موجک

سحر محمدی

|  |  |  |
| --- | --- | --- |
| اطلاعات گزارش |  | چکیده |
| **تاریخ: دی 1401** |  | در این گزارش، نحوه انجام تمرینات مربوط به فصل تبدیل موجک توضیح داده شده است. همچنین در ادامه، به مشاهده و بررسی نتایج پرداخته شده است. |
| **واژگان كليدي:**  موجک  هرم |  |

1-توصیف تکنیکال

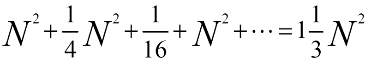
مسئله مورد بررسی اول(5.1.1)، قصد داریم برای تصویر mona lisa هرم پوسین و لاپلاسین 5 سطحی بسازیم.

مسئله 5.1.2، میخواهیم بررسی کنیم که چگونه جداپذیری و سلسله مراتبی کردن باعث افزایش سرعت فیلتر نرم کردن گوسین 3 مرحله ای میشود.

جداپذیری یعنی یک ماتریس دو بعدی را میتوان از ضرب داخلی دو ماترس یک بعدی (یکی سطری، یکی ستونی) بدست اورد. این خاصیت در فیلتر گوسین وجود دارد. که باعث میشود از حجم محاسبات کاسته شود پس سرعت بالا میرود. اگر کرنل k\*k باشد پیچیدگی زمانی 2k خیلی کمتر از k^2 میباشد.

هر بار که تصویر smooth میشود، از بیشترین فرکانس موجود در تصویر کاسته شده است پس با رعایت اصل شانون هم لازم نیست نرخ نمونه برداری دوبرابر حداکثر فرکانسی در تصویر باشد که دیگر موجود نیست. پس بعد از هر مرحله، نرخ نمونه برداری را نصف میکنیم. پس این کار باعث افزایش سرعت میشود.

مسئله 5.1.3، اگر تصویر N\*N باشد طوریکه N=2^j، بیشترین سطحی که میتواند هرم داشته باشد تا به انتها برسد(یعنی در اخرین مرحله فقط 1 پیکسل بماند)، به اندازه j میباشد.

کل اندازه مورد نیاز برای نگهداری هرم برای یک تصویر N\*N : 

کلا یک سوم بیشتر از تصویر اصلی.

پس کاربرد هرم میتواند درimage compression باشد که تصویر ممکن است از لحاظ بصری تغیر کیفیت واضح و مشخص نباشد ولی اندازه تصویر بشکل قابل توجهی کم میشود. همچنین میتوان تصویر را در scale های مختلف داشت و بسته به کاربرد از یکی از انها استفاده کرد.

مسئله 5.1.4، در این قسمت هرم تقریب 3 مرحله ای با فیلتر میانگین گیری 2\*2 یا همان box filter بدست اوردیم.

مسئله 5.1.5، هرم موجک 3 مرحله ای با استفاده از فیلتر haar بدست اوردیم.

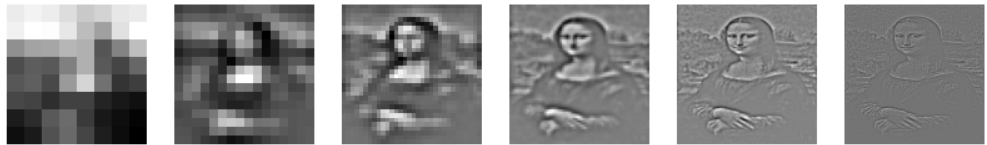
مسئله 5.1.6، خواسته که ضرایب بدست امده در تبدیل موجک را کوانتایز کنیم و تصویر را با ضرایب کوانتایز شده بازسازی کنیم.

2-نتایج

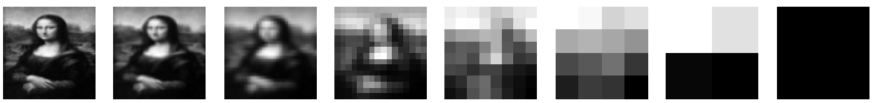
مسئله 5.1.1، هرم گوسین 5 سطحی:



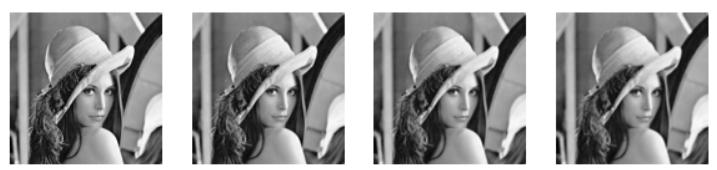
هرم لاپلاسین:



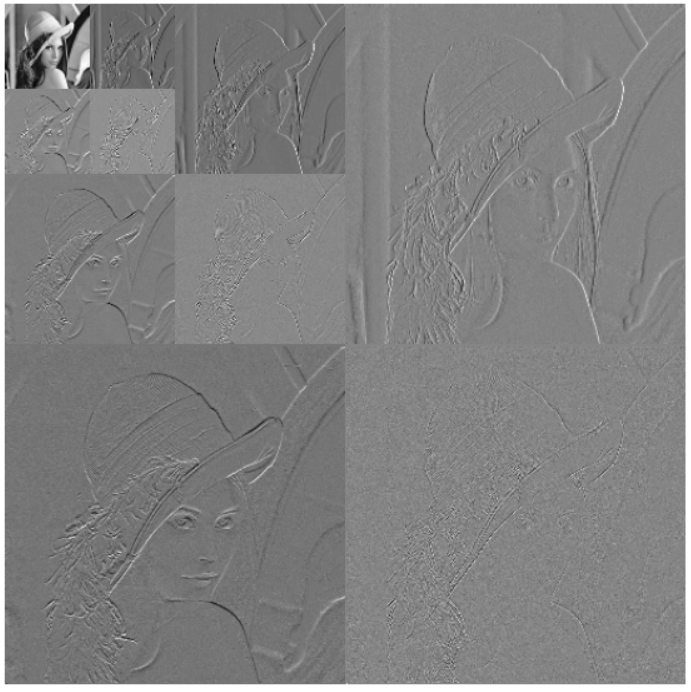
مسئله 5.1.3،



مسئله 5.1.4،



مسئله 5.1.5،



خروجی 5.1.4 شبیه هرم گوسین و خروجی این قسمت شبیه لاپلاسین است. و خب این تفاوت وجود دارد که تصاویر بالایی لبه های عمودی را دارند یعنی فیتر بصورت افقی اعمال شده و تصاویر پایین سمت چپ شامل لبه های افقی هستند.

مسئله 5.1.6،

3- ضمایم

کدها:

import cv2

import numpy as np

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

import pywt

# In[2]:

def imshow(\*args, figsize=10, title=None, fontsize=12):

if isinstance(figsize, int):

figsize = (figsize, figsize)

images = args[0] if type(args[0]) is list else list(args)

if title is not None:

assert len(title) == len(images), "Please provide a title for each image."

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='gray')

plt.axis('off')

plt.show()

# In[3]:

img = cv2.imread("Images/5/mona lisa.jpg", cv2.IMREAD\_GRAYSCALE)

# In[4]:

imshow(img)

# ## 5.1.1

# In[5]:

def gaussian\_pyramid(img, num\_levels):

lower = img.copy()

gaussian\_pyr = [lower]

for i in range(num\_levels):

lower = cv2.pyrDown(lower)

gaussian\_pyr.append(np.float32(lower))

return gaussian\_pyr

# In[6]:

def laplacian\_pyramid(gaussian\_pyr):

laplacian\_top = gaussian\_pyr[-1]

num\_levels = len(gaussian\_pyr) - 1

laplacian\_pyr = [laplacian\_top]

for i in range(num\_levels,0,-1):

size = (gaussian\_pyr[i - 1].shape[1], gaussian\_pyr[i - 1].shape[0])

gaussian\_expanded = cv2.pyrUp(gaussian\_pyr[i], dstsize=size)

laplacian = np.subtract(gaussian\_pyr[i-1], gaussian\_expanded)

laplacian\_pyr.append(laplacian)

return laplacian\_pyr

# In[7]:

img = cv2.resize(img, (256, 256))

g\_pyramid = gaussian\_pyramid(img, 5)

imshow(g\_pyramid, figsize=14)

# In[8]:

l\_pyramid = laplacian\_pyramid(g\_pyramid)

imshow(l\_pyramid, figsize=14)

# ## 5.1.3

# In[9]:

g\_pyramid = gaussian\_pyramid(img, 8)

imshow(g\_pyramid, figsize=14)

# In[10]:

print(img.shape)

g\_pyramid[8].shape

# ## 5.1.4

# In[11]:

img = cv2.imread("Images/5/Lena.bmp", cv2.IMREAD\_GRAYSCALE)

# In[12]:

img = cv2.resize(img, (256, 256))

kernel = np.ones((2,2),np.float32)/4

dst1 = cv2.filter2D(img,-1,kernel)

dst2 = cv2.filter2D(dst1,-1,kernel)

dst = cv2.filter2D(dst2,-1,kernel)

imshow(img, dst1,dst2,dst)

# ## 5.1.5

# In[13]:

img = cv2.imread("Images/5/Lena.bmp", cv2.IMREAD\_GRAYSCALE)

# In[14]:

def normalize(a):

if isinstance(a, list):

return list(map(normalize, a))

if isinstance(a, tuple):

return tuple(normalize(list(a)))

return ((a - a.min()) / (a.max() - a.min())) \* 255

# In[15]:

# wavelet transform

coeffs = pywt.wavedec2(img, 'haar', mode='periodization', level=3)

coeffs = normalize(coeffs)

# Put coefficients in a matrix

c\_matrix, c\_slices = pywt.coeffs\_to\_array(coeffs)

# In[16]:

imshow(c\_matrix)

# ## 5.1.6

# In[17]:

def psnr(original, compressed):

mse = np.mean((original - compressed) \*\* 2)

if(mse == 0): # MSE is zero means no noise is present in the signal .

# Therefore PSNR have no importance.

return 100

max\_pixel = 255.0

psnr = 20 \* log10(max\_pixel / sqrt(mse))

return psnr

# In[18]:

def quantize\_simulation(image, n\_bits):

coeff = 2\*\*8 // 2\*\*n\_bits

return (image // coeff) \* coeff

# In[19]:

quan = quantize\_simulation(coeffs,2)

img\_i = pywt.idwt2(quan, 'haar', mode='periodization')

img\_i = img\_i.astype('uint8')

imshow(img\_i)

# In[20]:

psnr(img, img\_i)

مراجع

[1] [FUM-ComputerVision/CV\_S04.ipynb at master · alifarrokh/FUM-ComputerVision (github.com)](https://github.com/alifarrokh/FUM-ComputerVision/blob/master/CV_S04.ipynb)