تمرین ۵ بینایی موجک

سارا قوام پور

اطلاعات گزارش

تاریخ:

واژگان کلیدی:
هرم گوسین
هرم لاپلاسین
فشرده سازی
تبدیل موجک
haar transform

چکیده

در این سری ۵ از تمرینات هدف آشنایی بـا MultiResolution Processing در حوزه پردازش تصویر میباشد. تمرینات اولیه به باز کرد ن مفاهیم هـرم هـای گوسـین و لاپلاسین و تشکیل عکس اصلی با استفاده از این هرم ها میباشد. تمرینات آخر به بررسی تبدیل موجک و استفادت از آن برای Compression میپردازد.

۱- مقدمه

این داکیومنت شامل توضیحات فـنی و تحلیـل نتـایج و پیاده سازی های تمرین ۵ موجـک بـا زبـان پـایتون و بـا استفاده از کتابخانه cv2 میباشد.

۲- توضیحات فنی تمرینات و تحلیل نتایج

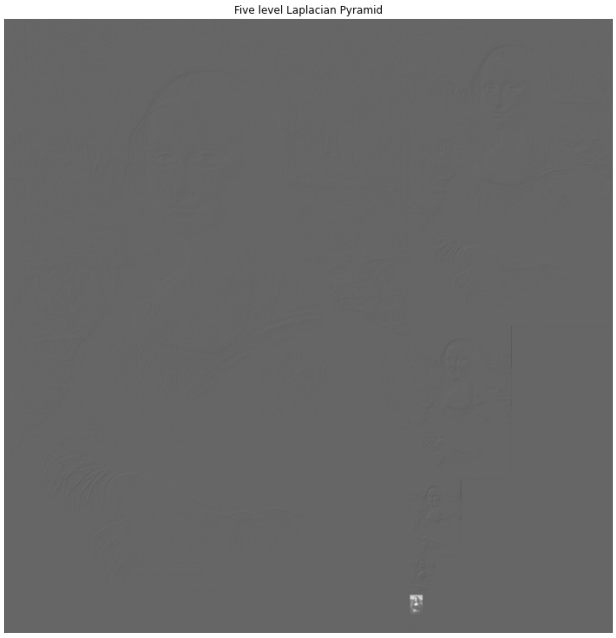
۲-۱ تمرین ۱-۱-۵

در این تمرین خواسته شده است تا هـرم گوسـی و هـرم لاپلاسین تا ۵ سطح بر روی عکس مونالیزا اجرا شود.
هرم گوسی به این گونه تشکیل میشـود کـه در سـطح ۰ خـــود عکس اصـــلی قـــرار دارد و ســـطح بعـــدی با downsample کردن سطح فعلی به دست می آید.
این downsample کردن در هـرم گوسـی, کانولوشـن کرنل گوسی با عکس ورودی یا به عبارتی محاسبه کـردن میانگین گوسی (وزن دار) عکس میباشد.
در این سوال هرم گوسی تا ۵ سطح خواسته شـده اسـت. چون سطح ۰ عکس اصلی است باید ۵ سطح دیگـر بـا ۵ بار اعمال فیلتر گوسی بر روی سطح قبل اعمـال شـود تـا هرم گوسی ۵ سطح به دست بیاید.
در ابتدا ارایه ای ۳ در ۳ به عنوان تخمین فیلتر گوسـی ۳ در ۳ تشکیل میشود به نام guassian_kernel.
برای پیاده سازی هرم گوسی تایع guassian_pyramid نوشته شده است. این تابع عکس ورودی, فیلتر لازم برای downsample کـردن و تعـداد سـطوح هـرم را دریـافت میکند.
این تابع در یک حلقه به تعـداد سـطوحی کـه در ورودی دریافت کرد, کرنـل گوسـی را بـا سـطح پیشـین کـانوالو میکند تا به سطع فعلی برسد.
چون در هر Iteration کرنل گوسـی را بـا عکس کـانوالو میکند و به عبارتی فرکانس عکس کاهش میابد باید طبق قضیه شانون پس از اعمال فیلتر نـرخ نمونـه بـرداری نـیز کـــاهش پیـــدا کنـــد و بـــه این منظـــور از تـــابع remove_row_col_downsampling اســتفاده شــده است. اگر میخواهیم عکس با نرخ ۲ downsample شود یعنی سایز ان در هر ۲ بعد نصف شود به این تـابع ورودی را ۲ میدهیم. پس از اعمال فیلتر و downsaple کردن با حذف ستون ها و ردیف ها, نتیجه به دست امـده حاصـل هرم گوسی در این سطح میباشد و حلقه به همین صورت تا اخرین لایه ادامه پیدا میکند.
به منظور نمایش هرم به شیوه که در اسلایدها بیان شـده است, یک آرایه numpy تمام صفر با ابعادسطر برابر سطر هـای عکس تعـداد سـتون هـای ۱.۵ برابـر عکس ایجـاد میکنیم و هـر یـک از سـطح هـا ی هـرم را در ایـنـدکس مربوطه آن قرار میدهیم. تابع guassian_pyramid به عنوان خروجی عکس که همه سطوح در آن قـرار گرفتـه شده اند وآرایه های شامل همه سطوح را برمیگرداند.

