باسمه تعالی

پردازش تصاویر پزشکی

بهار ۰۲-۱۴۰۱

تمرین سری سوم

استاد: دکتر فاطمی‌زاده

علی نوریان ۹۸۱۰۲۵۲۷

۱ سوالات تئوری

۱.۱ سوال اول

با توجه به عبارت داده شده داریم:

$$\texttt{if } x_i = 0 \Longrightarrow \lim_{\beta \to \infty} 1 - e^{-\beta x_i^2} = 1 - e^0 = 1 - 1 = 0$$

$$\texttt{if } x_i \neq 0 \Longrightarrow \lim_{\beta \to \infty} 1 - e^{-\beta x_i^2} = 1 - e^{-\infty} = 1 - 0 = 1$$

بنابراین برای x_iهای برابر صفر حاصل عبارت $1 - e^{-\beta x_i^2}$ صفر می‌شود و برای x_iهای ناصفر حاصل آن ۱ می‌شود. در نتیجه عبارت $1 - e^{-\beta x_i^2}$ زمانی که $\sum_{i=1}^{k}(1 - e^{-\beta x_i^2})$ برابر نرم صفر بردار $\boldsymbol{x}$ است.

۲.۱ سوال دوم

ابتدا به بررسی سه روش حذف نویز از روش‌های مبتی در یادگیری عمیق می‌پردازیم:

۱. شبکه‌های عصبی کانولوشنی (CNNs):

شبکه‌های عصبی کانولوشنی (CNNs) برای وظایف مختلف بینایی ماشین، از جمله حذف نویز تصاویر، به طور گسترده‌ای استفاده می‌شوند. ایده اصلی در این رویکرد، آموزش یک شبکه عصبی عمیق به منظور یادگیری تطبیق میان تصاویر نویزی و نسخه‌های پاک آن‌ها است. معماری شبکه معمولاً شامل چندین لایه کانولوشنی است که ویژگی‌های محلی تصویر را به دست می‌آورند و به آن‌ها تابع فعال‌سازی غیرخطی مانند ReLU اعمال می‌کنند. این شبکه ممکن است شامل لایه‌های پولینگ برای کاهش نمونه‌برداری و لایه‌های افزایش اندازه برای بزرگنمایی نمونه‌ها نیز باشد. لایه نهایی شبکه معمولاً عمل جستجوی مقدار پیکسل به پیکسل را برای تولید تصویر پاک‌سازی شده انجام می‌دهد.

برای آموزش یک CNN برای حذف نویز، لازم است یک مجموعه داده از تصاویر نویزی و پاک استفاده شود. شبکه آموزش داده می‌شود تا تفاوت بین تصاویر پاک پیش‌بینی شده و تصاویر پاک واقعی را کمینه کند. تابع خطای معمولاً استفاده شده در وظایف حذف نویز، خطای میانگین مربعات (MSE) یا خطای دریافتی است که اختلاف دریافتی بین تصاویر پاک‌سازی شده و تصاویر پاک را اندازه‌گیری می‌کند. شبکه با استفاده از الگوریتم‌های پس‌انتشار خطا backpropaation و گرادیان کاهشی gradient descent بهینه می‌شود تا وزن‌ها و بایاس‌ها به‌روز شده و به شبکه اجازه دهد که به طور موثر تصاویر را پاک‌سازی کند.

۲. اتوانکودرها (Autoencoders):

اتوانکودرها (Autoencoders) معماری شبکه عصبی‌ای هستند که شامل یک کدگذار و یک کدگشا می‌شوند. کدگذار تصویر ورودی را به یک نمایش کم‌بعد (فضای کدگذاری یا لاتنت (latent)) فشرده می‌کند و کدگشا تصویر را از این نمایش بازسازی می‌کند. در حوزه حذف نویز، یک اتوانکودر آموزش داده می‌شود تا تصاویر نویزی را به فضای لاتنت (latent) تبدیل کند و سپس آن‌ها را به تصاویر پاک بازسازی کند.

در طول آموزش، از مجموعه داده‌ای شامل تصاویر جفت نویزی و پاک استفاده می‌شود. اتوانکودر آموزش داده می‌شود تا خطای بازسازی بین تصاویر نویزی و متناظر پاک خود را به حداقل برساند. این باعث می‌شود اتوانکودر نمایشی قوی از تصاویر پاک را یاد بگیرد که به طور موثر از نویز جدا می‌شود. تابع خطای استفاده شده برای آموزش می‌تواند خطای میانگین مربعات (MSE) یا سایر توابع خطای دریافتی باشد.

پس از آموزش، اتوانکودر می‌تواند برای حذف نویز تصاویر نویزی جدید استفاده شود. تصویر نویزی از طریق کدگذار به فضای لاتنت منتقل شده و سپس توسط کدگشا بازسازی می‌شود.

۳. شبکه‌های مولد تخاصمی (GANs):

شبکه‌های مولد تخاصمی (GANs) شامل یک مولد generator و یک تمییزدهنده discriminator هستند. مولد با توجه به تصاویر نویزی ورودی به تولید تصاویر پاک‌شده، می‌پردازد، در حالی که تمییزدهنده سعی در تمییز بین تصاویر پاک واقعی و تصاویر تولید شده (پاک شده) دارد.

معمولاً مولد، یک مدل مبتنی بر CNN است که تصویر نویزی را به عنوان ورودی دریافت کرده و یک تصویر پاک شده را تولید می‌کند. تمییزدهنده نیز یک CNN است که هم تصاویر پاک واقعی و هم تصاویر پاک شده تولید شده را به عنوان ورودی دریافت کرده و سعی می‌کند بین آن‌ها تمایز بیندازد. مولد و تمییزدهنده به طور همزمان با هم آموزش داده می‌شوند. مولد سعی می‌کند تصاویری را تولید کند که تمییزدهنده نتواند آن‌ها را از تصاویر پاک واقعی تشخیص دهد و تمییزدهنده سعی می‌کند عملکرد تمایزدهی خود را بهبود بخشد.

فرآیند آموزش شامل به‌روزرسانی مکرر وزن‌های مولد و تمییزدهنده است. وزن‌های مولد به‌روز می‌شوند تا توانایی تمییزدهنده در تمییز بین تصاویر واقعی و تصاویر تولید شده را به حداقل برساند، در حالی که وزن‌های تمییزدهنده به‌روز می‌شوند تا دقت تمییزدهی خود را بهبود بخشد. این آموزش مقابله‌ای باعث می‌شود مولد تصاویر پاک شده با کیفیت بالا تولید کند که نزدیک به تصاویر پاک واقعی باشد. پس از آموزش، مولد می‌تواند برای حذف نویز تصاویر نویزی جدید استفاده شود.

حال به بررسی دو روش DnCNN و FFDNet می‌پردازیم:

DnCNN (Denoising Convolutional Neural Network):

DnCNN یک مدل یادگیری عمیق است که به طور خاص برای وظیفه حذف نویز تصاویر طراحی شده است. به دلیل عملکرد برترش، بسیاری از کاربردهای حذف نویز را دربر می‌گیرد. ایده اصلی پشت DnCNN، آموزش یک شبکه عصبی عمیق به منظور یادگیری نگاشتی است بین تصاویر نویزی و نسخه‌های پاک آن‌ها. با یادگیری این نگاشت، DnCNN به خوبی نویز را از تصاویر حذف می‌کند.

