EE 610 – Digital Image Processing

Assignment 1

Nihal Rajan Barde
170260010
Python Language

Q 1.
Observation -: Higher number of low intensity pixels
Conclusion -: Log transformation will give better results
Reason -: Log transformation maps lower intensity values to relatively higher values

original Image -:

log transformation -:

antilog transformation -:

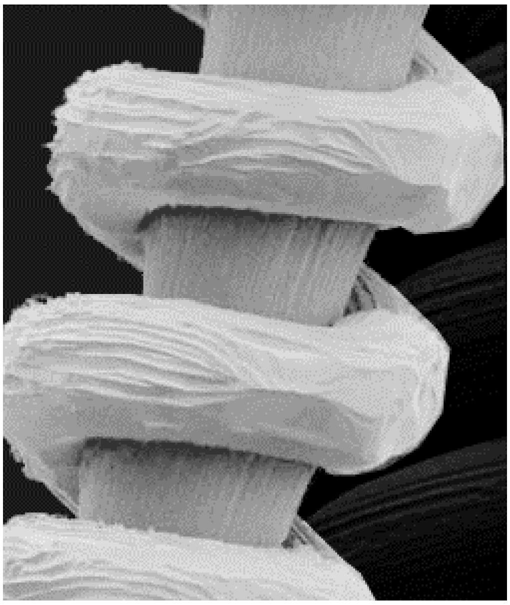
code -:

```
import cv2
import numpy as np

def logTrans(img):
	maxVal = np.amax(img) # max value in image
	cTarns = 255.0/np.log(1+maxVal) # scaling factor which sets max value to 255

	imgTrans = cTarns*(np.log(img+1.001)) # log transformation
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

def antilogTrans(img):
	maxVal = np.amax(img) # max value in image
	imgTemp = (np.exp(img*1.0/maxVal)-1) # taking antilog (range from 0 to e-1 )

	imgTrans = 255*(imgTemp/(np.e-1)) # scaling max value to 255
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q1.png",0) # reading image using openCV

	imgLog = logTrans(imgOriginal) # log transformation function
	imgAntilog = antilogTrans(imgOriginal) # antilog transformation function

	cv2.imwrite("./outputs/q1_log.png",imgLog)
	cv2.imwrite("./outputs/q1_antilog.png",imgAntilog)
```

Q 2.

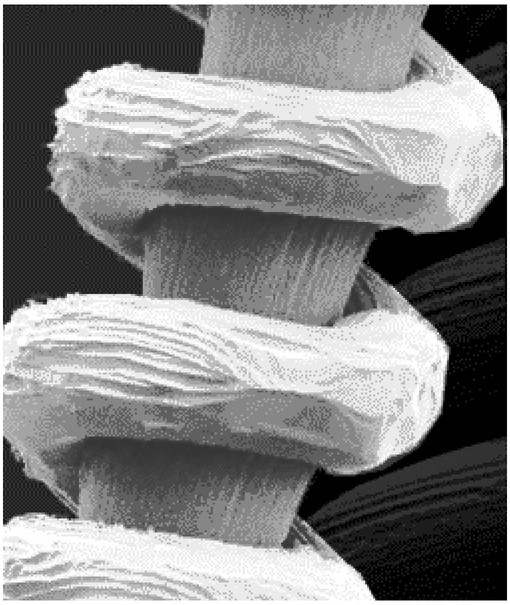
observation -: Image have different range of intensity values in different parts
conclusion & reason -: local histogram matching will give too much different output if every local patch is normalized pn scale 0 to 255. so we will scale about local min to local max. Global maximum will give good result as image have good number of high and low intensity values.

Original image -:

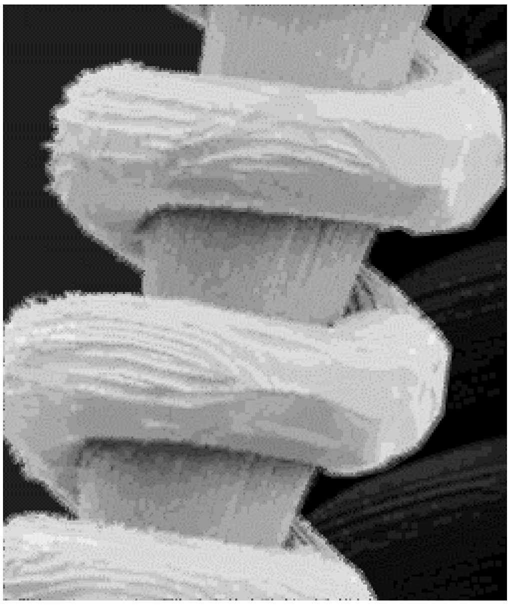


output images -:

global histogram equalization

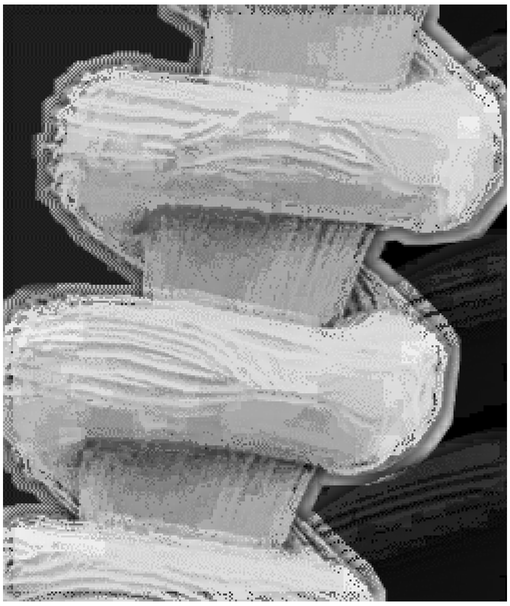


3x3 local histogram equalization



5x5 local histogram equalization

11x11 local histogram equalization



code -:

```
import cv2
import numpy as np

def globalHistTrans(img):
    # making histogram of complete image
    hist, bins = np.histogram(img.flatten(),256,[0,256])
    cdf = hist.cumsum()

    # normalizing histogram
    cdfMasked = np.ma.masked_equal(cdf,0)
    cdfNorm = (cdfMasked - cdfMasked.min())*255.0/(cdfMasked.max()-cdfMasked.min())

    # retaining result
    cdfResult = np.ma.filled(cdfNorm,0).astype("uint8")
    imgHist = cdfResult[img]

    return imgHist

def hist(img):
    # finding probability distribution in local domain
    probs = np.array([0.0 for i in range(256)])
    for x in img.flatten():
        probs[x] += 1
    # normalizing probability function
    probs /= np.sum(probs)

    s = img[img.shape[0]//2,img.shape[1]//2]
    imgMin, imgMax = img.min(), img.max()

    # getting result about min and max value
    return imgMin + np.sum(probs[:s+1])*(imgMax-imgMin)
```

```
def localHistTrans(img,kernel):
	imgResult = img.copy()
	# zero padding
	imgPad = 
cv2.copyMakeBorder(img,kernel.shape[0]//2,kernel.shape[0]//2,kernel.shape[1]//2,kernel.shape[1]//
2,borderType=0)

	imgMin, imgMax = img.min(), img.max()
	imgShape = img.shape

	# calculating for each pixel differently
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i][j] = 
round((hist(imgPad[i:i+kernel.shape[0],j:j+kernel.shape[1]])))
	return imgResult

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q2.png",0) # reading image using openCV
	x = 11
	kernel = np.ones((x,x))/(x*x)
	imgGlobalHist = globalHistTrans(imgOriginal)
	imgLocalHist = localHistTrans(imgOriginal,kernel)

	cv2.imwrite("./outputs/q2_global histogram.png",imgGlobalHist)
	cv2.imwrite("./outputs/q2_local histogram_11x11.png",imgLocalHist)
```

Q 3.

a) observation -: noise does not belong to salt & pepper noise class
   conclusion -: using smoothing filter
b) observation -: image have salt & pepper noise
   conclusion -: using median filter

