Wavelet

شیوا رادمنش

اطلاعات گزارش

تاریخ: ۱۳۹۹/۱۰/۲۱

واژگان کلیدی:
هرم لاپلاسین
هرم گاوسی
بازسازی تصویر
تبدیل wavelet
فشرده سازی

چکیده

در این گزارش در ابتدا به معرفی هرم لاپلاسین و گاوسی و طریقه‌ی ساخت آنها، همچنین شیوه‌ی بازسازی تصاویر از روی هرم و فشرده سازی هرم پرداخته شده. سپس تبدیل wavelet و شیوه‌ی فشرده سازی با استفاده از این تبدیل و نهایتا دو روش رفع نویز با استفاده از این تبدیل معرفی می‌شود.

۱- مقدمه

نوشتار حاضر در بخش اول به بررسی هرم لاپلاسین و گاوسی، شیوه‌ی بازسازی تصویر از روی هرم و کاربردهای آنها می‌پردازد. در بخش دوم تبدیل wavelet و بازسازی تصویر از روی آن و فشرده سازی تصویر wavelet و همچنین حذف نویز با استفاده از تبدیل wavelet مورد بررسی قرار گرفته است. تمامی پیاده سازی‌های این تمرین با استفاده از زبان python انجام شده است.

۲- شرح تکنیکال

۱-۲- سوال ۶.۱.۱

در این بخش به هرم گاوسی و هرم لاپلاسین پرداخته خواهد شد.

برای ساختن هرم گاوسی یک تصویر در هر سطح رزولوشن را پایین می‌آوریم و به صورت زیر عمل می‌شود:

- تصویر این مرحله را با استفاده یک فیلتر گاوسی(پایین گذر) smooth می‌کنیم.
- تصویر smooth شده را به اندازه‌ی $\frac{n}{2} \times \frac{n}{2}$ downsample می‌کنیم(با فرض اینکه اندازه‌ی تصویر در این مرحله $n \times n$ بوده است).

- تصویر بدست آمده(تصویر downsample شده) تصویر سطح بعد است. مراحل فوق را برای این تصویر انجام می‌دهیم.

هرم لاپلاسین با استفاده از تفاضل بین تصویر در رزولوشن خاصی از هرم گاوسی و نسخه‌ی بزرگتر شده با رزولوشن پایین‌تر ساخته می‌شود. اگر تصویر سطح i ام در هرم گاوسی را G_i و تصویر سطح i ام در هرم لاپلاسین را L_i بنامیم. تصویر در هر سطح هرم لاپلاسین از رابطه‌ی زیر به دست می‌آید.

$$L_i = G_i - expand(G_{i-1})$$

در نهایت برای بازسازی تصویر تنها هرم لاپلاسین و تصویر آخرین سطح هرم گاوسی نیاز داریم و فقط این تصاویر را ذخیره می‌کنیم.

از آنجایی که سایز تصویر در هر مرحله نصف می‌شود($n \times n \Rightarrow \frac{n}{2} \times \frac{n}{2}$) برای یک تصویر با اندازه‌ی $N \times N$ حداکثر تعداد مراحل(تعداد سطوح هرم) برابر است با $logN$. در این صورت در نهایت از downsample کردن آخرین سطح هرم گاوسی یک اسکالر(یک پیکسل) بدست می‌آید و دیگر قابل downsample شدن نخواهد بود. این مقدار در تصویر Lena برابر است با 117

در اینجا اندازه‌ی تصویر Lena برابر است با 512* 512 می‌باشد، بنابراین تعداد سطوح هرم برای این تصویر برابر با ۹ خواهد بود.

مجموع تعداد پیکسل‌هایی که در یک هرم وجود دارد برابر است با:

$$N^2 + \frac{1}{4}N^2 + \frac{1}{16}N^2 + \ldots = 1\frac{1}{3}N^2$$

این در حالی است که تعداد پیکسل‌های تصویر اصلی N^2 بوده است و با این روش تعداد پیکسل‌های بیشتری در حافظه ذخیره می‌کنیم.

