```python
# -*- coding: utf-8 -*-
"""Frequency_Domain_Sara_Ghavampour_9812762781.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/11zSfG3WFScbFX2w5zWnQRZ2H7qQ7yGYg
"""

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import cv2
from math import *
from sklearn.metrics import mean_squared_error
# %matplotlib inline

def show_img(*args, figsize=10, is_gray=True, title=None, fontsize=12):
    if isinstance(figsize, int):
        figsize = (figsize, figsize)
    images = args[0] if type(args[0]) is list else list(args)
    cmap=None
    if not is_gray:
        images = list(map(lambda x: cv2.cvtColor(x, cv2.COLOR_BGR2RGB), images))
    else:
        cmap = 'gray'
    plt.figure(figsize=figsize)
    for i in range(1, len(images)+1):
        plt.subplot(1, len(images), i)
        if title is not None:
            plt.title(title[i-1], fontsize=fontsize)

        plt.imshow(images[i-1], cmap=cmap)
        plt.axis('off')

lena_img=cv2.imread('Lena.bmp')
lena_img = cv2.cvtColor(lena_img,cv2.COLOR_BGR2GRAY)
plt.imshow(lena_img,cmap='gray')
plt.axis('off')
plt.show()

"""4.1. fourier transform

4.1.1
"""

###------------------------- 4.1.1

def pad_before_fft(img,pad_size):
  padded_img = np.zeros((2*pad_size,2*pad_size))
  padded_img[0:img.shape[0],0:img.shape[1]] = img
  return padded_img

show_img(pad_before_fft(lena_img,512))

"""plotting magnitude"""

def fourier_magnitude(x,pad_size):
  x_pad=pad_before_fft(x,pad_size)
  fft=np.fft.fftshift(np.fft.fft2(x_pad))
  return np.abs(fft)

filter_a=np.array([[1,2,1],[2,4,2],[1,2,1]])*(1/16)

show_img(fourier_magnitude(filter_a,512))

filter_a_1 = (1/4)*np.array([[1],[2],[1]])
show_img(fourier_magnitude(filter_a_1,512))

filter_a_1

filter_a_2 = (1/4)*np.array([[1,2,1]])
show_img(fourier_magnitude(filter_a_2,512))

filter_b=np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
show_img(fourier_magnitude(filter_b,512))

filter_c = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
show_img(fourier_magnitude(filter_c,512))

"""apply filters in frequency domain"""

def filter_frequency_domain(img,filter):
  img_pad = pad_before_fft(img,img.shape[0])
  x_fft=np.fft.fftshift(np.fft.fft2(img_pad))

  filter_pad = pad_before_fft(filter,img.shape[0])
  filter_fft=np.fft.fftshift(np.fft.fft2(filter_pad))

  out = x_fft * filter_fft
  # return np.fft.ifft2(np.fft.ifftshift(out))
  return abs(np.fft.ifft2(np.fft.ifftshift(out)))[0:img.shape[0],0:img.shape[1]]

freq_filtered_a_1 = filter_frequency_domain(lena_img,filter_a_1)
show_img(lena_img,freq_filtered_a_1,title=['original image','apply filter a_1 on image'])

freq_filtered_a_2 = filter_frequency_domain(lena_img,filter_a_2)
show_img(lena_img,freq_filtered_a_2,title=['original image','apply filter a_2 on image'])

freq_filtered_a = filter_frequency_domain(lena_img,filter_a)
show_img(lena_img,freq_filtered_a,title=['original image','apply filter a on image'])

freq_filtered_b = filter_frequency_domain(lena_img,filter_b)
show_img(lena_img,freq_filtered_b,title=['original image','apply filter b on image'])

freq_filtered_c = filter_frequency_domain(lena_img,filter_c)
show_img(lena_img,freq_filtered_c,title=['original image','apply filter c on image'])

