تمرین سوم

آریا ابراهیمی

چکیده

در این سری تمرین، با فیلتر ها در حوزه مکان آشنا می‌شویم و کارایی و تاثیر هرکدام را بر تصویر، بررسی می‌کنیم. همچنین بررسی میکنیم که کدام فیلتر برای کدام نویز تاثیر بهتری خواهد داشت.
در بخش اول تمرین به بررسی *box filter* می‌پردازیم، در بخش دوم که مربوط به *median filter* است، تاثیر آن‌را بر روی عکس هایی با نویز *salt and pepper* بررسی میکنیم که مشاهده میشود برای این نویز کارایی خوبی دارد. همچنین برای تصاویر با نویز گوسی نیز این کار را تکرار میکنیم و مقایسه‌ای با *box filter* انجام میدهیم که مشاهده میشود *box filter* برای نویز های گوسی کاربرد بهتری دارد.
در بخش های بعد نیز به ترتیب فیلتر های لبه یاب مختلف و همچنین *unsharp masking* بررسی می‌شوند.

۳.۱ *Box Filter*

۳.۱.۱

یک مشکلی که *box filter* ها دارند این است که از آنجایی که مقادیر آنها یکنواخت و همگن است، در نتیجه ضرایب یکسانی را برای ناحیه‌ای از عکس که فیلتر روی آن اعمال میشود در نظر میگیرند. نتیجه این خاصیت این می‌شود که در لبه ها که اختلاف بین دو پیکسل زیاد است، میانگین گرفته شود که باعث خراب شدن لبه می‌شود.

مشکل دیگر، این است که برای تصاویری که نویز *salt and pepper* دارند، استفاده از این فیلتر مناسب نمی باشد زیرا با میانگین گرفتن، مقادیر *salt* ها و *pepper* ها را نیز محاسبه میکنند در نتیجه باعث خراب شدن مقادیر میشود.

مشکل دیگری که میتوان به آن اشاره کرد این است که *box filter* ها در واقع *linear smoothing* میکنند و اگر بخواهیم زیاد *smooth* کنیم، تغییر مقادیر زیاد به چشم می‌آید و خیلی *sharp* است. برای رفع این مشکل میتوان از *gaussian filter* استفاده کرد که در لبه ها مقادیر *smooth* تری تولید میکنند.

۳.۱.۲

همانطور که گفته شد، اگر نویز *salt and pepper* داشته باشیم، *box filter* مناسب نمیباشد ولی اگر نویز گوسی داشته باشیم اعمال کردن *box filter* چند مرتبه باعث حذف نویز بیشتر میشود ولی تصویر خیلی *smooth* میشود و تقریبا لبه ها از بین میروند.

۳.۱.۳

همانطور که در تصاویر زیر مشاهده می‌شود، با اعمال فیلتر به صورت مکرر، تصویر *blur* میشود و جزئیات کمتر قابل مشاهده خواهند بود. اگر بیش از ۱۰۰ بار اعمال کنیم، تقریبا ورژن سگمنت شده ای از تصویر را مشاهده میکنیم که جزئیات خیلی کمی در آن قابل مشاهده است و اگر تعداد اعمال فیلترمان به بی‌نهایت میل کند، تصویری یک‌دست از یک مقدار *graylevel* به دست خواهد آمد.

شکل ۱) اعمال box filter با اندازه ۳ به تعداد ۱۵ بار.

شکل ۲) اعمال box filter با اندازه ۳ به تعداد ۵۰ بار.

شکل ۳) اعمال box filter با اندازه ۳ به تعداد ۱۰۰ بار.

۳.۱.۴

هر چه سایز فیلتر بزرگتر باشد، محدوده بیشتری را میانگین گیری میکند در نتیجه خاصیت *smoothing* افزایش پیدا میکند و برای نویز گوسی، کاهش نویز بیشتری خواهیم داشت.

شکل ۴) تصویر با نویز گوسی $\sigma = 0.01$

شکل ۵) اعمال *box filter* با سایز ۳ بر روی تصویر نویزی.

شکل ۶) اعمال *box filter* با سایز ۷ بر روی تصویر نویزی.

شکل ۷) اعمال *box filter* با سایز ۱۱ بر روی تصویر نویزی.

همان‌طور که مشاهده می‌شود، با افزایش سایز فیلتر، تصاویر *smooth* تری به دست می‌آوریم که نویز در آنها کمتر قابل مشاهده است ولی از طرفی باعث از بین رفتن لبه ها نیز می‌شود.

