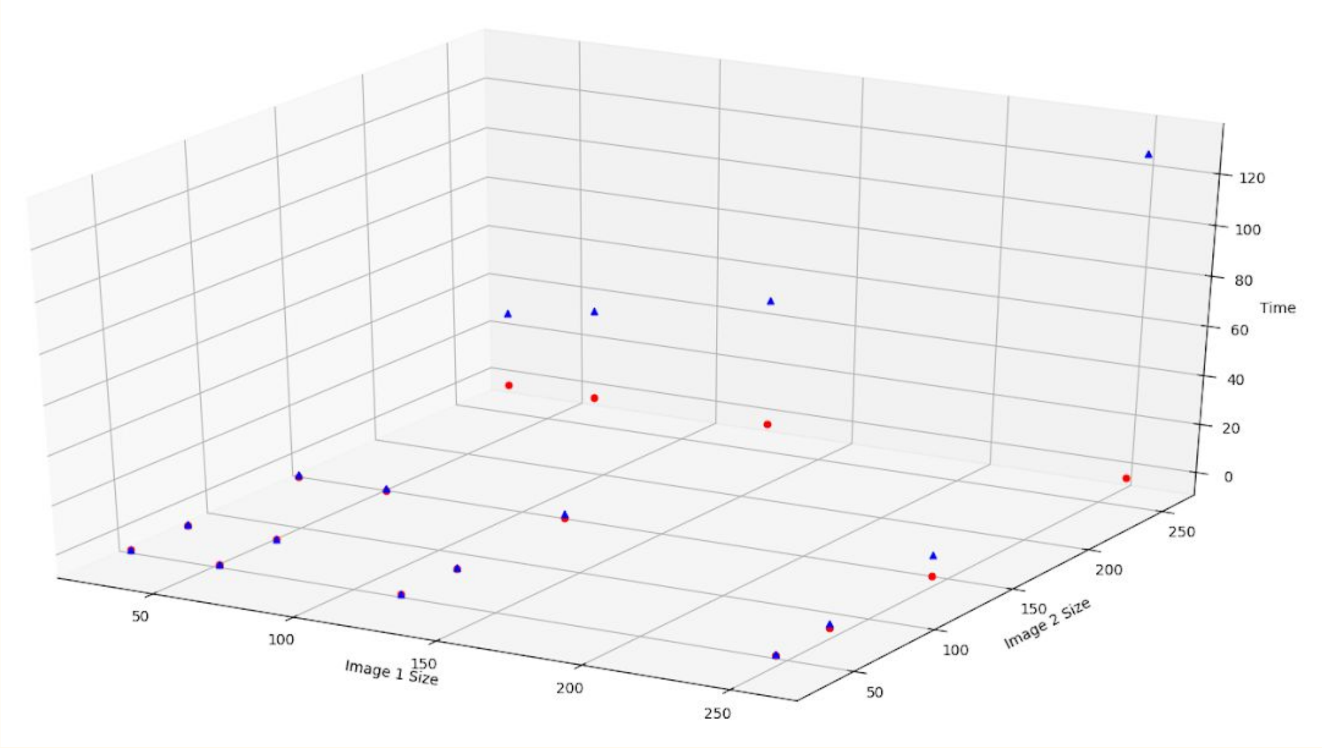
Steps:

