

تبدیل موجک

سحر شیخ الاسلامی

اطلاعات گزارش	چکیده
در این تمرین ابتدا به معرفی تبدیل موجک می پردازیم. سپس دلایل معرفی این تبدیل و نقاط قوت آن را بررسی می کنیم. سپس با تبدیل موجک و کاربردهای آن نظری فشرده سازی و از بین بردن نویز آشنا می شویم. همچنین با هرم های رزولوشن و کاربرد های آن نیز آشنا می شویم.	تاریخ: ۲۱/۱۰/۱۳۹۹
واژگان کلیدی: Wavelet Transform Gaussian Pyramid Laplacian Pyramid Denoising Haar Filter	

دارد. هر بار تصویر را downsample کردن برخی از جزئیات تصویر را از بین می برد و کلیت عکس باقی می ماند. به همین دلیل هر بخش عکس را می توان گفت در سطوح رزولوشنی مختلفی قرار دارد. با اعمال عملیات بر روی یک سطح رزولوشن و downsample کردن به یک سطح پایینتر می رویم. این کار را به اندازه ی دلخواه ادامه می دهیم. می توان ساخت هرم را تا آنجا ادامه داد تا به یک پیکسل برسیم. تمام این سطوح در کنار هم هرم رزولوشن را تشکیل می دهند. برای downsample کردن می توان از روش های متفاوتی استفاده کرد. در این قسمت ما با استفاده از فیلتر گوسی تصویر را smooth می کنیم. مراحل انجام کار به شکل زیر خواهد بود:

1. ابتدا تصویر فعلی این مرحله را از یک فیلتر گوسی عبور می دهیم. فیلتر گوسی یک فیلتر smooth کننده است که در مراحل قبل با آن آشنا شده ایم.

2. تصویر بدست آمده از مرحله قبل را downsample می کنیم. برای انجام این کار می توان به جای هر پنجره 2×2 در عکس اصلی، یک پیکسل در تصویر این مرحله قرار دارد. این پیکسل میتواند میانگین پیکسل های پنجره باشد یا می توان یک نماینده مثلاً پیکسل چپ بالا باشد.

۱- مقدمه
با تبدیل فوریه و کاربردهای بی شمار آن در قسمت های قبل آشنا شدیم. یکی از ایرادات اصلی تبدیل فوریه، در مورد کار با سیگنالهای غیر ایستا است. تبدیل فوریه در کار با این سیگنالها به شدت ضعیف عمل می کند. این ضعف را با استفاده از تبدیل SFTF می توان تا حدی حل کرد. اما سایز پنجره ثابت یکی از مشکلات اصلی این تبدیل است. اصل عدم قطعیت هایزنبرگ نشان می دهد که SFTF نیز نمی تواند اطلاعات مناسبی درباره هر دو حوزه زمان و فرکانس بدهد. به همین دلیل در این موقع از تبدیل موجک استفاده می کنیم. تبدیل موجک با سایز پنچره متغیر، برای سیگنال های متغیر با زمان بسیار مناسب است.

۲- شرح تکنیکال

۲.۱.۱: قسمت

هرم یا نمایش هرمی نوعی نمایش سیگنال multiscale است که در آن یک سیگنال یا یک تصویر تحت smoothing سازی و نمونه برداری مکرر قرار می گیرد. وظیفه نمایش هرمی بازنمایی فضای مقیاس و تجزیه و تحلیل multiresolution است. به طور کلی می توان گفت که اطلاعات تصویر در رزولوشن های مختلفی قرار

$$\begin{aligned} \text{Total pixels original} &= 2^J \times 2^J \\ &= 2^{J+1} \end{aligned}$$

مشاهده می کنیم که تعداد پیکسل های هرم، از تعداد پیکسل عکس اصلی بیشتر است.
هرم گوسین کاربردهای زیادی دارد. از کاربردهای این هرم می توان blending دو تصویر و فشرده سازی نام برد. همچنین این هرم برای پیدا کردن یک جسم در تصویر بسیار کارا است و سرعت خوبی دارد.

هرم لاپلاسین نیز هرمی است که با استفاده از آن می شود به هرم گوسی رسید. با اینکه در بالا مشاهده کردیم ک اندازه این هرم از تصویر اصلی بیشتر است و شاید در نگاه اول به نظر بیاید که این روش در حافظه بسیار بد عمل می کند و دلیلی برای استفاده از آن وجود ندارد. اما نکته قابل توجه این است که ماتریس های شامل تصاویر لاپلاسین تنک (sparse) است. بنابراین با استفاده از روش های نگه داری ماتریس sparse می توان حجم این هرم با به شکل قابل ملاحظه ای کاهش داد.

