# تمرین wavelet

آریا ابراهیمی ۹۸۲۲۷۶۲۱۷۵

**چکیده**

هدف از انجام این تمرین،آشنایی با هرم ها و تبدیلات pyramid است. در ابتدا برای تمرین اول، تابعی تعریف شده است تا هرم گوسی و هرم لاپلاسین مربوط به عکس را می‌سازد. این تابع، تابعی general است و برای حل پرسش های ابتدایی از آن استفاده میشود. همچنین تابعی معکوس برای محاسبه عکس اصلی از روی هرم گوسی و هرم لاپلاسین طراحی شده است. این تابع نیز به صورت کلی عمل میکند ومیتوان خروجی های تابع مربوط به ساخت هرم را برای ساخت مجدد عکس به این تابع به عنوان ورودی داد.

در ادامه، برای سوال های آخر که مربوط به مبحث wavelet هستند، همانند قسمت گذشته، تابعی general تعریف شده است که تبدیل wavelet را بر روی عکس اعمال میکند که در تمرین با جزئیات آن آشنا میشویم. همچنین تابعی معکوس برای بازگشت به عکس اصلی از روی ضرایب wavelet نیز طراحی شده است.

## ۱- تحلیل تکنیکال

**۱.۱**- همانطور که گفته شد، تابعی general برای ساخت هرم گوسی و لاپلاسین طراحی شده است. این تابع به عنوان ورودی مقادیر عکس، تعداد سطح های هرم، فیلتر، تابعی برای upsample و تابعی برای downsample دریافت میکند.

در بدنه این تابع حلقه ای به ازای مقدار وارد شده برای سطح های هرم وجود دارد که در هر iteration ، ابتدا تابعی برای padding به عکس اعمال میشود. این padding به این صورت است که اگر مقادیر سایز عمس عددی فرد بودند، تبدیل به عددی زوج شود. این کار برای این انجام میشود که در مرحله محاسبه هرم لاپلاسین به مشکل نخوریم. (برای عکس Mona-Lisa زمانی که ورژن upsample شده عکس را از عکس مرحله فعلی کم کنیم، ارور میدهد زیرا اندازه ها یکسان نخواهند بود) با اعمال این پدینگ، چون اندازه تصویر مقداری زوج میشود، در downsample و سپس upsample اندازه تصویر تغییری نمیکند.

در ادامه، عملیات convolution انجام میشود. در این مرحله از کرنلی که در ورودی به تابع هرم داده شده است استفاده میشود. بعد از این عملیات، دو ورژن از عکس ساخته میشود. یکی به نام downsampled که با استفاده از تابع downsample که در ورودی داده شده است انجام میشود و یکی دیگر هم upsampled که با استفاده از تابع upsample ورودی است.

ورژن downsample شده درواقع همان تصویر مربوط به سطح فعلی هرم گوسی است.

از ورژن upsample شده برای ساخت تصویر مربوط به هرم لاپلاسین در سطح فعلی استفاده میشود. با کم کردن مقدار upsampled از تصویر مرحله قبل هرم گوسی، میتوان به تصویر مربوط به هرم لاپلاسین در سطح فعلی رسید.

نتایج مربوط به هرم لاپلاسین و هرم گوسی به دو صورت ذخیره میشوند. یک حالت که تمامی تصویر های مربوط به هرم در یک ماتریس ذخیره میشوند که صرفا کاربرد نمایش دارد. حالت دیگر هر سطح را در یک لیست اضافه میکنیم. این لیست ها برای ساخت مجدد تصویر استفاده میشوند.

**۱.۲**- فرض کنید فیلتری گوسی با سایز $M \times M$ داریم و سایز تصویری که فیلتر قرار است بر روی آن اعمال شود $N \times N$ باشد، در نتیجه پیچیدگی محاسباتی عملیات convolution ، $O(N^2M^2)$ خواهد بود.

یک فیلتر دوبعدی separable نامیده میشود اگر بتوان آنرا به دو بردار یک بعدی تجزیه کرد به صورتی که ضرب ماتریسی آنها برابر با فیلتر اصلی دوبعدی شود.

