موجک

رضا عباس‌زاده

اطلاعات گزارش

تاریخ: ۱۴۰۰/۲/۳۱

واژگان کلیدی:
موجک
تبدیل
فوریه
هرم
هرم گوسی
هرم لاپلاسین
هرم موجک
حذف نویز

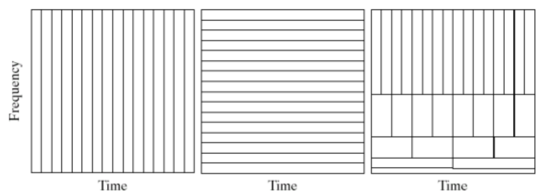
چکیده

موجک (به انگلیسی: Wavelet) دسته‌ای از توابع ریاضی هستند که برای تجزیه سیگنال پیوسته به مؤلفه‌های فرکانسی آن بکار می‌رود. بدیل موجک (Wavelet Transform) یکی از تبدیلات مهم ریاضی است که در حوزه‌های مختلف علوم کاربرد دارد. ایده اصلی تبدیل موجک این است که بر ضعف‌ها و محدودیت‌های موجود در تبدیل فوریه غلبه کند. این تبدیل را بر خلاف تبدیل فوریه، می‌توان در مورد سیگنال‌های غیر ایستا و سیستم‌های دینامیک نیز مورد استفاده قرار داد. به این صورت که می‌تواند مشخص کند در چه بازه‌ای از زمان چه فرکانس‌هایی وجود داشته‌اند. در این تبدیل رزولوشن زمانی و فرکانسی با توجه به فرکانس‌های موجود متغیر می‌باشد.

۱-مقدمه

در تمرین قبل با مبحث تبدیل فوریه و کاربردهای آن آشنا شدیم.تبدیل فوریه اطلاعات کاملی در فرکانس‌های موجود در تمام طول سیگنال در اختیار قرار می‌دهد اما نمی‌تواند مشخص کند که هر فرکانس در چه بازه‌ی زمانی وجود داشته است.

تبدیل موجک راه‌حل مشکل مطرح شده است. این تبدیل می‌تواند مشخص کند که کدام بازه از فرکانس در کدام بازه از زمان وجود داشته است. برای فرکانس‌های بالا می‌توان روزولوشن زمانی را تقویت کرد چرا که وجود مقدار اندکی خطا نمی‌تواند مشکل بزرگی ایجاد کند. برعکس در فرکانس‌های پایین تصویر شدیدا حساس به خطا می‌شود و نیاز است تا رزولوشن زمانی را افزایش دهیم. مطابق تصویر زیر:



a b c

FIGURE 7.21 Time-frequency tilings for (a) sampled data, (b) FFT, and (c) FWT basis functions.

در واقع بر اساس فرکانس‌های موجود ابعاد پنجره‌ای که روی سیگنال حرکت می‌دهیم را تغییر می‌دهیم تا به رزولوشن فرکانسی و زمانی مورد نظر برسیم.

۲-شرح تکنیکال

۱-۶ هرم

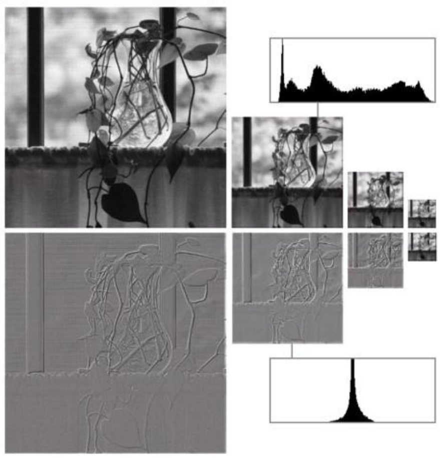
Gaussian pyramid ۶-۱-۱

در پایین‌ترین سطح این هرم تصویر اصلی قرار دارد. در هر مرحله روی تصویر مرحله قبل فیلتر گوسی اعمال می‌کنیم و ابعاد تصویر را نصف می‌کنیم.

طبق اصل شانون، نرخ نمونه برداری باید حداقل دوبرابر حداکثر فرکانس موجود در تصویر باشد تا همپوشانی (aliasing) ایجاد نشود. با اعمال فیلتر گوسی، فرکانس‌های بالای تصویر از بین می‌روند. درنتیجه، می‌توان نرخ نمونه-برداری را کاهش داد بدون اینکه همپوشانی ایجاد شود.

Laplacian pyramid

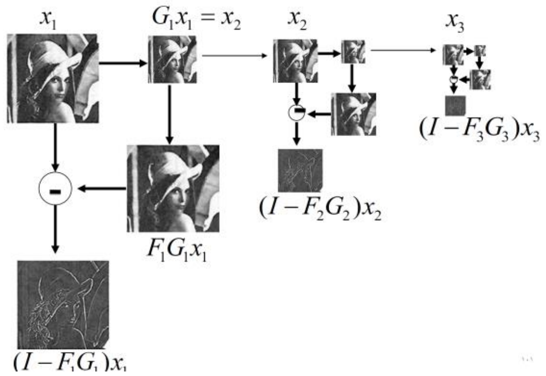
برای ساخت این هرم از هرم گوسی که در بخش قبل توضیح داده شده است، استفاده می‌کنیم. سطح اول این هرم از تفاضل تصویر اصلی و تصویر سطح دوم هرم گوسی که ابعاد آن ۲ برابر شده است (upsampling) تا با ابعاد تصویر سطح قبل مساوی شود به دست می‌آید و همین روند برای سطوح بعدی تکرار می‌شود. در واقع اطلاعاتی که در هر مرحله هرم گوسی از بین می‌رود را با استفاده از این تفاضل به دست می‌آوریم و ذخیره میکنیم. به تصویر زیر توجه کنید:



a
b

FIGURE 7.3 Two image pyramids and their statistics: (a) a Gaussian (approximation) pyramid and (b) a Laplacian (prediction residual) pyramid.

به این ترتیب تنها با ذخیره سازی آخرین سطح تصویر subsample شده هرم گوسی و تصاویر بقیه سطوح هرم لاپلاسین می‌توان تصویر اصلی را بازسازی کرد. با شروع از بالاترین سطح این هرم، ابعاد تصویر را دوبرابر می‌کنیم و با تصویر مرحله پیشین هرم لاپلاسین جمع می‌کنیم تا یک مرحله در هرم گاوسی پایین تر برویم.



نکته قابل توجه درباره هرم لاپلاسین این است که ذخیره این تفاضل ها با دلیل sparse بودن ماتریس تصاویر بسیار فضای کم‌تری اشغال خواهد کرد.