شکل ۱)هرم گوسین تصویر مونالیزا ۵ سطح

شکل ۱ هرم گوسین مونالیزا تا ۵ سطح را نشان میدهد.
چون خود تصویر اصلی سطح ۰ را تشکیل میدهد در تصویر بالا ۶ عکس وجود دارد که تصویر اصلی در سطح ۰ همان تصویر سمت چپ است. تصاویر سمت راست از بالا به پایین به ترتیب سطح های ۱ تا ۵ هرم را تشکیل میدهند. هر سطح از اعمال گوسین و downsample روی سطح قبل به دست می آید.

بخش دیگر این تمرین به پیاده سازی هرم لاپلاسین ۵ سطحی بر روی مونالیزا اختصاص دارد. به منظور پیاده سازی هرم لاپلاسین تابع laplacian_pyramid پیاده سازی شده است.
این تابع به عنوان ورودی آرایه شامل تمام سطوح گوسین را دریافت میکند چون هرم لاپلاسین از روی هرم گوسی به دست می آید به این صورت که هر سطح لاپلاسین حاصل تفریق همان سطح هرم گوسی با upsample شده با فاکتور ۲ سطح بعدی هرم گوسین میباشد.
ورودی بعدی این تابع تعداد سطوح میباشد.
در یک حلقه از ۰ تا تعداد سطح در هر بار تکرار حلقه, ابتدا با دریافت گوسین سطح بعدی از ارایه ورودی, آن را با استفادعت از تابع unsample_px_replication, با فاکتور ۲ upsample میکنیم با استفاده از روش pixel replication که اندازه تصویر را در فاکتور ضرب میکند و در سطر ها و ستون های خالی مقدار پیکسل قبلی را کپی میکند.
از آنجایی که خروجی این upsample باید از گوسین سطح بعد کم شود و در محاسبه هرم گوسین چون downsample صورت گرفته است ممکن است ابعاد گوسین فرد باشد و قابل تفریق با طرف دوم تفریق نباشد به همین دلیل تابعی با نام pad_even تعریف شده که با صفر گذاشت در بعد فرد تصویر, آن را زوج میکند.
خروجی تابع pad_evenبرروی سطح فعلی گوسین و تفریق آن با خروجی upsample سطح بعدی گوسین با فاکتور ۲, سطح فعلی هرم لاپیلاسن را تشکیل میدهد.
این مراحل در هر بار تکرار حلقه انجام میشود.
به منظور نمایش هرم به شیوه که در اسلایدها بیان شده است, یک آرایه numpy تمام صفر با ابعادسطر برابر سطر های عکس تعداد ستون های ۱.۵ برابر عکس ایجاد میکنیم و هر یک از سطح ها ی هرم را در ایندکس مربوطه آن قرار میدهیم. تابع laplacian_pyramid به عنوان خروجی عکس که همه سطوح در آن قرار گرفته شده اند وآرایه های شامل همه سطوح را برمیگرداند.



شکل۲)هرم لاپلاسین با با ۵ سطح روی تصویر مونالیزا

شکل ۲ هرم لاپلاسین مونالیزا تا ۵ سطح را نشان میدهد. سطح ۵ (سطح آخر) هرم لاپلاسین با سطح آخر هرم گوسی برابر است. سطح های ۰ تا سطح ۴ در واقع لبه های تصویر در scale های متفاوت میباشند.

۲-۲ تمرین۲-۱-۵

فیلتر جداپذیر یا seprable که در تمرین های قبلی(سوال ۱ تمرین ۴) هم به آن اشاره شد, فیلتری است که ماتریس ۲ بعدی آن رابتوان به صورت حاصل ضرب یک بردار سطری در بردار ستونی نوشت. به عنوان مثال میتوان فیلتر ۳ در ۳ برای کرنل گوی را مثال زد که حاصل ضرب بردار ستونی و سطری [۱,۲,۱] است.

اگر سایز تصویر n در n باشد و سایز فیلتر f در f انگار با اعمال فیلتر کل تصویر مرتبه زمانی به صورت توان ۲ از fn خواهد بود. اما اگر فیلتر جداپذیر باشد و در مرحلهاول ابتدا حاصل تصویر را با بردار ستونی و بعد حاصل این حاصل ضرب را در بردار سطری ضرب کنیم, نتیجه نهایی مانند کانوالو کل فیلتر در تصویر است اما مرتبه زمانی در حالتی که اجزا فیلتر جدا پذیر را به صورت حدا در تصویر ضرب کنیم کمتر خواهد بود. با ضرب بردار ستونی در تصویر مرتبه زمانی به صورت توان۲ f در n خواهد بود. مرتبه زمانی برای ضرب بردار سطری نیز همین خواهد بود به طور کلی ضرب تصویر در ۲ جز سطری و ستونی فیلتر از مرتبه توان۲ f در n خواهد بود در حالی که کانوالو فیلتر اصلی در تصویر از مرتبه توان ۲ از fn خواهد بود. پس اگر به جای کانوالو یک فیلتر جداپذیر به طور کلی در تصویر اجزا ان را به ترتیب ضرب کنیم در عکس مرتبه زمانی کمتر خواهد شد.

