تمرین ۳ بینایی

سارا قوام پور

اطلاعات گزارش

تاریخ: ۱۴۰۱/۹/۱۱

واژگان کلیدی:
فیلتر
میانگین
مدیان
گوسی
لبه
لاپلاس
تقویت بالا

چکیده

در تمرین ۳ هدف آشنایی با فیلتر ها میباشد. به این صورت که در هر یک از سؤال‌ها تأثیر فیلتر های متفاوت بر روی عکس‌ها مشاهده میشود. نویز های متفاوت و روش‌های denoising برای هر نویز آشنا میشویم.تمرین شامل چهار بخش مقدمه, بررسی فنی و کد میباشد.

۱-مقدمه

نوشتار حاضر، شامل شیوه حل هر کدام از تمرین‌ها با استفاده از زبانPython میباشد. به همرا توضیح الگوریتم ها و نتایج هر بخش در قالب عکس. همچنین جداولی برای ارائه نتایج نیز ارائه شده اند.

۲-بررسی فنی تمرینات

۲-۱ تمرین ۱-۱-۳

در این تمرین خواسته شده است تا در رابطه با ضعف‌های box filter که همان فیلتر میانگین یکنواخت بحث شود. عیوب فیلتر میانگین یکنواخت به صورت زیر است:

۱) این فیلترadaptive نیست وخطی است به این معنا که این فیلتر ها به تصویر اعمال می‌شوند بدون توجه به ویژگی‌ها و محتویات و اطلاعات آماری تصویر و در تمامی قسمت‌های تصویر یک رفتار را دارد.

۲) در این فیلتر وزن همه پیکسل ها ۱ است. به این معنا که به محتویات تصویر توجه نمیشد و این باعث blur شدن تصویر و کاهش کیفیت لبه ها در تصویر میشو. هر چه سایز فیلتر هم بزرگ‌تر باید این مات شدگی بیشتر خواهد بود چون حتی پیکسل های دورتر هم با یک میزان میتوانند در مقدار پیکسل تأثیر بگذارند و محتویات تصویر را تغییر دهند و مات کنند.

۳) این فیلتر اجازه میدهد تا high frequency sample data مانند نویز به ساختار تصویر خروجی نفوذ کند. به عبارتی مقادیر این نویز ها را در حساب کردن مقادیر همسایه هایشان تأثیر میدهد که باعث post aliasing میشودو کیفیت فیلترینگ را کاهش میدهد چون باعث افزایش فرکانس میشود.

۴) این فیلتر روی تصاویر با فرکانس پایین عمل‌کرد بدی دارد.

۵)این فیلتر اگر عکس ورودی دارای سیگنال smooth باشد هم خروجی که برای این تصویر برمیگرداند ناهموار است. و علت این امر این است که لبه های این فیلتر تیز و sharp هستند و باعث ایجاد این ناهمواری ها میشوند.

۲-۲ تمرین ۲-۱-۳

بعضی از فیلتر ها برای از بین بردن برخی از نویز ها مفید نیستند مانند فیلتر میانگین یکنواخت روی impulse noise اثر خوبی ندارد اما فیلتر مدیان برای این نوع نویز مناسب است. در این نوع نویز و با فیلتر مدیان هر چه تعداد بار بیشتری فیلتر را اعمال کنیم, نویز را بهتر از بین میبرد. اما اگر فیلتر میانگین یکنواخت را روی نویز نمک فلفل به تعداد زیاد تکرار کنیم تصویر دائماً مات تر می‌شود و در نهایت به یک تصویر خاکستری میرسیم. چون با اعمال فیلتر میانگین به تعداد زیاد عملاً مقداری که برای همه پیکسل ها به دست می‌آورد میانگین کلیه پیکسل ها میباشد به طور کلی با تکرار زیاد فیلتر های میانگین مستقل از نوع نویز به تصویر خاکستری میرسیم که مقدار همه پیکسل ها میانگین کل مقادیر پیکسل ها هست.علاوه بر این با تکرار چند باره فیلتر میانگین لبه ها بیشتر blur میشوند.

[1] ghavams2001@gmail.com

۲-۳ تمرین ۳-۱-۳

در این تمرین باید اثر چندبار اعمال شدن فیلتر box filter یا همان فیلتر میانگین یکنواخت ۳ در ۳ بر روی عکس Elaine بررسی شود. برای این تمرین تابعی نوشته شده است که ورودی سایز فیلتر را که ۳ است دریافت میکند و عکس ورودی را. در این تابع same padding به صورت zero padding اعمال می‌شود تا سایز ورودی و خروجی پس از اعمال فیلتر ثابت بماند. مقدار پدینگ داده شده برابر ۱ میباشد که هاز هر ۴ سمت تصویر اعمال شده است. بعد از اضافع کردن پدینگ فیلتر را از بالا سمت چپ وارد تصویر پدینگ شده کرده و هر بار یک واحد به راست و پایین شیفت داده می‌شود. میانگین این ۹ عدد حساب میوشد و در تصویر خروجی در موقعیت پیکسل مرکزی فیلتر قرار میگیرد.
سپس این تابع را در حلقه صدا میزنیم. در حلقه اول ۲۰ بار, حلقه دوم ۵۰ و حلقه سوم ۱۲۰ بار. با اعمال شده هر چه بیشتر فیلتر مشاهده می‌شود که لبه ها دائماً بیشتر blur می‌شوند و قسمت‌های اصلی اصلی مانند شکل صورت (قسمت های فرکانس پایین) باقی‌مانده اند. به گونه‌ای که در عکس ۱۲۰ بار میانگین گرفته شده گویا تصویر سگمنت شده است.

شکل ۱) ۵۰ بار اعمال فیلتر میانگین یکنواخت

شکل ۲) ۲۰ بار اعمال فیلتر میانگین یکنواخت

شکل ۳) ۱۲۰ بار اعمال فیلتر میانگین یکنواخت

همانطور که دیده می‌شود با ۵۰ بار اعمال blur شده است به طور قابل توجهی حالا با ۱۲۰ بار در شکل ۳ مشاهده می‌شود که بیشتر مات شده این و به گونه‌ای سگمنت شده استو تنها اجزا اصلی و محدوده شان قابل تشخیص است. خط مشکی اطراف ۳ عکس هم به خاط zero padding است. اگر تعداد انجام فیلتر زیاد باشد در نهایت به تصویر خاکستری میرسیم.

