به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

پردازش سیگنال‌های زمان-گسسته

تمرین کامپیوتری سری چهارم

نام و نام خانوادگی	یاسمن پرهیزکار
شماره دانشجویی	۸۱۰۱۹۵۵۵۹
تاریخ ارسال گزارش	۱۳۹۹/۵/۱۳

فهرست سوالات گزارش

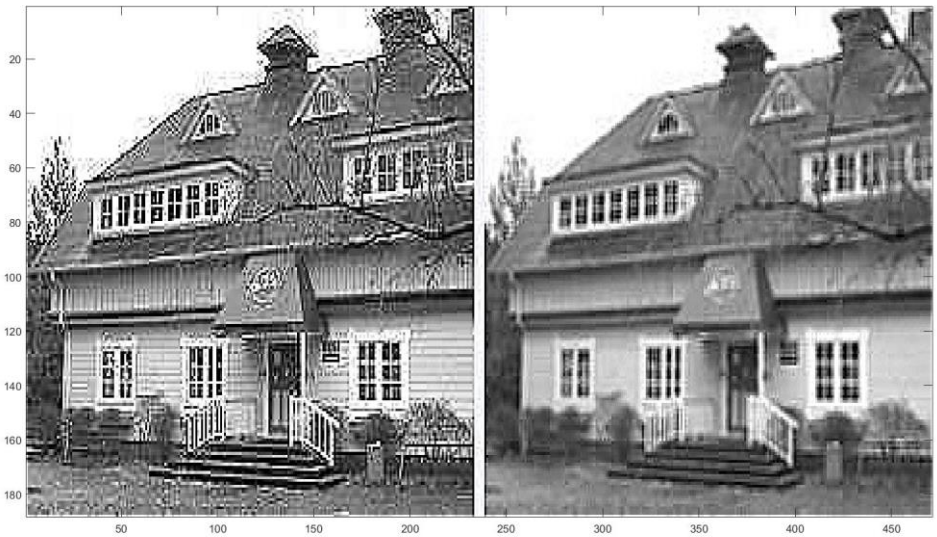
سوال ۱ – عملکرد کرنل های مختلف در Spatial Domain Filtering

کد مربوط به این سوال را می توانید در فایل spatial_filtering_01.m مشاهده کنید.

در این کد، بعد از خواندن تصویر house.jpg فیلتر های گفته شده را به آن اعمال کردیم؛ حاصل فیلتر ها را در شکل های ۱ تا ۹ مشاهده می کنید.

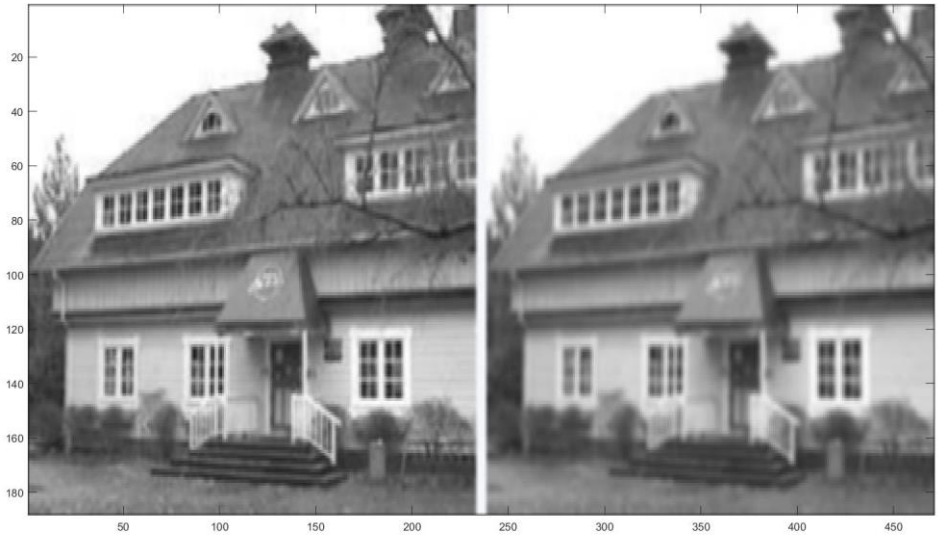
تنها نکته ای که در این کد وجود دارد، این است که هر جا به وزن منفی برخوردیم، آن را به 0 گرد کردیم، نه اینکه از آن قدر مطلق بگیریم. این کار کیفیت فیلترکردن تصاویر را بهتر کرد.

شکل 1- تصویر اصلی و فیلترنشده ی house.jpg



شکل 2- فیلتر اول: sharpen

فیلتر اول، تفاوت بین پیکسل ها را برجسته می کند؛ گویی اثر محوشدگی سیگنال را که با کرنل هایی مانند blur ایجاد می شوند، تا حدی خنثی می کند. همانطور که در شکل ۲ مشاهده می کنید، تصویر خانه ی سمت راست که در ابتدا محو بود، واضح تر شده است.