۳.۱.۵

با توجه به نتایج و تصاویر به دست آمده در قسمت ۳.۱.۴ ، میتوان مشاهده کرد که فیلتر با سایز ۷ ، نسبتا *tradeoff* خوبی میان *blurring* و کاهش نویز دارد. همچنین فیلتر با سایز ۹ نیز نتیجه قابل قبولی را ارائه میدهد. با بررسی *mse* نیز به همین نتیجه میرسیم. جدول *mse* برای تصویر ارائه شده در بخش ۳.۱.۴ به شرح زیر است:

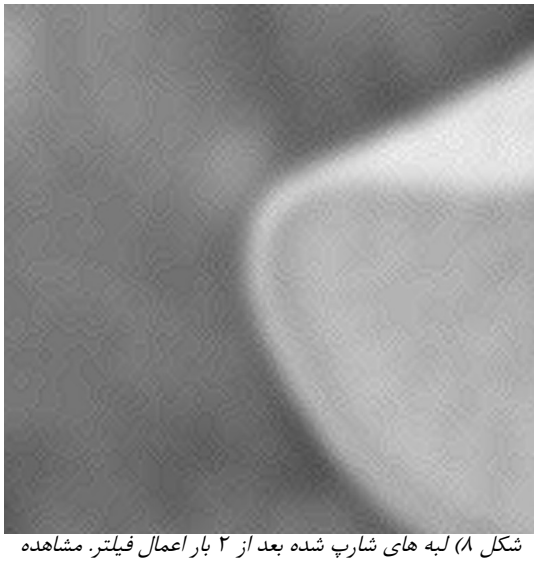
جدول ۱) *mse* های مربوط به *box filter* های مختلف اعمال شده

سایز فیلتر	۳	۵	۷	۹	۱۱
mse	۶۱.۷	۵۱.۲	۴۸.۴	۵۰.۳	۵۲.۹

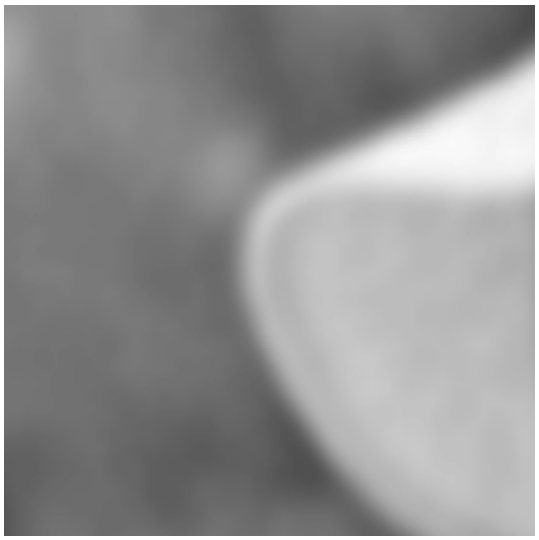
۳.۱.۶

این فیلتر برای *sharp* کردن تصاویر استفاده میشود. برای این تمرین از شکل ۱ استفاده شده است.

بعد از ۲ بار اعمال این فیلتر، شاهد بهبود نسبی در لبه های تصویر *blur* هستیم ولی مشکلی که به وجود می آید این است که یک سری تفاوت های *sharp* در تصویر ایجاد می‌شود. در دفعات ابتدایی این تفاوت خیلی قابل مشاهده نیست ولی در تصویر زوم شده میتوان این تفاوت را مشاهده کرد.



شکل ۸) لبه های شارپ شده بعد از ۲ بار اعمال فیلتر. مشاهده می‌شود علاوه بر شارپ تر شدن تصویر لبه هایی در کل تصویر ایجاد شده است

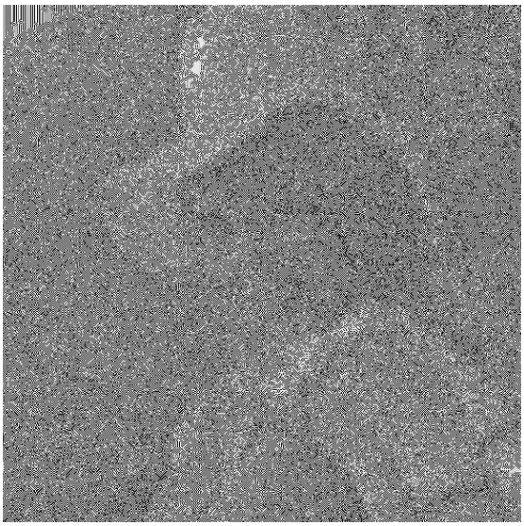


شکل ۹) ناحیه مربوط به شکل ۸ در تصویر *blur* شده.

این لبه های ایجاد شده باعث میشوند تا در دفعات بعدی اعمال فیلتر، آن‌ها نیز تقویت و شارپ شوند و که باعث ایجاد

نویز در تصویر میشود. نتایج این بررسی را میتوان در عکس های۱۰ و ۱۱ مشاهده کرد.

شکل ۱۰) همان‌طور که گفته شد، بعد از ۳ بار اعمال فیلتر، مشاهده می‌شود که لبه های ایجاد شده در مراحل قبلی اعمال فیلتر، در این تصویر شارپ تر شده اند که باعث ایجاد لبه هایی همانند نویز شده است.