۲-۴ تمرین ۴-۱-۳

در این تمرین هدف بررسی تأثیر سایز فیلتر در از بین بردن نویز و blurring میباشد. در ابتدا به تصویر Elaine نویز گوسی با واریانس ۰.۰۵ اضافه می‌شود و سپس تأثیر فیلتر میانگین یکنواخت از بخش قبل بر روی آن بررسی میشود.
تابعی به منظور اضافه کردن نویز استفاده شده است که در تمرین‌های بخش ۲-۳ هم از آن استفاده میشود. توضیح این تابع در قسمت ۲-۳ داده میشود. در اینجا از آن برای اضافه کردن نویز گوسی استفاده شده است.

شکل ۴)تصویر با نویز گوسی با واریانس ۰.۰۵

شکل ۵)باکس فیلتر با سایز ۵ روی شکل ۴

شکل ۶))باکس فیلتر با سایز ۷ روی شکل ۴

شکل ۷))باکس فیلتر با سایز ۱۱ روی شکل ۴

همانطور که مشاهده می‌شود نویز گوسی تا حدی در شکل ۴ با سایز فیلتر ۵ کاهش یافته است چون تصویر smooth شده توسط فیلتر میانگین و نویز کمتر مشاهده می‌شود که در عین حال بر روی لبه ها هم تأثیر گذاشته و آن‌ها هم blur شده اند. با افزایش سایز فیلتر میانگین یکنواخت, چون پیکسل های بیشتری در میانگین گیری شرکت میکنند, تصویرsmooth تر می‌شود و در نتیجه نویز گوسی هم بیشتر کاهی میابد به عنوان مثال در شکل ۷ با سایز فیلتر ۱۱ نویز کمتر قابل مشاهده است اما تصویر و لبه ها بیشتر smooth و blurr شده‌اند و لبه ها بیشتر از بین رفته اند.

۲-۵ تمرین ۵-۱-۳

برای تصمیم گیری برای بهترین سایز از بین سایز های اعمال شده در تمرین ۴-۱-۳ بهترین راه مقایسه بین mse های خروجی فیلتر روی عکس دارای نویز گوسی با عکس اصلی است.

جدول ۱)مقایسه mse سایز های مختلف box filter

سایز فیلتر	۵	۷	۹	۱۱
mse	۸۶.۳۰	۷۴.۹۱	۶۹.۲۷	۶۷.۷۶

کمترین mse بین تصویر denoise شده و تصویر اصلی مربوط به فیلتر میانگین یکنواخت با سایز ۱۱ میباشد. بنابراین از میان سایز های تست شده سایز ۱۱ برایذ فیلتر به دلیل داشتن کمترین mse نسبت به بقیه سایز ها, بهترین tradeoff را بین کاهش نویز و blurring برقرار میکند.

۱. شکل۸))شکل ۳ که یک تصویر مات شده بعد از اعمال ۱۲۰ بار فیلتر میانگین است

۲-۶ تمرین ۶-۱-۳

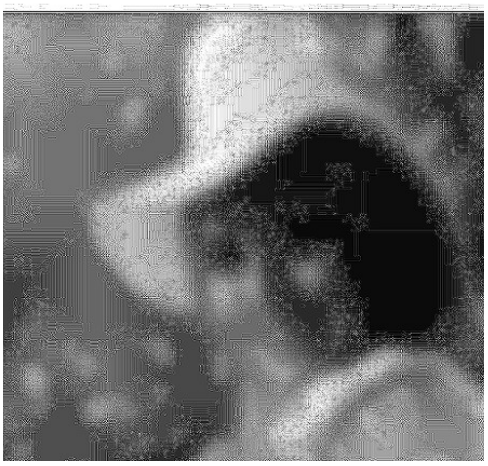
با correlate کردن فیلتر داده شده در این سؤال با تصویر درواقع ۲ عمل به صورت همزمان میگیرد. در فیلتر لاپلاسین لبه های افقی و عمودی مقداری که در خانه وسط قرار میگیرد ۴ است اما در این سؤال ۵ است به این معنا که با ضرب این فیلتر در عکس چون یک وزن بیشتر به پیکسل مرکزی از فیلتر لاپلاسین افقی و عمودی داده شده ۲ عمل همزمان انجام میشود:

۱) ابتدا فیلتر لاپلاسین در عکس ضرب می‌شود که باعث به دست آمدن لبه ها میشود. فیلتر لاپلاسین فیلتر شارپ کننده مشتق مرتبه ۲ در 2D میباشد و لبه ها و اختلافات موجود در عکس را برمیگرداند.

۲) لبه های به دست آمده را به عکس اصلی اضافه میکند که باعث بهبود کیفیت لبه ها و شارپ شدن لبه ها میشود.

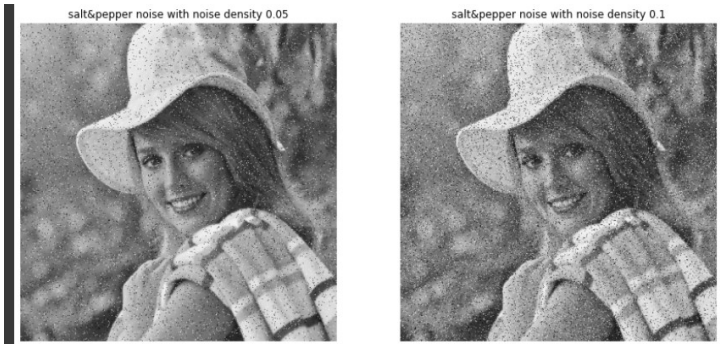
برای تست از شکل شماره ۳ در قسمت ۲-۳ استفاده میکنیم. روی این عکس ۱۲۰ بار box filter اعمال شده این و لبه ها blurr شده اند.