۶-۱-۲

جداپذیر بودن فیلتر گوسی به این معنی است که یک فیلتر دوبعدی گوسی می‌تواند به دوتابع یک بعدی تبدیل شود. یک تابع در راستای x و دیگری در راستای y. به این ترتیب یک convolution دوبعدی میتواند به دو تا یک بعدی تبدیل شود. پیچیدگی زمانی استفاده از یک فیلتر mxm روی یک تصویر با ابعاد nxn با این صورت است:

- $O(n^2 m^2)$

در صورتی که اگر هسته فیلتر جداپذیر باشد و بتوان به صورت موازی کار کرد پیچیدگی زمانی به این صورت خواهد بود:

- $O(n^2 m)$

شبه کد این فیلترینگ در زیر آمده است:

```
for i = ceil(size(weight1,1)/2) :1: size(I,1)-size(weight1,1)+ceil(size(weight1,1)/2)
     for j = ceil(size(weight2,2)/2) :1: size(I,2)-size(weight2,2)+ceil(size(weight2,2)/2)
        convol=0;
         %compute convolution for the neighbourhood associated to the vector
         for b = 1:size(weight1,1)

             convol = convol + (weight1(b)*I2(i-b+ceil(size(weight2,2)/2),j));

         end
         new_convol(i,j)=convol;
     end
end

for i = ceil(size(weight1,1)/2) :1: size(I,1)-size(weight1,1)+ceil(size(weight1,1)/2)
     for j = ceil(size(weight2,2)/2) :1: size(I,2)-size(weight2,2)+ceil(size(weight2,2)/2)
         convol2=0;
             %convolution with vector on the image generated before
             for a = 1:size(weight2,2)

                 convol2 = convol2 + weight2(a)*new_convol(i,j-a+ceil(size(weight2,2)/2));

             end
         OutputIm(i,j)=convol2;
     end
end
```

۶-۱-۳

این تمرین مشابه تمرین اول می‌باشد با این تفاوت که هرم را تا رسیدن به خشن‌ترین حالت (۱ پیکسل) ادامه می‌دهیم.

۶-۱-۴

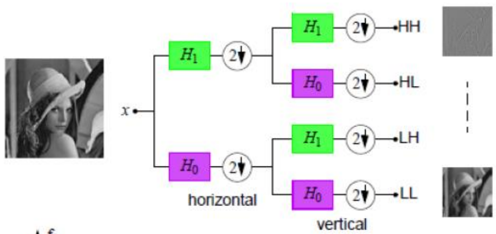
در این تمرین در مرحله down sampling از میانگین‌گیری 2x2 استفاده میکنیم و در مرحله up sampling به منظور ساخت هرم لاپلاسین از درونیابی تکرار پیکسل استفاده می‌کنیم.

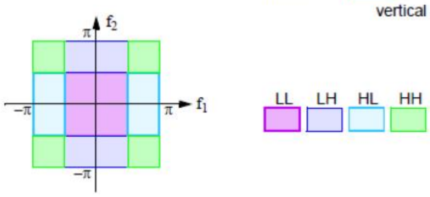
مشابه هرم گوسی می‌باشد با این تفاوت که به جای اعمال فیلتر گوسی، ابعاد تصویر را با میانگین‌گیری با استفاده از فیلتر ۲*۲ نصف می‌کنیم.

۶-۱-۵ هرم موجک

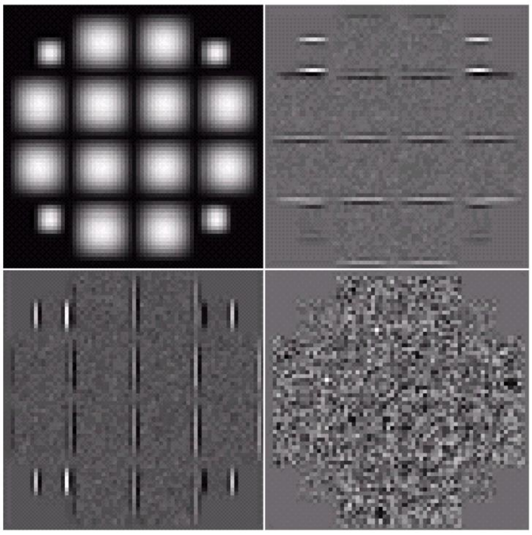
در این هم ۴ فیلتر زیر به تصویر اعمال می‌شود:

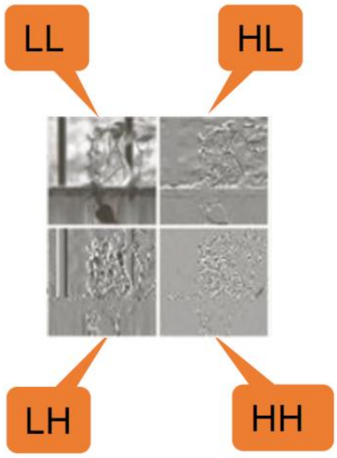
- فیلتر پایین گذر افقی و پایین گذر عمودی LL: نتیجه اعمال این فیلتر، باقی ماندن کلیت تصویر و حذف فرکانس‌های بالا و جزییات می‌باشد.
- فیلتر بالا گذر افقی و پایین گذر عمودی HL: این فیلتر لبه های افقی را نگه می‌دارد.
- فیلتر پایین گذر افقی و بالا گذر عمودی LH: این فیلتر لبه های عمودی را نگه می‌دارد.
- فیلتر بالا گذر افقی و بالا گذر عمودی HH: این فیلتر لبه های مورب را نگه می‌دارد.



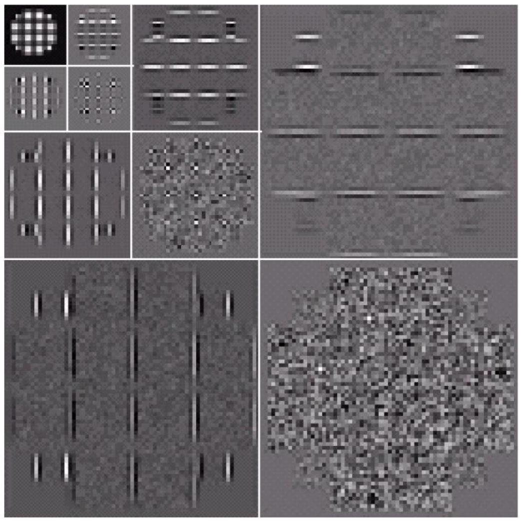


نمونه هرم موجک یه مرحله‌ای:



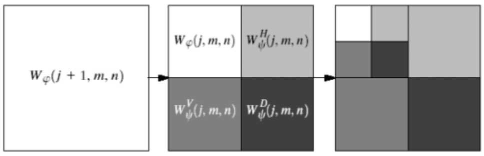


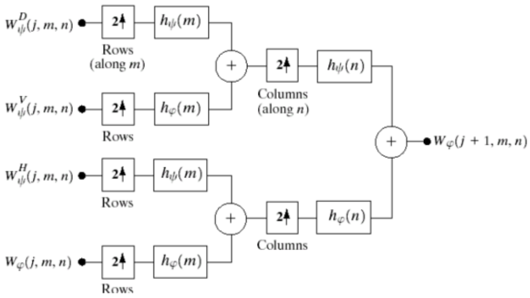
نمونه هرم موج ۳ مرحله‌ای:



هر مرحله که در هرم پیش می‌رویم سه sub-band جدید ایجاد می‌شود و زیر باند LL مرحله قبل با نصف ابعاد قبلی جایگزین می‌شود. بنابراین یک هرم موجک با n سطح، 3n+1 زیر باند دارد.

برای بازسازی تصویر روند معکوس را طی می‌کنیم:





در این تمرین با استفاده از تابع wavedec2 در کتابخانه pywt ضرایب هرم موجک چند مرحله‌ای را محاسبه می‌کنیم.

۶-۱-۶ quantized wavelet pyramid

برای quantize کردن ضرایب زیرباند ها، آن‌ها را به نزدیکترین عدد ضریب دو کمتر از آن گرد می‌کنیم.