با وجود اشغال حافظه‌ی بیشتر نسبت به حالت عادی، استفاده از هرم فوایدی دارد، که به برخی از آنها اشاره می‌شود.

- امکان مشاهده‌ی یک شی در اسکیل‌های مختلف مکانی را می‌دهد. این ویژگی باعث افزایش سرعت در template matching می‌شود. به این صورت که تصویر template در scale کوچک را در تصویر اصلی در scale کوچک جستجو می‌کنیم. پس از پیدا کردن location حدودی template در تصویر اصلی، آنها را expand می‌کنیم و در رزولوشن‌های بالاتر تنها همان ناحیه که در رزولوشن پایین تر پیدا کرده ایم جستجو می‌کنیم.
- پردازش خشن به ظریف(coarse to fine)، از این ویژگی برای تشخیص لبه استفاده می‌شود. بدین صورت که لبه‌ها را در سطوح بالای هرم(رزولوشن پایین) تشخیص داده و

در سطوح پایین(رزولوشن بالا) لبه‌های ظریف را تنها در اطراف لبه‌های پیدا شده در سطوح بالا جستجو می‌کنیم.

- از آنجایی که تصاویر در سطوح مختلف هرم لاپلاسین ماتریس اسپارس هستند، فشرده‌سازی آنها امکان پذیر است.
- این رویکرد امکان تبدیل تدریجی را فراهم می‌کند، چون تصاویر هر سطح که با فیلتر پایین گذر، فیلتر شده اند، یک تقریب مناسب از تصویر می‌باشند. این ویژگی سبب کاربردهایی مانند image blending و image mosaicing می‌شود. بدین صورت که تصاویر در همه‌ی سطوح هرم با هم ترکیب می‌شوند.
-

۲-۲- سوال ۶.۱.۲

در بخش ۲-۲ به بررسی هرم گاوسی و لاپلاسین پرداخته شد و روش ایجاد هرم بیان شد. در این بخش به چگونگی بازسازی تصویر در این روش، پرداخته می‌شود.

همانطور که در بخش ۲-۲ گفته شد، برای بازسازی تصویر تنها به هرم لاپلاسین و آخرین سطح هرم گاوسی احتیاج داریم.

برای بازسازی تصویر در هر سطح به صورت زیر عمل می‌کنیم.

- تصویر هرم گاوسی فعلی را upsample کرده. برای این کار به درون‌یابی احتیاج داریم(در اینجا از روش‌های pixel replication و bilinear استفاده شده و در بخش نتایج مقایسه و بررسی شده اند).
- حاصل جمع تصویر expand شده و تصویر نظیر آن در هرم لاپلاسین، تصویر سطح قبلی در هرم گاوسی می‌باشد.

$$G_i = L_i + G_{i-1}$$

- مراحل فوق را برای تصویر بدست آمده تکرار می‌کنیم تا به تصویر اصلی برسیم.

۲-۳- سوال ۶.۱.۳

در این بخش به بررسی تبدیل موجک دو بعدی پرداخته خواهد شد.

در این تبدیل با اعمال فیلترهای haar در هر سطح، تصویر به ۴ بخش LL و LH و HL و HH شکسته می‌شود. بخش LL تصویر با رزولوشن کمتر می‌باشد(horizontal lowpass و vertical lowpass)، بخش HL شامل لبه‌های افقی تصویر(horizontal highpass و vertical lowpass)، بخش LH شامل لبه‌های عمودی تصویر(horizontal

lowpass و vertical highpass) و بخش HH شامل جزئیات قطری تصویر است(horizontal highpass و vertical highpass). در برای تبدیل wavelet سطح، فرکانس‌های بالا را نگه داشته و دوباره فیلتر را به بخش LL اعمال می‌کنیم. در اینجا تبدیل wavelet با استفاده از فیلتر haar تا سطح ۳ با استفاده از تابع wavedec2 کتابخانه‌ی pywt پایتون بر روی تصویر Lena انجام شده است.