برای مثال ماتریس $G$ یک فیلتر جدا پذیر است زیرا میتوان آنرا به دو بردار $g_1$ و $g_2$ تجزیه کرد. ($G = g_1.g_2$)

$$G = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$g_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad g_2 = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

زمانی که فیلتر جداپذیر باشد، عملیات convolution را میتوان در دو گام انجام داد.

در گام اول عملیات convolution را با استفاده از بردار اول انجام میدهیم. پیچیدگی زمانی این عملیات $O(N^2M)$ خواهد بود. در گام بعد بردار دوم را با نتیجه مرحله قبل convolve میکنیم. پیچیدگی این بخش هم همانند قسمت قبل خواهد بود.

با استفاده از این خاصیت پیچیدگی زمانی عملیات به $O(N^2M)$ کاهش پیدا میکند.

مفهوم cascading filters ، به معنی اعمال چندین فیلتر پشت سر هم به جای اعمال فیلتری بزرگتر است. برای مثال اگر بردار گوسی نرمال نشده $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$ را درنظر بگیریم، با convolve کردن در خودش به فیلتر گوسی مرحله بالاتر میرسیم.

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

از آنجایی که پیچیدگی زمانی عملیات convolution همانطور که قبلا گفته شد $O(N^2M^2)$ است، با کاهش سایز فیلتر این پیچیدگی کمتر خواهد شد.

برای ایجاد هرم گوسی با استفاده از مفاهیم بالا میتوانیم به صورت زیر عمل کنیم.

۱. در سطح اول هرم، فیلتر گوسی با انحراف معیار $\sigma$ را به تصویر ورودی اعمال میکنیم. از آنجایی که فیلتر گوسی جدا پذیر است، این کرنل را میتوانیم به دو وکتور تبدیل کنیم که باعص سرعت بخشیدن به عملیات convolution می‌شود.

۲. در گام بعدی که مربوط به سطح بعدی است، فیلتر گوسی با انحراف معیار $\sqrt{2}\sigma$ را به عکس خروجی سطح قبل اعمال میکنیم. همانند سطح گذشته این فیلتر را به دو بردار یک بعدی تبدیل میکنیم و سپس عملیات convolution را انجام میدهیم تا پیچیدگی زمانی بهتری داشته باشیم.

۳. در گام آخر که مربوط به سطح آخر است، دو بردار گوسی جدا شده از فیلتر گوسی با انحراف معیار $2\sigma$ را به تصویر سطح قبلی اعمال میکنیم.

از آنجایی که از کانسپت فیلتر های جدا پذیر استفاده کردیم و هرم هم نوعی cascading فیلتر است، این الگوریتم پیچیدگی زمانی کمتری نسبت به ساخت هرم به روش معمولی دارد.

**۱.۳**- از آنجایی که تصویر در هر مرحله downsample میشود (تقسیم بر دو) و اندازه تصویر توانی از ۲ است، میتوان آنرا همانند عددی باینری در نظر گرفت که در هر سطح یک بیت به راست شیفت داده میشود. اگر $N$ برابر با $2^j$ باشد، بعد از $j$ شیفت به راست، سایز تصویر حاصل برابر با $1$ خواهد شد. بنابر این بیشینه سطح قابل ساخت، برابر با $j$ میباشد.

فرمولی که در زیر بیان شده است، برای محاسبه تمامی پیکسل ها در هرم است. عبارت جمع را میتوان به $\frac{4}{3}N^2$ تقریب زد که $N$ در آن، برابر با سایز تصویر اصلی میباشد.

$$p = N^2 + \frac{N^2}{4} + \frac{N^2}{16} + ... + \frac{N^2}{2^j} \approx \frac{4}{3}N^2$$

همانطور که بیان شد، تعداد پیکسل های مورد نیاز برای ساخت هرم تقریبا ۳۳ درصد بیشتر از پیکسلهای تصویر اصلی است و حجم بیشتری نیاز دارد. اما فواید و کاربرد هایی که هرم ها ارائه میکنند وزن بیشتری نسبت به حجم بیشتر نگه داری آنها دارد.