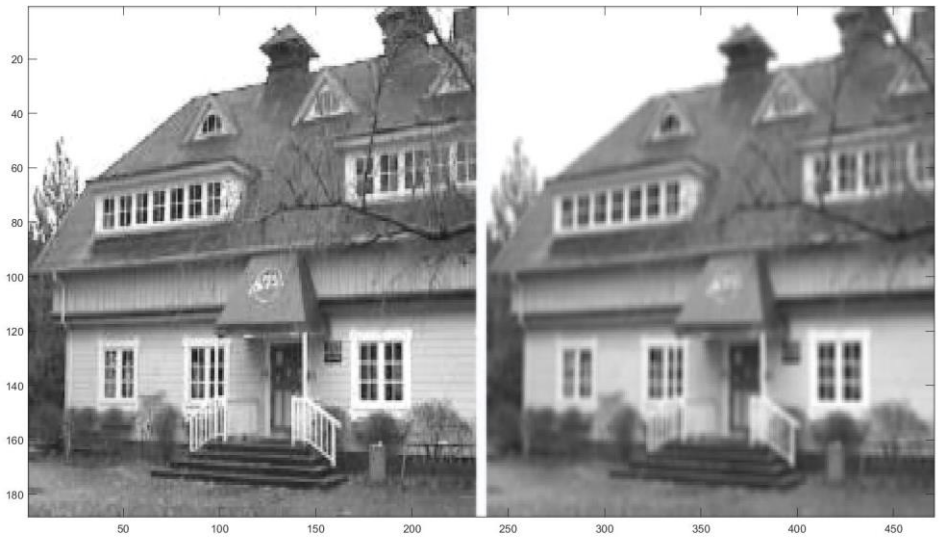


شکل 3- فیلتر دوم: blur

فیلتر دوم یا blur همانطور که از اسمش پیداست، تصویر را محو می کند.

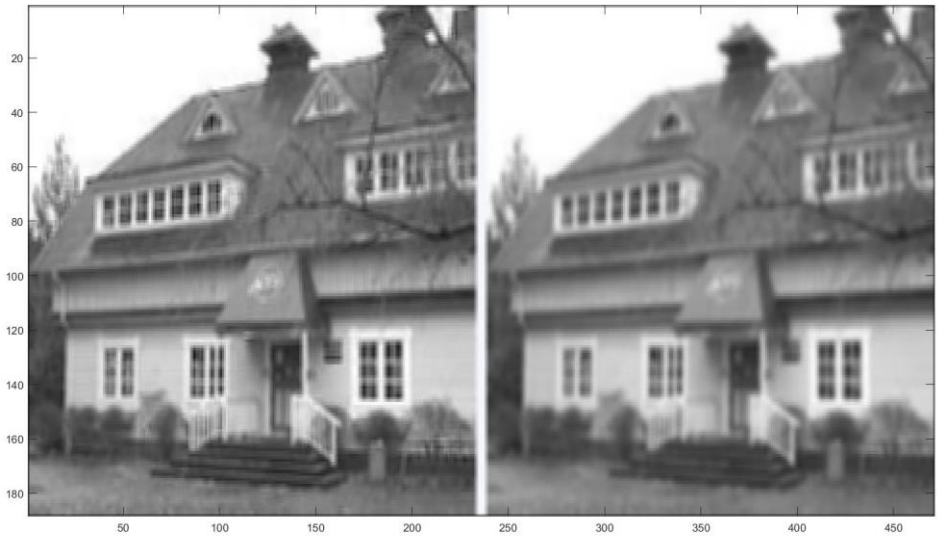
شکل 4- فیلتر سوم: outline

فیلتر سوم یا outline خطوط دور شکل ها را برجسته می کند. این خطوط را در نقاطی پیدا می کند که رنگ پیکسل ها ناگهان با شیب زیادی تغییر می کند؛ مثلا یکهو از حول و حوش 10 به حول و حوش 200 می رسند. هرچه این تغییرات ناگهانی بزرگتر باشند، آن خط را سفید تر نشان می دهد. به همین دلیل در تصویر محوشده ی سمت راست، کلا همه ی خط ها را کمرنگ تر نشان داده است.



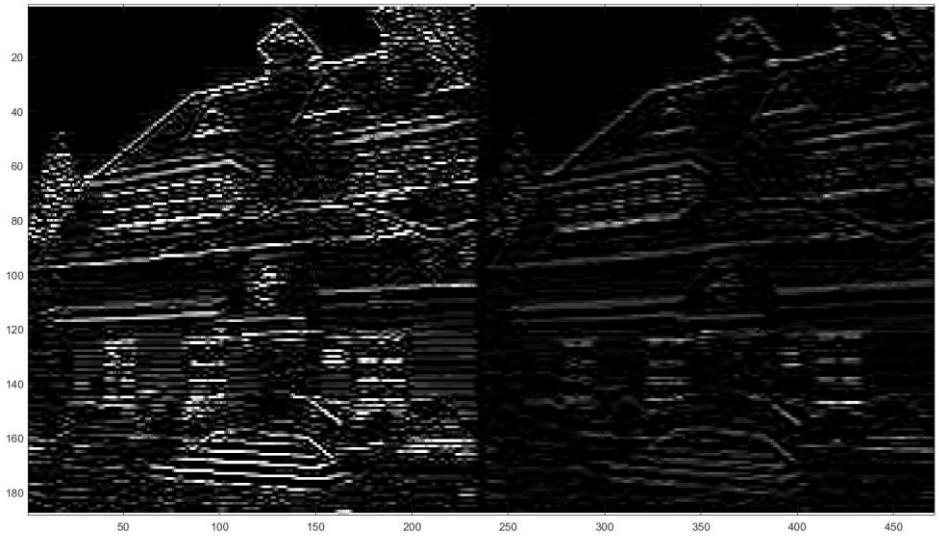
شکل 5- فیلتر چهارم: gauss

فیلتر چهارم یا gauss هم یک نوع فیلتر محو کننده است، مانند blur.



