ca03

October 28, 2024

1 COMPUTER ASSIGNMENT 03

1.1 PART 1

[22]:
```
!pip install PyWavelets

import numpy as np
import scipy
import math
import matplotlib.pyplot as plt
import cv2
import pywt
%matplotlib inline
```

```
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-
packages (1.7.0)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.10/dist-
packages (from PyWavelets) (1.26.4)
```

1.1.1 PART 1

[23]:
```
# generate the Gaussian and Laplacian pyramids for the image
# INPUT: image
# OUTPUT:  Gaussian and Laplacian pyramids
def pyramid(img):

    gp = []  # Store the Gaussian pyramid
    lp = []  # Store the Laplacian pyramid
    ##########################################    [TODO]   
 ↪##########################################
    # Use cv2.resize() to get the Gaussian Pyramid
    y, x = img.shape
    g1 = img # bottom level
    gp.append(g1)
    g2 = cv2.resize(g1,(x//2,y//2), fx=0, fy=0, interpolation=cv2.INTER_LINEAR)
 ↪      # middle level
    gp.append(g2)
```

```
    g3 = cv2.resize(g2,(x//4,y//4), fx=0, fy=0, interpolation=cv2.INTER_LINEAR)␣
↪       # top level
    gp.append(g3)
    ##########################################    [TODO]   ␣
↪##########################################
    # Get the Laplacian Pyramid correspondingly
    l1 = g1 - cv2.resize(g2, (x,y), fx=0, fy=0, interpolation=cv2.INTER_CUBIC) ␣
↪      # bottom level
    lp.append(l1)
    l2 = g2 - cv2.resize(g3, (x//2,y//2), fx=0, fy=0, interpolation=cv2.
↪INTER_CUBIC)       # middle level
    lp.append(l2)
    l3 = g3       # top level
    lp.append(l3)
    # RETURN GAUSSIAN AND LAPLACIAN PYRAMID
    return gp, lp
```

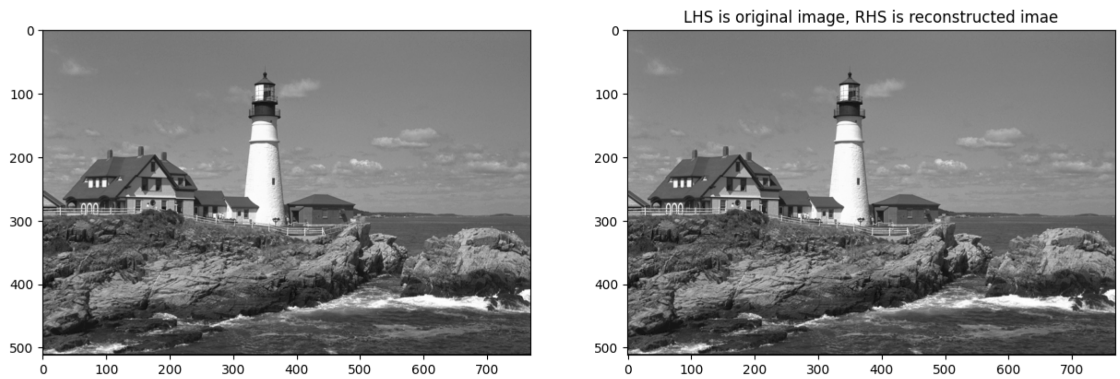
[24]:
```
# reconstruct the image from Laplacian Pyramid
# INPUT: Laplacian pyramid
# OUTPUT:  reconstructed image
def reconstruct_lp(lp):
    ##########################################    [TODO]   ␣
↪##########################################
    # reconstruct the image from the Laplacian pyramid
    # Use cv2.resize() to upsample the pyramid
    y, x = lp[0].shape
    reconstructed_g2 = lp[1] +  cv2.resize(lp[2], (x//2,y//2), fx=0, fy=0,␣
↪interpolation=cv2.INTER_CUBIC)
    reconstructed_g1 = lp[0] +  cv2.resize(reconstructed_g2, (x,y), fx=0, fy=0,␣
↪interpolation=cv2.INTER_CUBIC)
    reconstructed_img = reconstructed_g1
    # Return the reconstructed image
    return reconstructed_img
```