۲-۴- سوال ۶.۱.۴

در تبدیل موجک به دلیل اینکه ماتریس‌های بخش LH و HL و LL به نوعی اسپارس هستند، امکان فشرده سازی وجود دارد. در این بخش به بررسی یک روش فشرده سازی پرداخته شده است.

برای فشرده سازی،بر روی ضرایب wavelet در همه‌ی سطوح با استفاده از رابطه‌ی زیر عمل quantization انجام داده.(در این رابطه c ضرایب wavelet است)

$$c'(u,v) = \gamma \times sgn[c(u,v)] \times floor[\frac{|c(u,v)|}{\gamma}]$$

۲-۵- سوال ۶.۲

در این بخش به بررسی دو روش denoising با استفاده از تبدیل wavelet پرداخته می‌شود.

نویزهایی مانند نویز گاوسی به صورت مقادیر کوچکی در دامنه‌ی wavelet هستند و

در این سوال با استفاده از تابع denoise_wavelet که در کتابخانه scikit-image ارائه شده است، عمل حذف نویز انجام شده است. در این تابع، دو روش برای عمل حذف نویز ارائه شده است:

۱- VisuShrink و ۲- BayesShrink که در ادامه توضیحات آن ها ارائه می‌شود.

VisuShrink: رویکرد VisuShrink برای تمام ضرایب جزئیات موجک ، یک threshold واحد به کار می‌برد. این آستانه برای حذف نویز افزودنی گاوسی با احتمال زیاد طراحی شده است ، که منجر به ظاهر بیش از حد صاف تصویر می شود. با تعیین یک سیگما که کوچکتر از انحراف معیار نویز واقعی است ، می توان نتیجه مطلوبتری از نظر بصری بدست آورد.

BayesShrink: الگوریتم BayesShrink یک رویکرد انطباقی برای wavelet soft thresholding است که در آن یک threshold منحصر به فرد برای هر زیر باند wavelet تخمین زده می شود. این روش

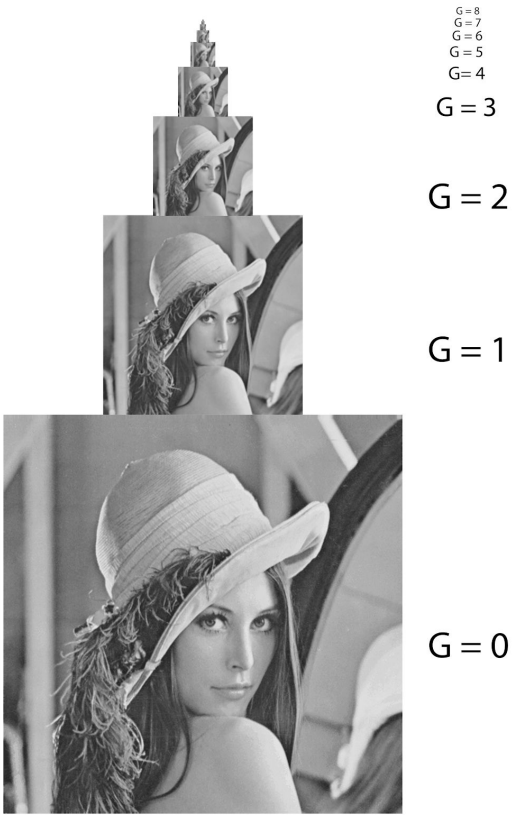
به طور کلی منجر به بهبود چیزی می شود که با یک threshold واحد بدست می آید.