قسمت ۶.۱.۲:

در قسمت قبل نحوه ایجاد هرم لاپلاسین توضیح دادیم. در این مرحله به بررسی جزئیات بیشتر بازسازی تصویر اصلی از هرم لاپلاسین می پردازیم.

در این قسمت ما هرم ماسه سطح دارد. سطح آخر یک تصویر گوسی و سطوح دیگر نیز لاپلاسین هستند. برای ساخت تصویر در این مرحله تصویر مرحله ای آخر را با ضریبی که upsample کردیم، downsample می کنیم. برای upsample کردن روش های متفاوتی وجود دارد. ما در این سوال از pixel replication استفاده می کنیم. این روش به جای یک پنجره، یک مقدار را به صورت تکراری جایگزین می کند. در بخش های ابتدایی درس خواندیم که این روش بهترین روش نیست و استفاده از درونیابی bilinear نتیجه مطلوب تری می دهد.

پس از upsample کردن تصویر پایین ترین سطح (که تصویر گوسی است) حال این تصویر را با لاپلاسین سطح بالاتر جمع می کنیم. با این روش در واقع گوسی سطح بالاتر بدست می آید. حال با این گوسی این سطح و به کمک لاپلاسین سطح بالاتر تصویر، گوسی سطوح های بالاتر بدست می آید و در نهایت تصویر اصلی می رسیم.

قسمت ۶.۱.۳:

3. حال تصویر فعلی را آبدیت می کنیم و برابر مقدار تصویر به دست آمده از مرحله قبل می گذاریم.
با طی کردن مراحل بالا، هرم گوسی بدست می آید. برای بازسازی تصویر اما علاوه بر هرم گوسی، نیاز به هرم لاپلاسی داریم. هرم لاپلاسین به شکل زیر بدست می آید:

1. مانند هرم گوسی، ابتدا یک فیلتر پایین گذر مانند گوسی را بر روی عکس اعمال می کنیم.
2. تصویر بدست آمده از مرحله قبل را، از تصویر اصلی کم می کنیم.

3. سپس تصویر بدست آمده را downsample می کنیم.

سپس تمام لاپلاسین ها و تصاویر مرحله آخر(تصویر گوسی) را بدست می اوریم.
دقت شود که در هر مرحله سایز تصویر نصف می شود.
می خواهیم بررسی کنیم که این کار را تا چند مرحله می توانیم انجام دهیم:

فرض کنیم تصویرمان N^j باشد که $N = 2^j$ باشد. فرض کنیم در هر مرحله طول و عرض عکس سطح بعد، نصف طول و عرض عکس سطح قبل باشد. در این صورت تعداد سطوح هرم گاوی برابر $\log_2 N$ خواهد بود که برابر $J+1$ است.

در ادامه خواسته شده است که مجموع تمام پیکسل های تمام مراحل را بدست آوریم:

در بالاترین سطح N^*N پیکسل داریم. هر سطحی که پایین می رویم تعداد پیکسل ها تصویر $4/1$ می شود.(طول و عرض نصف می شود).
يعنی داریم:

$$T = 1 * 1 + 2 * 2 + \dots + N * N$$

که این مقدار برابر است با:

$$T = 1 + 2^2 + 2^4 + 2^6 + \dots$$

می دانیم که این یک دنباله هندسی است، بنابراین برای بدست آوردن تعداد کل پیکسل ها از فرمول زیر استفاده می کنیم.

$$\text{Total pixels} = 1 \times \frac{4^{J+1}-1}{4-1}$$

بنابراین خواهیم داشت:

ابتدا درباره موجک و خواص آن توضیح می دهیم.
می دانیم که سیگنال ها را می توان به دو دسته تقسیم کرد.

سیگنال های ایستا:

حاوی اجزای طیفی می باشد که با زمان تغییر نمی کنند.
یعنی در این سیگنال ها تمام اجزای طیفی همیشه وجود دارند و نیازی به اطلاعات زمانی نیست. تبدیل فوریه برای سیگنالهای ایستا خوب عمل می کند.

سیگنال های غیر ایستا:

محتوای طیفی متغیر با زمان دارد. FT در این سیگنال تنها مشخص می کند که چه اجزایی در طیف وجود دارد و نه زمانی که آن طیف ها وجود دارند. به همین دلیل نیاز به روش هایی برای تعیین زمانی اجزای طیفی است.

تبدیل فوریه(FT) تمامی اجزای موجود در دل سیگنال را شناسایی می کند، اما هیچ اطلاعاتی در خصوص مکان این اجزا ارائه نمی کند.

تبدیل فوریه اطلاعات موجود در تصویر را بیان می کند، اما پاسخ به محل وقوع را نخواهد داد.