برای بازسازی تصویر اولیه از تابع waverec2 استفاده می‌کنیم که ضرایب سطح‌بندی شده را دریافت می‌کند و تصویر نتیجه را برمی‌گرداند.

برای محاسبه psnr از فرمول زیر استفاده کرده‌ایم:

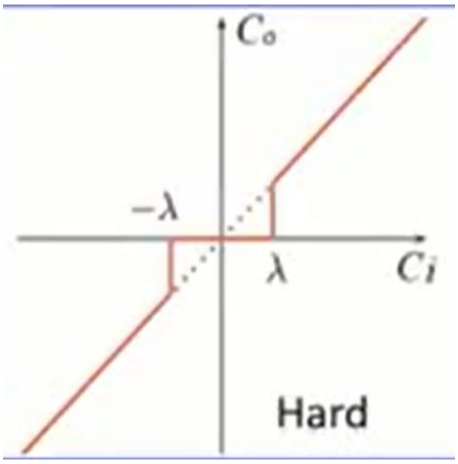
$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

۶-۲ حذف نویز

یکی از روش‌های حذف نویز به وسیله موجک، استفاده از thresholding است که به طور کلی به دو دسته تقسیم می‌شود:

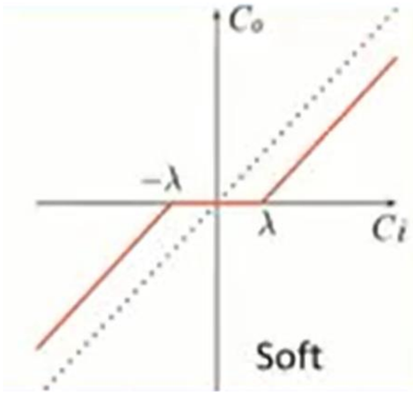
۱. hard thresholding

در این روش ضرایب کمتر از مقدار ترشولد را صفر می‌کنیم:



۲. soft thresholding

در این روش همه ضرایب را به یک مقدار مشخص به عدد صفر نزدیک می‌کنیم:



برای تعیین مقدار threshold نیز روش‌های متفاوتی وجود دارد:

۱. universal thresholding(VisuShrink)

در این روش یک ترشولد سراسری درنظر گرفته می‌شود و برای همه ضرایب موجک استفاده می‌شود.

در این روش نیاز است تا ترشولد را تعیین کنیم. برای اینکار می‌توان از تابع estimate_sigma استفاده کرد تا میانگین استاندارد نویز در کانال‌های مختلف را به دست آوریم.

۲. BayesShrink

در این روش برای هر زیر باند با توجه به مقادیر آن، یک ترشولد متفاوت با بقیه درنظر گرفته می‌شود. که باعث می‌شود عموما نتیجه بهتری نسبت به ترشولد سراسری داشته باشد.

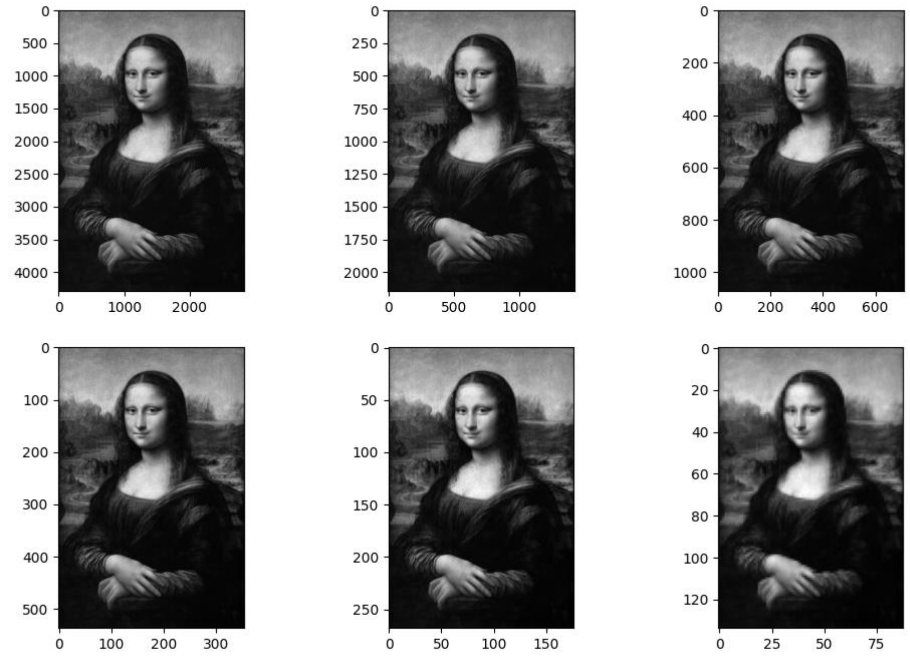
در این تمرین از کتابخانه skimage استفاده شده است.

ابتدا نویزی با سیگما 0.2 به تصویر اضافه می‌کنیم. سپس روش های مختلف گفته شده را با یکدیگر ترکیب می‌کنیم و نتایج را مقایسه می‌کنیم.