۳- نتایج

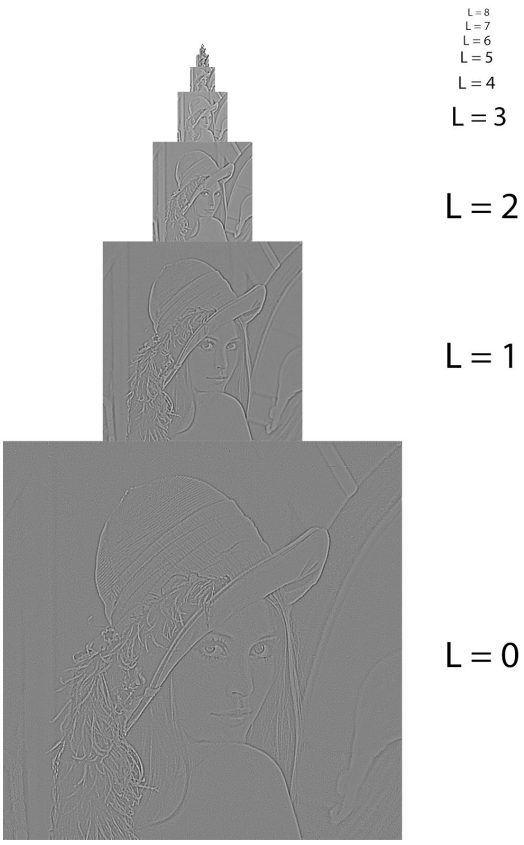
۱-۳- سوال ۶.۱.۱

در این بخش هرم گاوسی و هرم لاپلاسین تصویر Lena در ۹ سطح(حداکثر تعداد سطوح) بررسی شده است.

شکل۱- تصویر Lena



شکل۲- هرم گاوسی تصویر Lena در ۹ سطح



شکل۳- هرم لاپلاسین تصویر Lena

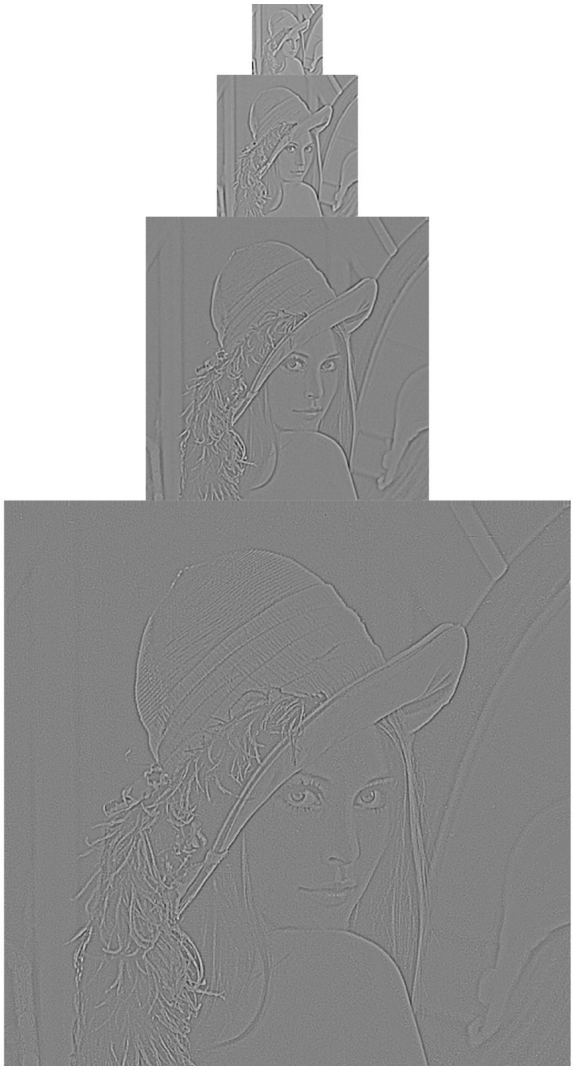
که البته تنها با استفاده از هرم لاپلاسین و عددی که از downsample کردن سطح آخر هرم گاوسی بدست می‌آید، می‌توان تصویر را باز سازی‌کرد.

همانطور که قابل مشاهده است در هرم لاپلاسین فرکانس‌های بالای تصویر نگهداری شده و در هرم گاوسی فرکانس‌های پایین تصویر، وقتی فرکانس‌های بالای تصویر را جدا کنیم ماکزیمم فرکانس در تصویر(تصویر در سطوح هرم گاوسی) کاهش یافته و طبق اصل شانون می‌توان نرخ نمونه برداری را بدون از دست دادن کیفیت کاهش داد و تصویر را downsample کرد.

