```python
# -*- coding: utf-8 -*-
"""cv_hw5_Wavelet_Sara_Ghavampour_9812762781.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vJiNYXnx0KtpHeGpGM5v_gBCbghz1pYH
"""

!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
import pywt
# %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()
#(512, 512) uint8

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

"""5

5.1. Pyramid

5.1.1.
"""

############### 5.1.1 ###############

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

def remove_row_col_downsampling(img,factor):
  return img[::factor,::factor]

def guassian_pyramid(img,kernel,level,q1=True):
  prev_level = img
  guass_array=[]
  guass_array.append(prev_level)
  out = np.zeros((img.shape[0],int(1.5*img.shape[1])))

  for i in range(0,level):
    filtered_img=cv2.filter2D(src=prev_level, ddepth=-1, kernel=kernel)
    next_level=remove_row_col_downsampling(filtered_img,2)
    guass_array.append(next_level)
    prev_level = next_level
  if q1==True:
    # put different levels of quassian pyramid in shape of pyramid

    #put level 0
    out[0:guass_array[0].shape[0],0:guass_array[0].shape[1]]=guass_array[0]

    #put level 1
    out[0:guass_array[1].shape[0],guass_array[0].shape[1]-1:guass_array[0].shape[1]-1+guass_array[1].shape[1]]=guass_array[1]

    #put level 2
    out[guass_array[1].shape[0]-1:guass_array[1].shape[0]-1+guass_array[2].shape[0],guass_array[0].shape[1]-1:guass_array[0].shape[1]-1+guass_array[2].shape[1]]=guass_array[2]

    #put level 3
    row=guass_array[1].shape[0]+guass_array[2].shape[0]-1
    col=guass_array[0].shape[1]
    out[row-1:row-1+guass_array[3].shape[0],col-1:col-1+guass_array[3].shape[1]]=guass_array[3]

    #put level 4
    row=row-1+guass_array[3].shape[0]
    out[row-1:row-1+guass_array[4].shape[0],col-1:col-1+guass_array[4].shape[1]]=guass_array[4]

    #put level 5
    row=row-1+guass_array[4].shape[0]
    out[row-1:row-1+guass_array[5].shape[0],col-1:col-1+guass_array[5].shape[1]]=guass_array[5]

  # if q1== False:
  #   #put level 5
  #   for j in range(6,level+1):
  #     row=row-1+guass_array[j-1].shape[0]
  #     out[row-1:row-1+guass_array[j-1].shape[0],col-1:col-1+guass_array[j].shape[1]]=guass_array[j]


  return out,guass_array

guassian_pyramid , guass_array=guassian_pyramid(monalisa_img,guassian_kernel,5)
show_img(guassian_pyramid,title=['Five level Guassian Pyramid'],figsize=13)

type(guass_array)

def unsample_px_replication(img, factor):
    out = np.repeat(img, factor, axis=0)
    out = np.repeat(out, factor, axis=1)
    return out

def pad_even(img):
    shape_0 = img.shape[0]
    shape_1 = img.shape[1]

    if(img.shape[0]%2):
        shape_0 = img.shape[0]+1
    if(img.shape[1]%2):
        shape_1 = img.shape[1]+1

    out = np.zeros((shape_0, shape_1))
    out[0:img.shape[0], 0:img.shape[1]] = img
    return out

def laplacian_pyramid(guass_array,level,q1=True):
  img=guass_array[0]
  laplacian_array=[]
  out = pad_even(np.zeros((img.shape[0],int(1.5*img.shape[1]))))

  for i in range(0,level):
    upsample = (unsample_px_replication(guass_array[i+1],2))
    laplacian_array.append(pad_even(guass_array[i])-upsample)
  laplacian_array.append((guass_array)[level])

  # put different levels of laplacian pyramid in shape of pyramid

  if q1==True:
    #put level 0
    out[:,0:laplacian_array[0].shape[1]]=laplacian_array[0]

    #put level 1
    out[0:laplacian_array[1].shape[0],laplacian_array[0].shape[1]-2:laplacian_array[0].shape[1]-2+laplacian_array[1].shape[1]]=laplacian_array[1]

    #put level 2
    out[laplacian_array[1].shape[0]-1:laplacian_array[1].shape[0]-1+laplacian_array[2].shape[0],laplacian_array[0].shape[1]-1:laplacian_array[0].shape[1]-1+laplacian_array[2].shape[1

    #put level 3
    row=laplacian_array[1].shape[0]+laplacian_array[2].shape[0]-1
    col=laplacian_array[0].shape[1]
    out[row-1:row-1+laplacian_array[3].shape[0],col-1:col-1+laplacian_array[3].shape[1]]=laplacian_array[3]

    #put level 4
    row=row-1+laplacian_array[3].shape[0]
    out[row-1:row-1+laplacian_array[4].shape[0],col-1:col-1+laplacian_array[4].shape[1]]=laplacian_array[4]

    #put level 5
    row=row-1+laplacian_array[4].shape[0]
    out[row-1:row-1+laplacian_array[5].shape[0],col-1:col-1+laplacian_array[5].shape[1]]=laplacian_array[5]


  return out,laplacian_array

laplacian_pyramid,laplacian_array = laplacian_pyramid(guass_array,5)
show_img(laplacian_array,title=['level 0','level 1','level 2','level 3','level 4','level 5'],figsize=22)

show_img(laplacian_pyramid,title=['Five level Laplacian Pyramid'],figsize=13)

for i,p in enumerate(laplacian_array):
  print(i,' ',p.shape)

"""5.1.3."""