- Coarse to Fine strategies: با استفاده از هرم ها، میتوانیم (scale space) سطوح مختلف تصویر را داشته باشیم و این سطوح میتوانند برای شناسایی آبجکت استفاده شوند. از آنجایی که پیدا کردن یک آبجکت در یک تصویر زمان بر است، میتوان آنرا در سطح پایینتر که اندازه کوچکتری دارند انجام داد و باعث کاهش پیچیدگی زمانی میشود.
- Image Blending: هرم های لاپلاسی میتوانند برای ترکیب دو عکس استفاده شوند. اگر در حالت عادی دو عکس را با یکدیگر ترکیب کنیم، تصویری غیر واقعی با لبه خیلی واضحی در مرز دو تصویر ایجاد خواهد شد.

اما اگر از هرم لاپلاسین دو تصویر استفاده کنیم میتوان با استفاده از آنها، هرم سومی تشکیل داد که باعث میشود تصویر واقعی تری داشته باشیم. (در قسمت هایی که مربوط به تصویر اول است از هرم لاپلاسی تصویر اول استفاده میکنیم و قسمت هایی که مربوط به تصویر دوم است را از مقادیر هرم لاپلاسی تصویر دوم استفاده میکنیم و در مرز میتوان از میانگین مقادیر هرم لاپلاسی دو تصویر استفاده کرد)

- Image Compression : هرم های لاپلاسی همچنین میتوانند برای کاهش حجم عکس ها نیز استفاده شوند. به جز سطح آخر هرم لاپلاسی که ورژن downsample شده عکس اصلی میباشد، بقیه سطح ها در هرم لاپلاسی مقادیر لبه ها هستند که معمولا یک ماتریس sparse هستند به این معنی که مقادیر ۰ و بدون اهمیت زیادی دارند. در نتیجه میتوان با استفاده از روش های ذخیره ماتریس sparse آنها را ذخیره کرد که باعث این مشوند که نیاز به ذخیره حجم کمتری از اطلاعات دارند و در کل باعث کاهش حجم عکس میشوند. (تنها اطلاعات ورژن کوچک عکس را نگهداری میکنیم و لبه ها که برای ساخت مجدد عکس استفاده میشوند را به صورت sparse نگهداری میکنیم)

**۱.۴**- برای این سوال همان تابع general در سوال اول استفاده شده است با این تفاوت که به جای کرنل گاسین، فیلتر میانگین یا همان box filter به عنوان ورودی به تابع داده شده است.

**۱.۵**- برای حل این سوال، همانند سوال اول که تابعی general برای ساخت هرم طراحی شد، تابعی برای محاسبه تبدیل موجک ارائه شده است.
این تابع مقادیر عکس، تعداد سطوح مورد نظر برای اعمال تبدیل موجک و یک mode که به صورت پیش‌فرض Haar است را دریافت میکند. خروجی های این تابع، ۱- Wrep که ماتریسی است که برای نمایش wavelet استفاده میشود. ۲- LLrep که مقادیر LL در هر سطح موجک را نگهداری میکند، ۳-coeffs که ضرایب تبدیل موجک مربوط به هر سطح میباشد و ۴- LL که مقدار LL سطح آخر است.
از خروجی های ۳ و ۴ برای بازسازی تصویر استفاده میشود.

در بدنه اصلی تابع، حلقه ای به ازای سطوح موجک وجود دارد که در هر iteration آن، ابتدا با استفاده از کتابخانه مربوط به تبدیل موجک و با استفاده از تابع dwt2 چهار مقدار ، LH ، LL HL و HH محاسبه میشوند و سه مقدار LH و HL و HH در Wrep برای نمایش قرار میگیرند همچنین در لیست coeffs نیز اضافه میشوند. از آنجایی که هدف این سوال مقایسه با سوال گذشته است، مقادیر LL همانند فرمت هرم ها داخل ماتریسی ذخیره میشوند که LLrep است.
از آنجایی که در هر سطح تبدیل، مقادیر LL ضرب در دو میشوند، برای ذخیره در ماتریس LL مقدار آنها را تقسیم بر دو میکنیم که مقادیر بین ۰ تا ۲۲۵ scale شوند.