بیان تصویر در حوزه مکان نیز، مکان وقوع را به خوبی خواهد داد، اما در مورد حوزه فرکانس در این مکان اطلاع خاصی ندارد. برای رفع این مشکل می توان از SFTF استفاده کرد. اما مشکل این تبدیل استفاده از پنجره با پنجره ثابت است.

طبق اصل عدم قطعیت Heisenberg

$$\Delta t \cdot \Delta f \geq \frac{1}{4\pi}$$

این اصل به ما نشان می دهد که نمی توان هر دو رزولوشن زمانی و فرکانسی را به دلخواه افزایش داد.

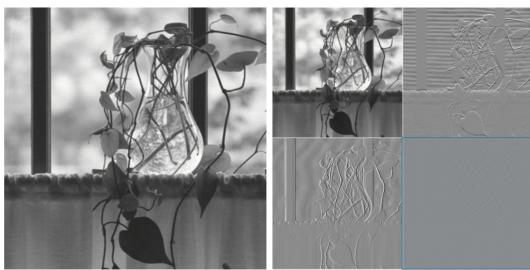
به همین دلیل تبدیل موجک معروفی شد. در این تبدیل سایز پنجره متفاوت گرفته شده است که می تواند بر مشکل از پیش تعیین کردن رزولوشن غلبه کرد.

در موجک به شکل زیر عمل می کنیم:

- آنالیز فرکانس های بالا: استفاده از پنجره های باریک تر برای رزولوشن زمانی بهتر

- آنالیز فرکانس های پایین: استفاده از پنجره های عریض برای رزولوشن فرکانسی بهتر
حال که با تبدیل موجک آشنا شدیم، هر موجک را توضیح می دهیم.

می دانیم که تبدیل موجک دارای خاصیت جدایزیری است و به کمک فیلترهای یک بعدی haar می توان این هرم را ایجاد کرد.



شکل ۱.۱: تصویر پس از اعمال فیلتر haar (اعمال تبدیل موجک)

همانطور که در شکل بالا دیده می شود، با هر بار اعمال فیلتر به تصویر، تصویر را به ۴ بخش تقسیم می کنیم:

- بخش horizontal: این بخش که از آن به LH نیز یاد می شود، شامل فرکانس های پایین در افقی و فرکانس بالا در عمودی می باشد. به همین دلیل به طور کلی این سیگنال شامل لبه ها و جزیبات افقی می باشد.
- بخش vertical: این بخش که از آن به HL نیز یاد می شود، شامل فرکانس های بالا در افقی و فرکانس های پایین در عمودی می باشد. به همین دلیل به طور کلی این سیگنال شامل لبه ها و جزیبات عمودی می باشد.
- بخش diagonal: این بخش که از آن به HH یاد می شود شامل سیگنالهای فرکانس بالا هم در افقی و هم در عمودی خواهد بود. این تصویر شامل لبه ها و جزیبات قطری خواهد بود.
- بخش Approximation: این بخش که همان LL است، شامل تصویر اصلی است که رزولوشن آن کاهش یافته است. بنابراین می توان از این عکس برای سطح بعدی هرم استفاده کرد. زیرا همان عکس است که سایز ان کاهش یافته است و رزولوشن آن نیز کم شده است.

بنابراین برای ساختن هرم موجک با استفاده از فیلتر haar این فیلتر را به عکس اعمال می کنیم. بخش approximation را به عنوان تصویر سطح بعد در نظر میگیریم و به طور مجدد این کار را ادامه می دهیم.
در ادامه از ما خواسته شده است که خروجی را با هرم لاپلاسی مقایسه کنیم.

از نظر پیچیدگی حافظه ای، هرم لاپلاس به دلیل اینکه ماتریس آن sparse هست و این گونه ماتریس ها را می

بیشتر مقدار آن برابر صفر است. یک راه برای از بین بردن نویز به کمک تبدیل موجک است که هرم موجک را تشکیل دهیم، و با در نظر گرفتن یک threshold مناسب، مقادیری از ماتریس را که مقداری نزدیک به صفر دارند را، صفر در نظر بگیریم. این مقدار در واقع نویز ها هستند.

برای تعیین threshold روش های متفاوتی وجود دارد. یکی از قوی ترین روش های که کتابخانه معروف scikit-image پایتون نیز از آن استفاده کرده است، threshold است.