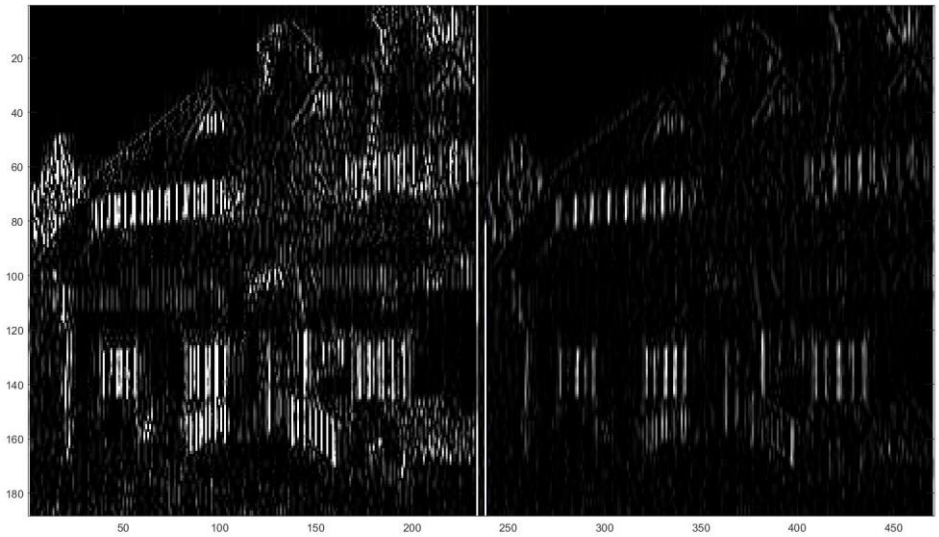
شکل 6- فیلتر پنجم: avg moving

فیلتر پنجم یا avg moving هم مانند فیلتر های gauss و blur یک فیلتر محوکننده است. به این صورت عمل می کند که میانگین رنگ هر پیکسل و ۸ پیکسل همسایه اش را جایگزین آن پیکسل در عکس فیلترشده می کند.



شکل 7- فیلتر ششم یا line H

فیلتر ششم یا line H خطوط افقی را در عکس پیدا می کند. روش کار آن، مشابه outline است؛ یعنی نواحی ای را که رنگ پیکسل های یک خط افقی ناگهان تغییر شدید کرده اند، برجسته می کند.



شکل 8- فیلتر هفتم: line V

فیلتر هفتم یا line V خطوط عمودی را در عکس پیدا می کند. نحوه ی کار آن کاملا مانند line H است و درواقع کرنل line V صرفا ترانهاده ی کرنل line H است.

شکل 9- فیلتر هشتم: identity

فیلتر هشتم یا identity همان عکس اصلی را برمی گرداند؛ زیرا کرنل آن طوری طراحی شده که دقیقا مقدار هر پیکسل را (بدون تغییر) جایگزین آن پیکسل در تصویر فیلترشده می کند.

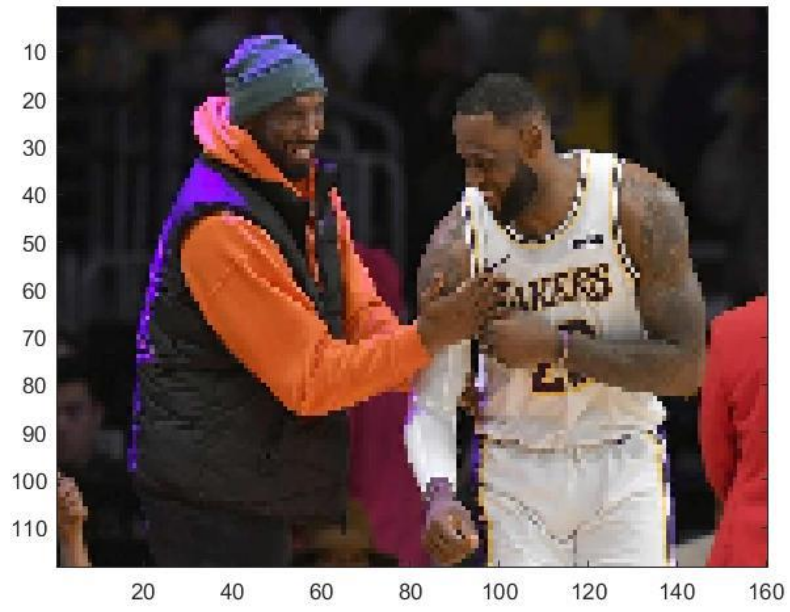
سوال ۲ – کوچک کردن تصویر با دو روش درون یابی مختلف

کد مربوط به این سوال را در فایل resize_02.m مشاهده می کنید.

شکل ۱۰ تصویر اصلی، شکل ۱۱ تصویر کوچک شده با متدِ درون یابِ پیش فرض متلب (bicubic) و شکل ۱۲ تصویر کوچک شده با متدِ درون یابِ nearest neighbor را نشان می دهد.