- Get images of various sizes, perform the following steps by taking to at a time
- Compute fourier transform of the images
- Multiply the fourier transform of the images.
- Compute the convolution of the two images.
- Measure the time taken by both operations and plot it.

```
def conv_normal(img1, img2):
    return convolve2d(img1, img2.astype(float))

def conv_fft(img1, img2):
    M = img1.shape[0] + img2.shape[0] - 1
    N = img1.shape[1] + img2.shape[1] - 1
    res = ifft2(ifftshift(fftshift(fft2(img1, s=(M, N))) *
fftshift(fft2(img2, s=(M, N)))))
    res = abs(res)
    return res
h = cv2.imread('bricks.jpg')
im1 = cv2.cvtColor(h, cv2.COLOR_BGR2GRAY)
im2 = im1[:128,:128]
im3 = im1[:64,:64]
im4 = im1[:32,:32]
im_dict = {0:im4, 1:im3, 2:im2, 3:im1}
print(im_dict)
norm_time = zeros((5,5))
fft_time = zeros((5,5))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for row in range(4):
    for col in range(4):
        start = time()
        res = conv_fft(im_dict[row], im_dict[col])
        end = time()
        fft_time[row][col] = end - start
        f = end - start
        start = time()
```

```
        res = conv_normal(im_dict[row], im_dict[col])
        end = time()
        norm_time[row][col] = end - start
        n = end - start
        ax.scatter((2**row)*32, (2**col)*32, f, c='r', marker='o')
        ax.scatter((2**row)*32, (2**col)*32, n, c='b', marker='^')
# print(fft_time)
# print(norm_time)
ax.set_xlabel('Image 1 Size')
ax.set_ylabel('Image 2 Size')
ax.set_zlabel('Time')
plt.show()
```

Output (graph):



Observations:

2D Convolution takes much more time than performing convolution using FFT, as FFT is highly optimised. The difference is evident when Image 2 size is larger.

Q7)

Aliasing can arise when you sample a continuous function or an image. The minimum sampling rate to avoid aliasing is called the nyquist rate.

1. **Sample this image at different spatial sampling frequencies n_x, n_y . Find the nyquist rate for the grayscale version of the image bricks.jpg.**

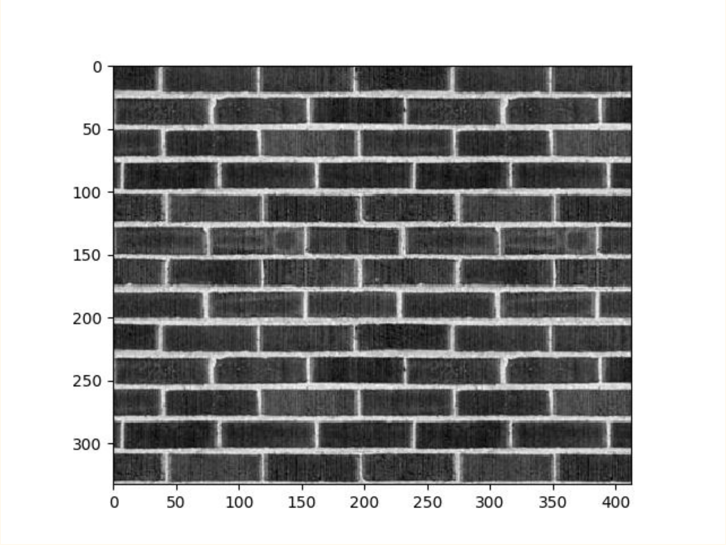
Steps:

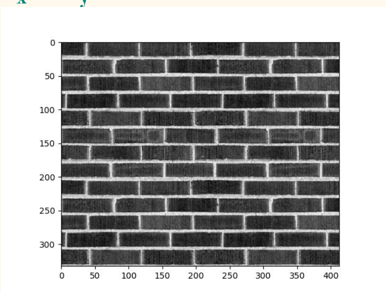
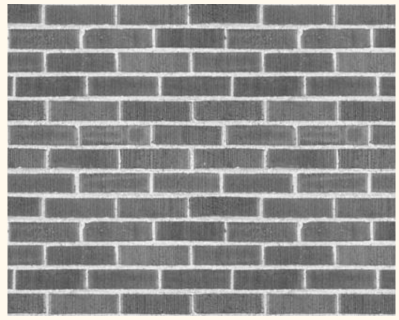
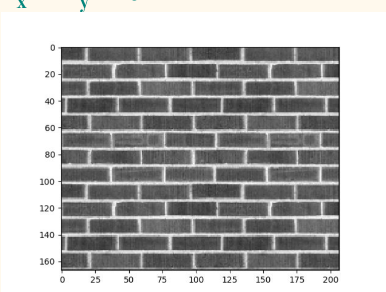
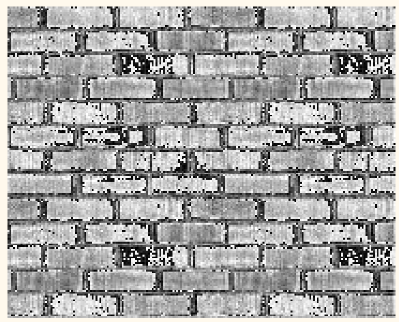
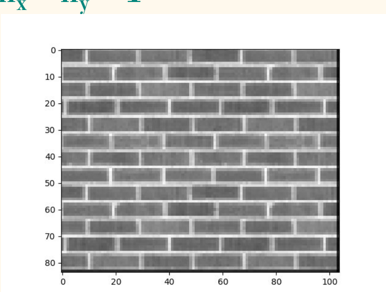
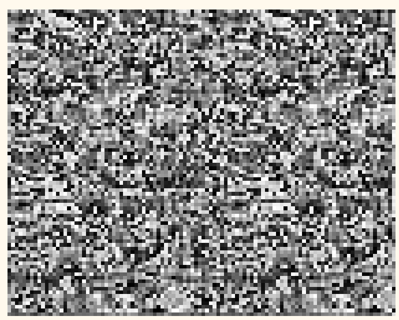
- Sample the image at different spatial frequencies.
- Compute the fourier transform of the image.
- Now take the inverse of the fourier transformed images to view the reconstructed image.

```
im = cv2.imread('bricks.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
cv2.imwrite("input.jpg",im)
sampled_1 = block_reduce(im, (1, 1))
sampled_2 = block_reduce(im, (2, 2))
sampled_4 = block_reduce(im, (4, 4))
sampled_8 = block_reduce(im, (8, 8))
sampled_16 = block_reduce(im, (16, 16))

sampled1_fft = fftshift(fft2(sampled_1))
sampled1_mag = abs(sampled1_fft)
res1 = ifft2(ifftshift(sampled1_fft)).astype("uint8")
cv2.imwrite("res1.jpg",res1)
cv2.imwrite("fft1.jpg",10*log(sampled1_mag +1))


sampled2_fft = fftshift(fft2(sampled_2))
sampled2_mag = abs(sampled2_fft)
res2 = ifft2(ifftshift(sampled2_fft)).astype("uint8")
cv2.imwrite("res2.jpg",res2)
cv2.imwrite("fft2.jpg",10*log(sampled2_mag +1))

sampled4_fft = fftshift(fft2(sampled_4))
sampled4_mag = abs(sampled4_fft)
res4 = ifft2(ifftshift(sampled4_fft)).astype("uint8")
```

```
cv2.imwrite("res4.jpg",res4)
cv2.imwrite("fft4.jpg",10*log(sampled4_mag +1))

sampled8_fft = fftshift(fft2(sampled_8))
sampled8_mag = abs(sampled8_fft)
res8 = ifft2(ifftshift(sampled8_fft)).astype("uint8")
cv2.imwrite("res8.jpg",res8)
cv2.imwrite("fft8.jpg",10*log(sampled8_mag +1))

sampled16_fft = fftshift(fft2(sampled_16))
sampled16_mag = abs(sampled16_fft)
res16 = ifft2(ifftshift(sampled16_fft)).astype("uint")
cv2.imwrite("res16.jpg",res16)
cv2.imwrite("fft16.jpg",10*log(sampled16_mag +1))
```

Input



Sampled image	Reconstructed
$n_x = n_y = 1$ 	
$n_x = n_y = 2$ 	
$n_x = n_y = 4$ 	

Observations:

It is observed that the image can be reconstructed properly when $n_x = n_y = 1$

2. **Investigate the effect of blurring the image on the nyquist rate. Show intermediate results wherever relevant.**