فرمول این روش برای سینگال های یک بعدی به شکل زیر است که میتوان آن را به دو بعد و یا بیشتر بسط داد:

$$y_{\text{soft}}(t) = \begin{cases} \text{sgn}(x(t)) \cdot \frac{|x(t) - \delta|}{|\delta|}, & |x(t)| > \delta \\ 0, & |x(t)| \leq \delta \end{cases}$$

همچین برای انتخاب threshold دو روش دیگر نیز وجود دارد:

VisuShrink:

در این روش یک threshold سراسری برای اعمال به کل ضرایب موجک انتخاب می شود. این روش وقتی مناسب است که نویز با توزیع زیاد به تصویر اضافه شده باشد. با استفاده از این روش و یک سیگمای مناسب، می توان نتیجه مطلوبتری از نظر بصری بدست آورد.

BayedShrink:

این روش یک رویکرد انطباقی برای تعیین استانه دارد که در آن به هر باند یک آستانه منحصر به فرد اعمال می شود.

توان به خوبی فشرده کرد، هرم لاپلاس پیچیدگی حافظه ای کمتری دارد.

همچنین واضح است که دستیابی به هرم گوسی نیاز به عملیات و محاسبات کمتری دارد، بنابراین پیچیدگی زمانی هرم موجک نیز بیشتر است.

قسمت ۶.۱.۴:

در قسمت قبل با ساختن هرم موجک آشنا شدیم. در این قسمت ابتدا به بررسی بازسازی تصویر از روی هرم موجک پرداخته و پس از آن، درباره فشرده سازی صحبت می کنیم. برای بازسازی تصویر از هرم موجک باید از تبدیل موجک inverse استفاده کنیم. این تابع \mathcal{F} مقدار vertical و diagonal Approximation horizontal را دریافت کرده و سپس تصویر سطح بالاتر (با رزوشن بیشتر) را خواهد ساخت. می دانیم که تصویر ساخته شده در واقع LL (یا approximation) سطح بالاتر است. حالا که LL سطح بالاتر را داریم، با توجه به اینکه LH و HL و HH سطح بالاتر را داریم، می توانیم LL سطح بعدی را بسازیم.

این کار را به این ترتیب ادامه میدهیم تا به تصویر اصلی برسیم.

در ادامه می خواهیم با استفاده از هرمی که در قسمت قبل ساختیم (هرم موجک) عمل فشرده سازی را انجام دهیم. می دانیم که یکی از کاربردهای موجک و به طور کل هرمها فشرده سازی تصاویر است.

در این بخش می خواهیم با استفاده از quantize (چندی سازی)، سایز تصویر را کاهش دهیم. در تمارین قبل مشاهده کردیم که quantize کردن تا چه اندازه می تواند باعث کاهش سایز تصویر شود، و دیدیم که کیفیت عکس پس از چند مرحله، به صورت قابل توجه تغییر نمی کند. برای انجام فشرده سازی در این مرحله، باید تابع زیر را به همه مولفه های سطوح اعمال کنیم.

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

پس از اعمال تابع بالا به همه مولفه های سطوح هرم موجک، حال به بازسازی تصویر می پردازیم. مراحل بازسازی تصویر را بالاتر توضیح دادیم.

قسمت ۶.۲:

در قسمت های قبلی، درباره این صحبت کردیم که هرم موجک همانند هرم لاپلاسین، یک ماتریس تنک است و



شکل ۲.۴: هرم لاپلاسین و گوسین بدست آمده در کنار هم (هرم لاپلاسین نرمالایز شده است)

همانطور که در شرح توضیح دادیم، در هر مرحله طول و عرض ۲/۱ می شود و به طور کل سایز تصویر ۴/۱ می شود. در مرحله نهم به یک پیکسل خواهیم رسید. در کل تعداد پیکسل ها 262144 خواهد بود.

حال تصویر را به کمک روش شرح داده شد در بخش تکنیکال، بازسازی می کنیم.



شکل ۲.۵: تصویر بازسازی شده با هرم ۹ سطحی تصویر بازسازی شده را با تصویر اصلی مقایسه می کنیم، برای مقایسه از MSE و PSNR استفاده می کنیم.

نتایج:
قسمت ۶.۱.۱ در قسمت به ساخت هرم گوسی و لاپلاسی پرداختیم.



شکل ۲.۱: تصویر اصلی تصویر اصلی که قرار است عملیات را بر روی آن انجام دهیم شکل ۱.۱ است. حال با توجه به آنچه در شرح توضیح دادیم، هرم را بدست می آوریم. می دانیم که هرم ۹ سطح دارد.



شکل ۱.۲: هرم گوسی ۹ سطحی



شکل ۲.۳: هرم لاپلاسی بدست آمده قبل از نرمالایز کردن



شکل ۲.۸: هرم لاپلاسین و گوسین بدهت آمده در کنار هرم (هرم لاپلاسین نرمالایز شده است) با توجه به آنچه در قسمت شرح توضیح دادیم، تصویر اصلی را بازسازی می کنیم.