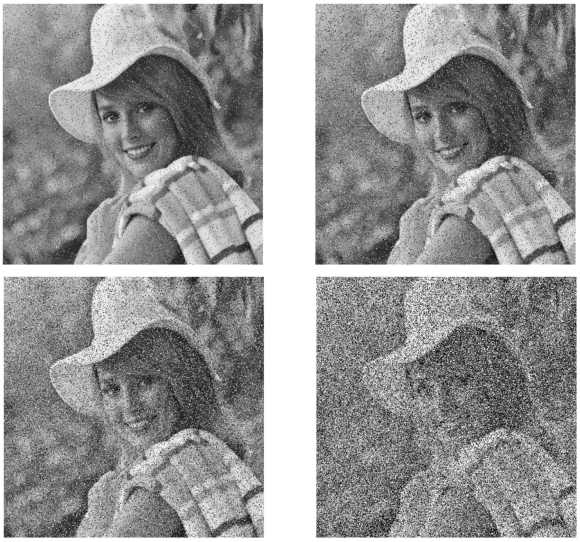
شکل ۱۱) بعد از ۵ بار اعمال کردن فیلتر، تقریبا جزئیات تصویر از بین رفته است و فقط میتوان کلیات آن را مشاهده کرد.

۳.۲ *Median Filter*

۳.۲.۱

در این سوال خواسته شده است تا بر روی تصویر *Elaine* نویز *salt and pepper* با *density* های متفاوت ایجاد کرده و سپس با استفاده از *median filter* آن نویز هارا برطرف کرده و نتیجه را با تصویر اصلی مقایسه کنیم.

برای ایجاد نویز از تابع *random_noise* موجود در کتابخانه *skimage* استفاده شده است. بدین منظور یک تابع جداگانه به نام *add_noise* تعریف شده که ۲ حالت *s&p* برای نویز *salt and pepper* و *gaussian* برای نویز *gaussian* دریافت کرده و با استفاده از تابع *random_noise*، نویز را به تصویر اضافه میکند. از آنجایی که تابع *random_noise* مقدار بین ۰ و ۱ برمیگرداند، برای خروجی در ۲۵۵ ضرب می‌شود. تابع *add_noise* را در پیوست میتوان مشاهده کرد.



شکل ۱۲) نویز salt and pepper اضافه شده به تصویر با density های $\rho = 0.05, \rho = 0.1, \rho = 0.2, \rho = 0.5$

بر روی تصاویر به دست آمده، *median filter* با ابعاد متفاوت را اعمال میکنیم. از آنجایی که فیلتر *median* فیلتری غیر خطی میباشد و میانه یک ناحیه را میگیرد، میزان تاثیر نویز در آن کمتر میشود. در نویز *salt and pepper* نویز به صورت پیکسل‌های سفید و سیاه میباشد در نتیجه در میانه گیری، مقادیر سیاه در ابتدای مقادیر سورت شده و مقادیر سفید در انتهای لیست سورت شده قرار می‌گیرند و مقادیر اصلی مربوط به یک ناحیه با احتمال زیادی در وسط قرار میگیرند که با میانه گیری میتوان آنها را به دست آورد و تخمین خوبی برای یک پیکسل با استفاده از همسایه هایش انجام داد.

برای اعمال فیلتر، تابعی به نام *filter* طراحی شده است که عکس، نوع فیلتر دریافتی (*box*، *median* و یا *gaussian*) و سایز فیلتر را دریافت کرده و بر اساس سایز فیلتر، *padding* انجام میدهد و سپس عملیات *convolve* را انجام میدهد. (کد این قسمت در پیوست قابل مشاهده است) نتایج به دست آمده برای این قسمت به شرح زیر است:

جدول ۲) مقادیر *mse* بین تصویر اصلی و نتیجه اعمال *median filter* بر روی تصاویر نویزدار شده با *salt and pepper*

	3×3	5×5	7×7	9×9	11×11
$\rho=0.05$	۲۸.۱۸	۳۲.۹۱	۳۵.۱۸	۳۸.۸۲	۴۱.۸۷
$\rho=0.1$	۲۹.۲۵	۳۳.۵۲	۳۵.۶۴	۳۹.۰۹	۴۲.۰۷
$\rho=0.2$	۳۱.۸۶	۳۵	۳۶.۶۶	۳۹.۹۸	۴۲.۷۵
$\rho=0.5$	۴۶.۴۳	۴۲.۳۹	۴۱.۶۶	۴۳.۸۷	۴۶.۱۳

مشاهده میشود که وقتی *density* نویز کم است، فیلتر کوچک با سایز ۳ بهتر عمل کرده است و زمانی که $\rho=0.5$ است، فیلتر با سایز ۷ بهتر عمل کرده است. میتوان نتیجه گرفت برای نویز با چگالی بیشتر، فیلتر بزرگتر میتواند بهتر عمل کند. در شکل زیر، نتایج مروبط به فیلتر با سایز ۳ برای $\rho=0.05$ و فیلتر با سایز ۷ برای $\rho=0.5$ قابل مشاهده است.