[25]:
```
# Perform quantization to the image with quantization step Q
# INPUT: image name and quantization step Q
# OUTPUT: reconstructed image after quantization, number of non-zeros in␣
↪Laplacian pyramid, PSNR
def quantize(image, Q):
    # Generate the Gaussian and Laplacian pyramids using your pyramid() function
    gp, lp = pyramid(image)
    num = 0 # Number of non-zeros of the quantized Laplacian pyramid
    ##########################################    [TODO]   ␣
↪##########################################
    # Quantize the pixel values in the Laplacian pyramid with a quantization␣
↪step Q
```

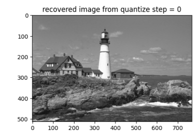
```
    # Assume that for the top level, mean = 128. For the other levels, mean = 0
    # And count the number of non-zero pixels in the pyramid after quantization
    # You can use np.count_nonzero() to count the number of non-zero elements␣
↪in an array
    # i.e., num = np.count_nonzero(input_array)
    mean = [0, 0, 128]
    quantized_lp = lp
    quantized_lp[2] = np.floor((lp[2]-mean[2]+Q/2)/Q)*Q + mean[2]
    quantized_lp[1] = np.floor((lp[1]-mean[1]+Q/2)/Q)*Q + mean[1]
    quantized_lp[0] = np.floor((lp[0]-mean[0]+Q/2)/Q)*Q + mean[0]
    num = np.count_nonzero(quantized_lp[0])+np.
↪count_nonzero(quantized_lp[1])+np.count_nonzero(quantized_lp[2])

    ##########################################    [TODO]   ␣
↪##########################################
    # Reconstruct the image with the quantized Laplacian Pyramid
    reconstructed_image_quant = reconstruct_lp(quantized_lp)

    ##########################################    [TODO]   ␣
↪##########################################
    # Calculate MSE and PSNR of the reconstructed image
    mse = np.mean((reconstructed_image_quant-image)**2)
    psnr = 10*np.log10(256**2/mse)

    # RETURN RECONSTRUCTED IMAGE, NUMBER OF NON-ZEROS AND PSNR
    return reconstructed_image_quant, num, psnr
```

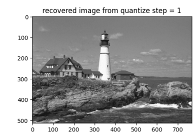
[26]:

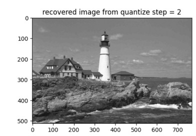
```
# Test your pyramid() and reconstruct_lp() here
##########################################    [TODO]   ␣
↪##########################################
# Read in an image
file_name = '/content/lighthouse.png'
image = cv2.imread(file_name,0)
if image is None:
    print("Error: Image file not found or unable to load.")
else:
    image = image.astype(float)
# Generate the Gaussian and Laplacian pyramid of the image by using pyramid()
gp, lp = pyramid(image)
# Reconstruct the image from Laplacian pyramid
rec_img = reconstruct_lp(lp)
# Plot the reconstructed image and the original image
_,ax = plt.subplots(1,2,figsize=(15,15))
ax[0].imshow(image,cmap='gray')
ax[1].imshow(rec_img,cmap='gray')
plt.title('LHS is original image, RHS is reconstructed imae')
plt.show()
```