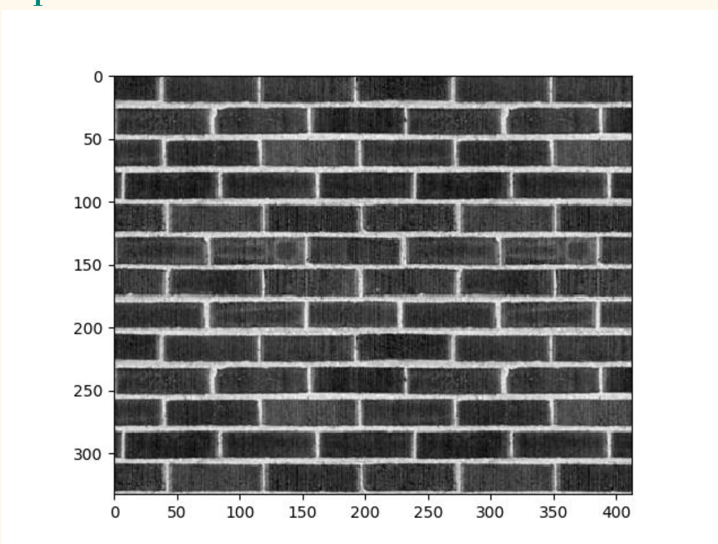
Steps:

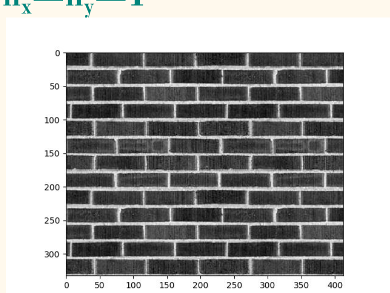
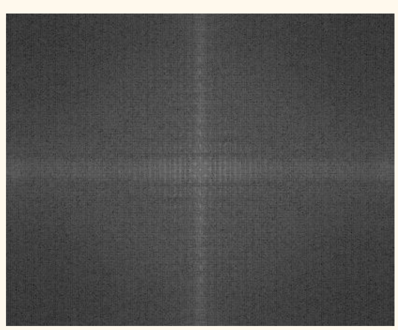
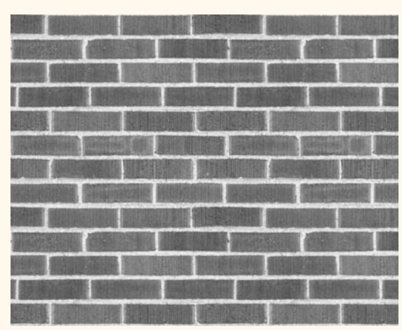
- Sample the image at different spatial frequencies.
- Compute the fourier transform of the image.
- Now take the inverse of the fourier transformed images to view the reconstructed image.