شکل ۱۳) نتایج مروبط به فیلتر با سایز ۳ برای $\rho=0.5$ و فیلتر با سایز ۷ برای $\rho=0.05$

۳.۲.۲

در این قسمت به مقایسه بین *median filter* و *box filter* برای نویز های گوسی می‌پردازیم. مشاهده میشود که *box filter* برای نویز های گوسی از *median filter* بهتر عمل میکند. همانند قسمت قبل عمل کرده و از همان توابع برای محاسبات استفاده شده است. نتایج به دست آمده به شرح زیر میباشد:

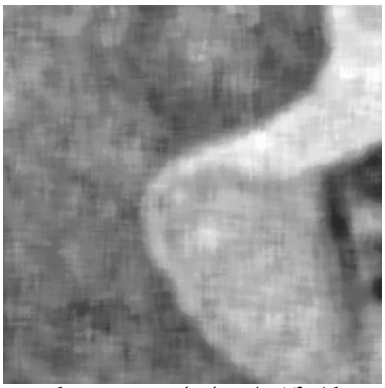
جدول ۳) تفاوت *mse* میان تصویر اصلی و *median filter* اعمال شده با سایز های مختلف بر روی عکس نویز دار با انحراف معیار های متفاوت.

	3×3	5×5	7×7	9×9	11×11
$\sigma=0.01$	۶۹.۳	۵۶.۳۴	۵۱.۴۹	۵۱.۷۹	۵۳.۴۸
$\sigma=0.05$	۹۲.۸۲	۸۰.۴۲	۷۲.۱۱	۶۷.۷	۶۵.۶۷
$\sigma=0.1$	۹۹.۵۹	۸۹.۶	۸۱.۶۲	۷۶.۵۱	۷۳.۵۴

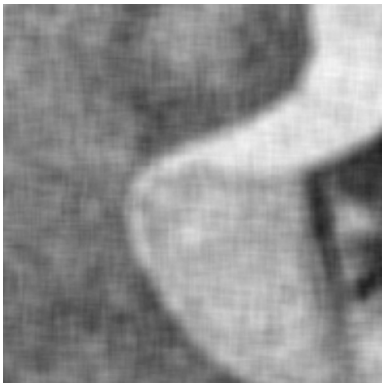
جدول ۴) تفاوت *mse* میان تصویر اصلی و *box filter* اعمال شده با سایز های مختلف بر روی عکس نویز دار با انحراف معیار های متفاوت.

	3×3	5×5	7×7	9×9	11×11
$\sigma=0.01$	۶۲.۰۸	۵۲.۱۳	۴۹.۷۱	۵۱.۹۸	۵۴.۹
$\sigma=0.05$	۸۶.۳۵	۷۴.۳۱	۶۸.۴۹	۶۶.۶۴	۶۶.۶۶
$\sigma=0.1$	۹۳.۸۴	۸۴.۳۶	۷۹.۴۵	۷۷.۶۴	۷۷.۲۹

همان‌طور که مشاهده میشود، *mse* در جدول مربوط به *box filter* کمتر است و میتوان نتیجه گرفت *box filter* برای نویز *gaussian* از *median filter* بهتر عمل میکند. اما نکته ای که حائز اهمیت است این است که مشاهده می‌شود برای $\sigma=0.1$ مقدار *mse* در *median* فیلتر کمتر از *box filter* است در صورتی که عکس خروجی مربوط به *box filter* قابل قبول تر است. این برای این است که در *box filter* با اندازه بزرگتر، *smoothing* بیشتر است و برای همین مقادیر *graylevel* در مقایسه با عکس اصلی تغییر بیشتری میکنند ولی خاصیت کاهش نویز بیشتر از *median filter* میباشد.
همچنین مشاهده میشود با افزایش انحراف معیار نویز، فیلتر های بزرگتر عملکرد بهتری از خود نشان میدهند.



شکل ۱۴) ناحیه ای از تصویر نویزی که *median filter* با سایز ۱۱ بر آن اعمال شده است.



شکل ۱۵) ناحیه ای از تصویر نویزی که *box filter* با سایز ۱۱ بر آن اعمال شده است.

۳.۲.۳

برای حذف نویز *salt and pepper* و *gaussian* به صورت همزمان، میتوان از آنجایی که فیلتر میانه برای حذف نویز *salt and pepper* خیلی خوب عمل میکند، ابتدا یک *median filter* به تصویر اعمال کرد تا نویز *salt and pepper* را از بین ببرد و در ادامه *gaussian filter* یا *box filter* برای حذف نویز گوسی اعمال کرد.

شکل ۱۶) تصویر Elaine با اعمال شدن دو نویز s&p با چگالی $\rho = 0.15$ *و gaussian با انحراف معیار* $\sigma = 0.05$

شکل ۱۷) اعمال median filter با سایز ۷ بر روی تصویر نویز دار شکل ۱۶ برای از بین بردن نویز s&p و در ادامه اعمال فیلتر gaussian با سایز ۷ و انحراف معیار ۱ برای از بین بردن نویز گوسی.