3a input

3a after applying smoothing filter

3b input

3b after median filter

3a code spatial + frequency

```
import cv2
import numpy as np
import math

def getDFT(img): # DFT function
	imgShape = img.shape
	imgResult = img.copy()
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*img[i,j]
	return np.fft.fft2(imgResult)

def getLogDFT(img): # DFT + log for getting DFT of image
	imgFFT = getDFT(img)
	imgFFT = np.abs(imgFFT)
	imgLogFFT = logTrans(imgFFT)

	return imgLogFFT
```

```
def IDFT(img): # IDFT function
	imgShape = img.shape
	imgResult = img.copy()
	imgIFFT = np.real(np.fft.ifft2(img))
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*imgIFFT[i,j]
	return imgResult.astype("uint8")

def logTrans(img):
	maxVal = np.amax(img) # max value in image
	cTarns = 255.0/np.log(1+maxVal) # scaling factor which sets max value to 255

	imgTrans = cTarns*(np.log(img+1.001)) # log transformation
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

def smooth(img,kernel):
	imgShape = img.shape
	# padding image by zeros
	imgPad =
cv2.copyMakeBorder(img,kernel.shape[0]//2,kernel.shape[0]//2,kernel.shape[1]//2,kernel.shape[1]//
2,borderType=0)
	# padding kernel by zeros to make size as same as padded image
	kernelPad = cv2.copyMakeBorder(kernel,0,imgShape[0]-1,0,imgShape[1]-1,borderType=0)

	# getting DFT of image and kernel
	kernelDFT = getDFT(kernelPad)
	imgDFT = getDFT(imgPad)

	# multiplying by kernel and retrieving result after IDFT
	resultFFT = imgPad*kernelPad
	result = IDFT(resultFFT)

	imgKernel = getLogDFT(kernelPad)
	cv2.imwrite('./outputs/q3a_filter.png', imgKernel)

	return result #returning the output


if __name__ == "__main__":
	kernel = np.ones((5,5))/(5*5)
	img = cv2.imread('./Data/q3a.png' , 0)
	r_im = getLogDFT(img)

	cv2.imwrite('./outputs/q3a_fft.png', r_im)

	result = smooth(img,kernel)
	cv2.imwrite('./outputs/q3a_out.png', result)
```

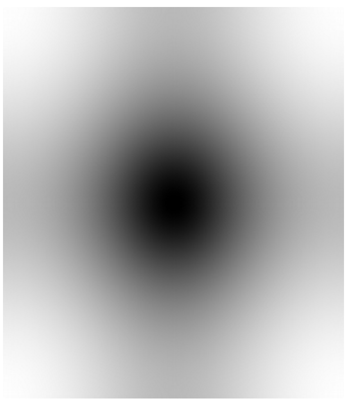
3b code spatial only

```
import cv2
import numpy as np

def spatialSmoothTrans(img,kernel):
	imgResult = img.copy()
	# padding by zeros
	imgPad = 
cv2.copyMakeBorder(img,kernel.shape[0]//2,kernel.shape[0]//2,kernel.shape[1]//2,kernel.shape[1]//
2,borderType=0)
	imgShape = img.shape
	# multiplying by kernel
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i][j] = 
np.sum(imgPad[i:i+kernel.shape[0],j:j+kernel.shape[1]]*kernel)
	return imgResult

def medianTrans(img,kernel):
	imgResult = img.copy()
	# zero padding
	imgPad = 
cv2.copyMakeBorder(img,kernel.shape[0]//2,kernel.shape[0]//2,kernel.shape[1]//2,kernel.shape[1]//
2,borderType=0)
	imgShape = img.shape
	# getting median for each image kernel
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i][j] = np.median(imgPad[i:i+kernel.shape[0],j:j+kernel.shape[1]])
	return imgResult

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q3b.png",0) # reading image using openCV
	kernel = np.array([[1,2,1],[2,4,2],[1,2,1]]).astype(float)
	kernel /= np.sum(kernel)
	imgSmooth = spatialSmoothTrans(imgOriginal,kernel)

	kernel = np.ones((3,3))/(3*3)
	imgMedian = medianTrans(imgOriginal,kernel)

	cv2.imwrite("./outputs/q3b_smooth.png",imgSmooth)
	cv2.imwrite("./outputs/q3b_median.png",imgMedian)
```

Q 4.

3x3 sharpening filter not giving good results as background is sharp
Laplacian -: [0,-1,0],[-1,4,-1],[0,-1,0]
4 input image