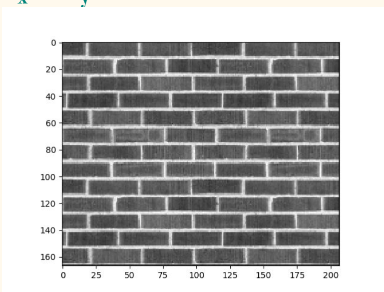
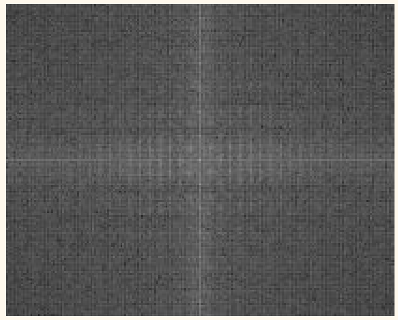
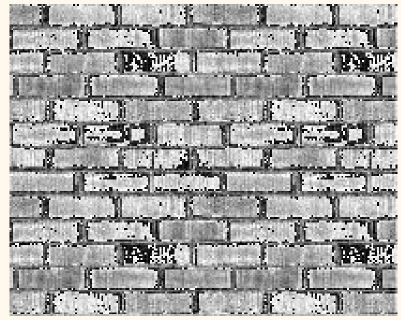
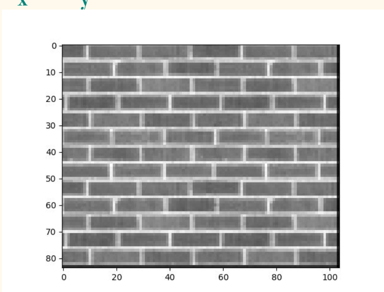
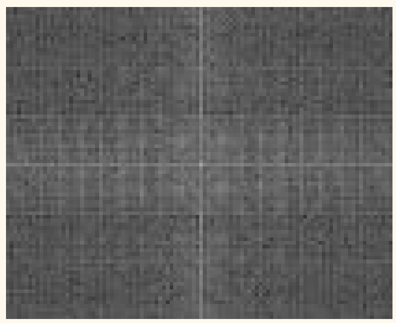
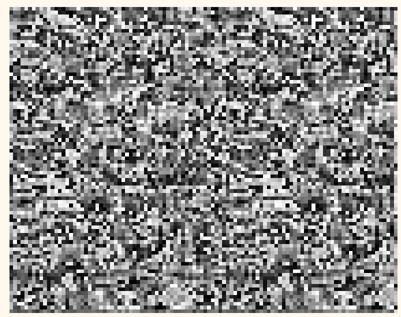
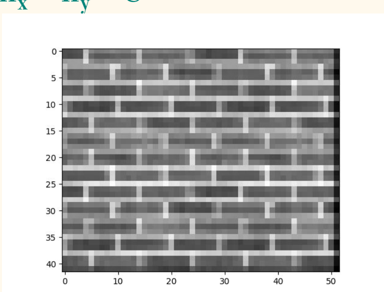
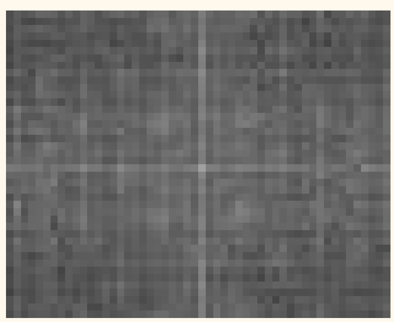
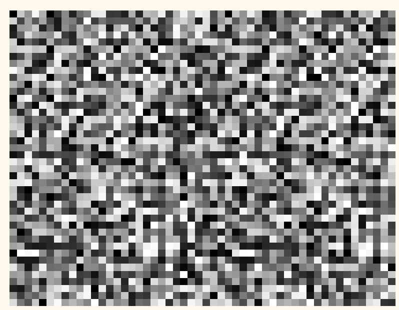
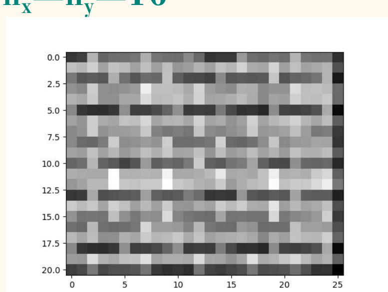
```
im = cv2.imread('bricks.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
cv2.imwrite("input.jpg",im)
sampled_1 = block_reduce(im, (1, 1))
sampled_2 = block_reduce(im, (2, 2))
sampled_4 = block_reduce(im, (4, 4))
sampled_8 = block_reduce(im, (8, 8))
sampled_16 = block_reduce(im, (16, 16))

sampled1_fft = fftshift(fft2(sampled_1))
sampled1_mag = abs(sampled1_fft)
res1 = ifft2(ifftshift(sampled1_fft)).astype("uint8")
cv2.imwrite("res1.jpg",res1)
cv2.imwrite("fft1.jpg",10*log(sampled1_mag +1))


sampled2_fft = fftshift(fft2(sampled_2))
sampled2_mag = abs(sampled2_fft)
res2 = ifft2(ifftshift(sampled2_fft)).astype("uint8")
cv2.imwrite("res2.jpg",res2)
cv2.imwrite("fft2.jpg",10*log(sampled2_mag +1))

sampled4_fft = fftshift(fft2(sampled_4))
sampled4_mag = abs(sampled4_fft)
res4 = ifft2(ifftshift(sampled4_fft)).astype("uint8")
cv2.imwrite("res4.jpg",res4)
cv2.imwrite("fft4.jpg",10*log(sampled4_mag +1))

sampled8_fft = fftshift(fft2(sampled_8))
sampled8_mag = abs(sampled8_fft)
res8 = ifft2(ifftshift(sampled8_fft)).astype("uint8")
cv2.imwrite("res8.jpg",res8)
cv2.imwrite("fft8.jpg",10*log(sampled8_mag +1))
```

```
sampled16_fft = fftshift(fft2(sampled_16))
sampled16_mag = abs(sampled16_fft)
res16 = ifft2(ifftshift(sampled16_fft)).astype("uint")
cv2.imwrite("res16.jpg",res16)
cv2.imwrite("fft16.jpg",10*log(sampled16_mag +1))
```