انتخاب سایز فیلتر ها بستگی چگالی و انحراف معیار های نویز های اعمال شده دارد. در اینجا با توجه به تجربه ای که از تمرین های قبل به دست آمده، متوجه میشویم فیلتری با سایز ۵ یا ۷ میتواند مناسب باشد که در اینجا فیلتر با سایز ۷ در نظر گرفته شده است.

میتوان *mse* ها را نیز مقایسه کرد تا به سایز هایی که *optimal* هستند برسیم.

۳.۳ *Sharpening, Blurring and Noise Removal*

۳.۳.۱

در این تمرین خواسته شده است تا تصاویری نویزی و *blur* با استفاده از دوربین گرفته شود و کیفیت آنها را با استفاده از روش های معرفی شده، بهبود بخشید.

شکل ۱۸) تصویر دارای نویز گرفته شده با استفاده از دوربین تلفن همراه.

مشاهده میشود تصویر هم دارای میزان کمی نویز *salt and pepper* و مقدار بیشتری نویز *gaussian* میباشد. در نتیجه میتوان ابتدا از *median filter* برای از بین بردن نویز *salt and pepper* استفاده کرد و سپس فیلتر *box* یا گوسی اعمال کرد تا نویز گوسی را کاهش دهد.

استفاده از فیلتر *box* و یا گوسی، باعث نرم تر شدن لبه ها میشود و در نتیجه تصویر کمی *blur* میشود. برای حل این مشکل از فیلتر شارپ کننده به وسیله لاپلاسین استفاده شده است تا لبه ها را شارپ تر کند.

نتیجه را میتوان در تصویر۱۹ مشاهده کرد. ابتدا فیلتر میانه با اندازه ۷ و سپس *box filter* با اندازه ۵ و در انتها فیلتر شارپ کننده بر روی تصویر اعمال شده است.

شکل ۱۹) اعمال median filter با سایز ۷ و سپس فیلتر box با سایز ۵ و در نهایت convolve کردن فیلتر شارپ کننده با استفاده از لاپلاسین بر روی نتیجه دو فیلتر قبلی.

مشاهده میشود که تصویر هنوز دارای نویز گوسی است. برای حل این مشکل میتوان box filter بزرگ تری بر روی تصویر convolve کرد ولی باعث blur شدن تصویر میشود. باید trade off میان نویز و blur شدن تصاویر را در نظر گرفت. میتوان نتیجه اعمال box filter بزرگتر را در تصویر ۲۰ مشاهده کرد.

شکل ۲۰) اعمال median filter با سایز ۷ و سپس فیلتر box با سایز ۹ و در نهایت convolve کردن فیلتر شارپ کننده با استفاده از لاپلاسین بر روی نتیجه دو فیلتر قبلی. مشاهده میشود که نسبت به شکل ۱۹، تصویر blur تر شده است و کمی از جزئیات را از دست داده ایم.

فیلتر شارپ کننده با استفاده از لاپلاسین استفاده شده به شرح زیر است:

$$L = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

در قسمت بعدی بر روی تصویری blur شده کار خواهیم کرد. برای بهبود این تصویر از فیلتر شارپ کننده لاپلاسین که در قسمت قبل ارائه شد استفاده شده است. نتایج را میتوان در تصاویر زیر مشاهده کرد:

شکل ۲۱) تصویر blur گرفته شده با استفاده از دوربین.

شکل ۲۲) مشاهده میشود که با convolve کردن فیلتر شارپ کننده با استفاده از لاپلاسین، لبه ها کمی شارپ تر شده اند.

۳.۴ *Edge Detection*

۳.۴.۱

در این تمرین خواسته شده است تا ۳ فیلتر لبه یاب را بر تصویر اعمال کرده و تفاوت های آنها را بررسی کنیم.
ابتدا گرادیان در تصاویر را معرفی میکنیم:

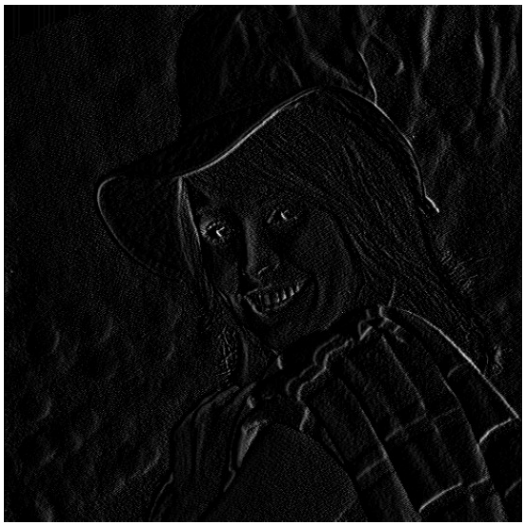
$$\nabla f \equiv grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$M(x,y) = ||\nabla f|| = \sqrt{g_x^2 + g_y^2} \approx |g_x| + |g_y|$$

حال میتوان برای این عبارت فیلتر هایی را تخمین زد که این عملیات را انجام دهند که فیلتر های ارائه شده در صورت سوال همه شامل آن می‌شوند. فیلتر های ارائه شده در سوال، برای شناسایی لبه های عمودی هستند. برای شناسایی لبه های افقی میتوان از ترانهاده این فیلتر ها استفاده کرد.
حال به بررسی جزئیات هر یک از فیلتر ها و نتایج آنها می‌پردازیم.

$$a)\frac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

نتیجه اعمال این فیلتر در شکل۲۳ نمایش داده شده است.