"""4.1.2"""

###----------------- 4.1.2
```

```
def log_transform(img):  # s = c log(r+1)   # 255/log(r_max +1)
  r_max = np.max(img)
  c = 255/log(r_max +1)
  out = img.copy()
  out = c * np.log(img+1)
  return out.astype('int32')

def Q_4_1_2(x):
  mag=abs(np.fft.fft2(x)) # without shift- without log
  mag_shifted=abs(np.fft.fftshift(np.fft.fft2(x))) # with shift- without log
  mag_log=log_transform(abs(np.fft.fft2(x))) # without shift- with log
  mag_log_shifted=log_transform(abs(np.fft.fftshift(np.fft.fft2(x)))) # with shift- with log
  return mag,mag_shifted,mag_log,mag_log_shifted

plt.imshow(lena_img,cmap='gray')
plt.axis('off')
plt.show()

# 4.1..2 on lena
lena_mag,lena_mag_shifted,lena_mag_log,lena_mag_log_shifted=Q_4_1_2(lena_img)#
show_img(lena_mag,lena_mag_shifted,lena_mag_log,lena_mag_log_shifted,figsize=35)

barbara_img=cv2.imread('Barbara.bmp')
barbara_img = cv2.cvtColor(barbara_img,cv2.COLOR_BGR2GRAY)
plt.imshow(barbara_img,cmap='gray')
plt.axis('off')
plt.show()

# 4.1..2 on barbara
barbara_mag,barbara_mag_shifted,barbara_mag_log,barbara_mag_log_shifted=Q_4_1_2(barbara_img)
show_img(barbara_mag,barbara_mag_shifted,barbara_mag_log,barbara_mag_log_shifted,figsize=35)

F16_img=cv2.imread('F16.bmp')
F16_img = cv2.cvtColor(F16_img,cv2.COLOR_BGR2GRAY)
plt.imshow(F16_img,cmap='gray')
plt.axis('off')
plt.show()

# 4.1..2 on f16
F16_mag,F16_mag_shifted,F16_mag_log,F16_mag_log_shifted=Q_4_1_2(F16_img)
show_img(F16_mag,F16_mag_shifted,F16_mag_log,F16_mag_log_shifted,figsize=35)

Baboon_img=cv2.imread('Baboon.bmp')
Baboon_img = cv2.cvtColor(Baboon_img,cv2.COLOR_BGR2GRAY)
plt.imshow(Baboon_img,cmap='gray')
plt.axis('off')
plt.show()

# 4.1..2 on Baboon
Baboon_mag,Baboon_mag_shifted,Baboon_mag_log,Baboon_mag_log_shifted=Q_4_1_2(Baboon_img)
show_img(Baboon_mag,Baboon_mag_shifted,Baboon_mag_log,Baboon_mag_log_shifted,figsize=35)

"""4.2

4.2.2
"""

#---------------------- 4.2.2

def normalize(img):
  min = np.min(img)
  max = np.max(img)
  return ((img-min)/(max-min) * 255).astype('uint8')

#----------------- part a

def Q_4_2_2_a(img,T):
  n = img.shape[0]
  dft = np.fft.fft2(img)
  filter_dim = int((1-T)*n) - int(T*n)
  dft[int(T*n):int((1-T)*n) ,int(T*n):int((1-T)*n) ] = np.zeros((filter_dim,filter_dim))
  out = np.abs(np.fft.ifft2(dft))
  out_freq= np.log(np.abs(dft))
  out_freq[out_freq == -inf] = 0
  return normalize(out),out_freq

# filter a , t = 1/4
temp,dft = Q_4_2_2_a(barbara_img,1/4)
show_img(dft,temp)

# filter a , t = 1/8
temp,dft = Q_4_2_2_a(barbara_img,1/8)
show_img(dft,temp)

#----------------- part b

def Q_4_2_2_b(img,T):
  n = img.shape[0]
  dft = np.fft.fft2(img)
  tn = int(T*n)
  t_1_n = int((1-T)*n)
  dft[0:tn ,0:tn] = np.zeros((tn,tn))
  dft[0:tn ,t_1_n:n] = np.zeros((tn,tn))
  dft[t_1_n:n ,0:tn] = np.zeros((tn,tn))
  dft[t_1_n:n ,t_1_n:n] = np.zeros((tn,tn))
  out = np.abs(np.fft.ifft2(dft))
  out_freq= np.log(np.abs(dft))
  out_freq[out_freq == -inf] = 0
  return normalize(out),out_freq

# filter b , t = 1/4
temp,dft = Q_4_2_2_b(barbara_img,1/4)
show_img(dft,temp)

# filter b , t = 1/8
temp,dft = Q_4_2_2_b(barbara_img,1/8)
show_img(dft,temp)
```