۱. شکل۹)۲ بار اعمال شدن فیلتر لاپلاسین روی عکس۸-شارپ شدن لبه ها و تغییرات پشت صحنه



با۲ بار اعمال فیلتر لاپلاسین بر روی تصویر ۸, همانط.ر که مشاهده می‌شود لبه ها تیز تر شده و تقویت یافته‌اند اما چون فیلتر لاپلاسین به طور کلی لبه ها و تغییرات را تیز میکند دیده میود که در شکل ۹ علاوه بر شارپ شدن لبه ها در پشت صحنه عکس هم تغییرات تیزتر شده اند.
در شکل ۱۰ که ۳ بار لاپلاسین اعمال شده هم لبه ها شارپ تر هستند و هم تغییرات در بک گراند عکس. با تکرار زیاد لاپلاس تغییرات پشت صحنه عکس هم شارپ می‌شوند علاوه بر لبه ها و عکس خراب می‌شود علاوه بر بهبود لبه ها.
در شکل ۹ که فیلتر لاپلاسین دو بار اعمال شده است, لبه های کلاه شارپ تر شده اند نسبت به شکل ۸ اما چون لاپلاسین به طور کلی تغییرات را تقویت میکند در شکل ۹ دیده میشود که در پشت تصویر هم برخی تغییرات شارپ تر شده اند که این شارپ تر شدن تغییرات پشت صحنه در شکل ۱۰ واضح تر است و تصویر را خراب کرده است.

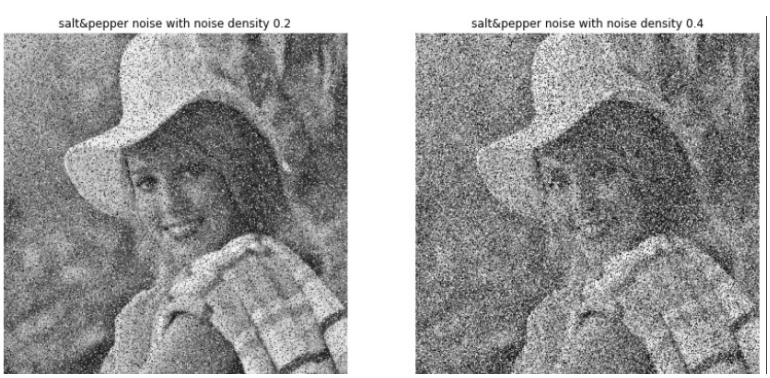
نویز برای نویز نمک فلفلو واریانس برای نویز گوسی را هم دریافت میکند. در صورتی که در ورودی برای نوع نویز نمک فلفل داده شده باشد, تابع random_noise از کتابخانه skimage.util استفاده میشود. چون خروجی random_noise بین ۰ و ۱ است باید در انتها تابع مقادیر در ۲۵۵ ضرب شوند تا بین ۲۵۵-۰ قرار بگیرند. بعد از اعمال نویز نمک فلفل با چگالی های متفاوت باید فیلتر مدیان اعمال شود.
تابع apply_median_filter برای اعمال فیلتر مدیان نوشته شده است.این تابع به عنوان ورودی عکس و سایز فیلتر را دریافت میکند. این تابع padding را به صورت valid که به معنای no padding است پیاده‌سازی میکند و چون پدینگ ندارد سایز تصویر خروجی پس از اعمال فیلتر کوچکتر از تصویر ورودی است.
در تایع میانه ابتدا اندازه و ابعاد عکس خروجی که کوچکتر از عکس ورودی است محاسبه میشود. سپس بر روی محدوده‌ای از عکس ورودی که در خروجی هم باقی میماند روی تصویر اصلی از بالا سمت چپ window به اندازه سایز فیلتر قرار میدهیم و هر بار آن را جابه جا میکنیم. Window که ارایه دو بعدی numpy است را به ارایه یک بعدی تبدیل کرده, و با np.med ianمدیان را به دست می اوریم. در انتها این مقدار میانه را در ایندکس مربوطه (پیکسل مرکزی که فیلتر اعمال شد)قرار میدهیم.
خروجی های نویز نمک فلفل اعمال شده به تصاویر:



۱. شکل۱۰.۱۱)تصویر سمت چپ نویزs&p چگالی۰.۰۵-تصویر سمت راست نویزs&p چگالی۰.۱

۲-۷ تمرین ۱-۲-۳

در این تمرین خواسته شده تا impulse noise با چگالی های نویز متفاوت (۰.۰۵,۰.۱,۰.۲,۰.۴) به تصویر Elaine اعمال شود. سپس با استفاده از فیلتر مدیان با valid padding این نویز ها از بین برده شوند و در انتها تأثیر سایز های مختلف فیلتر مدیان با mse بررسی شود.برای اعمال کردن نویز تابعی به نام noise_distributer نوشته شده است. پارامتر های این تابع عکس ورودی, نوع نویز که میتواند نمک فلفل یا نویز گوسی باشد.(استفاده از قسمت نویز گوسی برای سؤال بعد است). برای قسمت noise_mode مقدار s&p را قرار داد. این تابع چگالی



بر روی این تصاویر فیلتر میانه با سایز های مختلف اعمال میکنیم. فیلتر میانه یک فیلتر غیر خطی میباشد. فیلتر میانه روی نویز نمک فلفل تأثیر خوبی دارد.
تصاویر را برای نویز ۰.۴ بررسی میکنیم:

۱. شکل۱۲و۱۳)تصویر سمت چپ فیلتر میانه با سایز ۳ روی تصویر حاوی نویز sp با چگالی ۰.۴ و سمت راست تصویرفیلتر میانه با سایز ۵ روی تصویر حاوی نویز sp با چگالی ۰.۴

۱. شکل۱۴و۱۵))تصویر سمت چپ فیلتر میانه با سایز ۷ روی تصویر حاوی نویز sp با چگالی ۰.۴ و سمت راست تصویرفیلتر میانه با سایز ۹ روی تصویر حاوی نویز sp با چگالی ۰.۴

۱. شکل۱۶))تصویر فیلتر میانه با سایز ۱۱ روی تصویر حاوی نویز sp با چگالی ۰.۴

۱. جدول۲)جدول مقایسه mse های سوال ۱-۲-۳

سایز/ چگالی	۳	۵	۷	۹	۱۱
۰.۰۵	۸۵.۷۵	۵۶.۶۲	۶۱.۷۴	۷۱.۲۵	۸۳.۸۷
۰.۱	۲۰۱.۰۴	۵۸.۵۸	۶۳.۴۹	۷۲.۹۰	۸۵.۶۲
۰.۲	۶۴۹.۷۵	۶۲.۷۹	۶۶.۰۴	۷۵.۷۲	۸۸.۰۲
۰.۴	۲۴۶۰.۶	۱۹۶.۰۲	۸۲.۶۹	۸۷.۳۵	۹۷.۴۳