معماری DnCNN معمولاً شامل چندین لایه کانولوشنی است. این لایه‌ها مسئول استخراج ویژگی‌های محلی تصویر هستند و از توابع فعال‌سازی غیرخطی مانند ReLU برای بهبود عملکرد حذف نویز استفاده می‌کنند. شبکه بر روی مجموعه داده‌ای از جفت تصاویر نویزی و پاک آموزش می‌بیند و هدف آن کمینه کردن تفاوت بین تصاویر پاک پیش‌بینی شده و تصاویر پاک واقعی است. DnCNN عملکرد حذف نویز قابل تحسینی داشته و می‌تواند با سطوح و انواع مختلف نویز مقابله کند.

FFDNet (Fast and Flexible Denoising Network):

FFDNet یک مدل پرطرفدار دیگر برای حذف نویز است که بر روی حالت‌های حذف نویز به طور لحظه‌ای و قابلیت انعطاف‌پذیری تمرکز دارد. این مدل برای مقابله با سطوح متنوع نویز و سازگاری با خصوصیات مختلف نویز طراحی شده است. FFDNet از یک معماری پی در پی تشکیل شده است که شامل مرحله تخمین نویز و مرحله حذف نویز است.

در مرحله تخمین نویز FFDNet، مدل سطح نویز موجود در تصویر نویزی ورودی را تخمین می‌زند. این تخمین برای تعیین استراتژی حذف نویز بهینه حائز اهمیت است. در مرحله حذف نویز FFDNet با استفاده از سطح نویز تخمین‌زده شده، حذف نویز موثر انجام می‌دهد. این مدل از ویژگی‌های فیلترینگ غیرمحلی (non-local means) استفاده می‌کند و از فیلترهای مشترک برای دستیابی به عملکرد حذف نویز سریع استفاده می‌کند.

تفاوت اصلی بین DnCNN و FFDNet در طراحی معماری و استراتژی‌های حذف نویز آن‌ها قرار دارد. DnCNN سعی در پیدا کردن نگاشت بین تصاویر نویزی و پاک با استفاده از شبکه‌های عمیق کانولوشنی دارد. این مدل بر روی ویژگی‌های محلی تصویر عمل می‌کند و از توابع فعال‌سازی غیرخطی برای بهبود عملکرد حذف نویز استفاده می‌کند. از سوی دیگر، FFDNet از یک معماری پی در پی با مرحله تخمین نویز و مرحله حذف نویز استفاده می‌کند. استراتژی حذف نویز FFDNet بر اساس سطح نویز تخمین‌زده شده تنظیم می‌شود و از فیلترینگ غیرمحلی (non-local means) با فیلترهای مشترک استفاده می‌کند.

هر دو مدل قدرتمندی هستند و تصاویر پاک با کیفیت بالا تولید می‌کنند. DnCNN به عنوان یکی از بهترین مدل‌ها در حذف نویز شناخته شده است و با سطوح و انواع مختلف نویز مقابله می‌کند. FFDNet نیز به دلیل انعطاف‌پذیری و سرعت بالایش در حالت‌های حذف نویز لحظه‌ای مورد توجه قرار می‌گیرد.

۲ سوالات عملی

۱.۲ سوال اول Denoising with Total variation

کد ارائه شده در توابع TV_Chambolle و TV_GPCL دو الگوریتم برای حذف نویز Total Variation (TV) را پیاده‌سازی می‌کند. در ادامه یک شرح مرحله به مرحله از الگوریتم‌ها آمده است:

دو تابع داده شده چندین پارامتر ورودی را دریافت می‌کند: متغیرهای دوگان اولیه w1 و w2، تصویر نویزی f، ثابت پایداری lbd، طول قدم ثابت alpha، حداکثر تعداد تکرار NIT، تلرانس همگرایی GapTol و verbose برای نمایش نتایج میانی.

کد، متغیرها را مقداردهی اولیه می‌کند و گرادیان‌های لازم را براساس داده‌های ورودی محاسبه می‌کند. انرژی اولیه سیستم را براساس متغیرهای دوگان w1 و w2، بدست می‌آید. سپس این تابع متغیر اصلی u را براساس انرژی محاسبه شده و سایر متغیرها بدست می‌آورد. پس از آن کد وارد حلقه‌ای می‌شود که تکرارهای اصلی الگوریتم را انجام می‌دهد.

درون حلقه متغیرهای دوگان w1 و w2 را به‌روز می‌کند. متغیرهای به‌روزشده w1 و w2 نرمالیزه می‌شوند تا شرط $|w| \leq 1$ برقرار شود. سپس انرژی سیستم و همچنین متغیر اصلی u براساس متغیرهای دوگان به‌روزشده محاسبه می‌شود.

برای همگرایی مقدار dual gap نسبت به مقدار تلرانس مشخص شده، مقایسه می‌شود. اگر معیار همگرایی برآورده شود، کد حلقه را متوقف کرده و پایان می‌یابد. در غیر این صورت، به تکرار بعدی ادامه می‌دهد.

پس از پایان حلقه، کد مقادیر نهایی متغیر اصلی u، متغیرهای دوگان w1 و w2، انرژی، شکاف دوگانی، هزینه زمانی و تعداد کل تکرارها را برمی‌گرداند.

بطور کلی، الگوریتم TV_Chambolle با به‌روزرسانی متغیرهای دوگان براساس گرادیان تابع هدف، نرمال‌سازی متغیرها و محاسبه متغیر اصلی و شکاف دوگانی پیش می‌رود. این مراحل را تا زمانی که معیارهای همگرایی برآورده شود یا تعداد حداکثر تکرارها به پایان برسد، تکرار می‌کند. نتیجه‌ای که به‌وسیله حذف نویز TV به‌دست می‌آید، یک تصویر تمیز شده u است. همچنین کد ارائه شده در تابع TV_GPCL روش gradient projection را با طول گام ثابت برای حل فرمول‌بندی دوگان مدل بازیابی تصویر پیاده‌سازی می‌کند. تفاوت اصلی بین دو تابع TV_Chambolle و TV_GPCL در روش بهینه‌سازی مورد استفاده برای حل فرمول دوگانه مدل بازیابی تصویر Total Variation (TV) نهفته است.

