Assignment-2 Report

Question 1:

I have used the following formulae for calculating forward fourier transform, inverse fourier transform, discrete cosine transform and magnitude of the fourier transform for a 2D matrix of size 15 by 15.

Forward fourier transform:

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j) \, [\cos\left[\frac{2\pi}{N}(ui + vj)\right] - \sqrt{-1} \sin\left[\frac{2\pi}{N}(ui + vj)\right]]$$

In python, $\sqrt{-1} = 1j$

u = {0,14} v = {0,14} (as it is a 15 by 15 matrix)

I have implemented this formula by using some of the basic math functions

```
a = ((2 * math.pi)/N[0]) * ((u * k)+(v * l))
cosval = math.cos(a)
sinval = math.sin(a)
# print(1j*sinval)
#medium = medium + matrix[k,l] * (cosval - (1j*sinval))
outp = outp + (matrix[k,l] * (cosval - (1j*sinval)))
output[u,v] = outp
```

math.pi – for getting and using the value of pi.
math.cos() – to compute cos value.
math.sin() – to compute sin value.
For these functions we will import math.
To implement j value for complex number I have imported cmath.
I have cross checked the output values by passing the input matrix to the inbuilt fft function.
`np.fft.fft2(matrix)`

I got same values in both the ways.
We will get complex values as output.

Inverse fourier transform:

$$I(i,j) = \underbrace{\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)\,[\cos\left[\frac{2\pi}{N}(ui+vj)\right] + \sqrt{-1}\sin\left[\frac{2\pi}{N}(ui+vj)\right]]}$$

i = {0,..14}, j = {0,..14}

End up getting complex numbers

I have implemented this formula by using some of the basic math functions

```
a = ((2 * math.pi) / N[0]) * ((u * k) + (v * l))
cosval = math.cos(a)
sinval = math.sin(a)
# print(1j*sinval)
# medium = medium + matrix[k,l] * (cosval - (1j*sinval))
outp = outp + (matrix[k, l] * (cosval + (1j * sinval)))
output[u, v] = outp
```

math.pi – for getting and using the value of pi.
math.cos() – to compute cos value.
math.sin() – to compute sin value.
For these functions we will import math.
To implement j value for complex number I have imported cmath.
I have cross checked the output values by passing the input matrix to the inbuilt fft function.

```
np.fft.ifft2(matrix)
```

From both the methods the ratio of respective pixels is same.
Formula(0,0) / inbuilt(0,0) = Formula(0,1) / inbuilt(0,1)……….(ignoring complex values).
We will get complex values as output.

Discrete cosine transform:

I have implemented this formula by using some of the basic math functions

$$\underbrace{F(u,v) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} I(i,j)\,[\cos\left[\frac{2\pi}{N}(ui+vj)\right]]}_{\text{No complex part}}$$

I have implemented this formula by using some of the basic math functions

```
a = ((2 * math.pi) / N[0]) * ((u * k) + (v * l))
cosval = math.cos(a)

# print(1j*sinval)
# medium = medium + matrix[k,l] * (cosval - (1j*sinval))
outp = outp + (matrix[k, l] * (cosval))
output[u, v] = outp
```

math.pi – for getting and using the value of pi.
math.cos() – to compute cos value.
For these functions we will import math.
To implement j value for complex number I have imported cmath.
We don't get complex values in output here.

Magnitude of the fourier transform:

First I have calculate the fourier transform with the same formula as the first part of the question 1 and then calculated the magnitude for it.

$$M = |F(u,v)|$$

```
magnitude = abs(output)
```

I have used abs function to calculate absolute values.
In this we will get float values as output as I have given dtype as float.

Question 2:

For filtering we need to apply mask to input image. In this assignment we have total 6 filters.
I have implemented the following formulae for each of the filters.

Ideal low pass filter:

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

$$D(u,v) = \left[(u - P/2)^2 + (v - Q/2)^2 \right]^{1/2}$$

D(u,v) – distance between point (u,v) and center of the frequency rectangle.

Do – constant (we are passing it as cutoff)

```
c = shape[1]
r = shape[0]
mask = np.zeros((r, c), np.uint8)
for u in range(r):
    for v in range(c):
        value = ((u - (r/2)) ** 2 + (v - (c/2)) ** 2) ** (1 / 2)
        if (value <= cutoff):
            mask[u, v] = 1
        else:
            mask[u, v] = 0

return mask
```

Implementing this formula and returning the mask matrix.