با توجه به اعداد جدول۲ مشاهده می‌شود که برای چگالی های کمتر نویز مانند ۰.۰۵ سایز ۳ فیلتر مدیان بهتر عمل کرده است. برای چگالی ۰.۱ و ۰.۲ سایز فیلتر ۵ کمترین mse را داشته است و بهتردعملکرده است. برای چگالی نویز ۰.۴ بر اساس جدول ۲ سایز فیلتر ۷ کمترین mse را داشته است. با توجه به شکل‌های شماره ۱۲ تا ۱۶ نیز یه همین نتیجه میرسیم که بهترین سایز فیلتر برای نویز فلفل نمک با چگالی ۰.۴ سایز ۷ است.در تصاویر شکل ۱۲ و ۱۳ هنوز حاوی نویز هستند. تصاویر ۱۵ و ۱۶ هم مقداری کیفیتشان پایین آمده است و مات شده‌اند و بهترین تصویر, شکل ۱۴ مربوط به سایز فیلتر ۷ برای نویز ۰.۴ است.

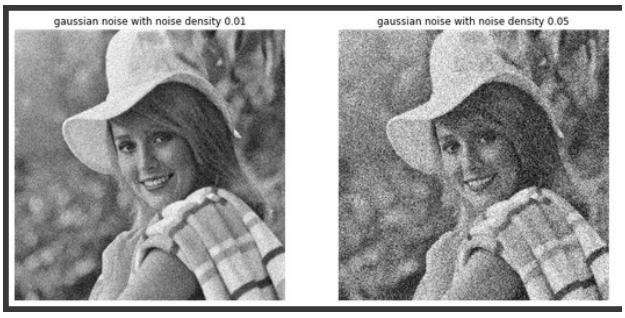
۲-۸ تمرین ۲-۲-۳

در این تمرین هدف مقایسه اثر فیلتر میانگین یکنواخت و فیلتر میانه بر روی نویز گوسی است.
در ابندا با استفاده از تابع noise_distributer که در بخش۲-۷ توضیح داده شد, نویز گوسی با واریانس های متفاوت را بعه تصویر اعمال میکنیم. باید به عنوان ورودی به noise_distributer, برای قسمت noise_mode مقدار gaussian را قرار داد.(برای نویز نمک فلفل باید s&p را قرار داد). فیلتر میانه که در قسمت ۲-۷ توضیح داده شد. تابع apply_box_filter فیلتر ماینگین یکنواخت با valid padding را اجرا میکند که به معنای no padding است وسایز تصویر خروجی کوچکتر از تصویر ورودی میشود. دلیل اینکه valid padding پیاده‌سازی شده این اینکه در تمرین ۱-۲-۳ گفته سده بود فیلتر میانه بدون پدینگ باشد و خب چون هدف در این سؤال هم مقایسه بین فیلتر میانه و میانگین یکنواخت است بهتر است که تابع apply_box_filter بدون پدینگ باشد. این تابه به عنوان ورودی عکس ورودی و سایز فیلتر را میگیرد. دراین تایع ابتدا اندازه و ابعاد عکس خروجی که کوچکتر از عکس ورودی است محاسبه میشود. سپس بر روی محدوده‌ای از عکس ورودی که در خروجی هم باقی میماند روی تصویر اصلی از بالا سمت چپ window به اندازه سایز فیلتر قرار میدهیم و هر بار

آن را جابه جا میکنیم.اعضا Window که ارایه دو بعـدی numpy است را با هم جمع کرده(با وزن ۱) و بعد حاصل جمع تقسیم بر توان ۲ سـایز فیلـتر می‌شـود طبـق رابطـع فیلتر میانگین یکنواخت. در انتها این مقدار میانگین را در ایندکس مربوطه (پیکسل مرکزی که فیلتر اعمال شد)قرار میدهیم.

ابتـدا نـویز گوسـی را اعمـال میکـنیم و بعـد فیلـتر هـای میانگین و میانه با سایز های متفاوت را و در انتها mse ها مقایسه میشوند.

خروجی های نویز گوسی اعمال شده به تصاویر:



۱. شکل۱۷و۱۸)شکل ۱۷ در سمت چپ نویز گوسی با واریانس ۰.۰۱ - شکل ۱۸ در سمت راست نویز گوسی با واریانس ۰.۰۵

۱. شکل۱۹) شکل ۱۹ در سمت راست نویز گوسی با واریانس ۰.۱

۱. جدول۳)جدول مربوط به mseهای فیلتر میانه بر روی نویز گوسی

سایز/ واراپانس	۳	۵	۷	۹	۱۱
۰.۰۱	۲۴۰.۸۸	۱۱۶.۷۱	۹۳.۷۸	۹۵.۸۸	۱۰۷.۳۳
۰.۰۵	۹۶۳.۹۵	۳۴۷.۷۳	۲۰۴.۲۸	۱۶۵.۳۱	۱۶۰.۲۲
۰.۱	۱۷۶۰.۲	۶۳۴.۳۹	۳۴۴.۷۳	۲۴۹.۷۲	۲۱۹.۲۲

۱. جدول۴)جدول مربوط به mseهای فیلتر میانگین یکنواخت بر روی نویز گوسی

سایز/ واریانس	۳	۵	۷	۹	۱۱
۰.۰۱	۶۱.۹۴	۵۱.۴۵	۴۸.۹۵	۵۱.۰۳	۵۳.۸۰
۰.۰۵	۸۶.۲۵	۷۴.۲۰	۶۷.۸۹	۶۶.۰۱	۶۵.۹۳
۰.۱	۹۳.۷۷	۸۳.۸۲	۷۹.۲۳	۷۷.۲۳	۷۶.۷۴