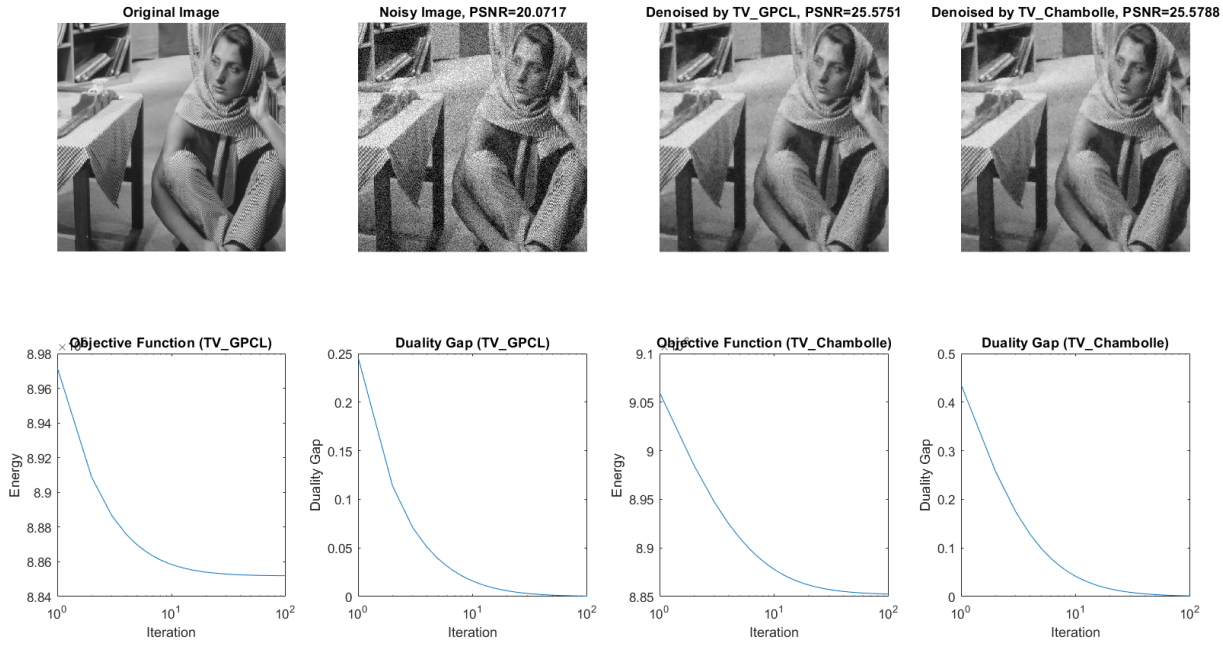
TV_Chambolle: این تابع روش Chambolle را که توسط Chambolle در سال ۲۰۰۴ پیشنهاد شد، پیاده سازی می کند. از طول گام ثابت برای کاهش گرادیان استفاده می کند. این الگوریتم متغیرهای دوگانه w1 و w2 را به طور مکرر با استفاده از روش کاهش گرادیان نیمه ضمنی Chambolle به روز می کند. فرایند گام‌به‌گام شامل محاسبات گرادیان، به روز رسانی متغیرهای دوگانه و محاسبه انرژی و شکاف دوگانه است. همگرایی با بررسی شکاف دوگانگی نسبی در برابر یک تلورانس مشخص تعیین می‌شود.

TV_GPCL: این تابع، روش gradient projection را با طول گام ثابت اعمال می‌کند. مشابه روش Chambolle، متغیرهای دوگانه (w1 و w2) را به طور مکرر به روز می‌شوند با این تفاوت که محاسبات گرادیان، به روز رسانی متغیرهای دوگانه با استفاده از gradient projection با طول گام ثابت مدل بازسازی ROF است. همگرایی اگوریتم نیز با مقایسه شکاف دوگانه با تلورانس مشخص شده تعیین می شود.

به طور خلاصه، هدف هر دو تابع حل یک فرمول دوگانه مدل بازیابی تصویر تلوی زیون است، اما آنها از روش‌های بهینه‌سازی متفاوتی استفاده می‌کنند. TV_Chambolle از نزول گرادیان نیمه ضمنی Chambolle استفاده می‌کند، در حالی که TV_GPCL

از روش اولیه gradient projection استفاده می‌کند. انتخاب بین دو روش ممکن است به عواملی مانند خواص همگرایی، کارایی محاسباتی و کابردهای خاص تسک بستگی داشته باشد.

پس از پیدا کردن پارامترهای مناسب توسط Cross-Validation خروچی دو تابع داده شده به صورت زیر است:



شکل ۱: تصاویر رفع‌نویز شده با روش‌های Chambolle و GPCL

۲.۲ سوال دوم Sparse representation and Image denoising using dictionary learning

Orthogonal Matching Pursuit (OMP)

الگوریتم Orthogonal Matching Pursuit (OMP) یک الگوریتم برای حل مسئله فشرده‌سازی سیگنال و بازسازی سیگنال به کمک یک توالی خطی از اتم‌ها است. این الگوریتم برای استخراج یک توالی خطی از اتم‌هایی که بهترین تطابق را با سیگنال موردنظر فراهم می‌کنند، استفاده می‌شود.

هدف اصلی OMP ، انتخاب مجموعه کوچکی از اتم‌ها از یک فضای اتم‌ها به نحوی است که سیگنال موردنظر را به بهترین شکل تقریب بزند. اتم‌ها معمولاً مجموعه از توابع بر پایه‌ی پایه‌هایی مثل توابع پایه‌ی مثلثی یا سینوسی هستند. با توجه به مجموعه‌ی اتم‌های، OMP تلاش می‌کند تا مجموعه‌ای از اتم‌ها را انتخاب کند و وزن‌های مناسبی برای آن‌ها محاسبه کند تا سیگنال را به بهترین نحو تقریب دهد.

الگوریتم OMP به صورت مرحله به مرحله اجرا می‌شود. در هر مرحله، اتمی که بهترین تطابق را با سیگنال دارد را انتخاب کرده و به مجموعه‌ی اتم‌های انتخاب شده اضافه می‌کند. سپس وزن‌های مربوط به اتم‌های انتخاب شده را با استفاده از روشی مانند روش حل مسئله کمترین مربعات محاسبه می‌کند. سپس این وزن‌ها را برای انتخاب اتم‌های بعدی استفاده می‌کند. این روند تا زمانی ادامه پیدا می‌کند که یک شرط خاتمه برآورده شود، مانند دستیابی به تعداد مشخصی از اتم‌ها یا رسیدن به یک حداکثر خطای مجاز.

Least Angle Regression (LARS)

الگوریتم Least Angle Regression (LARS) یک الگوریتم برای انتخاب ویژگی و حل مسئله رگرسیون خطی است. LARS یک روش فشرده‌سازی مدل است که به صورت تدریجی و قدم به قدم، متغیرهای مهم را شناسایی می‌کند و تاثیر آن‌ها را در مدل بررسی می‌کند.

هدف اصلی LARS، یافتن یک مدل تطبیقی کمینه برای رگرسیون خطی است، در حالی که تعداد متغیرها بسیار بزرگ است. در مسئله رگرسیون خطی، هدف ما پیدا کردن یک تابع خطی است که بین ورودی‌ها و خروجی‌ها تطابق داشته باشد. اما وقتی تعداد متغیرها بیشتر از تعداد نمونه‌ها است، مسئله به شکلی فشرده‌شده و غیر معکوس قابل حل نمی‌باشد. در اینجا می‌توانیم از LARS استفاده کنیم.

روند اصلی الگوریتم LARS به این صورت است:

۱. شروع با یک مدل خالی، به عنوان مدل پایه.

۲. در هر مرحله، متغیری که بیشترین همبستگی را با متغیرهای هدف دارد را انتخاب می‌کنیم.

۳. در هر مرحله، مقداری به متغیرهای انتخاب شده اضافه می‌کنیم تا مدل با دقت بیشتری تطبیق یابد.

۴. متغیرهایی که با اضافه کردن آن‌ها به مدل، بیشترین بهره را به دست می‌آوریم را انتخاب می‌کنیم.

۵. روند را تا زمانی که مدل نهایی به تعداد دلخواهی از متغیرها رسیده یا شرایط متوقف شدن دیگری برآورده شود، ادامه می‌دهیم.