شکل ۲۳) اعمال فیلتر a بر روی تصویر Elaine.

مشاهده می‌شود لبه های عمودی کوچک شناسایی شده اند.

$$b)\frac{1}{6}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

نتیجه اعمال این فیلتر در شکل۲۴ نمایش داده شده است.

شکل ۲۴) اعمال فیلتر b بر روی عکس Elaine.

$$c)\frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

نتیجه اعمال این فیلتر در شکل۲۴ نمایش داده شده است.

شکل ۲۵) اعمال فیلتر سوبل (c) بر روی تصویر Elaine.

تفاوتی که میتوان در فیلتر ها مشاهده کرد، این است که فیلتر a در مقایسه با فیلتر b و c لبه های کوچک تری را

تشخیص داده است که باعث کاهش دقت در تشخیص لبه شده است. (نویز های داخل تصویر را هم به عنوان لبه در نظر گرفته است) در شکل های ۲۶ و ۲۷ این تفاوت قابل مشاهده است. شکل ۲۶ مربوط به فیلتر a است که مشاهده میشود نقاط سفیدی همانند نویز دارد که لبه های تشخیص داده شده است.

شکل ۲۶) ناحیه ای از تصویر بعد از اعمال فیلتر a.

در شکل ۲۷ مشاهده میشود که لبه های بزرگتری در نظر گرفته شده است که باعث کاهش حساسیت به نویز است.

شکل ۲۷) ناحیه ای از تصویر بعد از اعمال فیلتر b.

نتیجه مربوط به فیلتر b و c خیلی با یکدیگر تفاوتی ندارند تنها تفاوتی که میتوان به آن اشاره کرد، ضریب ۲ برای پیکسل وسط در ستون اول و آخر میباشد که برای تاکید بیشتر بر روی پیکسل وسط میباشد که برای *smoothing* استفاده میشود.

۳.۴.۲

در این قسمت خواسته شده است تا فیلتر های روبرت را بر تصویر اعمال کنیم و نتیجه را با فیلتر های ارائه شده در قسمت قبل مقایسه کنیم.

$$a)\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad b)\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}$$

با توجه به به فیلتر ها میتوان گفت که با *convolve* کردن آنها بر روی تصویر، اختلاف میان پیکسل های اریب بررسی میشود. میتوان گفت این فیلتر ها لبه های اریب را شناسایی میکنند. فیلتر a لبه های با زاویه ۴۵ درجه از مبدا و فیلتر b لبه های با زاویه ۱۳۵ درجه از مبدا را بررسی و شناسایی میکند.

همانند قسمت های گذشته، فیلتر های ارائه شده را بر روی تصویر *convolve* می‌کنیم. نتایج به دست آمده به صورت زیر میباشد:

شکل ۲۸) اعمال فیلتر روبرت a بر روی تصویر Elaine.

شکل ۲۹) اعمال فیلتر روبرت b بر روی تصویر Elaine.

با بررسی و مقایسه فیلتر های مربوط به قسمت ۳.۴.۱ با فیلتر های روبرت، میتوان مشاهده کرد که فیلتر های روبرت لبه هایی را شناسایی کرده اند که فیلتر های قبلی شناسایی نکردند. (برای مثال لبه مربوط به بالای کلا در فیلتر روبرت *b* شناسایی شده است که در فیلتر سوبل و فیلتر های قبلی شناسایی نشده بود) ولی در قسمت لباس فیلتر های روبرت فقط لبه های مروبط به زاویه خودشان را شناسایی کرده اند. با جمع کردن نتایج دو فیلتر روبرت میتوان به نتیجه ای *general* تر رسید.

شکل ۳۰) مجموع نتایج دو فیلتر روبرت قبلی. میتوان مشاهده کرد که تقریبا تمامی لبه ها شناسایی شده اند و شاهد نتیجه نسبتا خوبی را هستیم.

۳.۵ *Unsharp Masking*

۳.۵.۱

در این تمرین خواسته شده است تا *unsharp masking* را پیاده سازی کنیم. برای این کار در کتاب ۳ مرحله گفته شده است.