**۱.۶**- برای حل این سوال دو تابع طراحی شده اند. تابع اول فرمول ارائه شده را به ضرایب تبدیل اعمال میکند. فرمول داده شده به این صورت عمل میکندکه با استفاده از مقدار گاما که دریافت میکند، ضرایب را گسسته میکند و دروائع تمامی ضرایب مضربی از مقدار گاما میشوند. از این روش هم میتوان برای کاهش حجم استفاده کرد زیرا مقادیر موجود کمتر میشوند.

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

تابع بعدی، برای بازسازی تصویر اصلی از روی LL سطح آخر و ضرایب تبدیل است. این تابع شامل یک حلقه است که روی سطوح پیمایش میکند. در هر iteration ، معکوس تبدیل موجک با استفاده از تابع idwt2 و LL و ضرایب آن سطح محاسبه میشود. خروجی این تابع درواقع LL سطح بعدی خواهد بود. این روند برای تمامی سطوح طی میشود تا بالاخره در سطح آخر، تصویر اصلی تشکیل میشود.

در پایان این تمرین خواسته شده است تا مقدار PSNR محاسبه و بررسی شود. این مقدار، معیاری است که نسبت بین بیشینه مقداری که سیگنال میتواند اختیار کند و توان نویز تخریب کننده را بررسی میکند. برای یک تصویر و ورژن کامپرس شده با استفاده از مقادیر ۸ بیتی، این مقدار اگر بین ۳۰ تا ۵۰ باشد نشان دهنده حفظ کیفیت خوبی میباشد.
این مقدار با استفاده از قرمول زیر محاسبه میشود.

$$PSNR = 20 \cdot log_{10}(MAX_I) - 10 \cdot log_{10}(MSE)$$

## ۲- بررسی نتایج

### ۲.۱-

تصویر شماره ۱ و تصویر شماره ۲، به ترتیب نشان دهنده هرم گوسی و هرم لاپلاسی برای تصویر مونالیزا هستند.

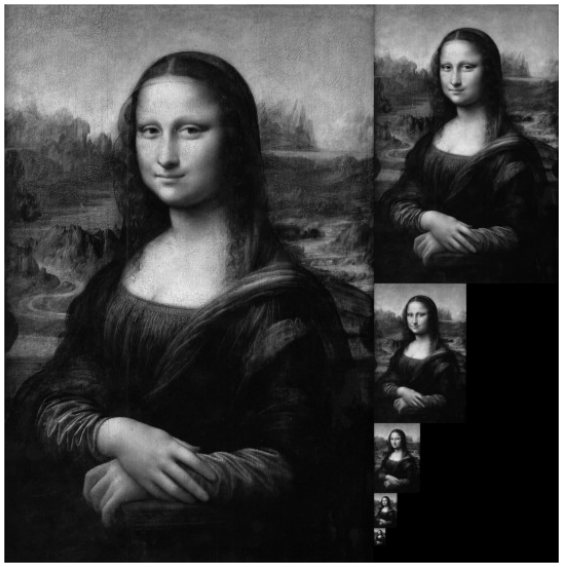

*Figure ۱: Gaussian pyramid of the Mona-Lisa image. The left part is the original image, and the right is the Gaussian pyramid's levels.*