اعمال فیلتر ها به صورت cascading به این معنا است که چندین فیلتر پشت سر هم به جای اعمال فیلتر بزرگتری است. به عنوان مثال هرم های گوسین را مثال زد که به جای اعمال یک گوسین بزرگ بر روی کل تصویر, در هر مرحله یک بار یک گوسین کوچکتر اعمال میشود. در هنگان کانوالو فیلتر بزرگ با عکس بدون cascading مرتبه زمانی به صورت توان۲ f در n خواهد بود. در حالی که اگر همین فیلتر به صورت cascading از فیلتر های کوچک اعمال شود, چون سایز فیلتر ها کوچکتر میشود مرتبه زمانی در cascading کاهش میابد.

برای الگوریتمی که در قسمت دوم سوال خواسته شده(هر سطح گوسین, انحراف معیار متفاوتی دارد) است میتوان به صورت زیر عمل کرد:
۱- در این سطح اول باید گوسین با انحراف معیار σ اعمال شود. چون فیلتر گوسین جدا پذیر است با ضرب کردم جز ستونی و سطری ان به صورت جداگانه باعث افزایش سرعت میشود.
۲- در سطح دوم باید گوسین با انحراف معیار $\sqrt{2}\sigma$ اعمال شود و مانند مرحله ۱ اگر ان را به بردار سطری و ستونی شکسته و بعد اعمال کنیم باعث کاهش مرنبه زمانی میشود.
۳- در سطح اخر باید باید گوسین با انحراف معیار 2σ اعمال شود.چون فیلتر گوسین جدا پذیر است با ضرب کردم جز ستونی و سطری ان به صورت جداگانه باعث افزایش سرعت میشود.
در هر یک از سه سطح ازجداپذیری فیلتر گوسین و تاثیر ان در افزتیش سرعت استفاده شد بعضافه اینکه عملا تعدادی فیلتر پشت سر هم همان cascading است که اثر ان نیز در بهبود سرعت به جای استفاده یک باره از یک فیلتر بزرگ بحث شد.

۲-۳ تمرین۳-۱-۵

در هرم ها در هر سطح اندازه تصویر نصف میشود, در نتیجه اگر سایز اولیه تصویر n در n باشد که n برابر است با 2^j بنابریان در هر سطح چون سایز ان تقسیم بر ۲ میشود از j یک مقدار کم میشود. این تقسیم بر ۲ و در نتیجه j-1 تا زمانی میتوان ادامه داد که j=0. بنابراین میتوان j+1 سطح از سطح ۰ تا سطح j ساخت.
مجموع تعداد پیکسل هایی که در هرم استفاده میشود از یک رابطه هندسی که در اسلاید شماره ۷ درس هم بیان میشود به دست می آید که به صورت زیر است:

$$N^2+1/4\,N^2+1/16\,N^2+\ldots=4/3\,N^2$$

در نتیجه تعداد پیکسل های استفاده شده در هرم تنها ۱/۳ بیشتر از تعداد پیکسل های قبلی است.(۳۳ درصد بیشتر)
این فرمول هم برای هرم گوسی و هم برای لاپلاسین درست است.
فواید استفاده از هرم ها که با توجه به اینمه تعداد پیکسل ها ۳۳ درصد بیشتر میشود همچنان مقرون به صرفه و منطقی هستند:

برخی فواید هرم گوسی :
۱- course to fine : به این معنا است که از قسمتی با جزییات کم به شیوه ای تدریجی به قسمتی با جزییات مهم تر برسیم. در هرم ها این عمل به این صورت انجام میشود که در یک سطح با جزییات کم که بررسی شویم اگر در ان محدوده به جزییات دقیق تر نیاز داشته باشیم میتوان زوولوشن بیشتر آن را از سطوح قبلی(با رزولوشن بیشتر) بررسی کرد.
۲- search for correspondence : دیتکت مکردن ابجکت در هرم گوسی میتوان با هزینه محاسباتی کمتر انجام شود و در زمان کمتر.

برخی فواید هرم لاپلاسین:
۱- image blending : زمانی که میخواهیم ۲ تصویر را با هم مخلوط کنیم که بین ان ها لبه مشخصی نباشد, لاپلاسین تصویر حاصل را با استفاده از لاپلاسین ۲ تصویر ورودی به دست می آوریم.
۲- image compression : در هرم لاپلاسین سطح آخر, سطح اخر هرم گوسی است اما بقیه سطح ها همه درواقع ماتریس های با مقدار زیاد ۰ (ماتریس saprse) هستند. در نتیجه نیاز به فضای کمتری برای ذخیره ان ها میباشد و چون میتوان دوباره از روی خود هرم لاپلاسین

تصویر اولی را ساخت در نتیجه استفاده از این هرم بـــرای فشرده سازی عکس مناسب است.

همچنین در توضیحات گفته شد که بر روی تصویر لنا نیز هرم گوسین و بعد از آن لاپلاسین استفاده شود و ســپس با کمک این ۲ هرم دوباره تصویر recontruct شود.