۲-۳- سوال ۶.۱.۲

در ادامه تصویر هرم گاوسی و لاپلاسین تصویر Lena و همچنین تصویر بازسازی شده از روی هرم لاپلاسین قابل مشاهده می‌باشد.

شکل۳- هرم گاوسی تصویر Lena تا سطح ۳



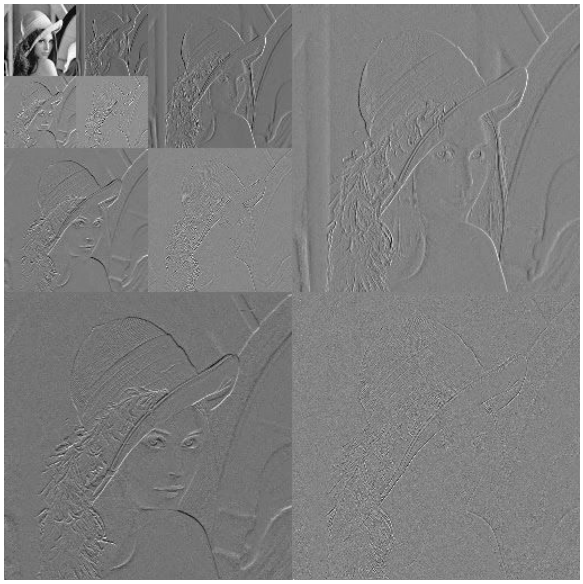
شکل۴- هرم لاپلاسین تصویر Lena تا سطح ۳

شکل۵- تصویر بازیابی شده از ۳ سطح با روش درونیابی pixel replication

همانطور که در شکل۵ مشاهده می‌شود، تصویری که با استفاده از روش pixel replication درونیابی شده است، از نظر بصری بسیار شبیه به تصویر اصلی می‌باشد. با وجود استفاده از روش pixel replication از نظر بصری جزئیاتی از دست نداده است. دلیل این امر آن است که جزئیات تصویر در هرم لاپلاسین نگهداری شده و در بازسازی تصویر جزئیات از دست نمی‌رود.

۲-۳- سوال ۶.۱.۳

تصویر هرم ۳ wavelet سطحی تصویر Lena و همچنین تصویر بازسازی شده از روی هرم wavelet در ادامه قابل مشاهده است.



شکل۶- تبدیل wavelet تصویر Lena در ۳ سطح

همانطور که مشاهده می‌شود، تعداد پیکسل‌های ذخیره شده در این روش دقیقا به اندازه‌ی تعداد پیکسل‌های تصویر اصلی است. و از نظر مصرف حافظه عملکرد بهتری دارد.

شکل۷- تصویر بازسازی شده از روی تبدیل wavelet در ۳ سطح

همانطور که مشاهده می‌شود تصویر بازسازی شده از نظر بصری از تصویر اصلی قابل تشخیص نیست.

دلیل این امر آن است که جزئیات در رزولوشن‌های مختلف ذخیره می‌شوند.

۴-۳- سوال ۶.۱.۴

در ادامه تصویر بازسازی شده از تبدیل wavelet فشرده سازی شده قابل مشاهده است.

شکل۸- تصویر بازسازی شده از تبدیل wavelet فشرده سازی شده.

این تصویر در مقایسه با تصویر اصلی و تصویر بازسازی شده از تبدیل wavelet، کیفیت کمتری دارد و psnr این تصویر و تصویر اصلی برابر ۳۰.۶۱ می‌باشد.

۵-۳- سوال ۶.۲

در این بخش به بررسی نتایج دو روش wavelet denoising پرداخته می‌شود. نتایج در ادامه قابل مشاهده است.