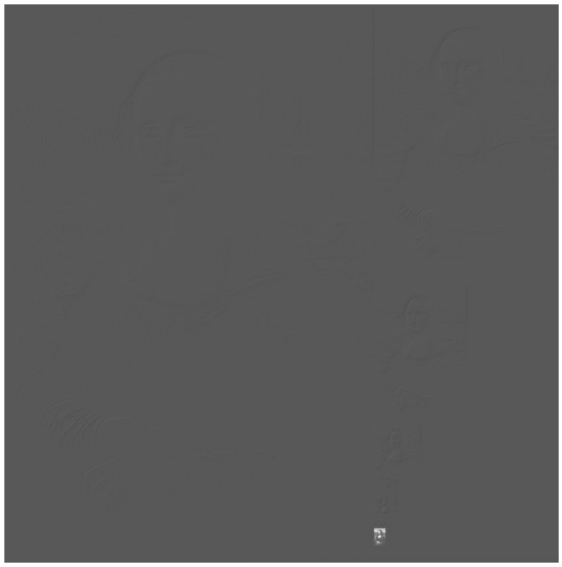

*Figure ۲: Laplacian pyramid of the Mona-Lisa image. The left part is the original image, and the right is the Laplacian pyramid's levels. Note that the last level of the Gaussian and Laplacian is the same.*

این تصاویر نتیجه تابع general محاسبه هرم میباشد و همانطور که مشاهده میشود، تصاویر در هر سطح، downsample شده اند. و با استفاده از این دو هرم میتوان تصویر اصلی را بازسازی کرد.

**۲.۲-** این تمرین به صورت الگوریتم در بخش قبل بیان شد و پیاده سازی و نتیجه ندارد.

**۲.۳-** برای حل این سوال از آنجایی که تصویری مشخص نشده بود، از تصویر camera-man برای حل استفاده شده است. تصویر camera-man تصویری $512 \times 512$ است و بنابراین $N = 2^j = 512$ و در نتیجه $j = 9$ میباشد. هرم های گوسی و لاپلاسی با ۹ سطح برای این تصویر ساخته شده است که در شکل های ۳ و ۴ قابل مشاهده است.

*Figure ۳: 9-level Gaussian pyramid of the camera man picture.*

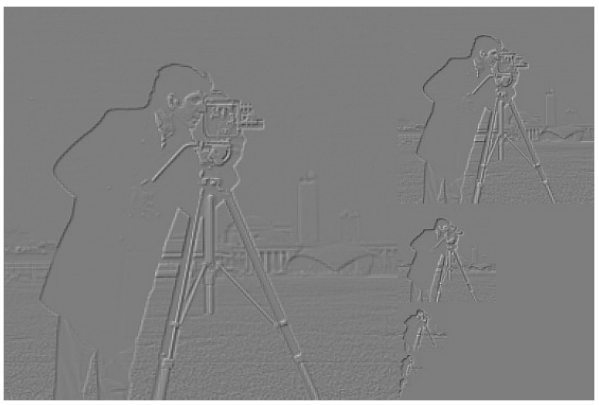

*Figure ۴: 9-level Laplacian pyramid of the camera man picture.*

**۲.۴-** در این تمرین با استفاده از همان تابع general سوال اول، هرم های گوسی و لاپلاسی برای تصویر Lena ساخته شده است. برای کرنل ورودی این تابع، فیلتر میانگین $2 \times 2$ استفاده شده است.

$$f = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

نتایج در شکل های ۵ و ۶ قابل مشاهده است.


*Figure ۵: 3-level pyramid using the averaging filter for Lena image.*

*Figure ۶: 3-level Laplacian pyramid for Lena image.*

**۲.۵**- نتیجه تبدیل موجک با استفاده از تابعی که در قسمت قبل توضیح داده شد محاسبه شده است. ازآنجایی که تابع موجک خواسته شده دارای سه سطح است، ۱۰ تصویر در Wrep وجود خواهد داشت که ۳ تا مربوط به LH و HL و HH هر لایه و یک LL مربوط به سطح آخر است. برای لایه های ابتدایی، LL با استفاده از LL لایه قبل و ضرایب آن لایه قابل محاسبه است.

در شکل ۸، ماتریس Wrep قابل مشاهده است که در آن LL لایه آخر به همراه ضرایب لایه آخر و لایه های ابتدایی تر نمایش داده شده است.

*Figure ۷: All the LLs in the wavelet transformation process.*

*Figure ۸: 3-level wavelet transform output.*

همچنین همانطور که گفته شد برای مقایسه با سوال ۴، نمایش دیگری از LL ها ساخته شده است که در LLrep ذخیره شده است. این ماتریس در تصویر ۷ قابل مشاهده است.