Input



Sampled image	Fourier transform	Reconstructed
$n_x = n_y = 1$ 		

$n_x=n_y=2$ 		
$n_x=n_y=4$ 		
$n_x=n_y=8$ 		
$n_x=n_y=16$ 		

Observations:

As nx, ny increases, it can be observed that as their values increases the image gets progressively more blurry.

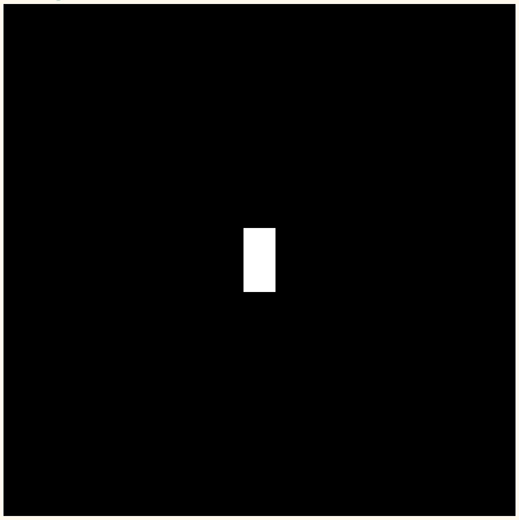
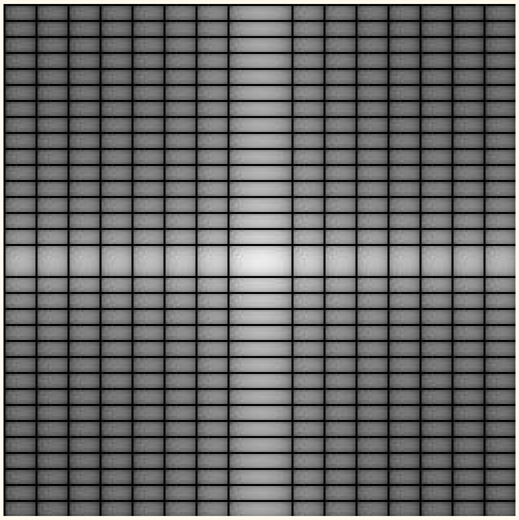
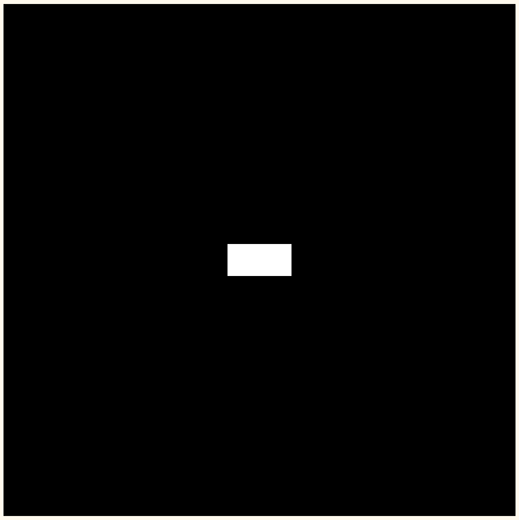
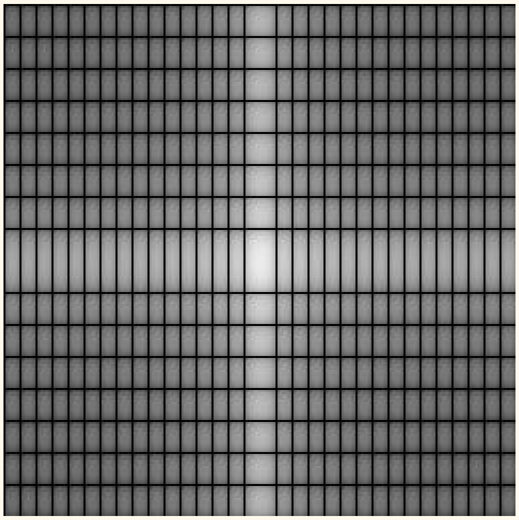
Q8)