شکل۹- تصویر اصلی و بدون نویز

شکل۱۰- تصویر نویزی با نویز گوسی

شکل۱۱- تصویر حاصل از حذف نویز گوسی با روش BayesShrink و $\sigma = 1$

شکل۱۳- تصویر حاصل از حذف نویز گوسی با روش BayesShrink و $\sigma = \frac{1}{4}$

شکل۱۲- تصویر حاصل از حذف نویز گوسی با روش BayesShrink و $\sigma = \frac{1}{2}$

شکل۱۴- تصویر حاصل از حذف نویز گوسی با روش VisuShrink و $\sigma = 1$

شکل۱۵- تصویر حاصل از حذف نویز گوسی با روش VisuShrink و $\sigma = \frac{1}{2}$

شکل۱۶- تصویر حاصل از حذف نویز گوسی با روش VisuShrink و $\sigma = \frac{1}{4}$

شکل۱۷- تصویر نویزی با نویز نمک و فلفل

شکل ۱۸- تصویر حاصل از حذف نویز نمک و فلفل با روش BayesShrink و $\sigma = 1$

شکل۲۰- تصویر حاصل از حذف نویز نمک و فلفل با روش BayesShrink و $\sigma = \frac{1}{4}$

شکل ۱۹- تصویر حاصل از حذف نویز نمک و فلفل با روش BayesShrink و $\sigma = \frac{1}{2}$

شکل۲۱- تصویر حاصل از حذف نویز نمک و فلفل با روش VisuShrink و $\sigma = 1$

همانطور که گفته شد، برای پیاده سازی این بخش، از تابع denoise_wavelet استفاده شده که روش حذف نویز آن را می‌توان در روش آن مشخص کرد. در نتایج حاصل، می‌توان مشاهده کرد که هرچقدر مقدار sigma بیشتر باشد،مقدار نویز حذف شده بیشتر است. همچنین روش BayesShrink مقدار حذف بیشتری را در یک sigma مشخص نسبت به روش VisuShrink برای یک نوع نویز خاص اعمال می‌کند. حال آنکه این تابع بر روی تصاویر با نویز گوسی، نتایج بهتری خروجی می‌دهد. که در تصاویر بالا قابل مشاهد است. این نکته بسیار مهمی است که هر چقدر مقدار sigma بیشتر باشد، عمل حذف نویز بیشتر انجام می‌شود. در عکس های بالا، تفاوت میان عکس های با مقدار $\sigma = 1$ و دیگر مقادیر آن تعداد نویز های موجود پس از عمل حذف نویز قابل مشاهده است.

به صورت کلی، روش BayesShrink مقداری بهتر از روش VisuShrink عمل حذف نویز را انجام داده است با این حال، در تصویر با نویز نمک و فلفل مقداری از نویز ها حذف نشده است. یکی از نکات منفی روش VisuShrink، شطرنجی کردن آن در مقایر بالاتر σ می باشد که مقداری شکل تصویر را بهم می‌ریزد.

شکل۲۲- تصویر حاصل از حذف نویز نمک و فلفل با روش BayesShrink و $\sigma = \frac{1}{2}$

شکل ۲۳- تصویر حاصل از حذف نویز نمک و فلفل با روش BayesShrink و $\sigma = \frac{1}{4}$

منابع

[1] Digital Image Processing, Rafael C. Gonzalez, Rechard E. Wood

[2]https://pywavelets.readthedocs.io/en/latest/ref/2d-dwt-and-idwt.html

[3]https://scikit-image.org/docs/dev/auto_examples/filters/plot_denoise_wavelet.html