۱. *blur* کردن تصویر اصلی
۲. کم کردن تصویر *blur* شده از تصویر اصلی (به حاصل این کار *mask* گفته میشود)
۳. اضافه کردن *mask* به تصویر اصلی

این پروسه همان فرمولی است که در صورت سوال داده شده است.

$$(1-\alpha)I + \alpha I' = I + \alpha(I' - I)$$

همانطور که مشاهده میشود، اگر α بسیار کوچک باشد و به ۰ میل کند، نتیجه فیلتر برابر با خود تصویر میشود زیرا عبارت $\alpha(I' - I)$ نیز به ۰ میل میکند و حاصل عبارت برابر با I میشود. و اگر α به بیشترین مقدار خود یعنی ۱ نزدیک شود، حاصل عبارت به I' نزدیک میشود. میتوان گفت *unsharp masking* با $0 \leq \alpha \leq 1$ تصویری بین I و I' را نتیجه میدهد.

در ادامه به انجام محاسبات و محاسبه اختلاف میان تصویر اصلی و تصویر *blur* شده با فیلتر گوسی می‌پردازیم.

شکل ۳۱) محاسبه اختلاف عکس اصلی و عکس blur شده با فیلتر گوسی با سایز های ۳، ۵، ۷، ۹ و ۱۱. (به ترتیب از بالا سمت چپ)

همانطور که مشاهده میشود، هرچقدر سایز فیلتر گوسی بزرگتر باشد، در تفاضل آن با عکس اصلی لبه های بزرگتری مشاهده میشود.

در گام بعدی عکس اصلی را با ضریبی از نتایج به دست آمده جمع میکنیم که در ادامه قابل مشاهده است.

شکل ۳۲) مجموع عکس اصلی و اختلاف محاسبه شده و نمایش داده شده در شکل ۳۱ با استفاده از $\alpha = 0.9$.

شکل ۳۳) مجموع عکس اصلی و اختلاف محاسبه شده و نمایش داده شده در شکل ۳۱ با استفاده از $\alpha = 0.5$.

شکل ۳۴) مجموع عکس اصلی و اختلاف محاسبه شده و نمایش داده شده در شکل ۳۱ با استفاده از $\alpha = 0.1$.

میتوان مشاهده کرد که با افزایش α تصویر به دست آمده به تصویر *blur* شده نزدیک تر میشود.

همچنین میتوان مشاهده کرد که نتیجه تصاویری که فیلتر گوسی بزرگتری به آنها اعمال شده است، لبه های شارپ تری نسبت به آنهایی که فیلتر گوسی کوچکتری اعمال شده است دارند.

برای پیدا کردن مقدار α باید *trade off* در نظر گرفت که چه چیزی برایمان مهم تر است. همان طور که مشاهده میشود تصویر با $\alpha = 0.9$ لبه های شارپ تری دارد ولی نویز ها هم شارپ تر شده اند. از طرف دیگر تصویر با $\alpha = 0.1$ لبه های خیلی شارپی ندارد ولی نویز نیز در آن کمتر است. و تصویر با $\alpha = 0.5$ میان دو آلفای قبلی می‌باشد.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math
from skimage.util import random_noise


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')


def add_noise(img, mode, val=0.05):
    if (mode == 's&p'):
        r = random_noise(img, mode, amount=val)
    elif (mode == 'gaussian'):
        r = random_noise(img, mode, var=val)

    return (r * 255).astype('uint8')

def mse_gray(img1, img2):
    diff = (img1 - img2) ** 2
    return np.sum(diff.ravel()) / (img1.shape[0] * img1.shape[1])

def box(size):
    return np.ones((size, size)) * (1/(size*size))

def create_gaussian_filter(kernel_size, sigma=1):
    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()

def convolve_gaussian_filter(raveled_img, kernel):
    filter = kernel.ravel()
    return np.sum(filter * raveled_img)

def filter(img, filter_size, mode='median', sigma=1):
    pad_size = int(math.floor(filter_size/2))
    x = np.pad(img, pad_size, 'mean')
    r = np.zeros((img.shape[0], img.shape[1]))

    is_g = False

```

```
    if (mode == 'median'):
        f = np.median
    elif (mode == 'box'):
        f = np.mean
    elif (mode == 'gaussian'):
        f = convolve_gaussian_filter
        is_g = True

    for i in range(pad_size, x.shape[0]-pad_size):
        for j in range(pad_size, x.shape[1]-pad_size):
            if is_g:
                g_filter = create_gaussian_filter(filter_size, sigma=sigma)
                r[i-pad_size, j-pad_size] = f(x[i-pad_size:i+pad_size+1, j-pad_size:j+pad_size+1].ravel(), g_filter)
            else:
                r[i-pad_size, j-pad_size] = f(x[i-pad_size:i+pad_size+1, j-pad_size:j+pad_size+1].ravel())


    return r.astype('uint8')


img = cv2.imread('Elaine.bmp')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

laplacian_filter = np.array([[-1, -1, -1,], [-1, 9, -1], [-1, -1, -1]])
sharpenning_filter = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

spnoise = []
for i in [0.05, 0.1, 0.2, 0.5]:
    spnoise.append(add_noise(img, 's&p', val=i))
show_img(spnoise , figsize=30)

out = add_noise(img, 's&p', val=0.1)
show_img(out, figsize=6)

out2 = add_noise(img, 'gaussian', val=0.01)
show_img(out2, figsize=6)