Ideal high pass filter:

Negation of ideal low pas filter gives you the ideal high pass filter

$$H_{HP}(u,v) = 1 - H_{LP}(u,v)$$

A 2-D ideal highpass filter (IHPL) is defined as

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

```
il_mask = self.get_ideal_low_pass_filter(shape, cutoff)
mask = 1 - il_mask
```

Implementing this formula and returning the mask matrix.

Gaussian low pass filter:

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

Do – cutoff

Implemented e with math.exp()

For this imported math.

```
c = shape[1]
r = shape[0]
mask = np.zeros((r, c))
for u in range(r):
    for v in range(c):
        value = ((u - (r / 2)) ** 2 + (v - (c / 2)) ** 2) ** (1 / 2)
        mask[u, v] = 1 / (math.exp(value ** 2 / (2 * (cutoff ** 2))))
```

Implementing this formula and returning the mask matrix.

Gaussian high pass filter:

Negation of ideal low pas filter gives you the ideal high pass filter

A 2-D Gaussian highpass filter (GHPL) is defined as

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

```
gl_mask = self.get_gaussian_low_pass_filter(shape, cutoff)
mask = 1 - gl_mask
```

Implementing this formula and returning the mask matrix.

Butterworth low pass filter:

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

Do – cutoff
N – order

```
c = shape[1]
r = shape[0]
n = 2*self.order
#print(n)
mask = np.zeros((r, c), np.uint8)
for u in range(r):
    for v in range(c):
        value = ((u - (r / 2)) ** 2 + (v - (c / 2)) ** 2) ** (1 / 2)
        mask[u, v] = 1/(1+((value/cutoff)**(n)))
```

Implementing this formula and returning the mask matrix.

Butterworth high pass filter:

Negation of ideal low pas filter gives you the ideal high pass filter

```
bl_mask = self.get_ideal_low_pass_filter(shape, cutoff)
mask = 1 - bl_mask
```

Implementing this formula and returning the mask matrix.

All the low pass filters give blurred images and all high pass images gives sharpened images.

Filtering function:

Magnitude of dft

1. First I have calculated fft for the input image(used numpy)

```
img_fft = np.fft.fft2(input)
```

2. Calculated fft shift

```
np.fft.fftshift(img_fft)
```

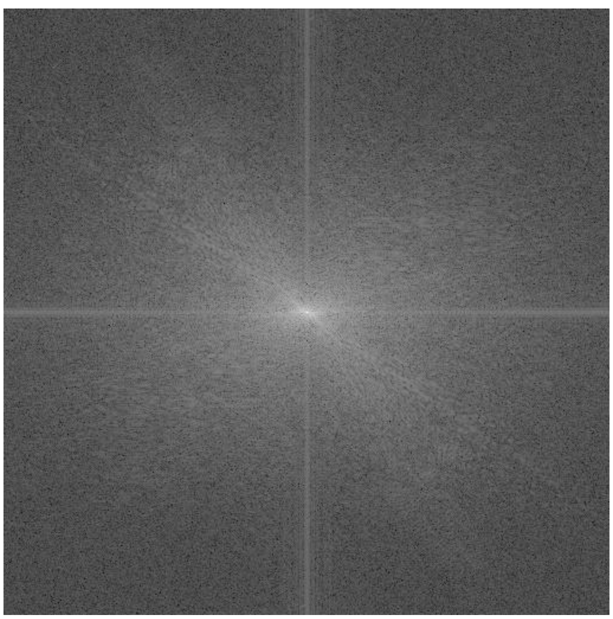
3. Calculated the magnitude of fft shift
4. Then applied log to the magnitude.

```
dft_log = np.log(1+magnitude)
```

5. At last applied full contrast stretch in order to make the image visible as after applying log we will get low values and image will not be clear.

```
coeff = (255) / (dft_log.max() - (dft_log.min()))
coeff1 = dft_log - (dft_log.min())
cont_stret = coeff * coeff1
```

This gives the magnitude of the dft.
one sample output for magnitude of dft



Ideal low pass filter, cutoff = 50

Magnitude of filtered dft:

1. Calculated shift for the selected mask

```
mask_shift = mask*self.shift
```

2. Then applied log to the magnitude

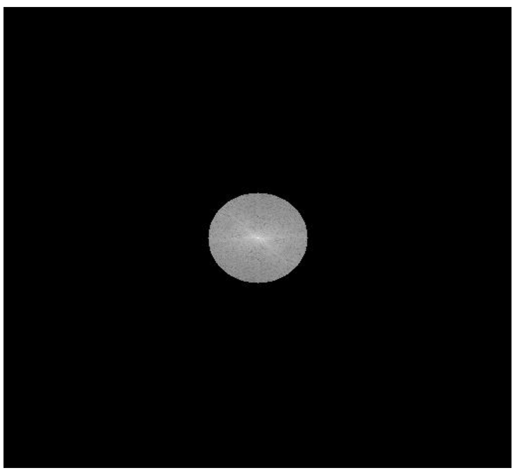
```
mask_abs = np.log(1 + abs(mask_shift))
```

3. At last applied full contrast stretch in order to make the image visible as after applying log we will get low values and image will not be clear.

```
maskcoeff = (255) / (mask_abs.max() - mask_abs.min())
maskcoeff1 = mask_abs - (mask_abs.min())
mask_strech = maskcoeff * maskcoeff1
```

4. This gives the magnitude of the filtered dft

one sample output for magnitude of the filtered dft



ideal low pass filter, cutoff = 50

Filtered image:

1. Calculated inverse shift and then inverse fft for shift of mask

```
mask_inverse = np.fft.ifft2(np.fft.ifftshift(mask_shift))
```

2. Calculated magnitude

```
mask_invabs = abs(mask_inverse)
```

3. At last applied full contrast stretch in order to make the image visible.

```
maskinv_coeff = (255) / (mask_invabs.max() - mask_invabs.min())
maskinv_coeff1 = mask_invabs - (mask_invabs.min())
mask_invstrech = maskinv_coeff * maskinv_coeff1
```

4. This gives the filtered image

one sample output for filtered image

ideal low pass filter, cutoff = 50

Observations:

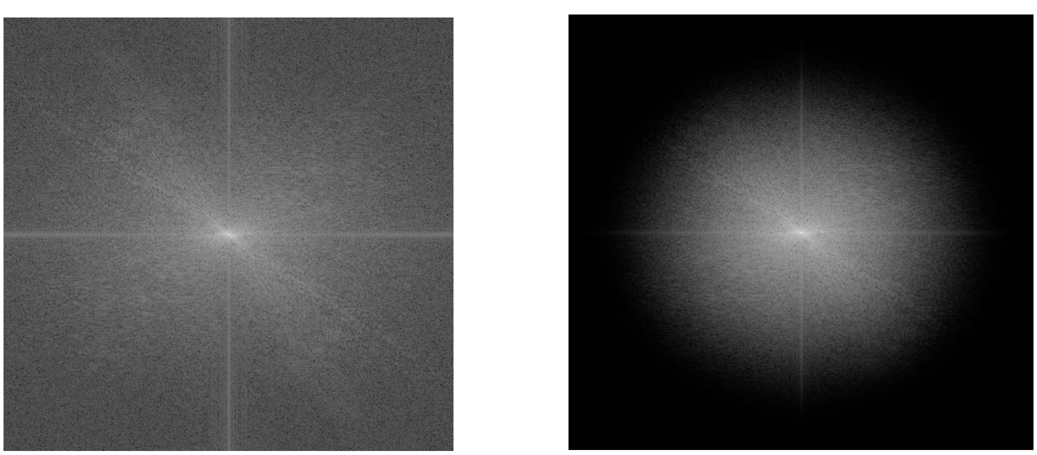
1. For low pass filters as the cutoff increases image becomes smoother with less blurring.

2. For high pass filters as the cutoff increases image becomes sharper (with edges clearly represented) and less noise

3. Butterworth filters are better that ideal pass filters as they give better results with respect to clarity and representation.

4. If we compare all the three filters then gaussian filters gives the best results.

sample gussian low pass outputs for cutoff = 50



sample gussian high pass outputs for cutoff = 50