4 output image
same for spatial and frequency domain



4 filter

q4 code

```
import cv2
import numpy as np
import math

def getDFT(img): # DFT function
	imgShape = img.shape
	imgResult = img.copy()
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*img[i,j]
	return np.fft.fft2(imgResult)

def getLogDFT(img): # DFT + log for getting DFT of image
	imgFFT = getDFT(img)
	imgFFT = np.abs(imgFFT)
	imgLogFFT = logTrans(imgFFT)

	return imgLogFFT

def IDFT(img): # IDFT function
	imgShape = img.shape
	imgResult = img.copy()
	imgIFFT = np.real(np.fft.ifft2(img))
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*imgIFFT[i,j]
	return imgResult.astype("uint8")

def logTrans(img):
	maxVal = np.amax(img) # max value in image
	cTarns = 255.0/np.log(1+maxVal) # scaling factor which sets max value to 255

	imgTrans = cTarns*(np.log(img+1.001)) # log transformation
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

def laplacianFourierTrans(img,kernel):
	imgResult = img.copy()

	imgShape = img.shape
	# padding image by zeros
	imgPad = 
cv2.copyMakeBorder(img,kernel.shape[0]//2,kernel.shape[0]//2,kernel.shape[1]//2,kernel.shape[1]//
2,borderType=0)
	# padding kernel by zeros to make size as same as padded image
	kernelPad = cv2.copyMakeBorder(kernel,0,imgShape[0]-1,0,imgShape[1]-1,borderType=0)

	# getting DFT of image and kernel
	kernelDFT = getDFT(kernelPad)
```

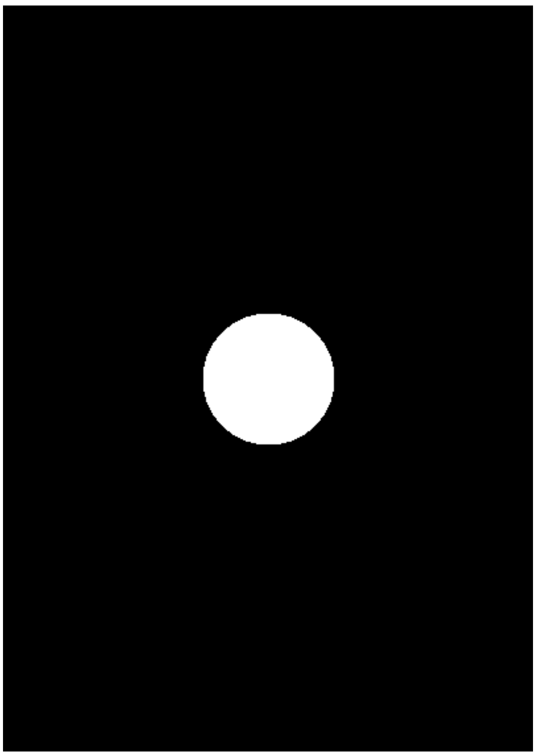
```
	imgDFT = getDFT(imgPad)

	# multiplying by kernel and retrieving result after IDFT
	resultFFT = imgPad*kernelPad
	result = IDFT(resultFFT)

	imgKernel = getLogDFT(kernelPad)
	cv2.imwrite('./outputs/q4_filter.png', imgKernel)

	return result #returning the output

def laplacianTrans(img,kernel):
	imgResult = img.copy()
	imgPad = cv2.copyMakeBorder(img,1,1,1,1,borderType=0)
	imgShape = img.shape
	imgResult[:,:] = imgResult[:,:] + 1*((4*imgPad[1:-1,1:-1]) - (1*imgPad[:-2,1:-1]) -
(1*imgPad[2:,1:-1]) - (1*imgPad[1:-1,:-2]) - (1*imgPad[1:-1,2:]))
	imgResult = np.clip(imgResult,0,255).astype("uint8")
	return imgResult

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q4.png",0) # reading image using openCV

	kernel = np.array([[0,1,0],[1,-4,1],[0,1,0]])
	imgLaplacian = laplacianTrans(imgOriginal,kernel)
	imgFourierLaplacian = laplacianFourierTrans(imgOriginal,kernel)

	cv2.imwrite("./outputs/q4_spatial.png",imgLaplacian)
	cv2.imwrite("./outputs/q4_fourier.png",imgFourierLaplacian)
```