"""--------3.1--------"""

# calculate mse between image and box filter on noisy image versions
v = []
for i in [3, 5, 7, 9, 11]:
    c = cv2.filter2D(src=out2, ddepth=-1, kernel=box(i))
    v.append(mse_gray(c, img))

print(v)

a = np.array(img)
for i in range(15):
    a = cv2.filter2D(src=a, ddepth=-1, kernel=box(3))
show_img(a)

a = cv2.filter2D(src=a, ddepth=-1, kernel=sharpenning_filter)
show_img(a)


filter_sizes = [3, 5, 7, 9, 11]
```

```
ro = [0.05, 0.1, 0.2, 0.5]
result = []
for p in ro:
    t = []
    noisy_img = add_noise(img, 's&p', val=p)
    for f in filter_sizes:
        t.append(filter(noisy_img, f))

    result.append(t)

show_img(result[0][0], result[3][2], figsize=30)

for i in range(len(result)):
    for j in range(len(result[0])):
        print(f'%.2f\t' % mse_gray(result[i][j], img), end=' ')
    print()

"""--------3.2--------"""


filter_sizes = [3, 5, 7, 9, 11]
sigma = [0.01, 0.05, 0.1]

median_result = []
box_result = []

for s in sigma:
    t = []
    u = []
    noisy_img = add_noise(img, 'gaussian', val=s)
    for f in filter_sizes:
        t.append(filter(noisy_img, f, mode='median'))
        u.append(filter(noisy_img, f, mode='box'))

    median_result.append(t)
    box_result.append(u)

print('median filter result on image with gaussian noise')
for i in range(len(median_result)):
    for j in range(len(median_result[0])):
        print(f'%.2f\t' % mse_gray(median_result[i][j], img), end=' ')
    print()

print('\nbox filter result on image with gaussian noise')
for i in range(len(box_result)):
    for j in range(len(box_result[0])):
        print(f'%.2f\t' % mse_gray(box_result[i][j], img), end=' ')
    print()

show_img(median_result[2][3], box_result[2][3], figsize=30)

noisy_img = add_noise(img, 's&p', 0.15)
noisy_img = add_noise(noisy_img, 'gaussian', 0.05)
show_img(noisy_img)

a = filter(noisy_img, 7, 'median')
b = cv2.filter2D(src=a, ddepth=-1, kernel=create_gaussian_filter(7))

show_img(b)
```

```
174
175 blur = cv2.blur(img, ksize=(7,7))
176 show_img(blur)
177
178 """--------3.3--------"""
179
180
181 noisy_img = cv2.imread('noise.jpeg')
182 noisy_img = cv2.cvtColor(noisy_img, cv2.COLOR_BGR2GRAY)
183
184 show_img(noisy_img)
185
186 a = filter(noisy_img, 7)
187
188 b = cv2.filter2D(src=a, ddepth=-1, kernel=box(5))
189
190 c = cv2.filter2D(src=b, ddepth=-1, kernel=laplacian_filter)
191
192 show_img(c)
193
194
195 blur = cv2.imread('blur.jpeg')
196 blur = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
197
198 show_img(blur)
199
200 a = filter(blur, 7)
201
202 b = cv2.filter2D(src=blur, ddepth=-1, kernel=box(3))
203
204 c = cv2.filter2D(src=b, ddepth=-1, kernel=laplacian_filter)
205
206 show_img(c)
207
208 """--------3.4--------"""
209
210
211 edge_detection_filter_1 = 1/2 * np.array([[1, 0, -1]])
212
213 a = cv2.filter2D(src=img, ddepth=-1, kernel=edge_detection_filter_1)
214
215 show_img(a)
216
217 edge_detection_filter_2 = 1/6 * np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
218
219 b = cv2.filter2D(src=img, ddepth=-1, kernel=edge_detection_filter_2)
220
221 show_img(b)
222
223 edge_detection_filter_3 = 1/8 * np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
224
225 c = cv2.filter2D(src=img, ddepth=-1, kernel=edge_detection_filter_3)
226
227 show_img(c)
228
229 robert1 = np.array([[1, 0], [0, -1]])
230
231 r1 = cv2.filter2D(src=img, ddepth=-1, kernel=robert1)
232
```

```
show_img(r1)

robert2 = np.array([[0, 1], [-1, 0]])

r2 = cv2.filter2D(src=img, ddepth=-1, kernel=robert2)

show_img(r2)

show_img(r1 + r2)

"""--------3.5--------"""

differences = []
for i in [3, 5, 7, 9, 11]:
    blur_image = cv2.GaussianBlur(img, (i, i), 10)
    a = blur_image - img
    differences.append(a)

show_img(differences[3], differences[4], figsize=30)

alpha = 0.5
r = []
for i in differences:
    b = img + alpha * i
    r.append(b)

show_img(r[0], r[1], r[2], figsize=30)

show_img(r[3], r[4], figsize=30)
```