با مقایسه هرم گوسی و LL ها در تبدیل موجک، میتوان مشاهده کرد که در موجک، عکس ها کیفیت بهتری دارند. قسمت هایی از هر کدام در تصویر های ۹ و ۱۰ قابل مشاهده هستند که میتوان آنها را با یکدیگر مقایسه کرد.

*Figure ۹: Pyramid with averaging filter final levels result.*



*Figure ۱۰: Wavelet transform. LL of the final levels.*

**۲.۶**- در تمرین آخر، از ما خواسته شده است تا ضرایب را گسسته کنیم و با استفاده از آن ضرایب تصویر را بازسازی کنیم.

همان طور که گفته شد تابعی برای بازسازی تصویر ساخته شده است که ضرایب تمامی لایه های و LL لایه آخر را میگیرد و تصویر را برمیگرداند.
حال ابتدا ضرایب را با استفاده از تابع اول که فرمول را به ضرایب اعمال میکند گسسته میکنیم و ضرایب گسسته شده را به همراه تصویر LL لایه آخر به تابع دوم برای ساخت تصویر میدهیم.
همانطور که در تصویر ۱۱ قابل مشاهده است، گسسته سازی تصویر میتواند کیفیت تصویر را کمتر کند ولی همچنین میتواند سایز تصویر را نیز کمتر کند. با استفاده از مقدار PSNR میتوانیم مقایسه کنیم که آیا تصویر گسسته شده خوب است یا خیر. برای گاما با مقدار ۲ که گفته شده بود، مقدار PSNR برابر با ۴۶.۷۷۱ شده است که مقدار خیلی خوبی است و نشان میدهد که نویز های ایجاد شده خیلی تصویر اصلی را خراب نکرده اند. اگر از گاما برابر با ۸ استفاده کنیم این مقدار برابر با ۳۷.۲۳۶ میشود که مقدار بدتری نسبت به گاما برابر با ۲ است. هرچقد مقدار گاما افزایش پیدا کند، تصویر تشکیل شده از تصویر اصلی بیشتر فاصله میگیرد ولی کاهش حجم بیشتری نیز امکان پذیر است.

*Figure ۱۱: The reconstructed Lena image using* $\gamma = 2$.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math
from pywt import dwt2, idwt2

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 = cv2.imread('Lena.bmp')
lena = cv2.cvtColor(lena, cv2.COLOR_BGR2GRAY)

mona_lisa = cv2.imread('mona-lisa.jpg')
mona_lisa = cv2.cvtColor(mona_lisa, cv2.COLOR_BGR2GRAY)

camera_man = cv2.imread('camera_man.bmp')
camera_man = cv2.cvtColor(camera_man, cv2.COLOR_BGR2GRAY)

#----------------------- general pyramid function -----------------------#

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

def pixel_replication(img, k):
    res = np.repeat(img, k, axis=0)
    res = np.repeat(res, k, axis=1)
    return res

def average_downsample(img, filter_size):
    filter = np.ones([filter_size, filter_size])
    filter /= filter_size**2

    r = np.zeros((math.floor(img.shape[0]/filter_size), math.floor(img.shape[1]/filter_size)))
    for i in range(math.floor(img.shape[0]/filter_size)):
        for j in range(math.floor(img.shape[1]/filter_size)):
            mat = img[i*filter_size:(i+1)*filter_size, j*filter_size:(j+1)*filter_size]
            mul = mat * filter
            sum = np.sum(mul)
            r[i, j] = sum

    return r

def remove_row_col(img, k):
    return img[::k,::k]


def pad(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

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

def pyramid(img, level, kernel=gaussian_kernel, upsample_function=pixel_replication,
downsample_function=remove_row_col):
    img = pad(img)
    G = np.zeros((img.shape[0], int(1.5*img.shape[1])))
    L = np.zeros((img.shape[0], int(1.5*img.shape[1])))

    g = []
    l = []