1. **Compute the FFT of the image rectangle.jpg. Now rotate the image in spatial domain and compute the FFT of the rotated image. Report your observations and justify it mathematically.**

Steps :

- Compute the fourier transform of the image.
- Rotate the given image by an angle(90^0).
- Compute the fourier transform of the rotated image.

```
rot = rot90(im)
cv2.imwrite("rot.jpg",rot)
fft = fft2(rot)
fft = fftshift(fft)
mag = abs(fft)
log_mag = 20*log(mag + 1)
cv2.imwrite("rotfft.jpg", log_mag)
```

Input	Output
Original 	
Rotated 	

Justification:

It can be seen clearly that rotating the image by any amount rotates the spectra by the same angle.

Proof:

Rotate image by $\theta = \hat{f}(x,y)$

$$\hat{f}(x,y) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} f(x,y). \quad -(1)$$

$$\hat{F}(x,v) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} \hat{f}(x,y)\, e^{-2\pi j\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

$$\cancel{\hat{f}(x,y) = f(x,y)[} \quad \hat{f}(x,y) = \cancel{f(x)}\, R(\theta) f(x,y).$$

$$\hat{F}(u,v) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} R(\theta) f(x,y)\, e^{-2\pi j\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

$$= R(\theta) \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x,y)\, e^{-2\pi j\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

$$= R(\theta) F(u,v)$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} F(u,v) \end{bmatrix}$$

$\therefore$ fourier transform gets rotated by same angle.