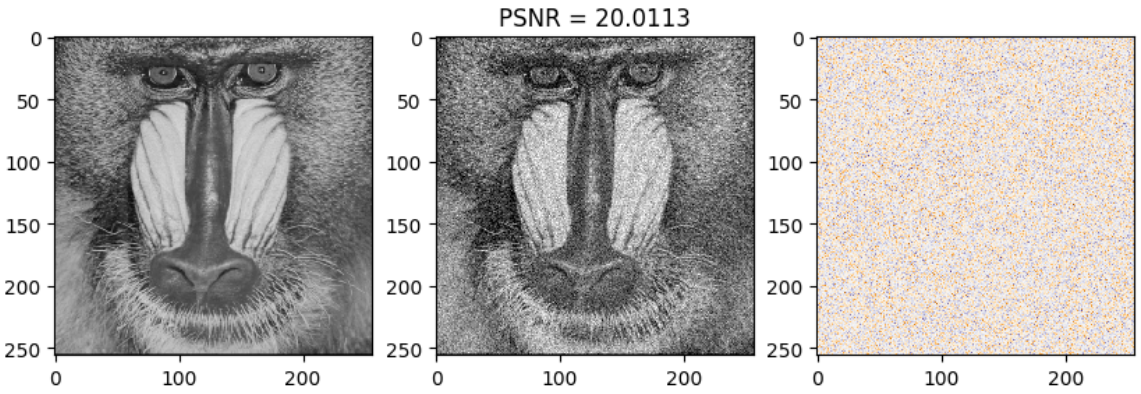
الگوریتم LARS به مرور زمان، متغیرهای مهم را به ترتیب شناسایی می‌کند و به آن‌ها وزن‌دهی می‌کند. با این حال، تفاوت اصلی بین LARS و الگوریتم‌های دیگر انتخاب ویژگی، این است که LARS به صورت تدریجی و با پیوستگی بهترین متغیرها را انتخاب می‌کند، به جای حذف یا اضافه کردن به صورت ناگهانی.

LARS می‌تواند در مسائلی که تعداد متغیرها زیاد است و رویکردهای سنتی ممکن است با مشکل مواجه شوند، مفید باشد.

اکنون به بررسی پیاره سازی و اعمال این دو الگوریتم برای رفع نویز می‌پردازیم.

ابتدا به صورت زیر تصویر را لود کرده و یک نویز گوسی به آن اضافه می‌کنیم:

```
image = load_image('mandrill.jpg')
width, height = image.shape
image = cv2.resize(image, (int(width / 2), int(height / 2)), interpolation = cv2.INTER_AREA)
width, height = image.shape

sigma = 0.1
noisy_image, noise = add_gaussian_noise(image, sigma)

show_with_diff(image, noisy_image, noise, True)
```



شکل ۲: تصویر اصلی (سمت چپ)، تصویر نویزی (وسط)، نویز گوسی اضافه شده (سمت راست)

با استفاده از کتابخانه sklearn از تصویر پچ‌های با اندازه 8×8 استخراج می‌کنیم.

```
patch_size = (8, 8)
patches_data = extract_patches(noisy_image, patch_size=patch_size)
```

سپس با تنظیم پارامترای دیکشنری، آن را روی پچهای بدست آمده آموزش میدهیم:

```
dico = MiniBatchDictionaryLearning(
   n_components=200,
   batch_size=200,
   alpha=1.0,
   max_iter=10,
   verbose=True
)
comps = dico.fit(patches_data).components_
```

خروجی ۱۰۰ کامپوننت اول به صورت زیر است:

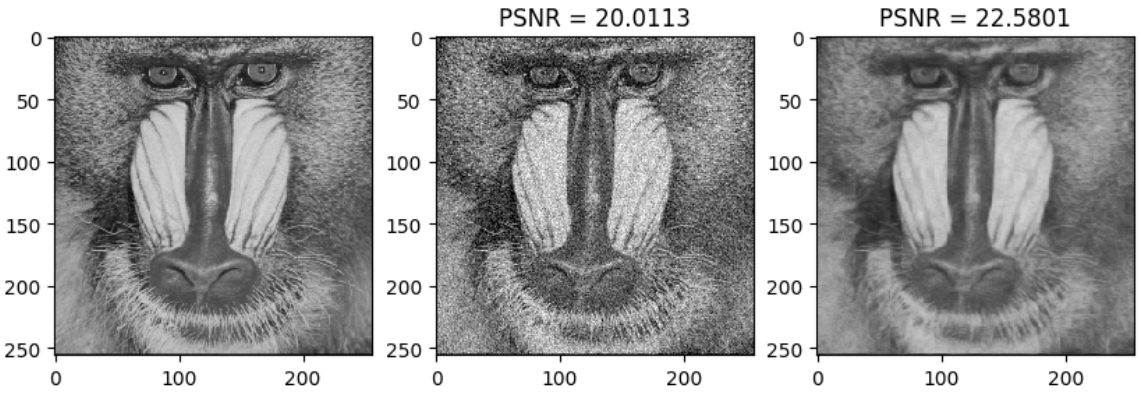
شکل ۳

اکنون با دو تبدیل Orthogonal Matching Pursuit و Least Angle Regression، در حالت دو تعداد اتم، تصویر را بازسازی می‌کنیم. در طی این فرایند مدت زمان صرف شده به ازای هر الگوریتم را نیز بدست می‌آوریم. همچنین از معیار PSNR برای بررسی کیفیت تصویر بازسازی شده استفاده می‌کنیم.

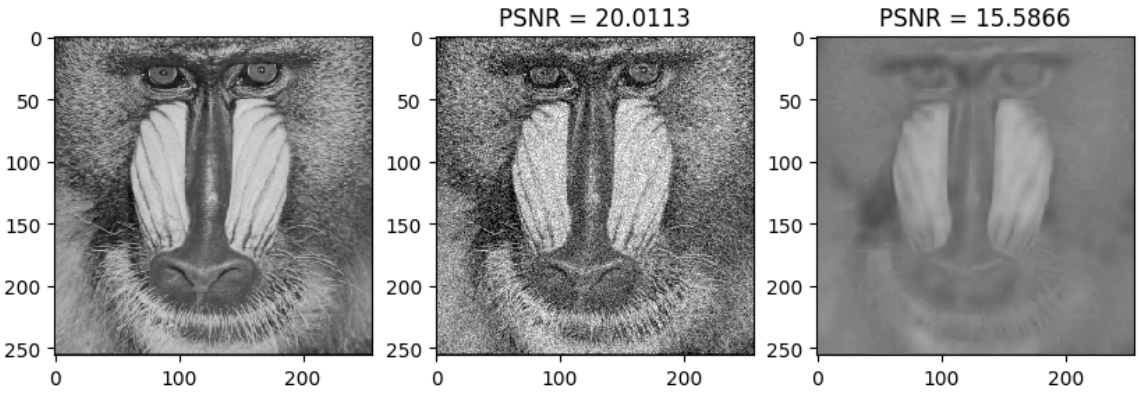
```
transform_algorithms = [
("Orthogonal Matching Pursuit\n2 atoms", "omp", {"transform_n_nonzero_coefs": 2}),
("Least-angle regression\n2 atoms", "lars", {"transform_n_nonzero_coefs": 2}),
]

data = extract_patches_2d(noisy_image, patch_size)
data = data.reshape(data.shape[0], -1)
intercept = np.mean(data, axis=0)
data -= intercept
```

```
reconstructions = {}
for title, transform_algorithm, kwargs in transform_algorithms:
   print(title + '...')
   t0 = time()
   reconstructions[title] = image.copy()

   dico.set_params(transform_algorithm=transform_algorithm, **kwargs)
   code = dico.transform(data)
   patches = np.dot(code, comps)
   patches += intercept
   patches = patches.reshape(len(data), *patch_size)

   reconstructions[title] = reconstruct_from_patches_2d(patches, (height, width))
   show_with_diff(image, noisy_image, reconstructions[title])
   print(f'time for this algorithm = {time() - t0} s')
plt.show()
```

خروجی تصویر بازسازی با دو روش ذکر شده در حالت دو تعداد اتم:



شکل ۴: تصویر اصلی (سمت چپ)، تصویر نویزی (وسط)، تصویر بازسازی شده با روش OMP (سمت راست)



شکل ۵: تصویر اصلی (سمت چپ)، تصویر نویزی (وسط)، تصویر بازسازی شده با روش LARS (سمت راست)

مدت زمان اجرای الگوریتم OMP برای تعداد اتم ۲ حدودا برابر با 8.5 ثانیه و مدت زمان اجرای الگوریتم LARS حدودا برابر 32 ثانیه بدست آمد. همچنین همانطور که در تصویر فوق مشهود است، تصویر در الگوریتم OMP با دو اتم، کمی رفع نویز شده است اما در الگوریتم LARS با ۲ اتم کیفیت تصویر بیشتر افت کرده است. این امر همانطور که در ابتدا توضیح داده شد مربوط به تفاوت روش این دو الگوریتم در این مسئله خاص می‌شود.