۱. شکل۲۰)نویز گوسی۰.۱ و فیلتر میانه با سایز ۵

۱. شکل۲۱)نویز گوسی۰.۱ و فیلتر میانگین یکنواخت با سایز ۵

۱. شکل۲۲)نویز گوسی۰.۱ و فیلتر میانه با سایز۱۱

۱. شکل۲۳)نویز گوسی۰.۱ و فیلتر میانگین یکنواخت با سایز ۱۱

با توجه به جداول۳ و۴ میتوان بیان کرد که فیلتر میانه بر روی تمام واریانس ها و تمام سایز ها از فیلتر میانگین یکنواخت mse بیشتری دارد. این به این معنا است که فیلتر میانه تأثیر کاهش نویز خوبی بر روی نویز گوسی ندارد. همچنین میتوان مشاهده کرد که فیلتر میانگین یکنواخت بر روی نویز گوسی عمل‌کرد بهتری دارد. با مقایسه شکل‌های ۲۰ و ۲۱ با یکدیگر که در هر دو واریانس نویز و سایز فیلتر یکسان است مشاهده می‌شود که در شکل ۲۱ که مربوط به box filter است, نسبت به شکل ۲۰ نویز ها بیشتر محو شده اند.
این شرایط در رابطه با مقایسه شکل‌های ۲۲ و ۲۳ هم صادق است. کیفیت شکل ۲۳ که با فیلتر میانگین است بهتر است و این فیلتر در نویز با واریانس ۰.۱ و ساسز فیلتر ۱۱ از فیلتر میانه با شرایط مشابه بهتر در از بین بردن نویز گوسی عمل کرده است.

با مقایسه شکل های۲۱ و ۲۳ که هر دو مربوط به فیلتر میانگین روی نویز گوسی با واریانس ۰.۱ هستند اما سایر فیلتر آن‌ها متفاوت است,مشاهده می‌شود که اثر noise reduction فیلتر میانگین با سایز ۱۱ نسبت به سایز ۵ بیشتر بوده است. به این معنا که برای نویز های بیشتر باید از فیلتر میانگین با سایز بیشتر استفاده کرد تا smotthing بیشتری اعمال کند.

۲-۹ تمرین ۱-۴-۳

باید خروجی این سه فیلتر در تصویر با هم مقایسه شوند.
خروجی هر سه فیلتر چون اختلاف افقی را محاسبه میکنند, لبه های عمودی است.
فیلتر a اما لبه های نازک تری نسبت به b,c به دست میدهد.
چون دقت a در لبه یابی کمتر است مشاهده می‌شود که برخی از نویز های محیط و پشت صحنه را هم بیشتر از b,c نشان میدهد.
اما لبه های b,c چون لبه های قوی‌تر را تشخیص میدهند به این نویز ها کمتر حساس اند و آن‌ها را کمتر نشان داده اند.
فیلتر b,c پایین گذار هستند. به این معنا که نویز هارا کاهش میدهند و همینطور که دیده می‌شود نویز های پس زمینه نتیجه فیلتر a در آن‌ها دیده نمیشود و کمتر است. تفاوت bوc این اینکه c وزن دار است و وزن بیشتری را به پیکسل مرکزی اختصاص داده که باعث کاهی نویز بیشتری میشود.
فیلتر c بهترین فیلتر برای گرادیان ۲ بعدی است چون به علت وزن دار بودن پیکسل میانی دقت بیشتری دار و با جرخاندن آن به اندازه ۹۰ دجه فیلتر لازم برا پیدا کردن لبه های افقی به دست می آید.

۱. شکل۲۴)نتیجه اعمال فیلتر a- شناسایی لبه های عمودی نازک تر و کوچکتر

۱. شکل۲۵)نتیجه اعمال فیلتر b- لبه های ذخیم تر و حساسیت کمتر نسبت به نویز نسبت به a

۱. شکل۲۶)نتیجه اعمال فیلتر c- لبه های ذخیم تر و

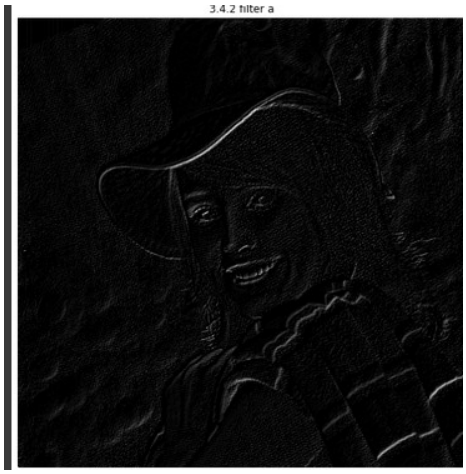
۲-۱۰ تمرین ۲-۴-۳

در این تمرین هر دو فیلتر داد هشده فیلتر های ۲ در ۲ روبرت هستند. فیلترa لبه های ۴۵ درجه را میباد و فیلتر b لبه های ۴۵- یا ۱۳۵ درجه را میابد. به همین دلیل در خروجی فیلتر a لبه های رو حوله که نزدیک به ۴۵ درجه هستند یافت شده‌اند اما در قسمت موها که لبه ها به ۱۳۵ درجه نردیک تر است هیچ لبه ای وجود ندارد و این لبه ها را فیلتر b دیتکت کرده است.

فیلتر روبرت نسبته به سوبل در سؤال قبل

۱) روبرت در پیدا کردن لبه ها ضعیف‌تر از سوبل عمل میکند. چون در سوبل میانگین پیکسل های بیشتری با هم مقایسه می‌شوند و برای همین کامل‌تر و بهتر است.

۲) روبرت نسبت به سوبل بیشتر به نویز حساس است و نویز هارا بیشتر تقویت میکند. این مورد هم دلیلی شبیه مورد ۱ دارد, چون با ضعیف بودن درم.رد یافتن لبه ها ممکن است نویز هارا بیلبد به صورت اشتباهی به جای لبه ها.



۱. شکل۲۷)اعمال فیلتر aوکشف لبه های ۴۵ درجه

شکل۲۸ ۱: لبه های ۱۳۵ درجه بعد از استفادهع از فیلتر b

۲-۱۱ تمرین ۱-۵-۳

در این تمـرین بایـد بـا اسـتفاده از فرمـول داده شده فیلتر تقویت بالا ایجاد کـنیم.در ابتـدا بایـد تصـویر blurیجـاد شـود. بـرای اینکـار از تـابع Guassian blur از cv2 استفاده میشود.
به عنوان ورودی, عکس ورودی و سـایز فیلـتر و واریـانس را دریـافت میکنـد کـه بـرای واریـانس مقدار استاندارد ۲ اعمال شده است. این فیلتر را با سایز های ۳و۵و۷و۹و۱۱ اعمال میکنیم.
برای اعمال کردن فیلتر فرمول داده شده فرمول سـؤال تـابع unsharp_masking_formula پیاده‌سازی شـده اسـت کـه مقـدار آلفـا و عکس ورودی و blur شده تصویر را دریافت میکند.