برای این عمل تابع recontruct_ نوشته شده است.
این تابع برای ورودی دو ارایه شـامل عکس هـا در سـطح هرم گوسین و هرم لاپلاسین دریافت میکند.
در یک حلقه به تعداد سطح ها تکرار میشود, در هر تکرار , عکس در ســطح فعلی گوســین بــا اســتفاده از تــابع unsample_px_replication کـه در سـوال ۱-۱-۵ توضیح داده شد, upsample میشود و خروجی آن عکس لاپلاسین سطح قبلی جمع میشود. در نهایت خروجی که تخمینی از عکس اصلی است به عنوان خروجی برگردانده میشود.

شکل۳)تصویر construct شده لنا با استفاده از هرم گوسین و لاپلاسین

شکل۴)تصویر لنا

بـا مقایســه شــکل ۳ و ۴ مشــاهده میشــود کــه تصــویر reconstruct شده شکل ۳ لبه های شارپ تری دارد که باعث شده است کیفیت شکل ۳ نسبت بـه تصـویر اصـلی (شکل ۴) افزایش پیدا کند.

۲-۴ تمرین۴-۱-۵

در این تمـرین خواسـته شـده اسـت تـا هـرم ۳ سـطحی approximation یا averaging (box filter) به دست آورده شــود و ســپس هــرم ۳ ســطحی residual یــا laplacian از روی هرم که با استفاده از باکس فیلــتر بــه دست آمده است, محاســبه شــود. هــرم هــا بایــد بــر روی تصویر لنا محاسبه شوند.
بــرای محاســبه هــرم بــاکس فیلــتر تــابعی بــه نــام boxfilter_pyramid نوشته شده است کـه ماننـد تـابع guassian_pyramid توضیح داده شده در سوال ۱-۱-۵ میباشد با این تفاوت که در مرحله کانوالو کرنل با عکس اینجا از کرنل باکی فیلتر استفاده میشود و بقیــه مراحل مشابه است.
برای محاسبه هرم لاپلاسین از روی هرم باکس فیلــتر هم تــابع box_filter_laplacian_pyramidنوشــته شــده است که مانند تـابع laplacian_pyramid در سـوال ۱-۵-۱ عمل میکند با این تفاوت که اینجا هرم لاپلاســین را از رو هرم باکس فیلتر میسازد و نه گوسین.

شکل۵)هرم ۳ سطحی باکس فیلتر بر روی تصویر لنا

شکل۶)هرم ۳ سطحی لاپلاسین که از روی هرم باکس فیلتر شکل ۵ ساخته شده است.

بر اساس مقایسه شکل های ۵ و ۶ با شکل های ۱ و ۲ به و چون فیلتر گوسین به علت وزن دار بودن در مرکز میتواند نویز هارا نیز تقویت کند اما باکس فیلتر این عمل را انجام نمیدهد, مشاهده میشود که به لیه های شارپ تری در شکل ۶ رسیده ایم.

۵-۲ تمرین۵-۱-۵

در این تمرین خواسته شده است که تبدیل موجک برای با استاده از آنالیر هار تا ۳ سطح بر روی عکس لنا انجام شود و هرم موجک به دست امده با هرم تمرین ۳-۱-۵ مقایسه شود.

برای پیاده سازی تبدیل موجک تابع wavelet_transform پیاده سازی شده است. این تابع به عنوان ورودی عکس ورودی و تعداد سطح هایی که باید تبدیل موجک انجام شود را دریافت میکند.

میدانیم که DWT در هر سطح از تبدیل ۴ خروجی بر میگرداند که یکی از آن ها LL یا approximation است و بقیه لبه های افقی و عمودی و ۴۵ درجه هستند.

عملکرد این تابه به این صورت است در هر مرحله approximation را نگه میدارد تا بعدا با این ها یک approximation_pyramid بسازد تا بتوان آن را با boxfilter pyramid سوال ۴-۱-۵ مقایسه کرد.

همچنین خروجی که بعد از ۳ بار تبدیل موجک به دست می آید که شامل ۹ قسمت حاوی لبه و یک aproximation میباشد با laplacian pyramid سوال ۴-۱-۵ مقایسه میشود.

این تابع با استفاده از تابع wavedec2 از pywt که به آن ورودی haar برای هار انالیز داده شده و عدد۳ به عنوان اینکه تبدیل تا چند سطح اجرا شود. Wavedec2 به عنوان خروجی اول خود ماتریس صرایب موجک را بعد از ۳ سطح خروجی میدهد. خروجی تبدیل موجک بعد از ۳ سطح بر روی تصویر لنا:

شکل۷)هرم موجک بعد از ۳ سطح تبدیل موجک بر روی تصویر لنا

شکل ۷ که خروجی و ضرایب موجک بعد از ۳ بار تبدیل موجک است تنها شامل ۱ عدد aproximation که مربوط به سطح ۳ است میباشد و بقیه ۹ ناحیه مربوط به لبه های افقی و عمودی و ۴۵ درجه در سطح های متفاوت هستند.(تعداد دفعات مختلف تبدیل موجک)