q5

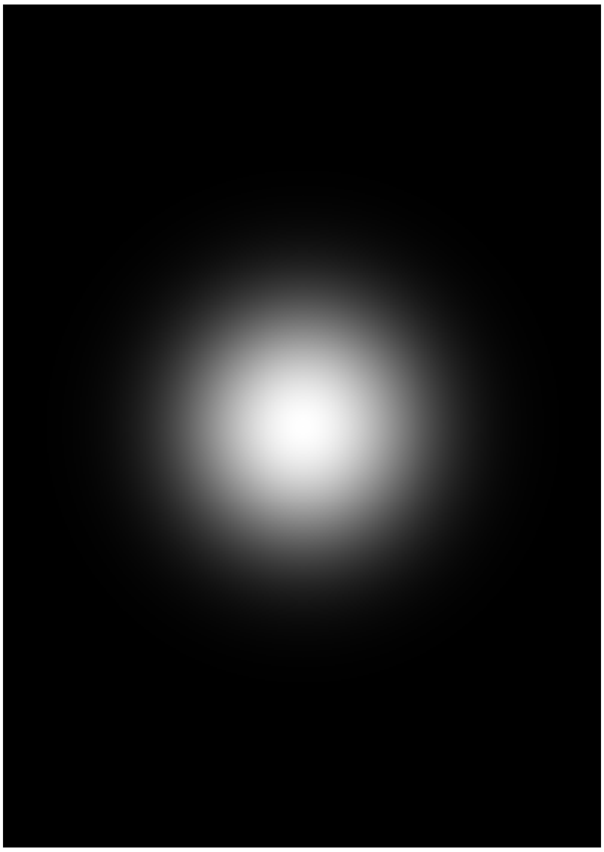
Ideal lowpass filter, Butterworth low pass filter and Gaussian low pass filter perform well.
However, Ideal lowpass filter introduces ringing effect
which is absent in the other two.
d = 50 pixels
Butterworth of order 2.

input image

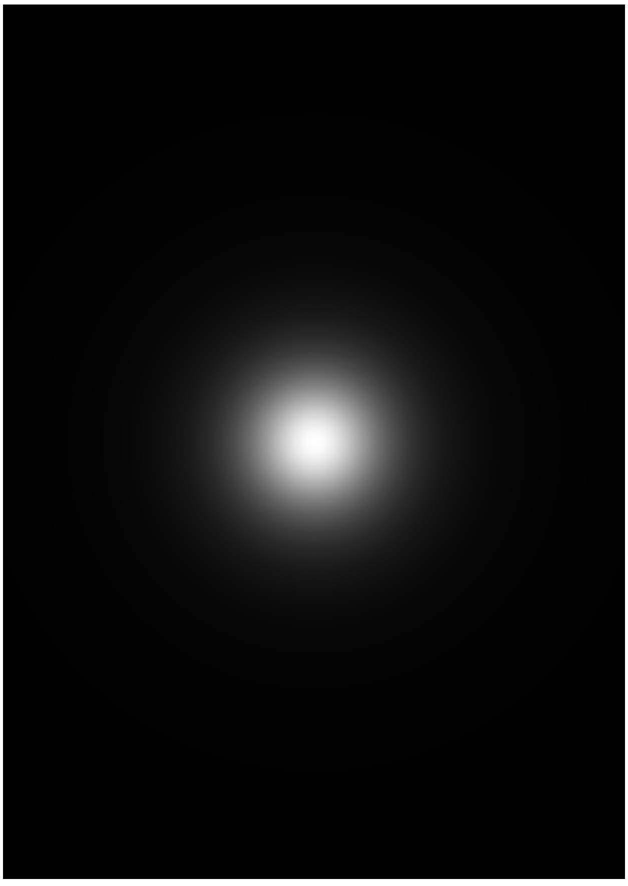
ideal low pass filter



gaussian low pass filter



butterworth low pass filter



Q 5 code

```
import cv2
import numpy as np
import math

def getDFT(img): # DFT function
	imgShape = img.shape
	imgResult = img.copy()
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*img[i,j]
	return np.fft.fft2(imgResult)

def getLogDFT(img): # DFT + log for getting DFT of image
	imgFFT = getDFT(img)
	imgFFT = np.abs(imgFFT)
	imgLogFFT = logTrans(imgFFT)

	return imgLogFFT

def IDFT(img): # IDFT function
	imgShape = img.shape
	imgResult = img.copy()
	imgIFFT = np.real(np.fft.ifft2(img))
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*imgIFFT[i,j]
	return imgResult.astype("uint8")

def logTrans(img):
	maxVal = np.amax(img) # max value in image
	cTarns = 255.0/np.log(1+maxVal) # scaling factor which sets max value to 255

	imgTrans = cTarns*(np.log(img+1.001)) # log transformation
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

def BLPF(img,d,n): #butterworth low pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using BLPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			mask[i][j] = 1/((1+(((i-imgShape[0]/2.0)**2+(j-imgShape[1]/2.0)**2)/
(d*d)))**n)
	cv2.imwrite("./outputs/q5_b.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
	# getting result by applying IDFT on DFT-result
```

```
	result = IDFT(resultDFT)

	return result

def GLPF(img,d): # gaussian low pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using GLPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			mask[i][j] = math.exp(((i-imgShape[0]/2.0)**2+(j-imgShape[1]/2.0)**2)/(-2.0*d*d))
	cv2.imwrite("./outputs/q5_g.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
	# getting result by applying IDFT on DFT-result
	result = IDFT(resultDFT)

	return result

def ILPF(img,d): # ideal low pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using ILPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			if((((i-imgShape[0]/2.0)**2+(j-imgShape[1]/2.0)**2)**0.5)<d):
				mask[i][j] = 1
	cv2.imwrite("./outputs/q5_i.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
	# getting result by applying IDFT on DFT-result
	result = IDFT(resultDFT)

	return result

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q5.png",0) # reading image using openCV
	imgILPF = ILPF(imgOriginal,50)
	imgGLPF = GLPF(imgOriginal,50)
	imgBLPF = BLPF(imgOriginal,50,2)

	cv2.imwrite("./outputs/q5_ILPF.png",imgILPF)
	cv2.imwrite("./outputs/q5_GLPF.png",imgGLPF)
	cv2.imwrite("./outputs/q5_BLPF.png",imgBLPF)
```