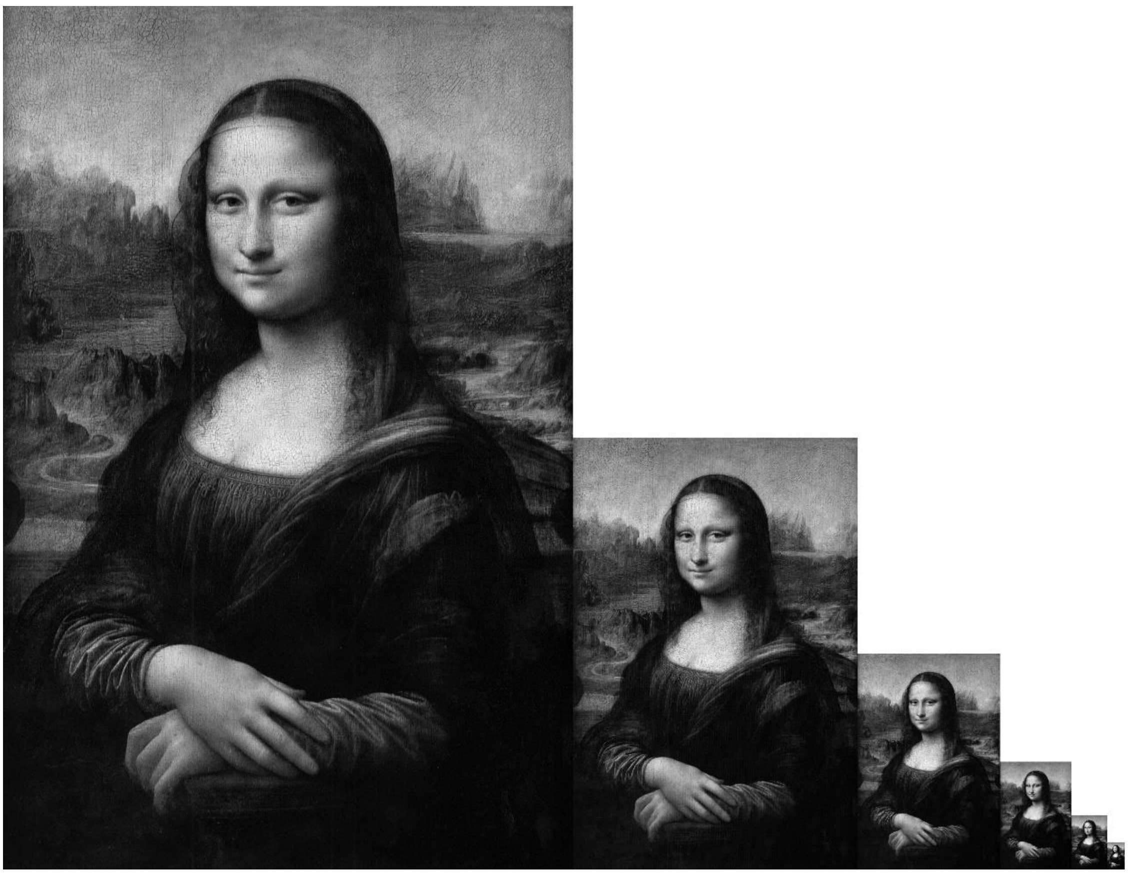
۳- شرح نتایج

۶-۱-۱

هرم گوسی

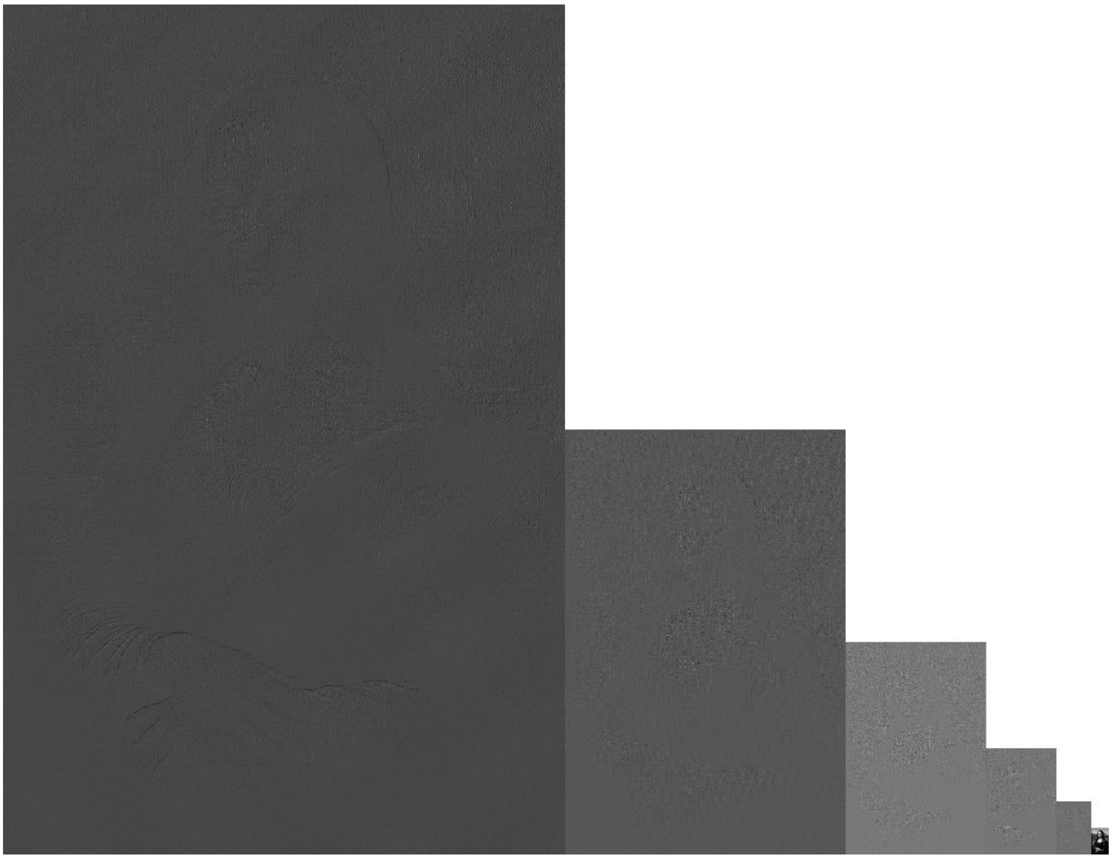


همانطور که مشخص است در هر مرحله ابعاد تصویر نصف شده است و تصویر تار می‌شود. همین نتیجه به صورت آبشاری به شکل زیر است که اولین تصویر از سمت چپ تصویر اولیه می‌باشد:



هرم لاپلاسین

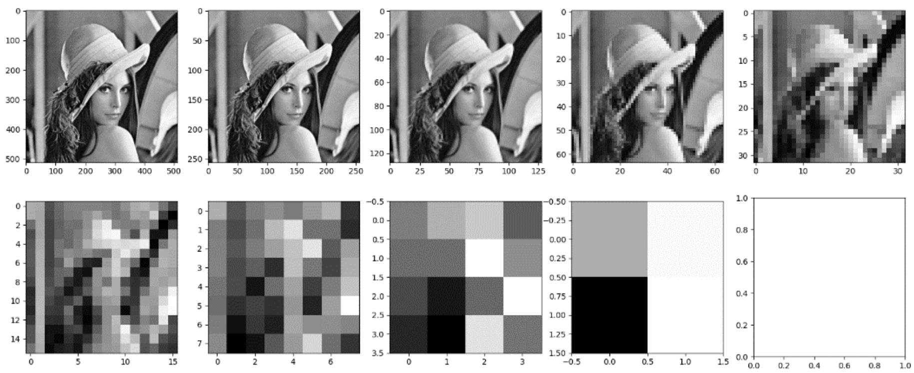
در این هرم جزییات تصویر که در هرم گوسی از بین می‌رود ذخیره می‌شوند.



۳-۱-۶

با توجه به اینکه در هر مرحله ابعاد تصویر نصف می‌شود بنابراین در تصویری مربعی با اندازه ضلع 2^J می‌توان J مرحله هرم را ادامه داد تا طول ضلع به یک پیکسل برسد. اگر تصویر اولیه را نیز در هرم در نظر بگیریم در مجموع J+1 مرحله خواهیم داشت. بنابراین در تصویر Lena با ابعاد 512x512، ۱۰ مرحله می‌توان تصویر را کوچک کرد.

هرم گوسی



هرم لاپلاسین

با بازسازی تصویر اولیه با استفاده از هرم لاپلاسین بدون هیچ خطایی و با mse = 0 به تصویر اولیه رسیدیم که مشخص می‌کند هیچ اطلاعاتی در این هرم از بین نمی‌رود و کاملا قابل بازیابی است.

تعداد پیکسل های هر مرحله یک دنباله هندسی تشکیل می‌دهند:

512, 256, 128, 64,…,1

مجموع این اعدا این دنباله برابر است با: $2N - 1 = 1023$

همانطور که محاسبه کردیم تعداد پیکسل‌ها ذخیره شده نسبت به تصویر اصلی بیشتر است.