در شکل ۷ دیده میشود که لبه هایی که بعد از بار سوم تبدیل موجک گرفتن محاسبه شده اند لبه های پهن تر و مشخص تر و ملموس تری نسبت به لبه ها در در سطح ۱ و ۲ هرم (بعد از ۱ و ۲ بار تبدیل موجک محاسبه کردن) این ضرایب (شکل ۷) را میتوان با laplacian pyramid سوال ۴-۱-۵ (شکل ۶) مقایسه کرد.
در شکل ۷ لبه ها علاوه بر تفکیک شدن بر اساس سطحی از هرم که در آن قرار دارند بر اساس اینکه لبه عمودی یا افقی یا ۴۵ درجه هستند هم تفکیک شده اند. به این معنا که در هرم موجک اطلاعات تفکیک شده تر هستند. همچنین میتوان مشاهده کرد که لبه هایی که در شکل ۶ مبینیم بیشترین شباهت را به لبه های سطح ۳ از هرم موجک دارند.
تابع wavelet_transform پس از محاسبه ضرایب موجک بعد از ۳ تبدیل, با استفاده از یک حلقه به اندازه تعداد بار های انجام تبدیل موجک(تعداد سطوح هرم موجک), در هر iteration یک تبدیل موجک را با استفاده از pywt.dwt2 اجرا میکند و قسمت approximation آن را ذخیره میکند به فرمت یک هرم. هدف از این حلقه ساختن یک approximation pyramid از approximation های موجک در سطح میباشد تا آن را با هرم باکس فیلتر سوال ۴-۱-۵ بتوان مقایسه کرد.
در انتنها نیز تابع wavelet_transform ضرایب موجک به شکل آرایه (چیزی که در شکل ۷ نمیاش داده شده) و ضرایب موجک خام (قبل از استفاده از coeffs_to_arra y) و aproximation pyramid تشکیل شده از approximation های هر تبدیل موجک را بر میگرداند.
aproximation pyramid تشکیل شده از approximation های هر تبدیل موجک:

شکل۸) aproximation pyramid تشکیل شده از approximation های هر بار اجرای تبدیل موجک

شکل ۸ را میتوان با شکل ۵ که هرم باکس فیلتر از ۴-۱-۵ است مقایسه کرد. در شکل ۸ تصویر سمت چپ همان تصویر اصلی لنا است و سه تصویر سمت راست appriximation های هر بار تبدیل موجک هستند که همانطور که مشاهده میشود از سطوح ۱ تا ۳ هرم باکس فیلتر شکل ۵ کیفیت بهتری دارند.
میدانیم که چه هرم ها و چه تبدیل موجک که به نوعی خود نیز هرم میسازد همه از روش های multi resolution processing هستند و همان طور که ذکر شد موجک لبه هارا تفکیک میکند بر اساس جهت و approximation ها هم کیفیت بهتری دارند نسبت به هرم های میانگین. پس میتوان نتیجه گرفت که تبدیل موجک روش multi resolution processing بهتری نسبت به هرم های میانگین و هرم لاپلاسین است.

۲-۶ تمرین۶-۱-۵

در این سوال خواسته شده است تا کلیه ضرایب بعد از اعمال ۳ سطح تبدیل موجک که تابع wavelet_transform آن را به عنوان خروجی دوم خود (coeffs) برمیگرداند را طبق فرمول خواسته شده سوال Quantize کرده و سپس با ضرایب جدید عکس دروباره ساخته شود و مقدار PSNR برای عکس حاصل حساب شود.
خروجی coeffs که درواقع خروجی pywt.wavedec2 است, به فرم زیر میباشد:

$$[cAn,(cHn,cVn,cDn),...(cH1,cV1,cD1)]$$

ایندکس ۰ در coeffs ماتریس مربوط به عکس approximation میباشد و تاپل های ۳ تایی از n تا ۱, شامل لبه های افقی و عمودی و قطری در هر سطح از تبدیل موجک هستند. به ترتیب از سطح آخر تا سطح اول.
اگر c هر ضریب موجک باشد فرمول خواسته شده توسط سوال برای کوانتایز کردن ضرایب به صورت زیر است:

$$x' = \gamma * sgn(c) * floor[|c|/\gamma]$$

برای کوانتایز کردن ضرایب موجک طبق فرمول بالا تابع coefficient_quantize پیاده سازی شده است.
این تابع ضرایب موجک بعد از سه سطح را به عنوان ورودی دریافت میکند.
فرمت این ضرایب در بالا توضیح داده شد که ایندکس ۰ آن مربوط به ضرایب approximation است و بقیه ایندکس ها تاپل های سه تایی از ضرایب لبه های افقی و عمودی و قطری از سطح آخر تا سطح اول.

این تـابع در ابتـدا مـاتریس ضـرایب ایـنـدکس ۰ را طبـق فرمول بالا کوانتـایز میکنـد و بعـد خـروجی را بـه عنـوان ایندکس ۰ آرایه new_coeff که قرار است شامل ضرایب کوانتایز شده باشد ودرواقع خروجی این تـابع اسـت, قـرار میدهد.
سپس یک حلقه از ۱ تا طول ارایه ضرایب وجود دارد کـه در هر تکرار این حلقه, تاپل حاوی سه آرایـه ضـرایب لبـه های افقس و عمودی و قطری را دریافت کرده و هر کـدام را جداگانـه توسـط فرمـول بـالا کوانتـایز میکنـد و بعـد خروجی هـر سـه تـا را بـه شـکل یـه تاپـل سـه تـایی در ایندکس مربوطه در new_coeffs قرار میدهد.
در نهایت این تـابع new_coeffs را بـه عنـوان خـروجی (ضرایب کوانتایز شده که فرمت قرار گـیری ضـرایب کنـار هم تغییر نکرده و به شکل توضیح داده در بالا میباشد) بر میگرداند.