۳.۲ سوال سوم Denoising with Diffusion filter

(۱)

کد تابع به نام "anisodiff" برای رفع نویز تصویر استفاده می‌شود. در زیر توضیحی از عملکرد تک تک بخش‌های کد آمده است:

ابتدا، تصویر ورودی "im" را به دوبل تبدیل کرده و سایز آن را در متغیرهای "$rows$" و "$cols$" ذخیره می‌کند. همچنین، تصویر نهایی را در متغیر "$diff$" تعریف می‌کند.

سپس یک حلقه به تعداد "$niter$" اجرا می‌شود تا مراحل تکراری الگوریتم اجرا شود.

در هر مرحله، تصویر "$diff$" را به اندازه یک پیکسل در هر جهت با صفرهای پر کنده شده در "$diffl$" کپی می‌کند.

سپس، مشتق‌های شمالی ($deltaN$) ، جنوبی ($deltaS$) ، شرقی ($deltaE$) و غربی ($deltaW$) تصویر "$diff$" را محاسبه می‌کند.

در مرحله بعد، با توجه به مقدار "$option$" ، ضریب‌های هادی را برای هر جهت محاسبه می‌کند. اگر "$option$" برابر با ۱ باشد، از تابع توزیع نرمال برای محاسبه ضریب‌ها استفاده می‌کند. اگر "$option$" برابر با ۲ باشد، از تابع توزیع رکورد برای محاسبه ضریب‌ها استفاده می‌کند.

در نهایت، تصویر "$diff$" را با استفاده از ضریب‌ها و تفاوت‌ها به روزرسانی می‌کند. این عمل شامل جمع ضرب تفاوت‌ها در ضریب‌ها در هر جهت است.

برای استفاده از این تابع، شما باید تصویر ورودی "im" ، تعداد تکرار "$niter$" ، ضریب "$kappa$" ، پارامتر "$lambda$" و گزینه "$option$" را به عنوان ورودی به تابع ارسال کنید. خروجی این تابع، تصویر رفع نویز شده است.

کد تابع "isodiff" نیز برای رفع نویز تصویر استفاده می‌شود. در مقایسه با تابع "anisodiff" که توضیح داده شد، تفاوت‌هایی در روش محاسبه ضرایب هادی و اعمال تفاوت‌ها وجود دارد. در این تابع، مانند قبل، تصویر ورودی "im" به نوع داده عددی دوبل تبدیل می‌شود و تصویر پایه مقداردهی می‌شود. سپس، مانند تابع قبل، مشتق‌های شمالی، جنوبی، شرقی و غربی هر پیکسل با مقادیر پیکسل مجاور آن محاسبه می‌شود. در اینجا، برای محاسبه ضرایب هادی در هر جهت، از یک ثابت به نام "$constant$" استفاده می‌شود که به عنوان ورودی به تابع ارسال می‌شود. تمام ضرایب هادی در این الگوریتم به همان مقدار ثابت برابر هستند.

در نهایت، با استفاده از ضرایب هادی و تفاوت‌ها، تصویر پایه به روزرسانی می‌شود. این عمل شامل جمع وزن‌دار تفاوت‌ها در هر جهت با توجه به مقادیر ضرایب هادی است.

تفاوت اصلی بین تابع "isodiff" و "anisodiff" در روش محاسبه ضرایب هادی و مقداردهی به آن‌ها است. در "isodiff" ، تمام ضرایب هادی به یک مقدار ثابت تنظیم می‌شوند، در حالی که در "anisodiff" ، محاسبه ضرایب هادی بر اساس شدت تغییرات روشنایی در هر جهت صورت می‌گیرد. این باعث می‌شود "isodiff" تصویر را با یک نرخ فیلترینگ یکنواخت تغییر دهد، در حالی که "anisodiff" نرخ فیلترینگ را در جهات مختلف تنظیم کرده و تغییرات روشنایی را بر اساس شدت آن‌ها انجام می‌دهد.

(۲)

یکی از معیارهای معروف برای ارزیابی کیفیت تصویر، شاخص شباهت ساختاری (Structural Similarity Index یا به اختصار SSIM) است. SSIM یک معیار ریاضی است که برای اندازه‌گیری شباهت بین تصاویر مبتنی بر ساختارهای آن‌ها استفاده می‌شود. این شاخص به مقایسه سه عامل مختلف بین تصویر مرجع (تصویر اصلی) و تصویر تحلیل شده (تصویر پردازش شده) می‌پردازد. این سه عامل عبارتند از: شباهت ساختاری (luminance structure similarity) ، شباهت کانتراست (contrast similarity) و شباهت سیگنال سیاه و سفید (luminance similarity). این معیار از ۰ تا ۱ مقادیری بین محدوده مقادیر خود قرار می‌دهد که مقدار بالاتر به معنای شباهت بیشتر است و مقدار پایین‌تر به معنای شباهت کمتر یا تغییرات زیاد در تصویر است. فرمول محاسبه این معیار به صورت زیر است:

$$SSIM(x(i,j), y(i,j)) = \frac{(2\mu_{x(i,j)}\mu_{y(i,j)} + c_1)(2\sigma_{x(i,j)y(i,j)} + c_2)}{(\mu^2_{x(i,j)} + \mu^2_{y(i,j)} + c_1)(\sigma^2_{x(i,j)} + \sigma^2_{y(i,j)} + c_2)}$$

$\mu_{x(i,j)}$ and $\sigma^2_{x(i,j)}$ are the mean and variances of the window around pixel (i,j), and $sigma^2_{x(i,j)y(i,j)}$ is the covariance between windows around (i,j) in x and y.

یکی دیگر از معیارهای مورد استفاده برای ارزیابی کیفیت تصویر، Naturalness Image Quality Evaluator (NIQE) است. این معیار یک مدل محاسباتی است که برای اندازه‌گیری کیفیت طبیعیت تصویر استفاده می‌شود. NIQE با تحلیل ویژگی‌های مختلف تصویر مانند روشنایی، کانتراست، و جزئیات تصویر، به ارزیابی کیفیت طبیعیت تصویر می‌پردازد. مقادیر NIQE بین ۰ تا ۱ است که مقدار کمتر به معنای کیفیت بهتر و مقدار بیشتر به معنای کیفیت پایین‌تر است.