شکل ۲.۹: تصویر بازسازی شده با هرم سه سطحی همانطور که در قسمت قبل توضیح دادیم، تصویر به دست آمده به تصویر اصلی بسیار نزدیک است. (به خاطر قانون شanon)

PSNR	MSE
100	0

قسمت ۶.۱.۳:

PSNR	MSE
100	0

همانطور که مشاهده می شود، تصاویر بازسازی شده و تصویر اصلی کاملاً مشابه هستند و هیچ بخشی از تصویر از بین نرفته است. بنابراین با استفاده از هرم لاپلاسین، هیچ داده ای از دست نمی روید. دلیل این امر این است که طی upsample و downsample های متواالی، فرکانس های بالا در لاپلاسین ذخیره می شود. به همین دلیل با کاهش سایز و کاهش ماکریتم فرکانس، طبق قانون شanon می توان نرخ نمونه برداری را بدون از دست داده کاهش داد.

دقت شود که حتما باید تصاویر سطح لاپلاسین را نرمال کرد. در صورتی که بدون نرمال سازی هرم را بسازیم، هنگام بازسازی تصویر کیفیت خوبی نخواهیم داشت.

قسمت ۶.۱.۲:

ابتدا همانطور که در قسمت قبل توضیح داده شد، ابتدا هرم گوسی و لاپلاس را تشکیل می دهیم. در این قسمت تعداد سطوح با قسمت قبل متفاوت است.



شکل ۲.۶: هرم گوسی سه سطحی



شکل ۲.۷: هرم لاپلاسین سه سطحی قبل از نرمالایز شدن

با توجه به آنچه در شرح توضیح دادیم، هرم موجک را در سه سطح تولید می کنیم.

PSNR	MSE
43.94	1.49

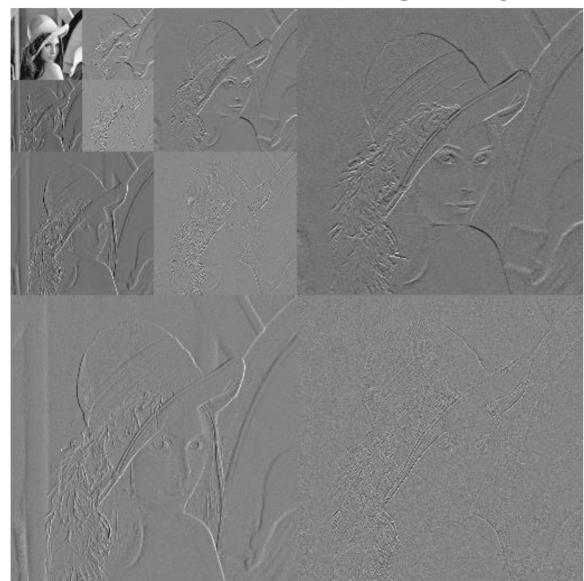
همانطور که نتایج نشان می دهد، به کمک هرم موجک و چندی سازی کمی از تصویر از دست رفته است. می دانیم که از تابع چندی سازی برای فشرده سازی استفاده می کنیم، اینکه با وجود فشرده سازی (کاهش حجم تصویر)

قسمت ۶.۲:

برای این قسمت ابتدا به تصویر نویز گوسی و نویز نمک و فلفل را اضافه می کنیم.



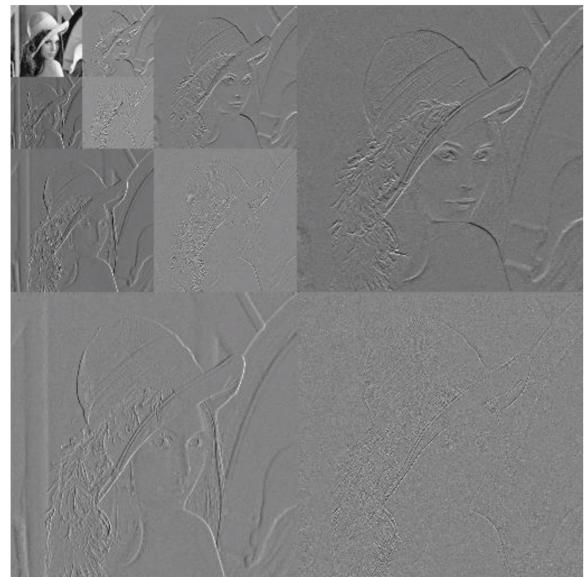
شکل ۲.۱۳: تصویر با اضافه شدن نویز گوسی



شکل ۲.۱۱: هرم موجک با استفاده از فیلتر haar برای سه سطح

قسمت ۶.۱.۴:

با توجه به آنچه در شرح توضیح دادیم، ابتدا هرم موجک را برای سه سطح می سازیم. حال به هر سطح به هر ۴ باند، تابع چندی سازی را اعمال می کنیم.