    G[0:img.shape[0], 0:img.shape[1]] = img
    g.append(img)

    imgcol = img.shape[1]
    pointer_g = 0
    pointer_l = 0
```

```
    for i in range(level):
        img = pad(img)
        kernelized = cv2.filter2D(img, -1, kernel)
        downsampled = downsample_function(kernelized, 2)
        upsampled = upsample_function(downsampled, 2)
        if (i == 0):
            L[:, 0:img.shape[1]] = img-upsampled
        else:
            L[pointer_l:pointer_l+img.shape[0], imgcol:imgcol+img.shape[1]] = img-upsampled
            pointer_l += img.shape[0]
        l.append(img-upsampled)
        img = downsampled
        G[pointer_g:pointer_g+img.shape[0], imgcol:imgcol+img.shape[1]] = img
        g.append(img)
        if (i == level-1):
            L[pointer_g:pointer_g+img.shape[0], imgcol:imgcol+img.shape[1]] = img
            l.append(img)
        pointer_g += img.shape[0]

    return G, L, g, l

#----------------------- 5.1.1 -----------------------#


g, l, _1, _2 = pyramid(mona_lisa, 5)
show_img(g)
show_img(l)

#----------------------- 5.1.3 -----------------------#


g1, l1, g2, l2 = pyramid(camera_man, 9)

show_img(g1)
show_img(l1)

def rec_pyr(g, l, upsample_function):
    n = len(l)-1
    gg = g[-1]
    for i in range(n, 1, -1):
        upsampled = upsample_function(gg, 2)
        gg = upsampled + l[i-1]

    return gg

show_img(rec_pyr(g2, l2, pixel_replication))

#----------------------- 5.1.4 -----------------------#


g2, l2, _, _ = pyramid(lena, 3, kernel = np.array([[0.25, 0.25], [0.25, 0.25]]))

show_img(g2, title=['gaussian pyramid of lena with 3 levels'])
show_img(l2)

#----------------------- 5.1.5 -----------------------#

def wavelet(img, level, mode='haar'):

    Wrep = np.zeros((img.shape[0], img.shape[1]))
    LLrep = np.zeros((img.shape[0], int(1.5*img.shape[1])))
    LLrep[0:img.shape[0], 0:img.shape[1]] = img
    img_width = img.shape[1]
    pointer = 0
    end = (img.shape[0], img.shape[1])
    coeffs = {}

    for i in range(level):
        LL, (LH, HL, HH) = dwt2(img, 'haar')

        coeffs[i] = (LH, HL, HH)

        Wrep[0:LL.shape[0], 0:LL.shape[1]] = LL
        Wrep[0:LH.shape[0], LL.shape[1]:end[1]] = LH
        Wrep[LL.shape[0]:end[0], 0:HL.shape[1]] = HL
        Wrep[LL.shape[0]:end[0], LL.shape[1]:end[1]] = HH

        LLrep[pointer:pointer + LL.shape[0], img_width:img_width + LL.shape[1]] = LL/(2**(i+1))
        pointer += LL.shape[0]

        end = (int(end[0]/2), int(end[1]/2))
        img = LL

    return Wrep, LLrep, coeffs, LL

wavelet, llrep, coeffs, LL = wavelet(lena, 3)
show_img(wavelet)
show_img(llrep)
```

```
178
179 #----------------------- 5.1.6 -----------------------#
180
181
182 def quantize_coeffs(coeffs, gamma=2):
183     def formula(x):
184         return gamma * np.sign(x) * np.floor(abs(x/gamma))
185
186     for i in range(len(coeffs)):
187         lh, hl, hh = coeffs[i]
188         coeffs[i] = (formula(lh), formula(hl), formula(hh))
189
190     return coeffs
191
192 coeffs_qunatized = quantize_coeffs(coeffs, gamma=2)
193
194 def rec(ll, coeffs):
195     for i in range(len(coeffs)-1, -1, -1):
196         ll = idwt2([ll, coeffs[i]], 'haar')
197
198     return ll
199
200 quan = rec(LL, coeffs_qunatized).astype('uint8')
201 show_img(quan)
202
203 cv2.PSNR(lena, quan)
204
205 show_img(lena)
206
207
```