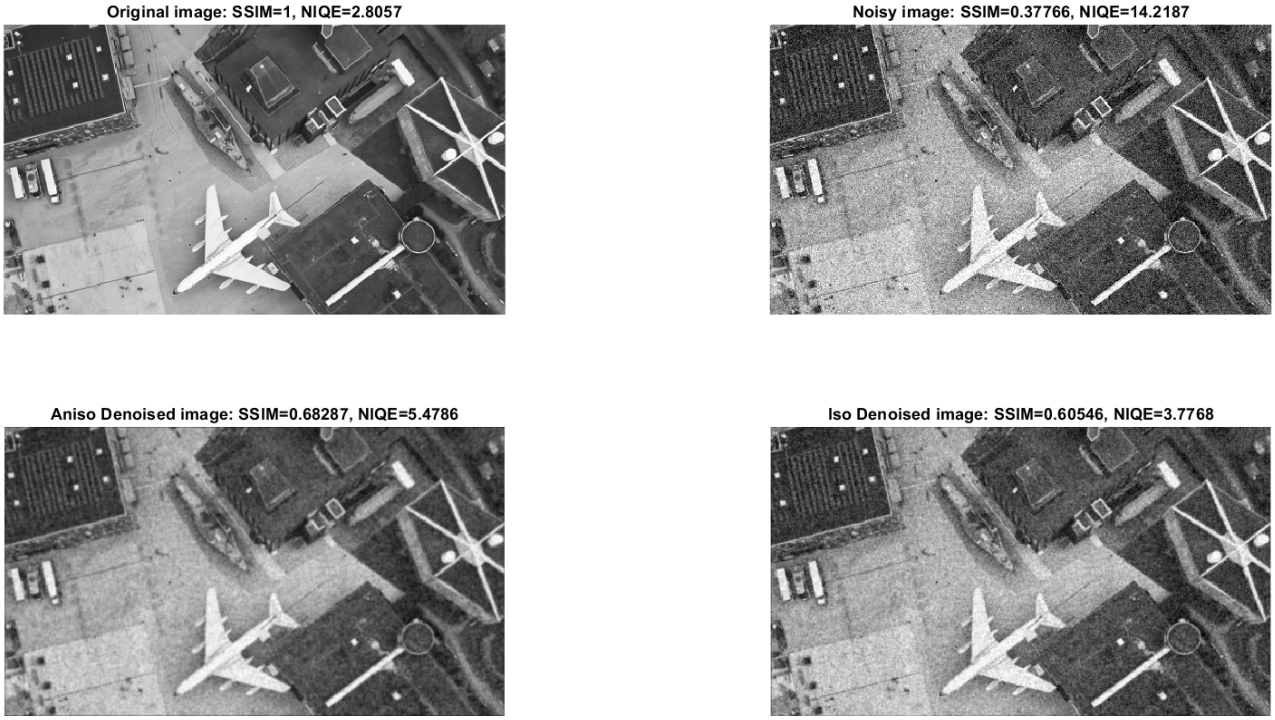
هر دو معیار SSIM و NIQE در صنعت و تحقیقات پردازش تصویر به عنوان ابزارهای مفیدی برای ارزیابی کیفیت تصاویر استفاده می‌شوند و به طور گسترده در بررسی الگوریتم‌ها و تکنیک‌های پردازش تصویر به کار می‌روند.

۳)

با استفاده از Cross-Validation مقادیر بهینه هایپرپارامترهای تابع را بدست می‌آوریم. ورودی و خروجی دو تابع داده شده به صورت زیر است:

```
% Denoise the image using anisotropic diffusion
niter = 2;        % Number of iterations
kappa = 25;       % Conduction coefficient
lambda = 0.18;     % Maximum value for stability
option = 1;       % Diffusion equation No 2
aniso_im_denoised = anisodiff(im_noisy, niter, kappa, lambda, option);
```

```
% Denoise the image using isotropic diffusion
lambda = 0.1;
constant = 1.7;
iso_im_denoised = isodiff(im_noisy, lambda, constant);
```



شکل ۶: تصویر اصلی (چپ بالا)، تصویر نویزی (راست بالا)، تصاویر رفع نویز شده با دو تابع سوال (دو تصویر پایین)

فیلترهای Anisotropic و Isotropic در پردازش تصویر برای کاهش نویز و بهبود جزئیات تصویر استفاده می‌شوند. در مورد فیلتر گوسی، این فیلتر معمولاً برای حذف نویز و تاری در تصاویر استفاده می‌شود.

فیلتر Gaussian، یک فیلتر خطی است که برای حذف نویز و نرم‌کردن تصاویر استفاده می‌شود. فیلتر گوسی عموماً بر اساس توزیع گوسی ساخته می‌شود و با استفاده از یک ماسک (kernel) دوبعدی یا سه‌بعدی به تصویر اعمال می‌شود. فیلتر گوسی اثرات نرم‌کنندگی و انتشار را بر روی تصویر ایجاد می‌کند و جزئیات ریز تصویر را کاهش می‌دهد. میزان اثرات نرم‌کنندگی فیلتر گوسی به وابستگی نسبت مستطیل نمونه (window size) و پارامتر انحراف معیار توزیع گوسی است.

در مقابل، فیلترهای Anisotropic (نامتقارن) توانایی حفظ لبه‌ها و جزئیات ریز در تصویر را دارند. این فیلترها برای حذف نویز و بهبود تصویر در مناطق با لبه‌های تند و جزئیات ریز مفید هستند. فیلترهای Anisotropic معمولاً بر اساس معیارهای متفاوتی مانند گرادیان تصویر، جهت لبه‌ها و نواحی همسایگی اعمال می‌شوند. این فیلترها به صورت هدفمند و به شکل محدودتری در نقاط مشخصی از تصویر به منظور حفظ جزئیات و جلوگیری از تغییرات غیرمطلوب در آن‌ها اعمال می‌شوند.

فیلترهای Isotropic (متقارن) نیز همانند فیلترهای Anisotropic می‌توانند برای حذف نویز و بهبود تصویر استفاده شوند، اما از روش‌های متقارن و یکنواخت برای اعمال تغییرات به تصویر استفاده می‌کنند. این فیلترها به ترتیبی همچون فیلتر گوسی عمل می‌کنند و نسبت به جهت و محل تغییرات تصویر حساسیت کمتری دارند.

۴.۲ سوال چهارم Denoising with anisotropic diffusion filtering

تابع Anisotropic را به صورت زیر تعریف می‌کنیم:

```
function diff = anisotropic(im, niter, kappa, lambda, option)

  im = double(im);
  [rows,cols] = size(im);
  diff = im;

  for i = 1:niter
     % Construct diffl which is the same as diff but has an extra padding of zeros around it.
    diffl = zeros(rows+2, cols+2);
    diffl(2:rows+1, 2:cols+1) = diff;

    % North, South, East and West differences
    deltaN = diffl(1:rows,2:cols+1) - diff;
    deltaS = diffl(3:rows+2,2:cols+1) - diff;
    deltaE = diffl(2:rows+1,3:cols+2) - diff;
    deltaW = diffl(2:rows+1,1:cols) - diff;

    % Conduction
    if option == 1
      cN = exp(-(deltaN/kappa).^2);
      cS = exp(-(deltaS/kappa).^2);
      cE = exp(-(deltaE/kappa).^2);
      cW = exp(-(deltaW/kappa).^2);
    elseif option == 2
      cN = 1./(1 + (deltaN/kappa).^2);
      cS = 1./(1 + (deltaS/kappa).^2);
      cE = 1./(1 + (deltaE/kappa).^2);
      cW = 1./(1 + (deltaW/kappa).^2);
    end

    diff = diff + lambda * (cN.*deltaN + cS.*deltaS + cE.*deltaE + cW.*deltaW);
  end
end
```