این فرمول زمانی که آلفا برابر ۱ باشـد خـروجی آن همان تصویر مـات شـده اسـت و اگـر آلفـا ۰ باشد خروجی آن تصویر ورودی است.
درواقع خـروجی هـا این تـابع بین تصـویر مـات شده با فیلتر گوسی و تصویر اصلی است.
ای فرمول لبه هارا تقویت نمیکند.

شکل۲۹ ۱: عکس اصلی و خروجی تابع با آلفا ۰ بر روی گوسی با سایز ۱۱ و تصویر اصلی, همان تصویر اصلی است.

شکل۳۰ ۱)عکس اصلی و خروجی تابع با آلفا ۱ بر روی گوسی با سایز ۱۱ و تصویر اصلی, همان تصویر مات شده است.

راه حل برای شارپ کـردن لبـه هـا این اسـت کـه تفـاوت عکس اصلی و عکس مات شده که که لبه ها هستند با آلفا بزرگ‌تر از ۱ به تصویر اصلی اضافه شوند.
این الگـــو در تـــابع unsharp_masking_improved

شکل۳۱ ۱))نتیجه اعمال فیلتر تقویت بالا بر روی عکس مات شده گوسی با فیلتر سایز ۳, با آلفا ۲ در تصویر چپ و آلفا ۳ در تصویر راست پیاده‌سازی شده است.

همان‌طور که مشاهده می‌شوند لبه ها شارپ تر شـده‌اند و در تصویر با آلفا ۳ این شارپ شدن بیشتر مشاه ه میشود.