شکل 10- تصویر اصلی پیش از resize شدن

شکل 11- تصویر کوچک شده با درون یابی bicubic



شکل 12- تصویر کوچک شده با درون یابی nearest neighbor

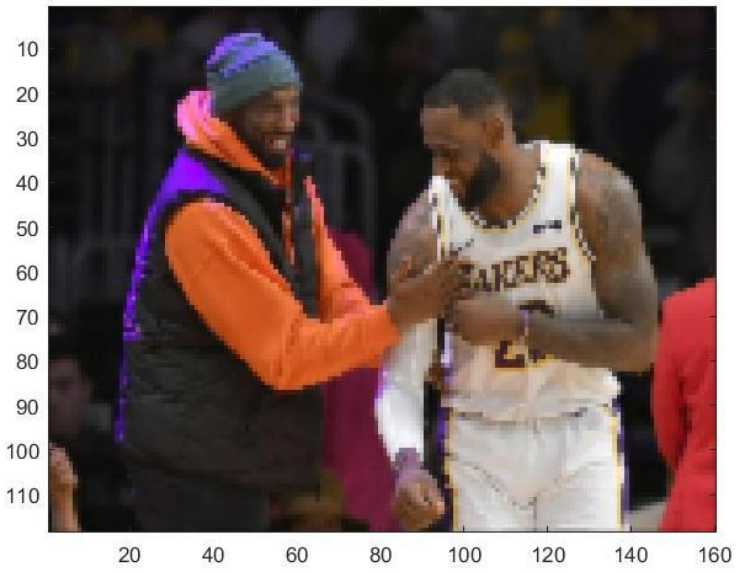
در روش دورن یابی nearest neighbor هر پیکسل در عکس کوچک شده، مقدار نزدیک ترین پیکسل به جایگاه آن را می گیرد. مثلا اگر بخواهیم تصویر را ۵ برابر کوچک کنیم، عکس اصلی را به مربع های 5x5 تقسیم کرده و پیکسل وسط هر کدام را در جایگاه مربوطه در عکس کوچک شده قرار می دهد.

این روش درون یابی بسیار سریع است، اما همانطور که از مقایسه شکل های ۱۱ و ۱۲ متوجه می شویم، عکس نهایی با کیفیتی تولید نمی کند.

روش درون یابی bicubic برای بدست آوردن مقدار هر پیکسل در عکس کوچک شده، از مقادیر یک مربع 4x4 در تصویر اصلی میانگین می گیرد. در نتیجه، همانطور که در شکل ۱۱ می بینیم، تصویر نهایی smooth تری تولید می کند.

همچنین، هر دو تصویر کوچک شده نسبت به تصویر اصلی کیفیت پایین تر و پیکسل های درشت تری دارند؛ چرا که اصلا معنای کوچک کردن تصویر همین است! پیکسل های تصاویر بازیابی شده در اینجا، مساحتی معادل 25 پیکسلِ تصویر اصلی را پوشش می دهند.

برای بهبود تصویر حاصل از nearest neighbor آن را با کرنل های گوسی و moving avg فیلتر می کنیم. نتیجه ی کار را در تصاویر ۱۳ و ۱۴ مشاهده می کنید. همانطور که می بینید، تصویر smooth تر شده است، اما هنوز به خوبی تصویر حاصل از bicubic نیست.



شکل 13- تصویر حاصل از nearest neighbor بعد از فیلترشدن توسط کرنل gauss

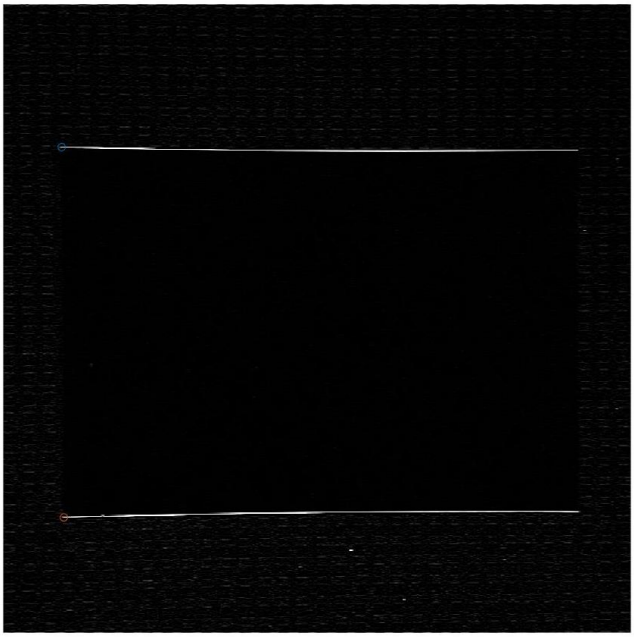
شکل 14- تصویر حاصل از nearest neighbor بعد از فیلتر شدن توسط کرنل های gauss و moving avg