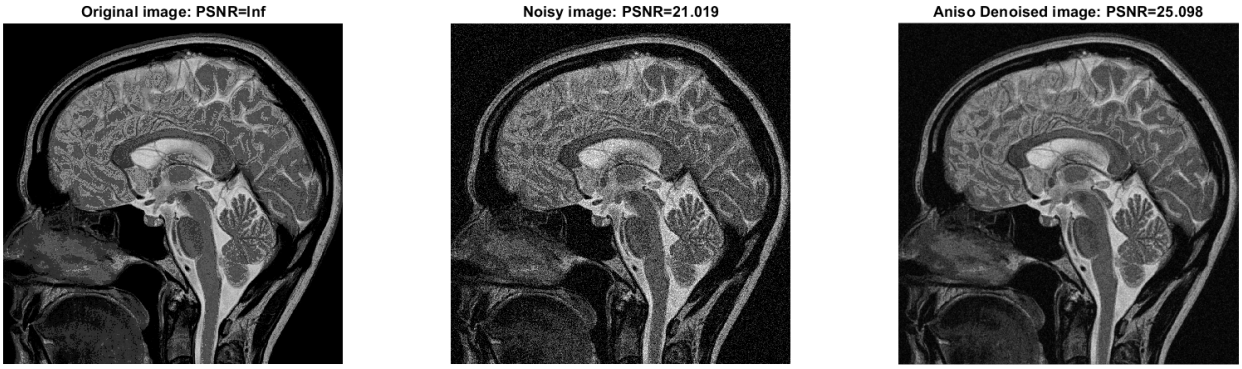
در این تابع مشتقات شمالی و جنوبی و شرقی و غربی بدست می‌آید و ضرایب هادی (conduction coefficients) به دو محاسبه می‌شود که روابط آن همانطور که در کد نیز مشخص است، بدین شکل است:

$$1) \quad c_i = e^{-(\frac{\delta}{\kappa})^2}$$

$$2) \quad c_i = \frac{1}{1 + (\frac{\delta}{\kappa})^2}$$

با اعمال Cross-Validation پارامترهای بهینه را پیدا می‌کنیم. در نتیجه پارمترها را به صورت زیر تعریف کرده و نتایج نیز به صورت زیر است:

```
% Denoise the image using anisotropic diffusion
niter = 7;          % Number of iterations
kappa = 5;          % Conduction coefficient
lambda = 0.05;      % Maximum value for stability
option = 1;         % Diffusion equation No 2
aniso_im_denoised = anisotropic(im_noisy, niter, kappa, lambda, option);
```



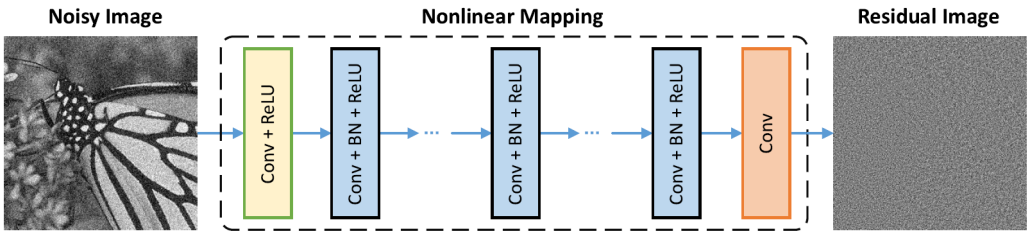
شکل ۷: تصویر سالم (سمت چپ)، تصویر نویزی (وسط) و تصویر رفع نویز شده (سمت راست)

۵.۲ سوال پنجم Deep Denoising

DnCNN (که مخفف Denoising Convolutional Neural Network است) یک شبکه عصبی عمیق برای حذف نویز در تصاویر است. هدف اصلی DnCNN ، بازسازی تصاویر نویزدار را به تصاویر پاک با استفاده از شبکه عصبی داخلی خود، انجام می‌دهد.

ساختار کلی DnCNN شامل لایه‌های کانولوشنی (Convolutional layers) و لایه‌های ReLU است. لایه‌های کانولوشنی وظیفه استخراج ویژگی‌های تصویر را دارند. این لایه‌ها با استفاده از فیلترهای کانولوشنی با اندازه‌ها و عمق‌های مختلف، تصویر را پردازش می‌کنند و ویژگی‌های مهم را استخراج می‌کنند. بعد از هر لایه کانولوشنی، یک لایه ReLU وجود دارد. ReLU (Rectified Linear Unit) یک تابع فعال‌سازی است که در خروجی لایه‌های کانولوشنی به کار می‌رود. ReLU وظیفه‌ی اعمال عملیات غیرخطی بر روی خروجی لایه‌های کانولوشنی را دارد و با حذف اثر نویز غیرضروری به بهبود عملکرد سیستم کمک می‌کند. همچنین، DnCNN شامل لایه‌های Batch Normalization است. این لایه‌ها وظیفه نرمال‌سازی ورودی‌های لایه‌های کانولوشنی را دارند تا فرآیند آموزش سریعتر و پایدارتر شود.

ساختار کلی DnCNN به گونه‌ای طراحی شده است که با عبور تصویر از شبکه، نویزها و جزئیات غیرضروری حذف شده و تصویر پاک‌تر و با کیفیت به دست می‌آید. به طور خلاصه، DnCNN یک شبکه عصبی عمیق برای حذف نویز در تصاویر است که با استفاده از لایه‌های کانولوشنی، لایه‌های ReLU و لایه‌های Batch Normalization ، تصاویر نویزدار را به تصاویر پاک تبدیل می‌کند.



شکل ۸: شمای گرافیکی ساختار DnCNN (خروجی این مدل می‌تواند نویز یا تصویر رفع‌نویز شده باشد)

دیتاست داده شده شامل ۱۰۰ نمونه عکس سیتی‌اسکن است. می‌توان هر تصویر را به چندین patch تبدیل کرد (این امر باعث بهبود عملکرد مدل خواهد شد). کلاس CTDataset به منظور خواندن و تبدیل هر تصویر به چندین patch نوشته شده است:

```
class CTDataset(Dataset):
  def __init__(self, dir, input_size=(512, 512), mode='train', patch_size=None, stride=None):
    self.main_dir = dir
    self.image_width, self.image_height = input_size
    df = pd.read_csv(self.main_dir + '/overview.csv')
    if mode == 'train':
      self.files_name = df.tiff_name.values[:80]
    else:
      self.files_name = df.tiff_name.values[80:]
    if patch_size is None:
      self.patch_size = (self.image_width, self.image_height)
      self.stride, self.n_patches = 1, 1
    else:
      self.patch_size, self.stride = patch_size, stride
      self.n_width_patches = (self.image_width - patch_size[0]) // stride + 1
      self.n_height_patches = (self.image_height - patch_size[1]) // stride + 1
      self.n_patches = self.n_width_patches * self.n_height_patches

  def __len__(self):
    return len(self.files_name) * self.n_patches

  def __getitem__(self, index):
    im = self.get_row_item(index)
    noise = np.random.normal(0, .1, im.shape)
    noisy_im = im + noise
    noisy_im[noisy_im < 0] = 0
    noisy_im[noisy_im > 1] = 1
    return torch.tensor(noisy_im, dtype=torch.float32), torch.tensor(im, dtype=torch.float32)

  def get_row_item(self, index):
    file_index = index // self.n_patches
    patch_index = index - file_index * self.n_patches
    im = skio.imread(self.main_dir + '/tiff_images' + '/' + self.files_name[file_index])
    im = im.reshape((1, self.image_width, self.image_height))
    im = (np.array(im) - np.min(im)) / (np.max(im) - np.min(im))
    start_patch_i = patch_index // self.n_width_patches
    start_patch_j = patch_index - start_patch_i * self.n_width_patches
    start_index_i, start_index_j = start_patch_i * self.stride, start_patch_j * self.stride
    im = im[:, start_index_i:start_index_i + self.patch_size[0], start_index_j:start_index_j +
        self.patch_size[1]]
    return im
```