Appendix

```
import cv2
import numpy as np
import math
import pywt

def avg_downsample(img, rate):
   avg_downsampled =np.zeros([int(np.size(img, 0)/rate), int(np.size(img,
1)/rate)], dtype = np.uint8)

   for i in range(np.size(avg_downsampled, 0)):
       for j in range(np.size(avg_downsampled, 1)):

           summ = 0
           #calculate the average
           for x in range(i*rate, i*rate + rate):
               for y in range(j*rate, j*rate + rate):
                   summ += img[x, y]

           avg = round(summ / (rate*rate))
           avg_downsampled[i, j] = avg

   return avg_downsampled

#-----------------------------------------------------

def replication_upsample(img, rate):
   upsampled = np.zeros([np.size(img, 0)* rate, np.size(img, 1)* rate], dtype =
np.uint8)

   for i in range(np.size(upsampled, 0)):
       for j in range(np.size(upsampled, 1)):

           upsampled[i, j] = img[math.floor(i/2), math.floor(j/2)]
```

```
    return upsampled

#-----------------------------------------------------------------------
# this method applys the input filter matrix on the input image
# my_filter is the matrix of filter
# returns the result of the input my_filter on img
def filtering(original_img, my_filter):
    filter_dim = np.size(my_filter, 0)
    filter_center = int(filter_dim/2)
    img = np.pad(original_img, (filter_dim -1, filter_dim-1), 'reflect')
    output = np.zeros_like(img, dtype=int)

    for i in range(np.size(img, 0)-filter_dim+1):
        for j in range(np.size(img, 1)-filter_dim+1):
            img_part = img[i:i+filter_dim, j:j+filter_dim]
            value = np.sum(np.multiply(img_part, my_filter))
            output[i+filter_center, j+filter_center ] = value
    output = unpad(output, filter_dim)
    # output = normal(output)
    return output
#-----------------------------------------------------------------------
# this method removes the padding
# takes the image with padding(as img) and size of the filter(as filter_dim)
# returns a image with the original size
def unpad(img, filter_dim):
    original_img = img[filter_dim-1 : np.size(img, 0)-filter_dim+1, filter_dim-1
: np.size(img, 1)-filter_dim+1]
    return original_img
#-----------------------------------------------------------------------
def normal(img):
    min_val = np.amin(img)
    max_val = np.amax(img)
    output = np.zeros_like(img)

    for i in range(np.size(img, 0)):
        for j in range(np.size(img, 1)):
            new_val = math.floor(((img[i, j] - min_val)*255) / (max_val -
min_val))
            output[i, j] = new_val
    return output
#-----------------------------------------------------------------------
def downsample(img, rate):
    downsampled =np.zeros([int(np.size(img, 0)/rate), int(np.size(img,
1)/rate)], dtype = np.uint8)
```

```
    for i in range(np.size(downsampled, 0)):
        for j in range(np.size(downsampled, 1)):
            downsampled[i, j] = img[i*rate, j*rate]

    return downsampled
#-----------------------------------------------------------------------

def wavelet_transform (img, level):
    # img = cv2.imread('./images/Lena.bmp', cv2.IMREAD_GRAYSCALE)
    coeffs = pywt.wavedec2(img, 'haar', level=level)



    [cA, (cH1, cV1, cD1), (cH2, cV2, cD2), (cH3, cV3, cD3)] = coeffs

    output = pywt.waverec2(coeffs, 'haar').astype('uint8')


    coeffs[0]= normal(coeffs[0])
    for i in range(1,len(coeffs)):
        coeffs[i] = [normal(d) for d in coeffs[i]]

    arr = pywt.coeffs_to_array(coeffs)


    return arr[0], output
#-----------------------------------------------------------------------
def wavelet_compression(img, l, gamma):
    coeff = pywt.wavedec2(img, 'haar', level=l)

    coeff[0]= gamma * np.sign(coeff[0]) * np.floor(coeff[0] / gamma)
    for i in range(1,len(coeff)):
            coeff[i] = [gamma * np.sign(d) * np.floor(d / gamma) for d in
coeff[i]]

    out = pywt.waverec2(coeff, 'haar').astype('uint8')

    coeff[0]= normal(coeff[0])
    for i in range(1,len(coeff)):
        coeff[i] = [normal(d) for d in coeff[i]]

    array = pywt.coeffs_to_array(coeff)
    return normal(array[0]), normal(out)
#-----------------------------------------------------------------------
```

```python
def psnr(img1, img2):
    mse = np.mean( (img1 - img2) ** 2 )