مزایای استفاده از هرم گوسی:

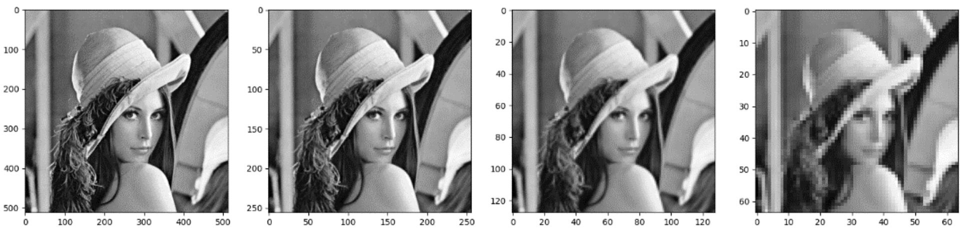
- ✓ مشاهده یک شی در اسکیل‌های مکانی مختلف که با استفاده از میانگین‌گیری گوسی down sample شده‌اند
- ✓ پردازش خشن به ظریف(Coarse to fine). منظور از این روش این است که به عنوان مثال برای پیدا کردن یک تصویر در تصویری دیگر به جای انکه روی تصویر اصلی پیمایش انجام دهیم، می‌توانیم روی تصاویر سطح بالاتر این هرم پردازش‌های لازم را انجام دهیم تا مکان‌های احتمالی شی مورد نظر را پیدا کنیم. سپس هرچه به مراحل پایین تر می‌رویم فقط نواحی اطراف آن ها را بررسی می‌کنیم به جای تمام تصویر.

مزایای استفاده از هرم لاپلاسین:

- ✓ پردازش خشن به ظریف(Coarse to fine)
- ✓ پردازش لبه ها که در سطوح بالای هرم بسیار راحت تر خواهد بود و پردازش کمتری دارد.
  - o Image Blending and Mosaicing به منظور ترکیب تصاویر با صورت واقعی و بدون لبه‌های واضح.
- ✓ فشرده‌سازی. همانطور که در شرح تکنیکال توضیح داده‌شد، ذخیره سازی ماتریس های sparse به فضای کمتری نیاز دارند و تنها یک تصویر غیر sparse داریم که در بالاترین سطح هرم قرار دارد و ابعادش بسیار کوچک و ناچیز خواهد بود.

۴-۱-۶

هرم تقریب با استفاده از میانگین‌گیری:

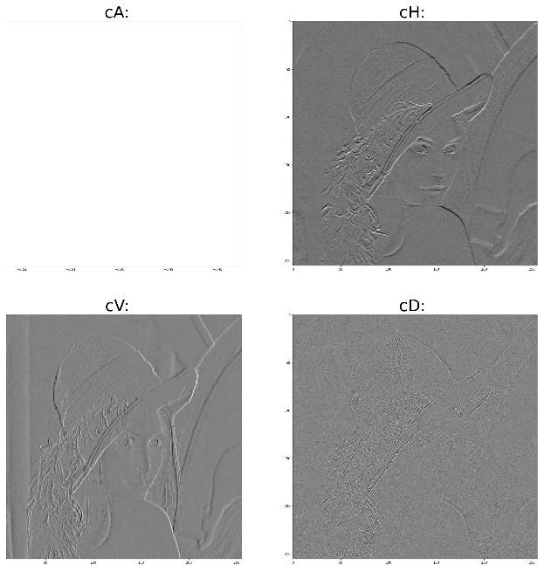


مشاهده می‌کنید که برخلاف هرم گوسی در هرمرحله تصویر به شدت وضوح خود را از دست می‌دهد و شطرنجی می‌شود در حالی که فیلتر گوسی تصویر smooth تری ایجاد میکرد.

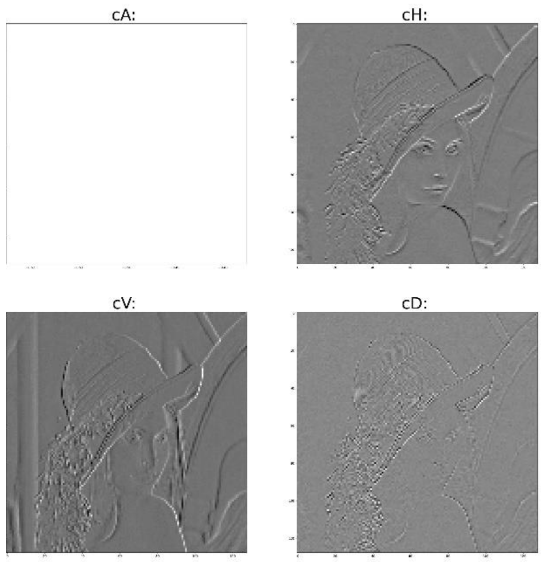
هرم لاپلاسین با استفاده از هرم تقریب بالا و درونیابی تکرار پیکسل‌ها:

۵-۱-۶

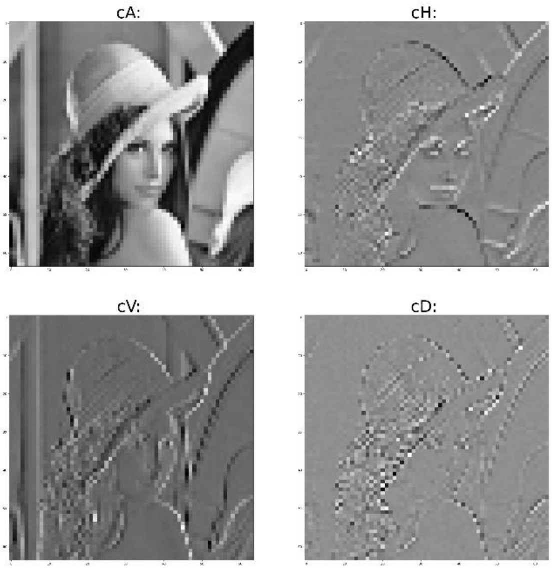
مرحله اول هرم موجک:



مرحله دوم هرم موجک:



مرحله سوم هرم موجک:



ترکیب سه مرحله در قالب هرم موجک

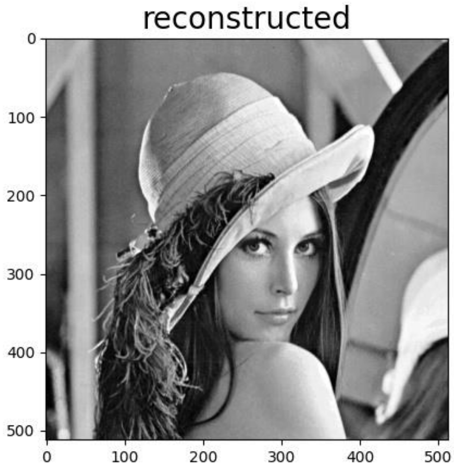
همانطور که مشخص است در لبه‌های افقی، عمودی و مورب در اسکیل‌های مختلف مشخص شده اند.

مقایسه هرم موجک، گوسی و لاپلاسین:

- در هرم گوسی و لاپلاسین(بدون فشرده سازی) فضای ذخیره سازی مورد نیاز دوبرابر تصویر اصلی می‌باشد در حالی که در هرم موجک مجموع تعداد پیکسل‌ها در همه مراحل برابر با تصویر اصلی است.
- در هرم گوسی اسکیل های مختلف تصویر را ذخیر میکنیم. درحالی که در هرم موجک و لاپلاسین فقط تصویر down sample شده آخرین مرحله باقی می‌ماند.
- در هرم گوسی جزییات و بازه فرکانسی موجود در مراحل مختلف همپوشانی دارد در حالی که در هرم لاپلاسین و موجک مراحل مختلف اطلاعات متفاوتی دارند و همپوشانی ندارند. البته در هرم موجک با توجه به بازه فرکانسی تصویر ممکن است مقداری همپوشانی داشته باشیم.