سوال ۳ – تشخیص لبه های کاغذ

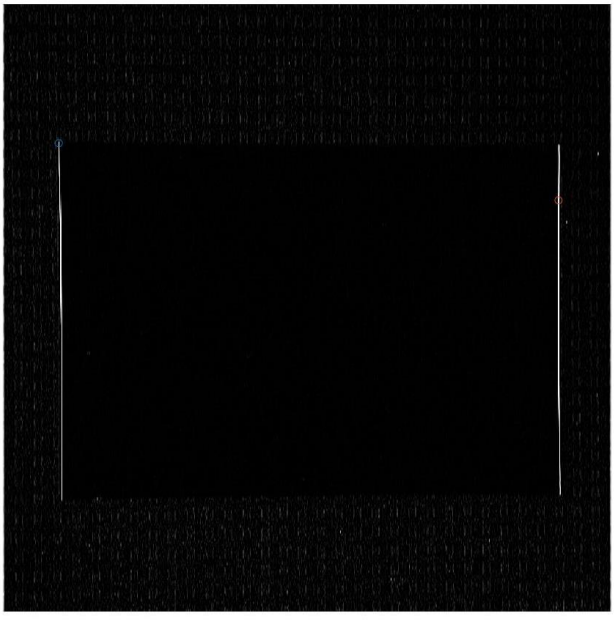
کد مربوط به این بخش را در فایل edge_detection.m مشاهده می کنید.

برای کشیدن مستطیل دور کاغذ، به اندیس i و j مربوط به چهار گوشه ی کاغذ نیاز داریم. اندیس های i را از روی خط های افقی تصویر (شکل ۱۵) و اندیس های j را از روی خط های عمودی تصویر (شکل ۱۶) به دست می آوریم.

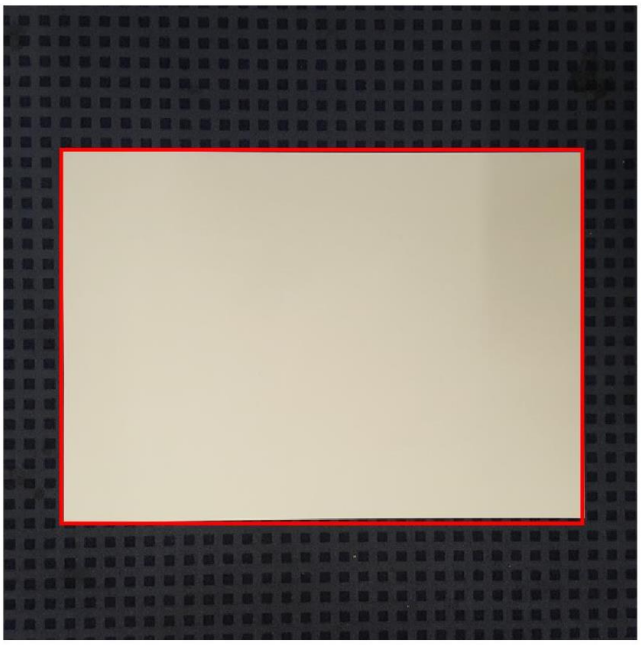
سپس با مقادیر به دست آمده، یک مستطیل قرمز دور لبه های کاغذ می کشیم (شکل ۱۷).



شکل ۱۵- به دست آوردن اندیس i لبه های کاغذ



شکل 16- به دست آوردن اندیس j لبه های کاغذ



شکل 17- تشخیص لبه های کاغذ که با یک مستطیل قرمز نشان داده است

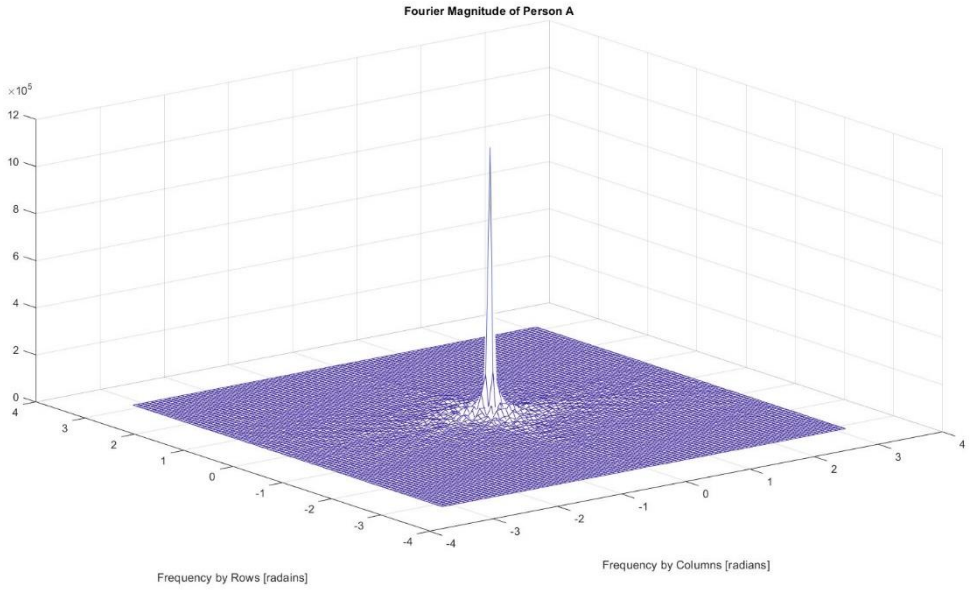
سوال ٤ – Frequency Domain Filtering

کد مربوط به این بخش را در فایل frequency_filtering_04.m مشاهده می کنید.