سوال میخواهد تا با این ضرایب دوباره عکس ساخته شود و تـابعی بـا نـام reconstruct_wawelet_coeff پیـاده سازی شـده اسـت کـه بـه عنـوان ورودی ضـرایب جدیـد کوانتایز شده را دریافت میکند. این تابع با اعمال معکوس تبدیل موجک به وسـیله pywt.idwt2 بـر روی ضـرایب کوانتایز شده جدید, با برگشـت بـه حـوزه مکـان از حـوزه تبدیل عکس را دوباره میسازد با ضـرایب جدیـد کوانتـایز شده ای که به عنوان ورودی دریافت کرد و عکس ساخته شده را به عنوان خروجی بر میگرداند.

شکل۹)عکس reconstruct شده توسط ضرایب موجک کوانتایز شده تصویر لنا

در ادامه سوال خواسته است تـا مقـدار PSNR بـرای این تصویر خواسته شده (شکل ۹) محاسبه شود.

PSNR بـه صـورت گسـترده بـرای انـدازه گـیری کیفیت فشرده سازی کـه روی عکس انجـام شـده اسـت, بـه کـار میرود.
کاری که در این سوال انجام شد نـیز درواقـع یـک lossy compression بود کـه بـا PSNR میتـوان کیفیت آن را سنجید.
مقدار PSNR برای شکل ۹ برابر است بـا 20.8 بـه این معنا که عکس نویزی (تصویر فشـرده شـده) کـه در واقـع شکل ۹ میباشد با عکس اصلی 20.8 db تفاوت دارد.
در واقع PSNR شبیه به MSE میباشد.