تصویر بازسازی شده با استفاده از تابع waverec2:

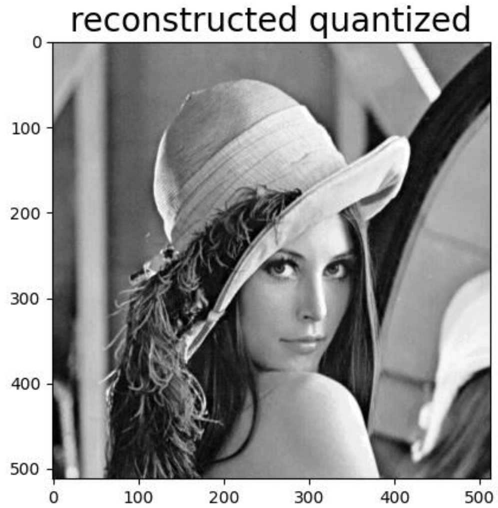
خطای mse = 8.392333984375e-05



۶-۱-۶

هرم موجک quantize شده:

تصویر بازسازی شده:



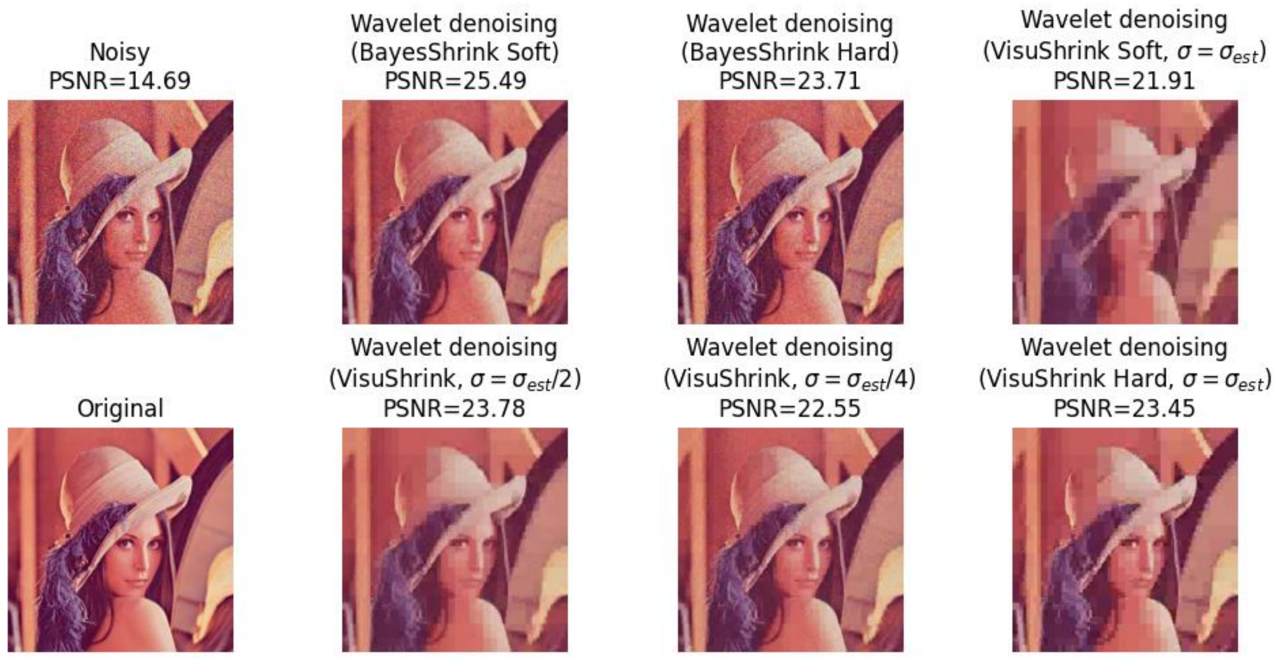
مقادیر mse و psnr این تصویر در مقایسه با تصویر اصلی به شرح زیر است:

Msr = 0.60784912109375

Psnr = 50.29284567792689

که مشخص می‌کند بازسازی تصویر با دقت بسیار خوبی انجام شده است.

۲-۶



همانطور که مشاهده می‌کنید تقریبا همه‌ی روش‌ها توانستند تا حد خوبی نویز را از بین ببرند.

در دو روش سخت و نرم bayes می‌بینیم که روش نرم عملکرد قابل توجهی داشته است و بهترین نتیجه را تولید کرده است زیرا تقریبا تمام نویز را از بین برده است و کیفیت تصویر نیز افت محسوسی نداشته است. در حالی که در روش سخت همین نوع، نویز به طور کامل از بین نرفته است.

در روش visuShrink میبینیم که عملکرد نصف سیگمای پیشنهاد شده توسط تابع estimate_sigma بهترین عملکرد را داشته است. هرچند که با چشم انسان اینطور به نظر نمیرسد زیرا لبه‌هایی به تصویر اضافه کرده است. در این روش با سیگمای یکسان نتیجه حاصل از روش سخت بهتر از روش نرم بوده است.

همچنین به نظر می‌رسد که در روش VisuShrink هرچه مقدار سیگما کوچکتر باشد، تصویر از لحاظ بصری به تصویر اصلی نزدیکتر می‌شود چرا که ضریب‌های کمتر از بین می‌روند.

پیوست

توابع مشترک مورد استفاده در تمام تمرین‌ها:

```
import cv2
import numpy as np
import copy
import matplotlib.pyplot as plt


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


def psnr(im1, im2):
    mse = np.mean((im1 - im2) ** 2)
    if mse == 0:
        return np.inf, 0
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))
    return psnr, mse


def gaussian_blur(img):
    return cv2.GaussianBlur(img, (5, 5), .1)


def average_blur(img):
    return cv2.blur(img, (2, 2))


def approximation_pyramid(img, level, mode='gaussian'):
    pyramid_list = [img]
    gauss = img
    for i in range(level):
```

```python
        gauss = average_blur(gauss) if (mode == 'average') else
gaussian_blur(gauss)
        gauss = cv2.resize(gauss, (int(gauss.shape[1] / 2),
int(gauss.shape[0] / 2)))
        pyramid_list.append(gauss)
    return pyramid_list


def up_sample(img, target_shape):
    result = np.ones(target_shape)
    is_height_odd = 0 if (target_shape[0] % 2 == 0) else 1
    is_width_odd = 0 if (target_shape[1] % 2 == 0) else 1
    for i in range(0, target_shape[0] - is_height_odd):
        for j in range(0, target_shape[1] - is_width_odd):
            result[i, j] = img[int(i / 2), int(j / 2)]
    return result


def laplacian_pyramid(imgs):
    pyramid_list = []
    for i in range(1, len(imgs)):
        upsample = up_sample(imgs[i], imgs[i - 1].shape)
        laplace = imgs[i - 1] - upsample
        pyramid_list.append(laplace)
    pyramid_list.append(imgs[-1])
    return pyramid_list


def reconstruct_from_laplacian(pyramid, img):
    upsample = copy.deepcopy(img)
    for i in range(len(pyramid) - 2, -1, -1):
        upsample = up_sample(upsample, pyramid[i].shape)
        upsample = upsample + pyramid[i]
    return upsample


def show_wt(cA, cH, cV, cD, file_name):
    plt.figure(figsize=(30, 30))

    plt.subplot(2, 2, 1)
    plt.imshow(cA, cmap=plt.cm.gray)
    plt.title('cA: ', fontsize=50)

    plt.subplot(2, 2, 2)
    plt.imshow(cH, cmap=plt.cm.gray)
    plt.title('cH: ', fontsize=50)

    plt.subplot(2, 2, 3)
    plt.imshow(cV, cmap=plt.cm.gray)
    plt.title('cV: ', fontsize=50)

    plt.subplot(2, 2, 4)
    plt.imshow(cD, cmap=plt.cm.gray)
    plt.title('cD: ', fontsize=50)
    plt.savefig(file_name)
```