شکل ۲.۱۲: هرم موجک برای سه سطح حال با توجه آنچه در شرح توضیح دادیم، تصویر را بازسازی خواهیم کرد.



شکل ۲.۱۶: تصویر با نویز نمک فلفل پس از رفع نویز

حال دو معیار mse و psnr را بررسی می کنیم.

	psnr	mse
شکل ۲.۱۴	18.57	590.65
شکل ۲.۱۵	16.39	675.530
شکل ۲.۱۶	24.6	275.87
شکل ۲.۱۷	27.3	320.99

همانطور که مشاهده می شود، برای رفع نویز روش های ذکر شده تاثیر مناسبی داشتند و تا حد خوبی نویز را رفع کرده است.



شکل ۲.۱۴: تصویر با اضافه شدن نویز نمک فلفل حال با توجه به آنچه در شرح توضیح داده شد، آستانه مناسب را پیدا کرده و سپس سعی با اعمال آستانه، نویز را حذف می کنیم.



شکل ۲.۱۵: تصویر با نویز گوسی پس از رفع نویز

```

for i in range(0,iteration):
    cur_img =
apply_box_filter(pre_img)
    #calcute all pixels in pyramid
    all_pixels +=
int(cur_img.shape[0] *
cur_img.shape[1])
    all_pixels = int(img.shape[0]
* img.shape[1])
    # print(iteration)
    path = "result/6.1/Lena" +
str(i) + ".jpg"
    # cv.imwrite(path, cur_img)
    #add margin to image
    margin_img = make_size(cur_img
, img.shape[0] , img.shape[1])
    #add to pyramid
    pyramid =
np.vstack((pyramid,margin_img))
    pre_img = cur_img

for i in range(0,iteration):
    cur_img =
apply_box_filter(pre_img)
    #calcute all pixels in pyramid
    all_pixels +=
int(cur_img.shape[0] *
cur_img.shape[1])
    all_pixels = int(img.shape[0]
* img.shape[1])
    #calculate laplacian
    laplacian = normalize(img -
cur_img)
    # print(iteration)
    path = "result/6.1/Lena" +
str(i) + ".jpg"
    # cv.imwrite(path, cur_img)
    #add margin to image
    margin_img = make_size(cur_img
, img.shape[0] , img.shape[1])
    #add to pyramid
    pyramid =
np.vstack((pyramid,margin_img))
    pre_img = cur_img

```

Appendix:

قسمت ۶.۱.۱ :

```

import cv2 as cv
import numpy as np

def normalize(img):
    minimum = np.amin(img)
    return (img - minimum) * (255
/ np.amax(img))

def apply_box_filter(img):
    w,l = img.shape
    result = np.zeros((int(w / 2)
, int(l / 2)) , np.uint8)
    for i in range(0,w,2):
        for j in range(0,l,2):
            index, jndex = int(i /
2) , int(j / 2)
            result[index][jndex] =
np.uint8(np.sum(img[i:i+2,j:j+2]) /
4)
    return result

def make_size(img , width ,
length):
    result = np.zeros((width ,
length) , np.uint8)
    mid_w , mid_l = int(width / 2)
, int(length / 2)
    img_w , img_l =
int(img.shape[0] / 2) ,
int(img.shape[0] / 2)
    result[mid_w - img_w : mid_w +
img_w , mid_l - img_l : mid_l +
img_l] = img
    return result

img = cv.imread("images/Lena.bmp"
, cv.IMREAD_GRAYSCALE)

pyramid = img
iteration =
int(np.log2(img.shape[0]))
pre_img = img
all_pixels = int(img.shape[0] *
img.shape[1])

```

```

pyramid = img
iteration =
int(np.log2(img.shape[0]))
pre_img = img
all_pixels = int(img.shape[0] *
img.shape[1])

for i in range(0,iteration):
    cur_img =
apply_box_filter(pre_img)
    #calcute all pixels in pyramid
    all_pixels +=
int(cur_img.shape[0] *
cur_img.shape[1])
    all_pixels = int(img.shape[0]
* img.shape[1])
    # print(iteration)
    path = "result/6.1/Lena" +
str(i) + ".jpg"
    # cv.imwrite(path, cur_img)
    #add margin to image
    margin_img = make_size(cur_img
, img.shape[0] , img.shape[1])
    #add to pyramid
    pyramid =
np.vstack((pyramid,margin_img))
    pre_img = cur_img

frame = cv2.cvtColor(frame_RGB,
cv2.COLOR_BGR2YCR_CB)
height = 10
Gauss = frame.copy()
gpA = [Gauss]
for i in xrange(height):
    Gauss = cv2.pyrDown(Gauss)
    gpA.append(Gauss)

lbImage = [gpA[height-1]]

for j in xrange(height-1,0,-1):
    GE = cv2.pyrUp(gpA[j])
    L = cv2.subtract(gpA[j-1],GE)
    lbImage.append(L)

ls_ = lbImage[0]
for j in range(1,height,1):
    ls_ = cv2.pyrUp(ls_)
    ls_ = cv2.add(ls_,lbImage[j])