۳- کد تمرینات

```
# -*- coding: utf-8 -*-
"""cv_hw5_Wavelet_Sara_Ghavampour_9812762781.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vJiNYXnx0KtpHeGpGM5v_gBCbghz1pYH
"""

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import cv2
from math import *
from sklearn.metrics import mean_squared_error
import pywt
# %matplotlib inline

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_img=cv2.imread('Lena.bmp')
lena_img = cv2.cvtColor(lena_img,cv2.COLOR_BGR2GRAY)
plt.imshow(lena_img,cmap='gray')
plt.axis('off')
plt.show()
#(512, 512) uint8

monalisa_img=cv2.imread('monalisa.bmp')
monalisa_img = cv2.cvtColor(monalisa_img,cv2.COLOR_BGR2GRAY)
plt.imshow(monalisa_img,cmap='gray')
plt.axis('off')
plt.show()

"""5

5.1. Pyramid

5.1.1.
"""

################ 5.1.1 ################

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

def remove_row_col_downsampling(img,factor):
  return img[::factor,::factor]

def guassian_pyramid(img,kernel,level,ql=True):
  prev_level = img
  guass_array=[]
  guass_array.append(prev_level)
  out = np.zeros((img.shape[0],int(1.5*img.shape[1])))

  for i in range(0,level):
    filtered_img=cv2.filter2D(src=prev_level, ddepth=-1, kernel=kernel)
    next_level=remove_row_col_downsampling(filtered_img,2)
    guass_array.append(next_level)
    prev_level = next_level
  if ql==True:
    # put different levels of quassian pyramid in shape of pyramid

    #put level 0
    out[0:guass_array[0].shape[0],0:guass_array[0].shape[1]]=guass_array[0]

    #put level 1
    out[0:guass_array[1].shape[0],guass_array[0].shape[1]-1:guass_array[0].shape[1]-1+guass_array[1].shape[1]]=guass_array[1]

    #put level 2
    out[guass_array[1].shape[0]-1:guass_array[1].shape[0]-1+guass_array[2].shape[0],guass_array[0].shape[1]-1:guass_array[0].shape[1]-1+guass_array[2].shape[1]]=guass_array[2]

    #put level 3
    row=guass_array[1].shape[0]+guass_array[2].shape[0]-1
    col=guass_array[0].shape[1]
    out[row-1:row-1+guass_array[3].shape[0],col-1:col-1+guass_array[3].shape[1]]=guass_array[3]

    #put level 4
    row=row-1+guass_array[3].shape[0]
    out[row-1:row-1+guass_array[4].shape[0],col-1:col-1+guass_array[4].shape[1]]=guass_array[4]

    #put level 5
    row=row-1+guass_array[4].shape[0]
    out[row-1:row-1+guass_array[5].shape[0],col-1:col-1+guass_array[5].shape[1]]=guass_array[5]

  # if ql== False:
  #   #put level 5
  #   for j in range(6,level+1):
  #     row=row-1+guass_array[j-1].shape[0]
  #     out[row-1:row-1+guass_array[j-1].shape[0],col-1:col-1+guass_array[j].shape[1]]=guass_array[j]


  return out,guass_array

guassian_pyramid , guass_array=guassian_pyramid(monalisa_img,guassian_kernel,5)
show_img(guassian_pyramid,title=['Five level Guassian Pyramid'],figsize=13)

type(guass_array)

def unsample_px_replication(img, factor):
```

```
    out = np.repeat(img, factor, axis=0)
    out = np.repeat(out, factor, axis=1)
    return out

def pad_even(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

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

def laplacian_pyramid(guass_array,level,q1=True):
  img=guass_array[0]
  laplacian_array=[]
  out = pad_even(np.zeros((img.shape[0],int(1.5*img.shape[1]))))

  for i in range(0,level):
    upsample = (unsample_px_replication(guass_array[i+1],2))
    laplacian_array.append(pad_even(guass_array[i])-upsample)
  laplacian_array.append((guass_array)[level])

  # put different levels of laplacian pyramid in shape of pyramid

  if q1==True:
    #put level 0
    out[:,0:laplacian_array[0].shape[1]]=laplacian_array[0]

    #put level 1
    out[0:laplacian_array[1].shape[0],laplacian_array[0].shape[1]-2:laplacian_array[0].shape[1]-2+laplacian_array[1].shape[1]]=laplacian_array[1]

    #put level 2
    out[laplacian_array[1].shape[0]-1:laplacian_array[1].shape[0]-1+laplacian_array[2].shape[0],laplacian_array[0].shape[1]-1:laplacian_array[0].shape[1]-1+laplacian_array[2].shape[1

    #put level 3
    row=laplacian_array[1].shape[0]+laplacian_array[2].shape[0]-1
    col=laplacian_array[0].shape[1]
    out[row-1:row-1+laplacian_array[3].shape[0],col-1:col-1+laplacian_array[3].shape[1]]=laplacian_array[3]

    #put level 4
    row=row-1+laplacian_array[3].shape[0]
    out[row-1:row-1+laplacian_array[4].shape[0],col-1:col-1+laplacian_array[4].shape[1]]=laplacian_array[4]

    #put level 5
    row=row-1+laplacian_array[4].shape[0]
    out[row-1:row-1+laplacian_array[5].shape[0],col-1:col-1+laplacian_array[5].shape[1]]=laplacian_array[5]


  return out,laplacian_array

laplacian_pyramid,laplacian_array = laplacian_pyramid(guass_array,5)
show_img(laplacian_array,title=['level 0','level 1','level 2','level 3','level 4','level 5'],figsize=22)

show_img(laplacian_pyramid,title=['Five level Laplacian Pyramid'],figsize=13)

for i,p in enumerate(laplacian_array):
  print(i,' ',p.shape)

"""5.1.3."""

############### 5.1.3 ###############

lena_img.shape   #512 = 2^9 --> j=9 ---> j+1 level in pyramids
lena_j=9

############### biulding guassian_pyramid and residual pyramid ###############

_ , lena_guass_array=guassian_pyramid(lena_img,guassian_kernel,9,q1=False)
show_img(lena_guass_array,figsize=30)

_,lena_laplacian_array = laplacian_pyramid(lena_guass_array,lena_j,q1=False)
show_img(lena_laplacian_array,figsize=30)

############### reconstructing original lena image using guassian_pyramid and residual pyramid ###############

def reconstruct_(guass_array,laplacin_array):
  levels= np.flip([i for i in range(0,len(guass_array))])
  constructed_array  = []
  out = (guass_array[-1])
  for i in levels:
    if i ==0 : continue