تمرین ۶-۱-۱

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

from common import approximation_pyramid, laplacian_pyramid,
reconstruct_from_laplacian, psnr, normalize

img = cv2.imread("mona lisa.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imwrite('6-1-1/img.png', img)
img = img.astype(float)
gaussian_pyramid = approximation_pyramid(img, 5)
gauss_pyramid_joined = np.ones((img.shape[0], 2 * img.shape[1])) * 255
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
last_pixel = 0
for i, image in enumerate(gaussian_pyramid):
    axes[i // 3, i % 3].imshow(image, cmap='gray')
    gauss_pyramid_joined[img.shape[0] - image.shape[0]:img.shape[0],
last_pixel:last_pixel + image.shape[1]] = normalize(image)
    last_pixel += image.shape[1]

fig.savefig('6-1-1/approximation_pyramid.png')
cv2.imwrite('6-1-1/approximation_pyramid_joined.jpg', gauss_pyramid_joined)

laplacian_pyramid = laplacian_pyramid(gaussian_pyramid)
laplacian_pyramid_joined = np.ones((img.shape[0], 2 * img.shape[1])) * 255

fig, axes = plt.subplots(2, 3, figsize=(12, 8))
last_pixel = 0
for i, image in enumerate(laplacian_pyramid):
    laplacian_pyramid_joined[img.shape[0] - image.shape[0]:img.shape[0],
last_pixel:last_pixel + image.shape[1]] = normalize(image)
    axes[i // 3, i % 3].imshow(image, cmap='gray')
    last_pixel += image.shape[1]

fig.savefig('6-1-1/laplace_pyramid.png')
cv2.imwrite('6-1-1/laplace_pyramid_joined.jpg', laplacian_pyramid_joined)
```

تمرین ۶-۱-۳

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

from common import approximation_pyramid, laplacian_pyramid,
reconstruct_from_laplacian, psnr, normalize

img = cv2.imread("Lena.bmp", cv2.IMREAD_GRAYSCALE)
cv2.imwrite('6-1-3/img.png', img)
img = img.astype(float)
gaussian_pyramid = approximation_pyramid(img, 8)
gauss_pyramid_joined = np.ones((img.shape[0], 2 * img.shape[1])) * 255
fig, axes = plt.subplots(2, 5, figsize=(20, 8))
last_pixel = 0
for i, image in enumerate(gaussian_pyramid):
    axes[i // 5, i % 5].imshow(image, cmap='gray')
    gauss_pyramid_joined[img.shape[0] - image.shape[0]:img.shape[0],
last_pixel:last_pixel + image.shape[1]] = normalize(image)
    last_pixel += image.shape[0]
```

```python
fig.savefig('6-1-3/approximation_pyramid.png')
cv2.imwrite('6-1-3/approximation_pyramid_joined.jpg', gauss_pyramid_joined)

laplacian_pyramid = laplacian_pyramid(gaussian_pyramid)
laplacian_pyramid_joined = np.ones((img.shape[0], 2 * img.shape[1])) * 255

fig, axes = plt.subplots(2, 5, figsize=(20, 8))
last_pixel = 0
for i, image in enumerate(laplacian_pyramid):
    laplacian_pyramid_joined[img.shape[0] - image.shape[0]:img.shape[0],
last_pixel:last_pixel + image.shape[1]] = normalize(image)
    axes[i // 5, i % 5].imshow(image, cmap='gray')
    last_pixel += image.shape[0]

fig.savefig('6-1-3/laplace_pyramid.png')
cv2.imwrite('6-1-3/laplace_pyramid_joined.jpg', laplacian_pyramid_joined)
reconstruct = reconstruct_from_laplacian(laplacian_pyramid,
gaussian_pyramid[len(gaussian_pyramid) - 1])
cv2.imwrite('6-1-3/reconstruct.png', reconstruct)
psnr, mse = psnr(img, reconstruct)
print("mmse={}".format(mse))
print("psnr={}".format(psnr))
```

تمرین ۶-۱-۴

```python
import matplotlib.pyplot as plt
import cv2
from common import *

img = cv2.imread('Lena.bmp', cv2.IMREAD_GRAYSCALE)
img = img.astype(float)
gaussian_pyramid = approximation_pyramid(img, 3, 'average')
gauss_pyramid_joined = np.ones((img.shape[0], 2 * img.shape[1])) * 255

fig, axes = plt.subplots(1, 4, figsize=(20, 5))
last_pixel = 0
for i, im in enumerate(gaussian_pyramid):
    axes[i].imshow(im, cmap='gray')
    gauss_pyramid_joined[0:im.shape[0], last_pixel:last_pixel +
im.shape[1]] = im
    last_pixel += im.shape[0]

fig.savefig('6-1-4/approximation_pyramid.png')
cv2.imwrite('6-1-4/approximation_pyramid_joined.jpg', gauss_pyramid_joined)

l_pyramid_im = np.ones((img.shape[0], 2 * img.shape[1])) * 255
laplace_pyramid = laplacian_pyramid(gaussian_pyramid)
fig, axes = plt.subplots(1, 4, figsize=(20, 5))
last_pixel = 0
for i, im in enumerate(laplace_pyramid):
    l_pyramid_im[0:im.shape[0], last_pixel:last_pixel + im.shape[1]] =
normalize(im)
    axes[i].imshow(im, cmap='gray')
    last_pixel += im.shape[0]

fig.savefig('6-1-4/laplace_pyramid.png')
cv2.imwrite('6-1-4/laplace_pyramid_im.jpg', l_pyramid_im)
reconstructed = reconstruct_from_laplacian(laplace_pyramid,
gaussian_pyramid[len(gaussian_pyramid) - 1])
```

```
cv2.imwrite('6-1-4/reconstruct_im.jpg', reconstructed)
psnr, mmse = psnr(img, reconstructed)
print("mmse={}".format(mmse))
print("psnr={}".format(psnr))
```