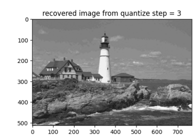


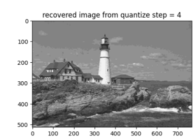
[27]:
```python
# Test your quantize() function and plot the results here
##########################################     [TODO]    ##########################################
# Read in an image
file_name = '/content/lighthouse.png'
image = cv2.imread(file_name,0).astype(float)
n = 10     # The level of quantization ( Q = 1, 2, 4, ... , 2^(n-1) )
rec_image_quant = np.zeros([n, image.shape[0], image.shape[1]]) # Reconstructed images array
num = np.zeros(n)  # Number of non-zeros array
psnr = np.zeros(n) # PSNR array
Q = np.zeros(n)    # Q array
# Use quantize() to quantize the Laplacian pyramid of the image with Q = 1, 2, 4, ... , 2^(n-1)
_,ax = plt.subplots(n+1,1,figsize=(5, 60))

for i in range(n):
    Q[i] = 2**i
    rec_image_quant[i], num[i], psnr[i] = quantize(image,Q[i])
    ax[i].imshow(rec_image_quant[i],cmap='gray')
    ax[i].set_title('recovered image from quantize step = ' + str(i))
# Plot the reconstructed images correspondingly .
# Plot the curve of psnr vs number of non-zeros
ax[n].plot(psnr, num)
plt.show()
```

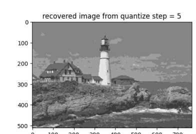

recovered image from quantize step = 0

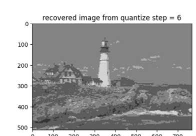

recovered image from quantize step = 1

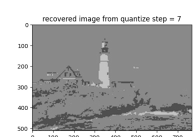

recovered image from quantize step = 2

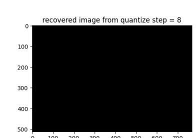

recovered image from quantize step = 3

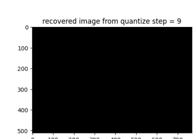

recovered image from quantize step = 4

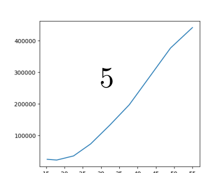

recovered image from quantize step = 5


recovered image from quantize step = 6


recovered image from quantize step = 7


recovered image from quantize step = 8


recovered image from quantize step = 9



1.2 PART 2

-

[28]:

```
# Generate the wavelet coefficients for the image
# INPUT : 2D NP ARRAY REPRESENTING THE IMAGE
# OUTPUT : WAVELET COEFFICIENTS

# NOTE: THE WAVELET COEFFICIENTS SHOULD BE RETURNED IN THE FORM OF A LIST SHOWN␣
↪BELOW
# [[SMALL SCALE REPRESENTATITON OF IMAGE,[LIST OF LEN 3 LEVEL 3 REPRESENTATION␣
↪OF IMAGE]],[LIST OF LEN 2 LEVEL 2 COEFFICIENTS],[LIST OF LEN 3 LEVEL 1␣
↪COEFFICIENTS]]
def gen_wavelet(img):
    levels = 3
    img_wavlets = [None]*levels
    temp = img
    # padding zeros to make the dimentions even
    for i in range(levels-1,-1,-1):
        r,c = temp.shape
        if c%2 != 0:
            temp = np.hstack((temp,np.zeros((r,1))))
            c += 1
        if r%2 != 0:
            temp = np.vstack((temp,np.zeros((1,c))))
            r += 1
        ##########################################    [TODO]   ␣
↪##########################################
        # Similarly do it for rows
        ##########################################    [TODO]   ␣
↪##########################################
        # call the dwt2 function to generate wavelet coefficients
        temp, (B,C,D) = pywt.dwt2(temp, wavelet='db1', mode='symmetric')
        ##########################################    [TODO]   ␣
↪##########################################
        # Store B,C,D in the structure shown above
        img_wavlets[i] = [B,C,D]
    img_wavlets[0] = [temp,img_wavlets[0]]
    return img_wavlets
```

[29]:

```
# Function used purely to display the coefficients of the wavelet transform
# Stack all the wavelet subimages together in the shape of the original image.
def wv_stack(ls):
    for i in range(len(ls)):
        if i == 0:
```

```
            A,(B,C,D) = ls[i]
            H1 = np.hstack((A,B))
            H2 = np.hstack((C,D))
            temp = np.vstack((H1,H2))
        else:
            B,C,D = ls[i]
            r,c = temp.shape
            r1,c1 = B.shape
            if r1 != r:
                temp = temp[:-1,:]
            if c1 != c:
                temp = temp[:,:-1]
            H1 = np.hstack((temp,B))
            H2 = np.hstack((C,D))
            temp = np.vstack((H1,H2))
    return temp
```