    upscale = unsample_px_replication((guass_array[i]),2)
    out = (upscale) + (laplacin_array[i-1])

  return out

reconstructed_lena = reconstruct_(lena_guass_array,lena_laplacian_array)
show_img(reconstructed_lena)

"""5.1.4. """

######## 5.1.4 ###############

box_filter_kernel = (1/9) * np.array([[1,1,1],[1,1,1],[1,1,1]])
box_filter_kernel

def boxfilter_pyramid(img,kernel,level,q1=True):
  prev_level = img
  boxfilter_array=[]
  boxfilter_array.append(prev_level)
  out = np.zeros((img.shape[0],int(1.5*img.shape[1])))

  for i in range(0,level):
    filtered_img=cv2.filter2D(src=prev_level, ddepth=-1, kernel=kernel)
    next_level=remove_row_col_downsampling(filtered_img,2)
    boxfilter_array.append(next_level)
    prev_level = next_level
  if q1==True:
    # put different levels of quassian pyramid in shape of pyramid

    #put level 0
    out[0:boxfilter_array[0].shape[0],0:boxfilter_array[0].shape[1]]=boxfilter_array[0]

    #put level 1
    out[0:boxfilter_array[1].shape[0],boxfilter_array[0].shape[1]-1:boxfilter_array[0].shape[1]-1+boxfilter_array[1].shape[1]]=boxfilter_array[1]
```

```
    #put level 2
    out[boxfilter_array[1].shape[0]-1:boxfilter_array[1].shape[0]-1+boxfilter_array[2].shape[0],boxfilter_array[0].shape[1]-1:boxfilter_array[0].shape[1]-1+boxfilter_array[2].shape[1

    #put level 3
    row=boxfilter_array[1].shape[0]+boxfilter_array[2].shape[0]-1
    col=boxfilter_array[0].shape[1]
    out[row-1:row-1+boxfilter_array[3].shape[0],col-1:col-1+boxfilter_array[3].shape[1]]=boxfilter_array[3]



  return out,boxfilter_array

boxfilter_pyramid , boxfilter_array=boxfilter_pyramid(lena_img,box_filter_kernel,3)
show_img(boxfilter_pyramid,title=['Five level boxfilter Pyramid'],figsize=13)

def boxfilter_laplacian_pyramid(boxfilter_array,level,ql=True):
  img=boxfilter_array[0]
  laplacian_array=[]
  out = pad_even(np.zeros((img.shape[0],int(1.5*img.shape[1]))))

  for i in range(0,level):
    upsample = (unsample_px_replication(boxfilter_array[i+1],2))
    laplacian_array.append(pad_even(boxfilter_array[i])-upsample)
  laplacian_array.append((boxfilter_array)[level])

  # put different levels of laplacian pyramid in shape of pyramid

  if ql==True:
    #put level 0
    out[:,0:laplacian_array[0].shape[1]]=laplacian_array[0]

    #put level 1
    out[0:laplacian_array[1].shape[0],laplacian_array[0].shape[1]-2:laplacian_array[0].shape[1]-2+laplacian_array[1].shape[1]]=laplacian_array[1]

    #put level 2
    out[laplacian_array[1].shape[0]-1:laplacian_array[1].shape[0]-1+laplacian_array[2].shape[0],laplacian_array[0].shape[1]-1:laplacian_array[0].shape[1]-1+laplacian_array[2].shape[1

    #put level 3
    row=laplacian_array[1].shape[0]+laplacian_array[2].shape[0]-1
    col=laplacian_array[0].shape[1]
    out[row-1:row-1+laplacian_array[3].shape[0],col-1:col-1+laplacian_array[3].shape[1]]=laplacian_array[3]

    # #put level 4
    # row=row-1+laplacian_array[3].shape[0]
    # out[row-1:row-1+laplacian_array[4].shape[0],col-1:col-1+laplacian_array[4].shape[1]]=laplacian_array[4]

    # #put level 5
    # row=row-1+laplacian_array[4].shape[0]
    # out[row-1:row-1+laplacian_array[5].shape[0],col-1:col-1+laplacian_array[5].shape[1]]=laplacian_array[5]


  return out,laplacian_array

laplacian_pyramid,laplacian_boxfilter_array = boxfilter_laplacian_pyramid(boxfilter_array,3)
show_img(laplacian_boxfilter_array,title=['level 0','level 1','level 2','level 3'],figsize=22)

show_img(laplacian_pyramid,title=['Three level Laplacian of boxfilter Pyramid'],figsize=13)

def normalize(img):
  min = np.min(img)
  max = np.max(img)
  return ((img-min)/(max-min)*255).astype('uint8')

def wawelet_transform(img,levels):
  wt_out  = np.zeros((img.shape[0],img.shape[1]))
  appriximations= np.zeros((img.shape[0],int(1.5*img.shape[1])))

  appriximations[:,0:img.shape[1]]=img
  img_col = img.shape[1]
  # wavelet transform
  coeffs = pywt.wavedec2(img, 'haar', mode='periodization', level=levels)

  # Put coefficients in a matrix
  c_matrix, c_slices = pywt.coeffs_to_array(coeffs)

  ##  get approximations
  row_pointer = 0
  for i in range(0, levels):
    cA, _ = pywt.dwt2(img, 'haar', mode='periodization')
    print(cA.shape)
    appriximations[row_pointer:row_pointer+cA.shape[0],img_col:img_col+cA.shape[1]] = normalize(cA)
    row_pointer+=cA.shape[0]
    img=cA


  # return wt_out,appriximations_wrap
  return c_matrix,coeffs,appriximations

wt_out,coeffs,appriximations_wrap = wawelet_transform(lena_img,3)
show_img(wt_out)

show_img(appriximations_wrap)

"""5.1.6."""

(coeffs[1])

######## 5.1.6 ###########

def coefficiant_quantize(coeffs,step_size):
  length = len(coeffs)
  new_coeff = []
  new_coeff.append(step_size * np.sign(coeffs[0]) * np.floor(np.abs(coeffs[0])/step_size))
  for i in range(1,length):
    h,v,d = coeffs[i]
    new_h = step_size * np.sign(h) * np.floor(np.abs(h)/step_size)
    new_v = step_size * np.sign(v) * np.floor(np.abs(v)/step_size)
    new_d = step_size * np.sign(d) * np.floor(np.abs(d)/step_size)
    new_coeff.append((new_h,new_v,new_d))

  return new_coeff

new_coeffs = coefficiant_quantize(coeffs,2)
len(new_coeffs[0:2])

def reconstruct_wawelet_coeff(new_coeffs):
  length = len(new_coeffs)
  reconstructed_img = pywt.idwt2(new_coeffs[0:2], 'haar', mode='periodization')
  for i in range(2,length):
```

```
    reconstructed_img = pywt.idwt2([reconstructed_img,new_coeffs[i]], 'haar', mode='periodization')
  return reconstructed_img

reconstructed_img = reconstruct_wawelet_coeff(new_coeffs)
reconstructed_img

show_img(reconstructed_img)

psnr = cv2.PSNR(lena_img,reconstructed_img.astype('uint8'))
psnr
```