```

```

print(all_pixels)
#
cv.imwrite("result/6.1/pyramid.jpg"
, pyramid)
cv.waitKey(0)

```

قسمت ٦.١.٢

```

import cv2 as cv
import numpy as np

def normalize(img):
    minimum = np.amin(img)
    return (img - minimum) * (255
/ np.amax(img))

def apply_box_filter(img):
    w,l = img.shape
    result = np.zeros((int(w / 2)
, int(l / 2)) , np.uint8)
    for i in range(0,w,2):
        for j in range(0,l,2):
            index, jndex = int(i / 2) ,
            int(j / 2)
            result[index][jndex] =
np.uint8(np.sum(img[i:i+2,j:j+2]) /
4)
    return result

def make_size(img , width ,
length):
    result = np.zeros((width ,
length) , np.uint8)
    mid_w , mid_l = int(width / 2)
    , int(length / 2)
    img_w , img_l =
int(img.shape[0] / 2) ,
int(img.shape[0] / 2)
    result[mid_w - img_w : mid_w +
img_w , mid_l - img_l : mid_l +
img_l] = img
    return result

img = cv.imread("images/Lena.bmp"
, cv.IMREAD_GRAYSCALE)

```

```

dec_part =
output(1:R/(2^(level-1)),1:C/(2^(level-1)));

[r,c] = size(dec_part);
cA = dec_part(1:r/2,1:c/2);
cV = dec_part(r/2+1:r,1:c/2);
cH = dec_part(1:r/2,c/2+1:c);
cD = dec_part(r/2+1:r,c/2+1:c);

output(1:R/(2^(level-1)),1:C/(2^(level-1))) =
idwt2(cA,cH,cV,cD, 'haar');

output = iwt(output,level-1);

end

function output =
wt(img,level,present)

if(level==0)
    output = img;
    return
end
[R,C] = size(img);
output= zeros(R,C,'double');

%[LoD,HiD] =
wfilters('haar','d');
[cA,cH,cV,cD] =
dwt2(img, 'haar');

if(present)
    cA = norm(cA);
    cV = norm(cV);
    cH = norm(cH);
    cD = norm(cD);
end
output(1:R/2,1:C/2) =
wt(cA,level-1,present);
output(R/2+1:R,1:C/2)=cV;
output(1:R/2,C/2+1:C)=cH;
output(R/2+1:R,C/2+1:C)=cD;

if (present)
    output = uint8(output);
end

end

function output = norm(img)
output = mat2gray(img);
Max = max(max(output));
Min = min(min(output));
output = (255/(Max-Min))*output;

```

```

ls_ = cv2.cvtColor(ls_,cv2.COLOR_YCR_CB2BGR)
cv2.imshow("Pyramid reconstructed Image",ls_)
cv2.waitKey(0)

for i in range(0,iteration):
    cur_img =
apply_box_filter(pre_img)
    #calcute all pixels in pyramid
    all_pixels +=
int(cur_img.shape[0] *
cur_img.shape[1])
    all_pixels = int(img.shape[0]
* img.shape[1])
    #calculate laplacian
    laplacian = normalize(img -
cur_img)
    # print(iteration)
    path = "result/6.1/Lena" +
str(i) + ".jpg"
    # cv.imwrite(path, cur_img)
    #add margin to image
    margin_img = make_size(cur_img,
img.shape[0] , img.shape[1])
    #add to pyramid
    pyramid =
np.vstack((pyramid,margin_img))
    pre_img = cur_img

print(all_pixels)
#
cv.imwrite("result/6.1/pyramid.jpg",
" , pyramid)
cv.waitKey(0)

```

این دو قسمت را به علت مشکل پایتون، با متلب زدم:
قسمت ۶.۱.۳

```

function output =
iwt(wt_pyramid,level)
if(level==0)
    output = wt_pyramid;
    return
end
output = wt_pyramid;
[R,C] = size(output);