با توجه به کلاسی که تعریف کرده‌ایم دو دیتاست train و test را می‌سازیم. ۸۰ درصد داده‌ها را به train و ۲۰ درصد را به test اختصاص می‌دهیم.

```
image_size = (512, 512)
patch_size = (512, 512)
stride = 1
batch_size = 5

train_ds = CTDataset(dir=main_dir, input_size=image_size, mode='train', patch_size=patch_size,
    stride=stride)
test_ds = CTDataset(dir=main_dir, input_size=image_size, mode='test', patch_size=patch_size,
    stride=stride)

train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=False, num_workers=2)
test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False, num_workers=2)
```

در اینجا متغیر patch_size برابر با اندازه تصویر تنظیم شده است. می‌توان با تغییر پارامترها تعداد نمونه‌ها را افزایش یا کاهش داد.
اکنون مدل DnCNN را به صورت زیر تعریف می‌کنیم:

```
class DnCNN(nn.Module):
  def __init__(self, num_layers=17, num_filters=64):
    super(DnCNN, self).__init__()
    self.num_layers = num_layers
    self.num_filters = num_filters

    self.first_layer = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=self.num_filters, kernel_size=3, padding='same'), nn.ReLU()
    )
    hidden_layers = []
    for _ in range(num_layers - 2):
      hidden_layers.append(nn.Conv2d(in_channels=num_filters, out_channels=num_filters,
          kernel_size=3, stride=1, padding=1))
      hidden_layers.append(nn.BatchNorm2d(num_filters))
      hidden_layers.append(nn.ReLU())
      self.hidden_layers = nn.Sequential(*hidden_layers)
    self.final_layer = nn.Conv2d(in_channels=num_filters, out_channels=1, kernel_size=3, stride=1,
        padding=1)

  def forward(self, x):
    out = self.first_layer(x)
    out = self.hidden_layers(out)
    out = self.final_layer(out)
    return out
```

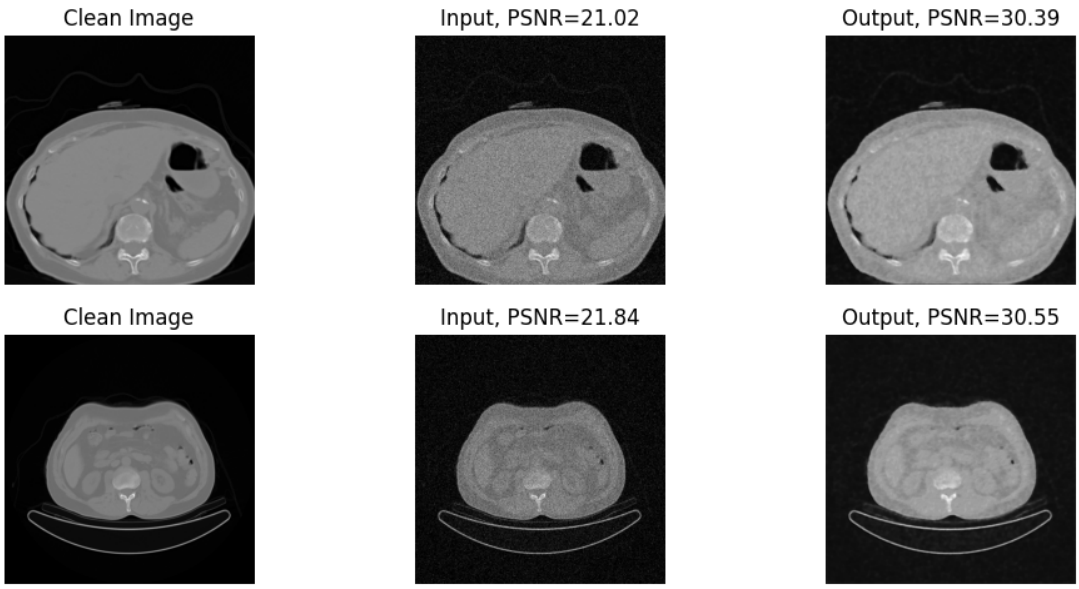
با تعیین تعداد لایه‌های شبکه برابر با ۱۲ و تعداد فیلترها برابر با ۶۴، مشخصات دیگر شبکه به صورت زیر خواهد بود:

```
Total params: 186,112
Trainable params: 186,112
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.00
Forward/backward pass size (MB): 2178.00
Params size (MB): 0.71
Estimated Total Size (MB): 2179.71
```

پس از آموزش مدل مرحله تست مدل است. با توجه به اینکه خروجی مدل یک path می‌باشد، نیاز است تصویر با ابعاد درست استخراج شود. تابع زیر بدین منظور نوشته شده است:

```
def reconstruct_img_from_patches(result, ds, index, image_size, patch_size, stride):
  image_width, image_height = image_size
  n_width_patches = (image_width - patch_size[0]) // stride + 1
  n_height_patches = (image_height - patch_size[1]) // stride + 1
  reconst_input = torch.zeros(image_size, dtype=torch.float32)
  reconst_output = torch.zeros(image_size, dtype=torch.float32)
  reconst_clean_image = torch.zeros(image_size, dtype=torch.float32)
  for i in range(n_width_patches):
    for j in range(n_height_patches):
      item_index = i * n_width_patches + j
      start_index_i = i * stride
      end_index_i = start_index_i + patch_size[0]
      start_index_j = j * stride
      end_index_j = start_index_j + patch_size[1]
      noisy_image, _ = ds.__getitem__(index + item_index)
      reconst_input[start_index_i:end_index_i, start_index_j:end_index_j] = noisy_image
      reconst_output[start_index_i:end_index_i, start_index_j:end_index_j] = result[index +
          item_index][0].to('cpu')
      clean_image = ds.get_row_item(index + item_index)
      reconst_clean_image[start_index_i:end_index_i, start_index_j:end_index_j] =
          torch.tensor(clean_image, dtype=torch.float32)
  return reconst_clean_image, reconst_input, reconst_output
```

در پایان تصویر نویزی موجود در داده تست را به ورودی مدل می‌دهیم و خروجی مدل که تصویر رفع‌نویز شده می‌باشد را بدست می‌آوریم. نتیجه برای دو نمونه تست در ادامه آمده است. (نمونه‌های بیشتر در فایل کد قایل مشاهده است.)



شکل ۹: ورودی و خروجی شبکه و تصویر رفع نویز شده برای دو نمونه تست

با افزایش تعداد دادگان، افزایش تعداد لایه‌های مدل و افزایش تعداد فیلترهای مدل می‌توان عملکرد مدل را بهبود بخشید. مقادیر ذکر شده با درنظر گرفتن ملاحظات سخت‌افزاری تعیین شده است.