################ 5.1.3 ################

lena_img.shape   #512 = 2^9 --> j=9 ---> j+1 level in pyramids
lena_j=9

################ biulding guassian_pyramid and residual pyramid ################

_ , lena_guass_array=guassian_pyramid(lena_img,guassian_kernel,9,q1=False)
show_img(lena_guass_array,figsize=30)

_,lena_laplacian_array = laplacian_pyramid(lena_guass_array,lena_j,q1=False)
show_img(lena_laplacian_array,figsize=30)

################ reconstructing original lena image using guassian_pyramid and residual pyramid ################

def reconstruct_(guass_array,laplacin_array):
  levels= np.flip([i for i in range(0,len(guass_array))])
  constructed_array  = []
  out = (guass_array[-1])
  for i in levels:
    if i ==0 : continue
    upscale = unsample_px_replication((guass_array[i]),2)
    out = (upscale) + (laplacin_array[i-1])

  return out

reconstructed_lena = reconstruct_(lena_guass_array,lena_laplacian_array)
show_img(reconstructed_lena)

"""5.1.4. """

######## 5.1.4 ##############

box_filter_kernel = (1/9) * np.array([[1,1,1],[1,1,1],[1,1,1]])
box_filter_kernel

def boxfilter_pyramid(img,kernel,level,q1=True):
  prev_level = img
  boxfilter_array=[]
  boxfilter_array.append(prev_level)
  out = np.zeros((img.shape[0],int(1.5*img.shape[1])))

  for i in range(0,level):
    filtered_img=cv2.filter2D(src=prev_level, ddepth=-1, kernel=kernel)
    next_level=remove_row_col_downsampling(filtered_img,2)
    boxfilter_array.append(next_level)
    prev_level = next_level
  if q1==True:
    # put different levels of quassian pyramid in shape of pyramid

    #put level 0
    out[0:boxfilter_array[0].shape[0],0:boxfilter_array[0].shape[1]]=boxfilter_array[0]

    #put level 1
    out[0:boxfilter_array[1].shape[0],boxfilter_array[0].shape[1]-1:boxfilter_array[0].shape[1]-1+boxfilter_array[1].shape[1]]=boxfilter_array[1]
    #put level 2
    out[boxfilter_array[1].shape[0]-1:boxfilter_array[1].shape[0]-1+boxfilter_array[2].shape[0],boxfilter_array[0].shape[1]-1:boxfilter_array[0].shape[1]-1+boxfilter_array[2].shape[1

    #put level 3
    row=boxfilter_array[1].shape[0]+boxfilter_array[2].shape[0]-1
    col=boxfilter_array[0].shape[1]
    out[row-1:row-1+boxfilter_array[3].shape[0],col-1:col-1+boxfilter_array[3].shape[1]]=boxfilter_array[3]



  return out,boxfilter_array

boxfilter_pyramid , boxfilter_array=boxfilter_pyramid(lena_img,box_filter_kernel,3)
show_img(boxfilter_pyramid,title=['Five level boxfilter Pyramid'],figsize=13)

def boxfilter_laplacian_pyramid(boxfilter_array,level,q1=True):
  img=boxfilter_array[0]
  laplacian_array=[]
  out = pad_even(np.zeros((img.shape[0],int(1.5*img.shape[1]))))

  for i in range(0,level):
    upsample = (unsample_px_replication(boxfilter_array[i+1],2))
    laplacian_array.append(pad_even(boxfilter_array[i])-upsample)
  laplacian_array.append((boxfilter_array)[level])

  # put different levels of laplacian pyramid in shape of pyramid

  if q1==True:
    #put level 0
    out[:,0:laplacian_array[0].shape[1]]=laplacian_array[0]

    #put level 1
    out[0:laplacian_array[1].shape[0],laplacian_array[0].shape[1]-2:laplacian_array[0].shape[1]-2+laplacian_array[1].shape[1]]=laplacian_array[1]