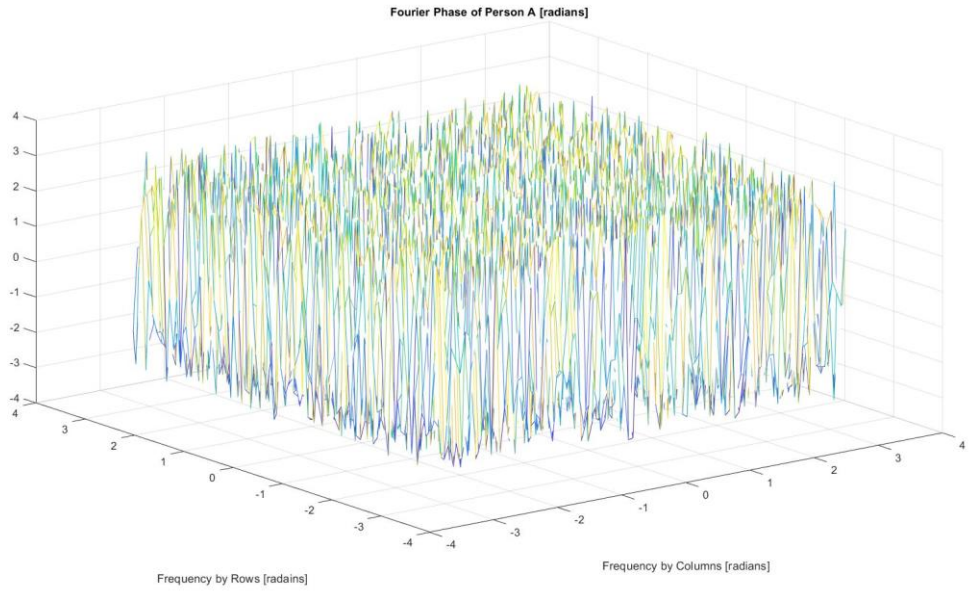
در این بخش، تصاویر فولدر s7 را (به عنوان شخص A) انتخاب کردیم. در شکل ۱۸ تصویر آقای A را مشاهده می کنید.

شکل 18- تصویر شخص A

سپس، از تصویر شکل ۱۸ تبدیل فوریه ی دو بعدی گرفتیم و اندازه و فاز آن را رسم کردیم؛ که به ترتیب در شکل های ۱۹ و ۲۰ مشاهده می کنید.



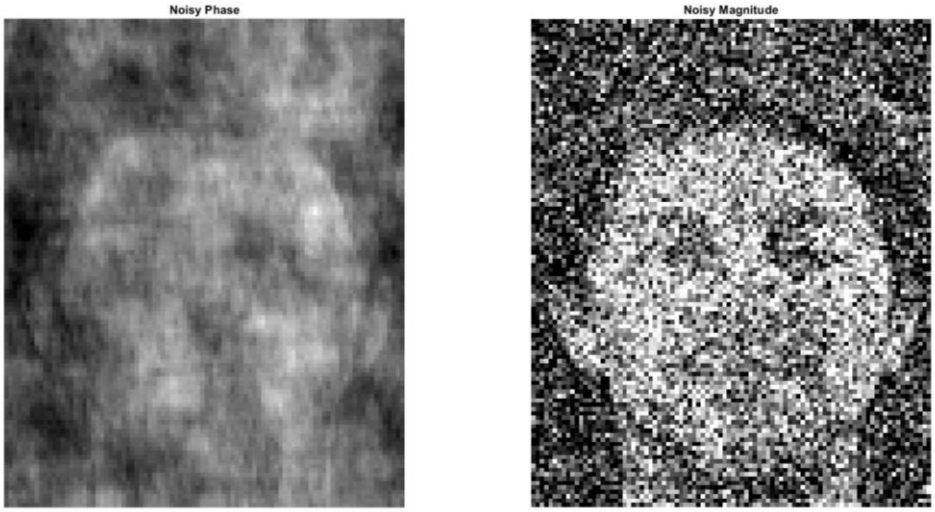
شکل 19- اندازه تبدیل فوریه ی دو بعدی تصویر شخص A



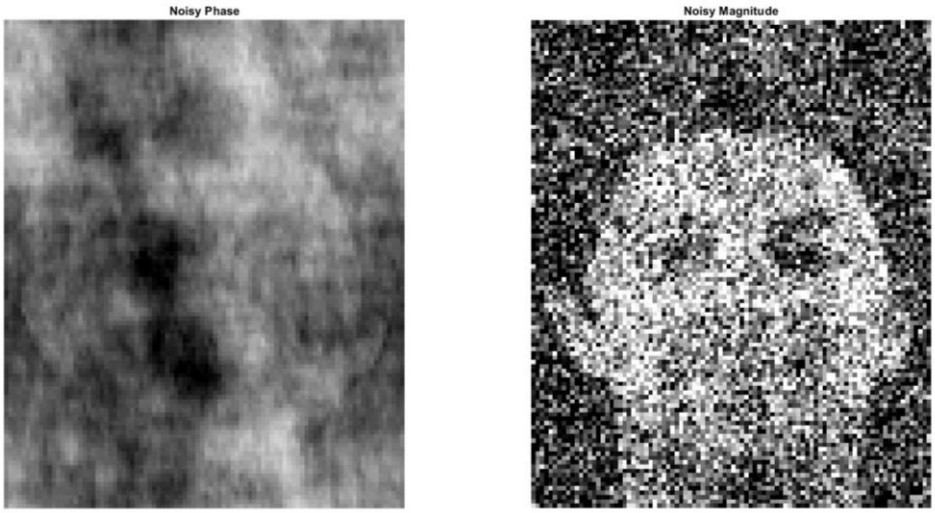
شکل 20- فاز تبدیل فوریه ی دو بعدی تصویر شخص A

حال طبق خواسته صورت پروژه، عکس فرد را یک بار با فاز نویزی و یک بار با دامنه ی نویزی رسم کردیم.