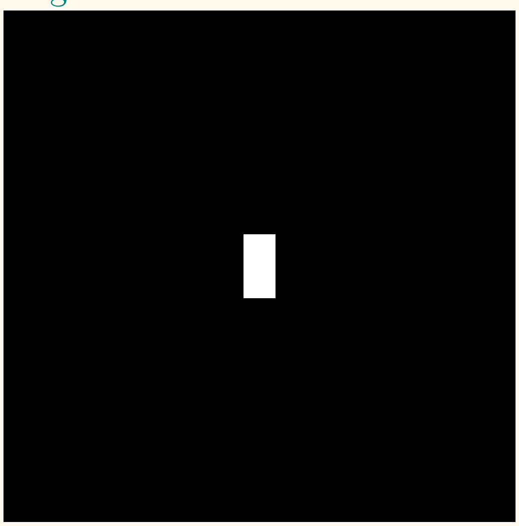
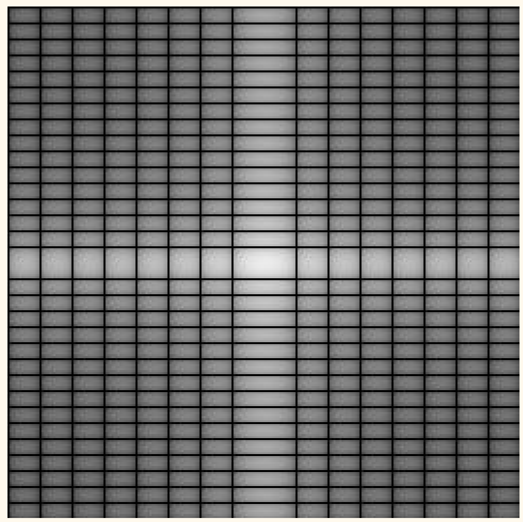
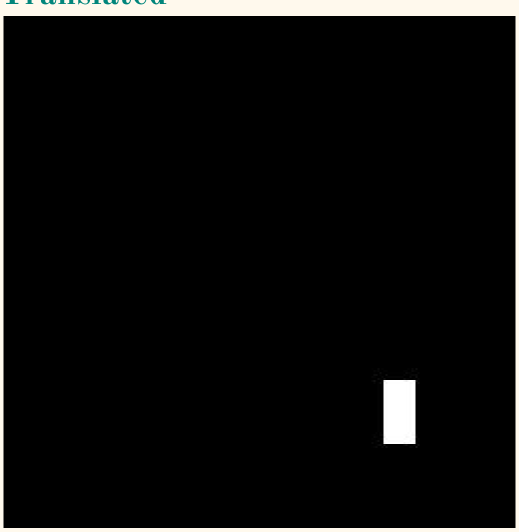
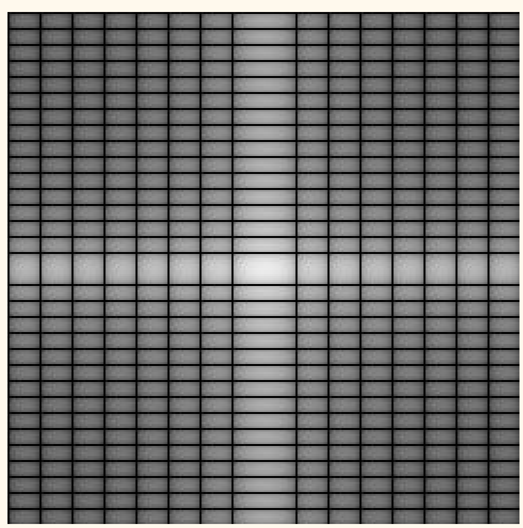
2. **Take the image rectangle.jpg and translate the image by few pixels Find the FFT of original and translated images. Report your observations and justify it mathematically.**

Steps :

- Compute the fourier transform of the image.
- translate the given image by a position.
- Compute the fourier transform of the translated image.

```
translation_matrix = np.float32([ [1,0,70], [0,1,70] ])
trans = cv2.warpAffine(im, translation_matrix, (im.shape))
cv2.imwrite("trans.jpg",trans)
```

```
fft = fft2(trans)
fft = fftshift(fft)
mag = abs(fft)
log_mag = 20*log(mag + 1)
cv2.imwrite("transfft.jpg", log_mag)
```

Input	Output
Original 	
Translated 	

Justification:

It can be seen that the process of shifting the origin of the frequency plane make the fourier transform translation invariant, i.e. no matter where we may translate the image, the origin of the spectra can always be centered on the image.

Proof:

$f(x,y) \leftrightarrow F(u,v)$.

for images

$f(x,y) \leftrightarrow F(u-M/2, v-N/2)$ (fourier shifted).

Translation in spatial domain:

$f(x,y) \leftrightarrow F(u,v) \xrightarrow{\text{fourier shift}} F(u-M/2, v-N/2)$.

$f(x+x_0, y+y_0) \leftrightarrow \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} (f(x+x_0, y+y_0)) e^{-2\pi j \left(\frac{ux}{M} + \frac{vy}{N}\right)}$

$x + x_0 = x' \Rightarrow x = x' - x_0$

$y + y_0 = y' \qquad y = y' - y_0$

$\Rightarrow f(x+x_0, y+y_0) \leftrightarrow \sum_{y'=y_0}^{N-1} \sum_{x'=x_0}^{M-1} (f(x', y')) e^{-2\pi j \left(\frac{u(x')}{M} + \frac{vy'}{N}\right)} \times e^{2\pi j \left(\frac{ux_0}{M} + \frac{vy_0}{N}\right)}$.

$\leftrightarrow F(u+u_0, v+v_0)$

$\leftrightarrow F(u', v')$.

fft shift $\leftrightarrow F(u-M/2, v-M/2)$

$\therefore$ It just shifts the origin of frequency plane.