۳ کد تمرینات

```
# -*- coding: utf-8 -*-
"""cv_hw3_filters_saraGhavampour_9812762781.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16EApS67OkKQpXiBmjVgm4eGGgQJYHW0I
"""

!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
# %matplotlib inline

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

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()

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')

"""3.1.3"""

def apply_box_filter_same_padding(img,window_size):
  out=img.copy()
  pad = floor((window_size-1)/2)
  img=np.pad(img.copy(),((pad,pad),(pad,pad)),mode='constant',constant_values=(0,0))
  filter = np.ones((window_size,window_size))
  for i in range(img.shape[0]-window_size+1):
    for j in range(img.shape[1]-window_size+1):
      window=img[i:i+window_size,j:j+window_size]
      out[i,j]=np.sum(window*filter)/(window_size**2)

  return out.astype('uint8')

# for i in range (0,20):
#   elaine_box_filter_same_1 = apply_box_filter_same_padding(elaine_img,3)

# show_img(elaine_box_filter_same_1)

for i in range (0,50):
  elaine_box_filter_same_2 = apply_box_filter_same_padding(elaine_img,3)

show_img(elaine_box_filter_same_2)

# for i in range (0,150):
#   elaine_box_filter_same_3 = apply_box_filter_same_padding(elaine_img,3)

# show_img(elaine_box_filter_same_3)

# for i in range (0,350):
#   elaine_box_filter_same_3 = apply_box_filter_same_padding(elaine_img,3)

# show_img(elaine_box_filter_same_3)

"""3.1.4"""

from skimage.util import random_noise
def noise_distributer(img,noise_mode,noise_density,var):
  if noise_mode == 's&p':
    out=random_noise(img,noise_mode,amount=noise_density)

  if noise_mode == 'gaussian': # mean?
    out=random_noise(img,noise_mode,var=var)
  return (out*255).astype('uint8')

elaine_noise = noise_distributer(elaine_img,'gaussian',0,0.05)
show_img(elaine_noise,figsize=8)

elaine_denoise1=apply_box_filter_same_padding(elaine_noise,3)
show_img(elaine_denoise1,figsize=6)

elaine_denoise2=apply_box_filter_same_padding(elaine_noise,5)
show_img(elaine_denoise2,figsize=6)

elaine_denoise3=apply_box_filter_same_padding(elaine_noise,7)
show_img(elaine_denoise3,figsize=6)

elaine_denoise4=apply_box_filter_same_padding(elaine_noise,11)
show_img(elaine_denoise4,figsize=6)

"""3.1.5"""

print(mean_squared_error(elaine_img,elaine_denoise1))
print(mean_squared_error(elaine_img,elaine_denoise2))
print(mean_squared_error(elaine_img,elaine_denoise3))
print(mean_squared_error(elaine_img,elaine_denoise4))
```

```
"""3.1.6"""

elaine_120_blured=cv2.imread('120.jpeg')
elaine_120_blured = cv2.cvtColor(elaine_120_blured,cv2.COLOR_BGR2GRAY)
plt.imshow(elaine_120_blured,cmap='gray')
plt.axis('off')
plt.show()

laplacian_HV_filter = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
elaine_120_blured_laplacian_1 = cv2.filter2D(src=elaine_120_blured, ddepth=-1, kernel=laplacian_HV_filter)
show_img(elaine_120_blured_laplacian_1)

copy = elaine_120_blured.copy()
for i in range(0,2):
  copy = cv2.filter2D(src=copy, ddepth=-1, kernel=laplacian_HV_filter)
show_img(copy)

copy = elaine_120_blured.copy()
for i in range(0,3):
  copy = cv2.filter2D(src=copy, ddepth=-1, kernel=laplacian_HV_filter)
show_img(copy)

copy = elaine_120_blured.copy()
for i in range(0,4):
  copy = cv2.filter2D(src=copy, ddepth=-1, kernel=laplacian_HV_filter)
show_img(copy)

copy = elaine_120_blured.copy()
for i in range(0,5):
  copy = cv2.filter2D(src=copy, ddepth=-1, kernel=laplacian_HV_filter)
show_img(copy)

"""3.2

3.2.1
"""

noisy_sp_img_results=[]

noisy_sp_img_results.append(noise_distributer(elaine_img,'s&p',0.05,0))
noisy_sp_img_results.append(noise_distributer(elaine_img,'s&p',0.1,0))
noisy_sp_img_results.append(noise_distributer(elaine_img,'s&p',0.2,0))
noisy_sp_img_results.append(noise_distributer(elaine_img,'s&p',0.4,0))
titles=['salt&pepper noise with noise density 0.05','salt&pepper noise with noise density 0.1','salt&pepper noise with noise density 0.2','salt&pepper noise with noise density 0.4']

show_img(noisy_sp_img_results, figsize=30,title=titles)

def apply_median_filter(img,window_size):
  img_copy=img.copy()
  shift = floor(window_size/2)
  x,y=img_copy.shape
  out = np.zeros((x-2*shift,y-2*shift))
  for i in range(shift,x-shift):
    for j in range(shift,y-shift):
      window = img_copy[i-shift:i+shift,j-shift:j+shift]
      window = window.ravel()
      median=np.median(window)
      out[i-shift,j-shift]=median

  return out

elaine_sp_01_median_5=apply_median_filter(noisy_sp_img_results[3],3)
show_img(elaine_sp_01_median_5)

elaine_sp_01_median_5

filter_wondow_sizes=[3,5,7,9,11]
median_filter_results=[]
median_filter_results_titles=[]
noise_densities=[0.05,0.1,0.2,0.4]

for (index,noisy_img) in enumerate(noisy_sp_img_results):
  median_results_window=[] # each row for each imahe with density
  median_results_window_title=[]
  for size in  filter_wondow_sizes:
    median_img=apply_median_filter(noisy_img,size)
    median_results_window.append(median_img)# each col is size
    median_results_window_title.append(f'median filter size: {size} on s&p noise: {noise_densities[index]}')
    shift=floor(size/2)
    x,y=elaine_img.shape
    bounded_noisey_image = elaine_img[shift:x-shift,shift:y-shift]
    mse = mean_squared_error(bounded_noisey_image,median_img)
    print(f'mse elaine_img & median filter {size} size on sp noise density {noise_densities[index]} is :{mse}')
  print('-----------------------------------------------------------')
  median_filter_results.append(median_results_window)
  median_filter_results_titles.append(median_results_window_title)

for i in range(0,np.array(median_filter_results).shape[0]):
  show_img(median_filter_results[i], figsize=30,title=median_filter_results_titles[i])

"""3.2.2"""

noisy_gaussian_img_results=[]

noisy_gaussian_img_results.append(noise_distributer(elaine_img,'gaussian',0,0.01))
noisy_gaussian_img_results.append(noise_distributer(elaine_img,'gaussian',0,0.05))
noisy_gaussian_img_results.append(noise_distributer(elaine_img,'gaussian',0,0.1))

titles=['gaussian noise with noise density 0.01','gaussian noise with noise density 0.05','gaussian noise with noise density 0.1']

show_img(noisy_gaussian_img_results, figsize=20,title=titles)

filter_wondow_sizes=[3,5,7,9,11]
median_filter_results=[]
median_filter_results_titles=[]
noise_densities=[0.01,0.05,0.1]

for (index,noisy_img) in enumerate(noisy_gaussian_img_results):
  median_results_window=[] # each row for each imahe with density
  median_results_window_title=[]
  for size in  filter_wondow_sizes:
    median_img=apply_median_filter(noisy_img,size)
    median_results_window.append(median_img)# each col is size
    median_results_window_title.append(f'median filter size: {size} on gaussian noise: {noise_densities[index]}')
    shift=floor(size/2)
    x,y=elaine_img.shape
    bounded_noisey_image = elaine_img[shift:x-shift,shift:y-shift]
    mse = mean_squared_error(bounded_noisey_image,median_img)