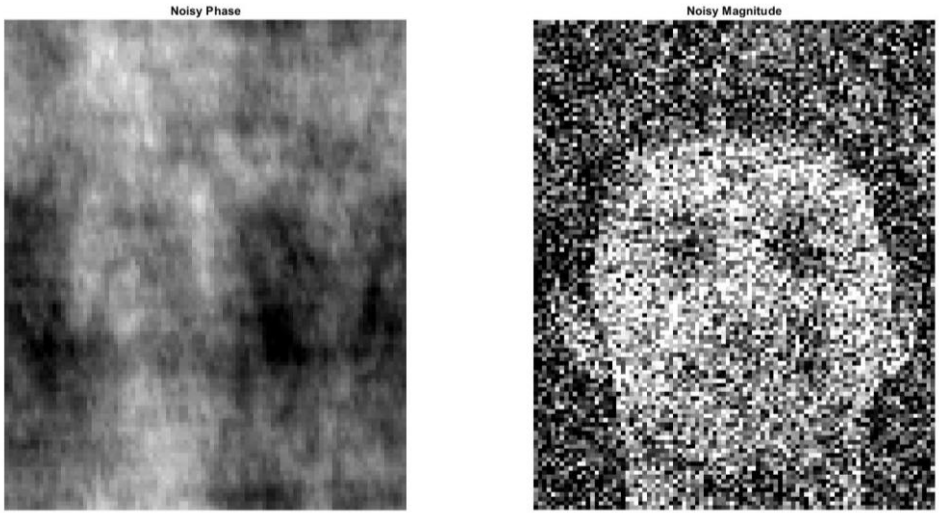
همچنین این کار را برای SNR های ۱و ۰/۵ و ۰/۲۵ انجام دادیم که نتایج آن ها را به ترتیب در شکل های ۲۱ و ۲۲و ۲۳ مشاهده می کنید.



شکل 21- تصویر شخص A با دامنه ی نویزی (سمت راست) و فاز نویزی (سمت چپ)؛ SNR = 1

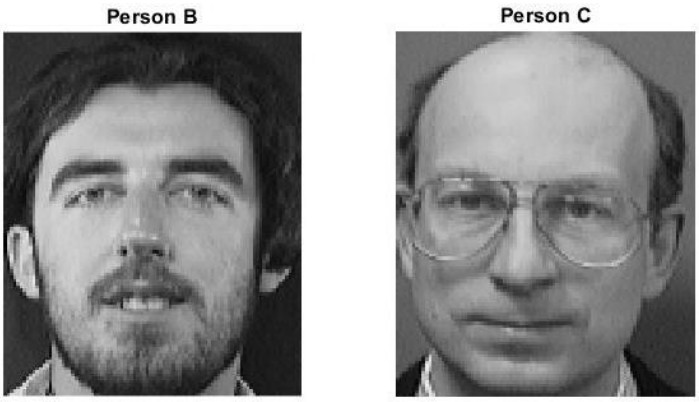


شکل 22- تصویر شخص A با دامنه ی نویزی (سمت راست) و فاز نویزی (سمت چپ)؛ SNR = 0.5



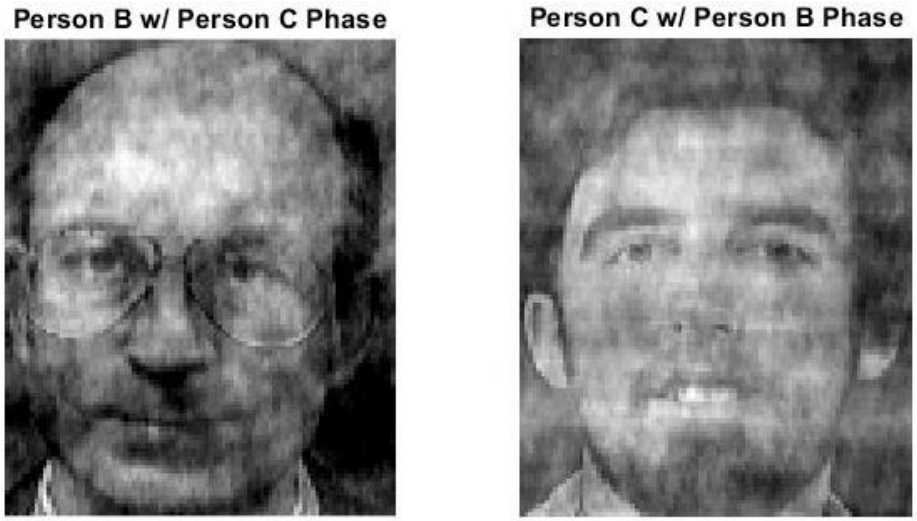
شکل 23- تصویر شخص A با دامنه ی نویزی (سمت راست) و فاز نویزی (سمت چپ)؛ SNR = 0.25

حال، فولدر s11 را به عنوان فرد B و فولدر s13 را به عنوان فرد C انتخاب می کنیم. تصاویر این افراد را در شکل ۲۴ مشاهده می کنید.