[30]:

```
# reconstruct the image from the wavelet coeffcients
# INPUT: WAVELET COEFFICIENTS IN THE FORM AS MENTIONED IN THE GEN_WAVELET␣
 ↪FUNCTION
# OUTPUT: RECONSTRUCTED IMAGE
def gen_iwavelet(coeff):
    temp = pywt.idwt2(coeff[0],'db1','symmetric')
    for lv in coeff[1:]:
        ##########################################    [TODO]    ␣
 ↪##########################################
        # Write code segment to make sure the image reconstruction(temp) is of␣
 ↪the same size as the coefficients on one higher level
        # The temp variable generated abpve may not have the same dimension as␣
 ↪the higher level wavelet coefficients.
        # run an if statement to verify if temp.shape == lv.shape

        if temp.shape[0] != lv[0].shape[0]:
            temp = temp[0:-1,:]
        if temp.shape[1] != lv[0].shape[1]:
            temp = temp[:,0:-1]

        temp = (temp,lv) # pywt.idwt2() takes in coefficients in the form of␣
 ↪[A,(B,C,D)]
        ##########################################    [TODO]    ␣
 ↪##########################################
        # Perform inverse wavelet transform
        temp = pywt.idwt2(temp,'db1','symmetric')
    return temp
```

[31]:
```python
# function to calculate the psnr
# input : #1- noisy image , #2- original image as reference
# OUTPUT: PSNR value
def psnr(img,ref):
    ##########################################    [TODO]   
 ↪##########################################
    # Write the code to find the psnr
    mse = np.mean((img-ref)**2)
    psnr = 10*np.log10(256**2/mse)
    return psnr
```

[32]:
```python
def Q_wavelet(img,Q): # image,Quantization step
    ##########################################    [TODO]   
 ↪##########################################
    # generate teh wavelet coefficnets using gen_wavelet
    coeff = gen_wavelet(img)
    levels = 3
    for i in range(levels):
        if i == 0:
            mean = 128
            ##########################################    [TODO]   
 ↪##########################################
            # perform quantization using the formula provided
            # quantized_lp[2] = np.floor((lp[2]-mean[2]+Q/2)/Q)*Q + mean[2]
            coeff[i][0] = np.floor((coeff[i][0]-mean+Q/2)/Q)*Q + mean
            mean = 0
            for j in range(levels):
                ##########################################    [TODO]   
 ↪##########################################
                # perform quantization using the formula provided
                coeff[i][1][j] = np.floor((coeff[i][1][j]-mean+Q/2)/Q)*Q + mean
        else:
            mean = 0
            ##########################################    [TODO]   
 ↪##########################################
            # perform quantization using the formula provided
            for j in range(3):
                coeff[i][j] = np.floor((coeff[i][j]-mean+Q/2)/Q)*Q + mean

    # finished quantization
    # count number of zero values

    nz_ele = 0
    ##########################################    [TODO]   
 ↪##########################################
    # Write code segment to count number of non-zero values. Use np.
 ↪count_nonzero function
```

```
    coeff_stacked = wv_stack(coeff)
    nz_ele =  np.count_nonzero(coeff_stacked)

    ##########################################    [TODO]   ␣
↪##########################################
    # Reconstruct the image
    img_rec = gen_iwavelet(coeff)

    ##########################################    [TODO]   ␣
↪##########################################
    # Make sure the reconstructed image and original are of the same shape

    if img_rec.shape[0] != img.shape[0]:
        img_rec = img_rec[0:-1,:]
    if img_rec.shape[1] != img.shape[1]:
        img_rec = img_rec[:,0:-1]

    ##########################################    [TODO]   ␣
↪##########################################
    # Calculate the psnr
    psnr_ = psnr(img, img_rec)
    return psnr_, nz_ele,img_rec
```

[33]:

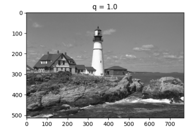
```
# Main function for part 2
def main2(filename):
    ##########################################    [TODO]   ␣
↪##########################################
    # Read in the image
    img = cv2.imread(file_name,0).astype(float)

    ##########################################    [TODO]   ␣
↪##########################################
    # Generate wavelet coefficients and plot the coefficients in standard form␣
↪using wv_stack function
    img_wavlets = gen_wavelet(img)
    stacked_wavlets = wv_stack(img_wavlets)

    ##########################################    [TODO]   ␣
↪##########################################
    # Reconstruct the image using gen_iwavelet function and display output
    img_reconstruct = gen_iwavelet(img_wavlets)

    plt.show()

    ##########################################    [TODO]   ␣
↪##########################################
```

```
    # Set quantization levels
    Q = np.logspace(0, 9, num=10, base=2) # array saving quantization values
    print('Q:',Q)
    _,ax = plt.subplots(len(Q)+4,1,figsize=(5,60))
    ax[len(Q)].imshow(stacked_wavlets,cmap='gray')
    ax[len(Q)+1].imshow(img_reconstruct,cmap='gray')

    Q_imgs = [None]*len(Q) # list to hold quantized images
    psnr_arr = [None]*len(Q) # list to hold the psnr values
    nz_arr = [None]*len(Q) # list to hold `non-zero element` values for␣
↪different quantization levels

    ##########################################    [TODO]   ␣
↪##########################################
    # Write code segment to calculate the wavelet transform for the image,␣
↪quantize with different Q,
    # Add the values to the Q_imgs,psnr_arr,nz_arr lists accordingly
    for i in range(len(Q)):
        psnr_arr[i], nz_arr[i], Q_imgs[i] = Q_wavelet(img, Q[i])

    ##########################################    [TODO]   ␣
↪##########################################
    # plot Q vs non-zero piexl count
    # plot non-zero pixel count vs psnr
    ax[len(Q)+2].plot(np.arange(10),nz_arr)
    ax[len(Q)+3].plot(psnr_arr, nz_arr)

    # PLotting all figures side by side with original to show the difference
    for i in range(len(Q)):
        ax[i].imshow(Q_imgs[i],cmap='gray')
        ax[i].set_title('q = {}'.format(Q[i]))
        pass
    plt.show()
```

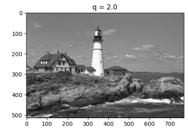
[34]:
```
# Call the main function
file_name = '/content/lighthouse.png'
main2(file_name)
```

```
Q: [  1.   2.   4.   8.  16.  32.  64. 128. 256. 512.]
```

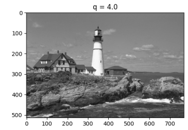
q = 1.0



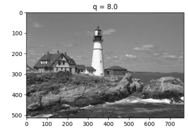
q = 2.0



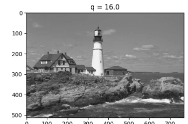
q = 4.0



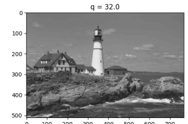
q = 8.0



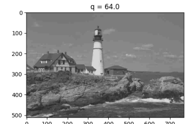
q = 16.0



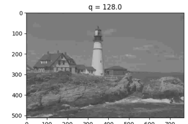
q = 32.0



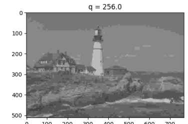
q = 64.0



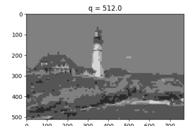
q = 128.0

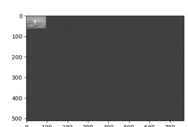


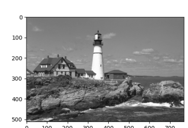
q = 256.0

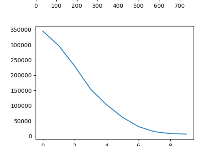


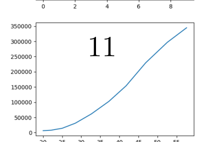
q = 512.0











- wavelet transform is much more effective than Laplacian, using 350000 non-zero to reach PSNR 58, but Laplacian using 450000 to reach 58, the difference is around 3:4