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
#_________________6.1.1_________________

lena = cv2.imread("image/Lena.bmp", 0)
cv2.imwrite("output/lena.jpg", lena)

gaussian_pyramid = []
laplacian_pyramid = []
lena_shape = lena.shape
level = math.log2(lena_shape[0])
gaussian = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]]) / 16

current_img = lena
last_level = 0
for i in range(int(level) + 1):
    if(i < int(level)):
        g = filtering(current_img, gaussian)
        l = current_img - g
        gaussian_pyramid.append(current_img)
        laplacian_pyramid.append(l)
        current_img = avg_downsample(g, 2)
    else:
        last_level = current_img

for i in range(len(gaussian_pyramid)):
    cv2.imwrite('output/6_1_1/G' + str(i) + '.jpg', gaussian_pyramid[i])
    cv2.imwrite('output/6_1_1/L' + str(i) + '.jpg',
normal(laplacian_pyramid[i]))

#_________________6.1.2 -> reconstruction _________________

laplacian = laplacian_pyramid
level_num = int(level)

current = gaussian_pyramid[3]

for i in range(len(laplacian) - 6):
    gauss = replication_upsample(current, 2)
    # dim = (current.shape[0]*2, current.shape[1]*2)
    # current = current.astype('float64')
```

```
    # gauss = cv2.resize(current, dim, interpolation = cv2.INTER_LINEAR)
    lap = laplacian[level_num - i -7]
    current = gauss + lap
    current = normal(current)

# cv2.imwrite('output/6_1_1/reconstructed_replication.jpg', current)
#_____________________6.1.3_____________________________
wave_img, rec = wavelet_transform(lena, 3)
cv2.imwrite('output/6_1_3/wave_img.jpg', wave_img)
cv2.imwrite('output/6_1_3/reconstructed.jpg', rec)

#_____________________6.1.4_____________________________

arr_img, out_img = wavelet_compression(lena, 3, 2)

cv2.imwrite('output/6_1_4/rec.png', out_img)
cv2.imwrite('output/6_1_4/wave.png', arr_img)

#_____________________6.2_____________________________
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.color import rgb2gray
from skimage.io import imsave

import cv2
import math
import numpy as np



def psnr(img1, img2):
    mse = np.mean( (img1 - img2) ** 2 )
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))


original = cv2.imread('./image/Lena.bmp', cv2.IMREAD_GRAYSCALE)
original = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
```

```
# sigma = 0.12
sigma = 0.3
# noisy = random_noise(original, var=sigma**2)

noisy = cv2.imread('./image/noisy/salt.jpg', cv2.IMREAD_GRAYSCALE)
noisy = cv2.cvtColor(noisy, cv2.COLOR_GRAY2BGR)

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, multichannel=True, average_sigmas=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 = denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True,
                           method='BayesShrink', mode='soft',
                           rescale_sigma=True, wavelet='haar')


im_bayes2 = denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True,
                           method='BayesShrink', mode='soft',
                           rescale_sigma=True, wavelet='haar',sigma=sigma_est/2)


im_bayes4 = denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True,
                           method='BayesShrink', mode='soft',
                           rescale_sigma=True, wavelet='haar',sigma=sigma_est/4)

im_visushrink = denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True,
                                method='VisuShrink', mode='soft',
                                sigma=sigma_est, rescale_sigma=True,
wavelet='haar')

# 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,
method='VisuShrink',mode='soft',sigma=sigma_est/2,rescale_sigma=True,
wavelet='haar')
im_visushrink4=denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True,
                                 method='VisuShrink', mode='soft',
```

```
                                sigma=sigma_est/4, rescale_sigma=True,
wavelet='haar')



imsave('./output/6_2/noisy.png', noisy)
imsave('./output/6_2/original.png', original)
imsave('./output/6_2/im_bayes.png', im_bayes)
imsave('./output/6_2/im_bayes2.png', im_bayes2)
imsave('./output/6_2/im_bayes4.png', im_bayes4)
imsave('./output/6_2/im_visushrink.png', im_visushrink)
imsave('./output/6_2/im_visushrink2.png', im_visushrink2)
imsave('./output/6_2/im_visushrink4.png', im_visushrink4)
```