```

```

        return
    end
    output = wt_pyramid;
    [R,C] = size(output);
    dec_part =
output(1:R/(2^(level-
1)),1:C/(2^(level-1)));

[r,c] = size(dec_part);
cA = dec_part(1:r/2,1:c/2);
cV = dec_part(r/2+1:r,1:c/2);
cH = dec_part(1:r/2,c/2+1:c);
cD = dec_part(r/2+1:r,c/2+1:c);

output(1:R/(2^(level-
1)),1:C/(2^(level-1))) =
idwt2(cA,cH,cV,cD, 'haar');

output = iwt(output,level-1);
end

function output =
wt(img,level,present)

if(level==0)
    output = img;
    return
end
[R,C] = size(img);
output= zeros(R,C, 'double');

%[LoD,HiD] =
wfilters('haar','d');
[cA,cH,cV,cD] =
dwt2(img, 'haar');

cA = quantizer(cA,2);
cV = quantizer(cV,2);
cH = quantizer(cH,2);
cD = quantizer(cD,2);

if(present)
    cA = norm(cA);
    cV = norm(cV);
    cH = norm(cH);
    cD = norm(cD);
end
output(1:R/2,1:C/2) =
wt(cA,level-1,present);
output(R/2+1:R,1:C/2)=cV;
output(1:R/2,C/2+1:C)=cH;
output(R/2+1:R,C/2+1:C)=cD;

if (present)
    output = uint8(output);
end

```

```

end

img = imread('Images/Lena.bmp');
img = rgb2gray(img);

level = 3;
output = wt(img,level,0);

x = iwt(output,level);
%x = norm (x);
x = uint8(x);
imshow(output);
figure
imshow(x);
m = immse(img,x);
p = psnr(img,x);

imwrite(x, 'x.png');


```

قسمت ۶.۱.۴

```

img = imread('Image/Lena.bmp');
img = rgb2gray(img);

level = 4;
output = wt(img,level,0);

x = iwt(output,level);
%x = norm (x);
x = uint8(x);
imshow(output);
figure
imshow(x);
p = psnr(x,img);
m = immse(x,img);

imwrite(x, 'x.png');

function output= quantizer
(img,gamma)
[M,N] = size(img);
output = zeros(M,N, 'double');

for i=1:M
    for j=1:N
        output(i,j) = gamma*
sign(img(i,j))*floor(
abs(img(i,j))/gamma);
    end
end
function output =
iwt(wt_pyramid,level)
if(level==0)
    output = wt_pyramid;

```

```

gussian_noise = noised_img =
random_noise(img , mode="gaussian"
, var = 0.01)
sigma = 0.12
noisy = random_noise(original,
var=sigma**2)

fig, ax = plt.subplots(nrows=2,
ncols=3, figsize=(8, 5),
sharex=True, sharey=True)

plt.gray()

sigma_est = estimate_sigma(noisy,
multichannel=True,
average_sigmas=True)

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

im_visushrink2 =
denoise_wavelet(noisy,
multichannel=True,
convert2ycbcr=True,
method='VisuShrink', mode='soft',
sigma=sigma_est/2,
rescale_sigma=True)

```

```

end
function output = norm(img)
output = mat2gray(img);
Max = max(max(output));
Min = min(min(output));
output = (255/(Max-Min))*output;
end

```

قسمت ۶.۲

```

import matplotlib.pyplot as plt

from skimage.restoration import
(denoise_wavelet, estimate_sigma)
import cv2 as cv
from skimage.util import
random_noise
from skimage.metrics import
peak_signal_noise_ratio

def
apply_median_filter(img,window_size):
    w,l = img.shape
    result = np.zeros((w ,l) ,
np.uint8)
    hw = int(window_size / 2)

    for i in range(hw , w - hw):
        for j in range( hw,l -
hw):
            result[i - hw][j - hw] =
np.median(np.squeeze(np.asarray(im
g[i-hw:i+hw+1,j-hw:j+hw+1])))
    print(result)
    return result

original =
cv.imread("images/Lena.bmp" ,
cv.IMREAD_GRAYSCALE)

salt_paper = noise_img =
random_noise(img , mode="s&p" ,
salt_vs_pepper = 0.1)

```

```
im_visushrink4 =
denoise_wavelet(noisy,
multichannel=True,
convert2ycbcr=True,
method='VisuShrink', mode='soft',
sigma=sigma_est/4,
rescale_sigma=True)

psnr_noisy =
peak_signal_noise_ratio(original,
noisy)
psnr_bayes =
peak_signal_noise_ratio(original,
im_bayes)
psnr_visushrink =
peak_signal_noise_ratio(original,
im_visushrink)
psnr_visushrink2 =
peak_signal_noise_ratio(original,
im_visushrink2)
psnr_visushrink4 =
peak_signal_noise_ratio(original,
im_visushrink4)

plt.show()
```