    #put level 2
    out[laplacian_array[1].shape[0]-1:laplacian_array[1].shape[0]-1+laplacian_array[2].shape[0],laplacian_array[0].shape[1]-1:laplacian_array[0].shape[1]-1+laplacian_array[2].shape[1

    #put level 3
    row=laplacian_array[1].shape[0]+laplacian_array[2].shape[0]-1
    col=laplacian_array[0].shape[1]
    out[row-1:row-1+laplacian_array[3].shape[0],col-1:col-1+laplacian_array[3].shape[1]]=laplacian_array[3]

    # #put level 4
    # row=row-1+laplacian_array[3].shape[0]
    # out[row-1:row-1+laplacian_array[4].shape[0],col-1:col-1+laplacian_array[4].shape[1]]=laplacian_array[4]

    # #put level 5
    # row=row-1+laplacian_array[4].shape[0]
    # out[row-1:row-1+laplacian_array[5].shape[0],col-1:col-1+laplacian_array[5].shape[1]]=laplacian_array[5]


  return out,laplacian_array

laplacian_pyramid,laplacian_boxfilter_array = boxfilter_laplacian_pyramid(boxfilter_array,3)
show_img(laplacian_boxfilter_array,title=['level 0','level 1','level 2','level 3'],figsize=22)

show_img(laplacian_pyramid,title=['Three level Laplacian of boxfilter Pyramid'],figsize=13)

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

def wawelet_transform(img,levels):
  wt_out  = np.zeros((img.shape[0],img.shape[1]))
  appriximations= np.zeros((img.shape[0],int(1.5*img.shape[1])))

  appriximations[:,0:img.shape[1]]=img
  img_col = img.shape[1]
  # wavelet transform
  coeffs = pywt.wavedec2(img, 'haar', mode='periodization', level=levels)

  # Put coefficients in a matrix
  c_matrix, c_slices = pywt.coeffs_to_array(coeffs)

  ##  get approximations
  row_pointer = 0
  for i in range(0, levels):
    cA, _ = pywt.dwt2(img, 'haar', mode='periodization')
    print(cA.shape)
    appriximations[row_pointer:row_pointer+cA.shape[0],img_col:img_col+cA.shape[1]] = normalize(cA)
    row_pointer+=cA.shape[0]
    img=cA


  # return wt_out,appriximations_wrap
  return c_matrix,coeffs,appriximations

wt_out,coeffs,appriximations_wrap = wawelet_transform(lena_img,3)
show_img(wt_out)

show_img(appriximations_wrap)

"""5.1.6."""

(coeffs[1])

######## 5.1.6 ###########

def coefficiant_quantize(coeffs,step_size):
  length = len(coeffs)
  new_coeff = []
  new_coeff.append(step_size * np.sign(coeffs[0]) * np.floor(np.abs(coeffs[0])/step_size))
  for i in range(1,length):
    h,v,d = coeffs[i]
    new_h = step_size * np.sign(h) * np.floor(np.abs(h)/step_size)
    new_v = step_size * np.sign(v) * np.floor(np.abs(v)/step_size)
    new_d = step_size * np.sign(d) * np.floor(np.abs(d)/step_size)
    new_coeff.append((new_h,new_v,new_d))

  return new_coeff

new_coeffs = coefficiant_quantize(coeffs,2)
len(new_coeffs[0:2])

def reconstruct_wawelet_coeff(new_coeffs):
  length = len(new_coeffs)
  reconstructed_img = pywt.idwt2(new_coeffs[0:2], 'haar', mode='periodization')
  for i in range(2,length):
    reconstructed_img = pywt.idwt2([reconstructed_img,new_coeffs[i]], 'haar', mode='periodization')
  return reconstructed_img

reconstructed_img = reconstruct_wawelet_coeff(new_coeffs)
reconstructed_img

show_img(reconstructed_img)

psnr = cv2.PSNR(lena_img,reconstructed_img.astype('uint8'))
psnr
```