تمرین ۶-۱-۵

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pywt

from common import normalize, show_wt, psnr

img = cv2.imread('Lena.bmp', cv2.IMREAD_GRAYSCALE)
C = pywt.wavedec2(img, 'haar', 'periodization', 3)

plt.figure(figsize=(30, 30))

cA3 = C[0]
(cH1, cV1, cD1) = C[-1]
(cH2, cV2, cD2) = C[-2]
(cH3, cV3, cD3) = C[-3]
show_wt(np.zeros((0, 0)), cH1, cV1, cD1, '6-1-5/Level 1.jpg')
show_wt(np.zeros((0, 0)), cH2, cV2, cD2, '6-1-5/Level 2.jpg')
show_wt(C[0], cH3, cV3, cD3, '6-1-5/Level 3.jpg')

arr, coeff_slices = pywt.coeffs_to_array(C)
plt.figure()
plt.imshow(arr, cmap=plt.cm.gray)
cv2.imwrite('6-1-5/pyramid.jpg', normalize(arr))

imgr = pywt.waverec2(C, 'haar', 'periodization')
imgr = np.uint8(imgr)
plt.figure()
plt.imshow(imgr, cmap=plt.cm.gray)
plt.title('reconstructed', fontsize=20)
plt.savefig('6-1-5/reconstructed.jpg')

psnr, mse = psnr(img, imgr)
print('psnr: {}'.format(psnr))
print('mse: {}'.format(mse))
```

تمرین ۶-۱-۶

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pywt

from common import normalize, psnr


def quantize(arr, value=2):
    return (np.int64(arr) / value) * value


img = cv2.imread('Lena.bmp', cv2.IMREAD_GRAYSCALE)
C = pywt.wavedec2(img, 'haar', 'periodization', 3)
```

```
plt.figure(figsize=(30, 30))

cA3 = C[0]
(cH1, cV1, cD1) = C[-1]
(cH2, cV2, cD2) = C[-2]
(cH3, cV3, cD3) = C[-3]

cA3_quantized = quantize(cA3)
cH3_quantized = quantize(cH3)
cV3_quantized = quantize(cV3)
cD3_quantized = quantize(cD3)
cH2_quantized = quantize(cH2)
cV2_quantized = quantize(cV2)
cD2_quantized = quantize(cD2)
cH1_quantized = quantize(cH1)
cV1_quantized = quantize(cV1)
cD1_quantized = quantize(cD1)
C_quantized = [cA3,
               (cH3_quantized, cV3_quantized, cD3_quantized),
               (cH2_quantized, cV2_quantized, cD2_quantized),
               (cH1_quantized, cV1_quantized, cD1_quantized)]

arr, coeff_slices = pywt.coeffs_to_array(C_quantized)
plt.figure()
plt.imshow(arr, cmap=plt.cm.gray)
cv2.imwrite('6-1-6/pyramid.jpg', normalize(arr))

imgr_quantized = pywt.waverec2(C_quantized, 'haar', 'periodization')
imgr_quantized = np.uint8(imgr_quantized)
plt.figure()
plt.imshow(imgr_quantized, cmap=plt.cm.gray)
plt.title('reconstructed quantized', fontsize=20)
plt.savefig('6-1-6/reconstructed quantized.jpg')

psnr, mse = psnr(img, imgr_quantized)
print('psnr: {}'.format(psnr))
print('mse: {}'.format(mse))
```

تمرین ۶-۲

```
import matplotlib.pyplot as plt
from skimage.restoration import (denoise_wavelet, estimate_sigma)
from skimage import data, img_as_float
from skimage.util import random_noise
from skimage.metrics import peak_signal_noise_ratio
from skimage.io import imread

original = img_as_float(imread('Lena.bmp'))

sigma = 0.2
noisy = random_noise(original, var=sigma ** 2)

fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(10, 5),
                       sharex=True, sharey=True)

plt.gray()

# Estimate the average noise standard deviation across color channels.
```

```
sigma_est = estimate_sigma(noisy, average_sigmas=True, multichannel=True)
# Due to clipping in random_noise, the estimate will be a bit smaller than
the
# specified sigma.
print(f"Estimated Gaussian noise standard deviation = {sigma_est}")

im_bayes_soft = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                method='BayesShrink', mode='soft',
                                rescale_sigma=True)
im_bayes_hard = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                method='BayesShrink', mode='hard',
                                rescale_sigma=True)
im_visushrink_soft = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                     method='VisuShrink', mode='soft',
                                     sigma=sigma_est, rescale_sigma=True)
im_visushrink_hard = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                method='VisuShrink', mode='hard',
                                sigma=sigma_est, rescale_sigma=True)

# VisuShrink is designed to eliminate noise with high probability, but this
# results in a visually over-smooth appearance.  Repeat, specifying a
reduction
# in the threshold by factors of 2 and 4.
im_visushrink2 = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                 method='VisuShrink', mode='soft',
                                 sigma=sigma_est / 2, rescale_sigma=True)
im_visushrink4 = denoise_wavelet(noisy, multichannel=True,
convert2ycbcr=True, wavelet='haar',
                                 method='VisuShrink', mode='soft',
                                 sigma=sigma_est / 4, rescale_sigma=True)

# Compute PSNR as an indication of image quality
psnr_noisy = peak_signal_noise_ratio(original, noisy)
psnr_bayes_soft = peak_signal_noise_ratio(original, im_bayes_soft)
psnr_bayes_hard = peak_signal_noise_ratio(original, im_bayes_hard)
psnr_visushrink_soft = peak_signal_noise_ratio(original,
im_visushrink_soft)
psnr_visushrink_hard = peak_signal_noise_ratio(original,
im_visushrink_hard)
psnr_visushrink2 = peak_signal_noise_ratio(original, im_visushrink2)
psnr_visushrink4 = peak_signal_noise_ratio(original, im_visushrink4)

ax[0, 0].imshow(noisy)
ax[0, 0].axis('off')
ax[0, 0].set_title('Noisy\nPSNR={:0.4g}'.format(psnr_noisy))
ax[0, 1].imshow(im_bayes_soft)
ax[0, 1].axis('off')
ax[0, 1].set_title(
    'Wavelet denoising\n(BayesShrink
Soft)\nPSNR={:0.4g}'.format(psnr_bayes_soft))

ax[0, 2].imshow(im_bayes_hard)
ax[0, 2].axis('off')
ax[0, 2].set_title(
    'Wavelet denoising\n(BayesShrink
Hard)\nPSNR={:0.4g}'.format(psnr_bayes_hard))
```

```
ax[1, 0].imshow(original)
ax[1, 0].axis('off')
ax[1, 0].set_title('Original')
ax[1, 1].imshow(im_visushrink2)
ax[1, 1].axis('off')
ax[1, 1].set_title(
    'Wavelet denoising\n(VisuShrink, $\\sigma=\\sigma_{est}/2$)\n'
    'PSNR=%0.4g' % psnr_visushrink2)
ax[1, 2].imshow(im_visushrink4)
ax[1, 2].axis('off')
ax[1, 2].set_title(
    'Wavelet denoising\n(VisuShrink, $\\sigma=\\sigma_{est}/4$)\n'
    'PSNR=%0.4g' % psnr_visushrink4)

ax[0, 3].imshow(im_visushrink_soft)
ax[0, 3].axis('off')
ax[0, 3].set_title(
    'Wavelet denoising\n(VisuShrink Soft, $\\sigma=\\sigma_{est}$)\n'
    'PSNR=%0.4g' % psnr_visushrink_soft)

ax[1, 3].imshow(im_visushrink_hard)
ax[1, 3].axis('off')
ax[1, 3].set_title(
    'Wavelet denoising\n(VisuShrink Hard, $\\sigma=\\sigma_{est}$)\n'
    'PSNR=%0.4g' % psnr_visushrink_hard)
fig.tight_layout()

plt.savefig('denoising.png')
```