شکل 24- تصویر فرد B و فرد C

تبدیل فوریه تصاویر این افراد را در متلب به دست آوردیم. سپس، فاز عکس ها را با هم عوض کردیم. نتیجه را در شکل ۲۵ می بینید.



شکل 25- تصاویر افراد B و C ؛ بعد از جا به جا کردن فاز آنها

همانطور که مشاهده می کنید، شکل ۲۵ تقریبا مثل این است که جای عکس فرد B و C را عوض کرده ایم.

همچنین، در شکل های ۲۱ تا ۲۳ هم تصاویری که فاز بدون نویز داشتند، کیفیت بسیار بهتری داشتند و عکس فرد بیشتر قابل تشخیص بود.

از این دو آزمایش، نتیجه می گیریم که بیشتر اطلاعات چهره افراد در **فاز** تبدیل فوریه آن ذخیره شده است.

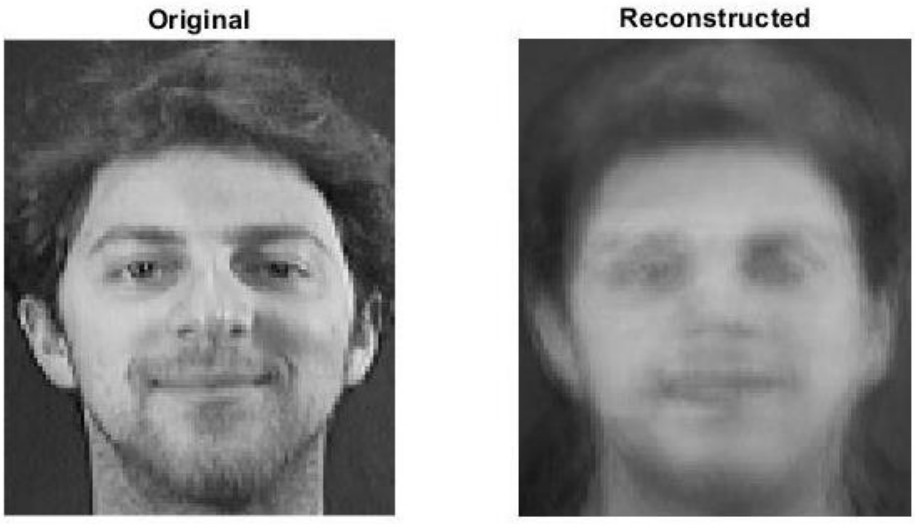
سوال ۵ – Face Recognition

کد مربوط به این بخش را در فایل face_recognition_05.m مشاهده می کنید.

عکس های شماره ۱ تا ۸ را برای هر شخص به دسته train و عکس های ۹ و ۱۰ را به دسته test اختصاص می دهیم.

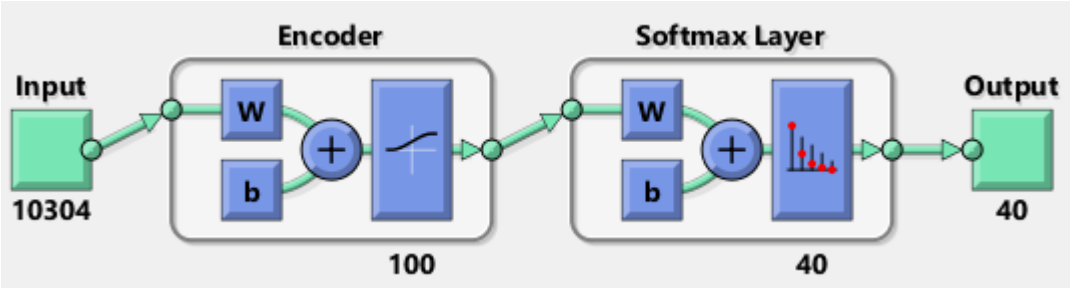
ابتدا تمام عکس های موجود در این دو دسته را به شکل یک cell array از ماتریس های دو بعدی در می آوریم. سپس، Autoencoder را با دسته ی train آموزش می دهیم. ضرایب نهایی Autoencoder پس از آموزش دیدن را با نام autoenc.mat ذخیره کرده ام (اما به دلیل زیاد شدن حجم فایل نتوانستم در cecm هم آپلودش کنم).

حال، برای تست عملکرد Autoencoder تصویر شخص A را به آن می دهیم. خود تصویر و خروجی آن را می توانید در ۲۶ مشاهده کنید.



شکل 26- تصویر شخص A (سمت چپ) و تصویر بازیابی شده وی پس از عبور از Autoencoder

حال، طبق صورت پروژه Softmax Layer را با target هایی که طبق قائده ی one-hot coding درست کرده ایم، آموزش می دهیم. در نهایت، سیستم حاصل از Autoencoder و Softmax Layer به شکل زیر می شود.



شکل ۲۷ - سیستم نهایی حاصل از یک Autoencoder و یک Softmax Layer

نسبت acc را برای داده های train و test به دست آوردیم که برای داده های train برابر ۱ و برای داده های test برابر 0.9375 شد.

پایان