    print(f'mse elaine_img & median filter {size} size on gaussian noise density {noise_densities[index]} is :{mse}')
  print('-----------------------------------------------------------')
```

```
  median_filter_results.append(median_results_window)
  median_filter_results_titles.append(median_results_window_title)

for i in range(0,np.array(median_filter_results).shape[0]):
  show_img(median_filter_results[i], figsize=30,title=median_filter_results_titles[i])

def apply_box_filter(img,window_size):
  img_copy=img.copy()
  shift = floor(window_size/2)
  x,y=img_copy.shape
  out = np.zeros((x-2*shift,y-2*shift))
  for i in range(shift,x-shift):
    for j in range(shift,y-shift):
      window = img_copy[i-shift:i+shift+1,j-shift:j+shift+1]
      #print(window)
      box_avg = (np.sum(window)) / (window_size**2)
      out[i-shift,j-shift]=box_avg

  return out.astype('uint8')

filter_wondow_sizes=[3,5,7,9,11]
boxfilter_filter_results=[]
boxfilter_filter_results_titles=[]
noise_densities=[0.01,0.05,0.1]

for (index,noisy_img) in enumerate(noisy_gaussian_img_results):
  boxfilter_results_window=[] # each row for each imahe with density
  boxfilter_results_window_title=[]
  for size in  filter_wondow_sizes:
    boxfilter_img=apply_box_filter(noisy_img,size)
    boxfilter_results_window.append(boxfilter_img)# each col is size
    boxfilter_results_window_title.append(f'boxfilter filter size: {size} on gaussian noise: {noise_densities[index]}')
    shift=floor(size/2)
    x,y=elaine_img.shape
    bounded_noisey_image = elaine_img[shift:x-shift,shift:y-shift]
    mse = mean_squared_error(bounded_noisey_image,boxfilter_img)
    print(f'mse elaine_img & boxfilter filter {size} size on gaussian noise density {noise_densities[index]} is :{mse}')
  print('--------------------------------------------------------')
  boxfilter_filter_results.append(boxfilter_results_window)
  boxfilter_filter_results_titles.append(boxfilter_results_window_title)

for i in range(0,np.array(boxfilter_filter_results).shape[0]):
  show_img(boxfilter_filter_results[i], figsize=30,title=boxfilter_filter_results_titles[i])

def apply_guessian_filter(img,window_size):
  img_copy=img.copy()
  shift = floor(window_size/2)
  x,y=img_copy.shape
  out = np.zeros((x-2*shift,y-2*shift))
  for i in range(shift,x-shift):
    for j in range(shift,y-shift):
      window = img_copy[i-shift:i+shift+1,j-shift:j+shift+1]
      #print(window)
      box_avg = (np.sum(window)) / (window_size**2)
      out[i-shift,j-shift]=box_avg

      x, y = np.mgrid[-kernel_size//2 + 1:kernel_size//2 + 1, -kernel_size//2 + 1:kernel_size//2 + 1]
      g = np.exp(-((x**2 + y**2)/(2.0*sigma**2)))
      return g/g.sum()

  return out.astype('uint8')

"""3.4"""

edge_detector_341a=1/2*np.array([1,0,-1])
edge_detector_341b=1/6*np.array([[1,0,-1],[1,0,-1],[1,0,-1]])
edge_detector_341c=1/8*np.array([[1,0,-1],[2,0,-2],[1,0,-1]])

def convolve2D(img,filter):
  f=filter.shape[0]
  pad = floor((f-1)/2)
  out=img.copy()
  img=np.pad(img.copy(),((pad,pad),(pad,pad)),mode='constant',constant_values=(0,0))
  for i in range(img.shape[0]-f+1):
    for j in range(img.shape[1]-f+1):
      window=img[i:i+f,j:j+f]
      out[i,j]=np.sum(window*filter)

  return out

elaine_341a = convolve2D(elaine_img,edge_detector_341a)
show_img(elaine_341a,figsize=10)

elaine_341a = cv2.filter2D(src=elaine_img, ddepth=-1, kernel=edge_detector_341a)
show_img(elaine_341a,figsize=10)

elaine_341b = cv2.filter2D(src=elaine_img, ddepth=-1, kernel=edge_detector_341b)
show_img(elaine_341b,figsize=10)

elaine_341c = cv2.filter2D(src=elaine_img, ddepth=-1, kernel=edge_detector_341c)
show_img(elaine_341c,figsize=10)

edge_detector_342a=np.array([[1,0],[0,-1]])
edge_detector_342b=np.array([[0,1],[-1,0]])
elaine_342a = cv2.filter2D(src=elaine_img, ddepth=-1, kernel=edge_detector_342a)
elaine_342b = cv2.filter2D(src=elaine_img, ddepth=-1, kernel=edge_detector_342b)
show_img(elaine_342a,elaine_342b,title=['3.4.2 filter a','3.4.2 filter b'],figsize=20)

"""3.5     unsharp masking"""

lena_gus3=cv2.GaussianBlur(lena_img,(3,3),2)
lena_gus5=cv2.GaussianBlur(lena_img,(5,5),2)
lena_gus7=cv2.GaussianBlur(lena_img,(7,7),2)
lena_gus9=cv2.GaussianBlur(lena_img,(9,9),2)
lena_gus11=cv2.GaussianBlur(lena_img,(11,11),2)

show_img(lena_gus11,lena_img)

def unsharp_masking_formula(alpha,img,smooth_img):
  out = (1-alpha)*img + alpha*smooth_img
  return out

lena_gus3_alpha0=unsharp_masking_formula(0,lena_img,lena_gus3)
lena_gus3_alpha1=unsharp_masking_formula(1,lena_img,lena_gus3)
show_img(lena_img,lena_gus3_alpha0,lena_gus3_alpha1,title=['lena_img','lena_gus3_alpha0','lena_gus3_alpha1'],figsize=16)

lena_gus5_alpha0=unsharp_masking_formula(0,lena_img,lena_gus5)
lena_gus5_alpha1=unsharp_masking_formula(1,lena_img,lena_gus5)
show_img(lena_img,lena_gus5_alpha0,lena_gus5_alpha1,title=['lena_img','lena_gus5_alpha0','lena_gus5_alpha1'],figsize=16)
```

```
lena_gus7_alpha0=unsharp_masking_formula(0,lena_img,lena_gus7)
lena_gus7_alpha1=unsharp_masking_formula(1,lena_img,lena_gus7)
show_img(lena_img,lena_gus7_alpha0,lena_gus7_alpha1,title=['lena_img','lena_gus7_alpha0','lena_gus7_alpha1'],figsize=16)

lena_gus9_alpha0=unsharp_masking_formula(0,lena_img,lena_gus9)
lena_gus9_alpha1=unsharp_masking_formula(1,lena_img,lena_gus9)
show_img(lena_img,lena_gus9_alpha0,lena_gus9_alpha1,title=['lena_img','lena_gus9_alpha0','lena_gus9_alpha1'],figsize=16)

lena_gus11_alpha0=unsharp_masking_formula(0,lena_img,lena_gus11)
lena_gus11_alpha1=unsharp_masking_formula(1,lena_img,lena_gus11)
show_img(lena_img,lena_gus11_alpha0,lena_gus11_alpha1,title=['lena_img','lena_gus11_alpha0','lena_gus11_alpha1'],figsize=16)

def unsharp_masking_improved(alpha,img,smooth_img):
  out = img + alpha*(img-smooth_img)
  return out

lena_gus3_alpha0=unsharp_masking_improved(2,lena_img,lena_gus3)
lena_gus3_alpha1=unsharp_masking_improved(3,lena_img,lena_gus3)
show_img(lena_img,lena_gus3_alpha0,lena_gus3_alpha1,title=['lena_img','lena_gus3_alpha2','lena_gus3_alpha3'],figsize=16)

lena_gus5_alpha0=unsharp_masking_formula(2,lena_img,lena_gus5)
lena_gus5_alpha1=unsharp_masking_formula(3,lena_img,lena_gus5)
show_img(lena_img,lena_gus5_alpha0,lena_gus5_alpha1,title=['lena_img','lena_gus5_alpha2','lena_gus5_alpha3'],figsize=16)

lena_gus7_alpha0=unsharp_masking_formula(2,lena_img,lena_gus7)
lena_gus7_alpha1=unsharp_masking_formula(3,lena_img,lena_gus7)
show_img(lena_img,lena_gus7_alpha0,lena_gus7_alpha1,title=['lena_img','lena_gus7_alpha2','lena_gus7_alpha3'],figsize=16)

lena_gus9_alpha0=unsharp_masking_formula(2,lena_img,lena_gus9)
lena_gus9_alpha1=unsharp_masking_formula(3,lena_img,lena_gus9)
show_img(lena_img,lena_gus9_alpha0,lena_gus9_alpha1,title=['lena_img','lena_gus9_alpha2','lena_gus9_alpha3'],figsize=16)

lena_gus7_alpha0=unsharp_masking_formula(2,lena_img,lena_gus7)
lena_gus7_alpha1=unsharp_masking_formula(3,lena_img,lena_gus7)
show_img(lena_img,lena_gus7_alpha0,lena_gus7_alpha1,title=['lena_img','lena_gus7_alpha2','lena_gus7_alpha3'],figsize=16)
```