Q 6.

Ideal lowpass filter, Butterworth low pass filter and Gaussian low pass filter perform well.
Ideal lowpass filter introduces ringing effect
gaussian > butterworth > ideal (smoothness)
which is absent in the other two.
d = 10 pixels
Butterworth of order 2.

input image

idea high pass filter

gaussian high pass filter

butterworth low pass filter

q6 code

```
import cv2
import numpy as np
import math

def getDFT(img): # DFT function
	imgShape = img.shape
	imgResult = img.copy()
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*img[i,j]
	return np.fft.fft2(imgResult)

def getLogDFT(img): # DFT + log for getting DFT of image
	imgFFT = getDFT(img)
	imgFFT = np.abs(imgFFT)
	imgLogFFT = logTrans(imgFFT)

	return imgLogFFT

def IDFT(img): # IDFT function
	imgShape = img.shape
	imgResult = img.copy()
	imgIFFT = np.real(np.fft.ifft2(img))
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			imgResult[i,j] = ((-1)**(i+j))*imgIFFT[i,j]
	return imgResult.astype("uint8")

def logTrans(img):
	maxVal = np.amax(img) # max value in image
	cTarns = 255.0/np.log(1+maxVal) # scaling factor which sets max value to 255

	imgTrans = cTarns*(np.log(img+1.001)) # log transformation
	imgTrans = imgTrans.astype("uint8") # converting back to image format

	return imgTrans

def BHPF(img,d,n): # butterworth high pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using BHPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			xxx = ((i-imgShape[0]/2.0)**2+(j-imgShape[1]/2.0)**2)
			if xxx:
				mask[i][j] = 1/((1+((d*d)/xxx))**n)
	cv2.imwrite("./outputs/q6_b.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
```

```
	# getting result by applying IDFT on DFT-result
	result = IDFT(resultDFT)

	return result

def GHPF(img,d): # gaussian high pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using GHPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			mask[i][j] = 1.0 - math.exp(((i-imgShape[0]/2.0)**2+(j-
imgShape[1]/2.0)**2)/(-2.0*d*d))
	cv2.imwrite("./outputs/q6_g.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
	# getting result by applying IDFT on DFT-result
	result = IDFT(resultDFT)

	return result

def IHPF(img,d): # ideal high pass filter
	imgShape = img.shape
	mask = np.zeros(imgShape)
	# calculating mask using IHPF function
	for i in range(imgShape[0]):
		for j in range(imgShape[1]):
			if((((i-imgShape[0]/2.0)**2+(j-imgShape[1]/2.0)**2)**0.5)>d):
				mask[i][j] = 1
	cv2.imwrite("./outputs/q6_i.png",logTrans(mask))
	# getting DFT of image and multiplying by mask
	imgDFT = getDFT(img)
	resultDFT = np.multiply(mask,imgDFT)
	# getting result by applying IDFT on DFT-result
	result = IDFT(resultDFT)

	return result

if __name__ == "__main__":
	imgOriginal = cv2.imread("./Data/q6.png",0) # reading image using openCV
	imgIHPF = IHPF(imgOriginal,10)
	imgGHPF = GHPF(imgOriginal,10)
	imgBHPF = BHPF(imgOriginal,10,2)

	cv2.imwrite("./outputs/q6_IHPF.png",imgIHPF)
	cv2.imwrite("./outputs/q6_GHPF.png",imgGHPF)
	cv2.imwrite("./outputs/q6_BHPF.png",imgBHPF)
```