به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

پردازش سیگنال‌های زمان-گسسته

تمرین کامپیوتری سری ۴

نام و نام خانوادگی	سهند خوشدل
شماره دانشجویی	۸۱۰۱۹۶۶۰۷
تاریخ ارسال گزارش	۹۹/۵/۱۳

فهرست گزارش سوالات

1.Spatial domain filtering

۱.۱) توصیف عملکرد کرنل ها :

شیوه انجام کانولوشن:

- هر یک از کرنل ها یک ماتریس سه در سه هستند که حین انجام کانولوشن مانند کانولوشن سیگنال ها، کرنل باید برعکس شود (که در اینجا معادل یک بار وارون جانبی شدن و یک بار بالا پایین شدن (horizontal and vertical flip) می باشد) و سپس روی تمام عناصر (پیکسل) ماتریس حرکت کند و ماتریس کرنل در ماتریس حاصل از پیکسل هایی که در آن واحد پوشش داده می شوند، ضرب داخلی شود و حاصل این ضرب به عنصر (پیکسل) وسط اطلاق شود. که در ادامه تمرین به این پیکسل **" پیکسل هدف"** گفته ام. از آنجا که برای پیکسل هایی که در مجاورت لبه ها و گوشه های تصویر قرار دارند به جای ۸ پیکسل همسایه، به ترتیب ۵ و ۳ پیکسل مجاور وجود دارد. خود تابع conv2 حین اعمال شدن، تصویر را به اندازه $\frac{(kernelsize-1)}{2}$ لبه ها باید extend شوند.

سپس کرنل پس از انجام ضرب روی هر بلوک ۳ در ۳ به پیکسل مجاور می رود و دوباره همین کار را تکرار می کند تا تمام پیکسل های تصویر را پوشش دهد. در ادامه به توصیف عملکرد برخی کرنل ها با توجه به تحلیل خروجی آن ها خواهیم پرداخت:

توصیف رفتار کرنل های نمونه :

0	0	0	-1	2	-1	-1	-1	-1	0.1111	0.1111	0.1111	0.0113	0.0838	0.0113	-1	-1	-1	0.0625	0.125	0.0625	0	-1	0
0	1	0	-1	2	-1	2	2	2	0.1111	0.1111	0.1111	0.0838	0.6193	0.0838	-1	8	-1	0.125	0.25	0.125	-1	5	-1
0	0	0	-1	2	-1	-1	-1	-1	0.1111	0.1111	0.1111	0.0113	0.1111	0.0113	-1	-1	-1	0.0625	0.125	0.0625	0	-1	0
identity (ح)			line V (ز)			line H (و)			avg moving (ه)			gauss (د)			outline (ج)			blur (ب)			sharpen (آ)		

Sharpen : (آ)

0	-1	0
-1	5	-1
0	-1	0

sharpen (آ)

فیلتر sharpen با اختصاص وزن های 1- به پیکسل های همسایه عمودی و افقی پیکسل هدف (وسطی)، به نوعی مشتق گیری در جهت های عمودی و افقی می پردازد (از آنجا که وزن 5 را می توان به چهار وزن 1+ برای انجام مشتق گیری با همسایه ها (در حوره گسسته difference پیدا کردن معادل مشتق گیری است) و همچنین ۱ وزن که برای پیکسل هدف باقی می ماند تا تاثیر محتوای پیکسل هدف در پیکسل متناظر تصویر خروجی بیشتر باشد (زمانی که جمع عناصر کرنل صفر نباشد چنین حالتی داریم.)

Figure 1.1: Horizontal/Vertical Sharpen Filter Output Image

مشاهده میکنیم که اگر وزن اعمالی روی پیکسل هدف را به ۴ تغییر دهیم. تنها پیکسل هایی محتوای غیر صفر خواهند داشت که در همسایگی شان تغییر رنگ (یا در اینجا intensity) شدیدی رخ داده باشد. در غیر این صورت با توجه به آنکه پیسکل های مجاور تقریبا همان سطح رنگی (یا intensity) دارا خواهند بود حاصل ضرب داخلی انجام شده حین کانولوشن برای پیکسل هدف با چنین همسایگی ای صفر خواهد بود و پیکسل متناظر در خروجی سیاه خواهد شد: (نوعی لبه یابی یا outline) برای خطوط افقی و عمودی که مشتق intensity یا گرادیان rgb آن ها در جهت های نرمال یعنی به ترتیب عمودی و افقی است، اتفاق می افتد. در کرنل شماره "ج" با لبه یابی outline با وزن ۸ نیز آشنا خواهیم شد.

0	-1	0
-1	4	-1
0	-1	0

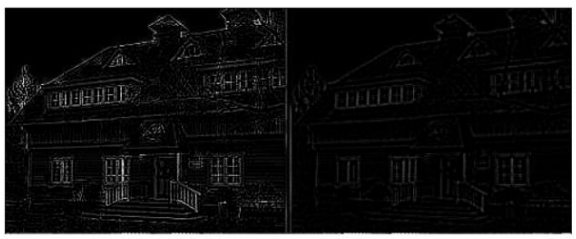

Figure 1.2: Vertical/Horizontal Outline with respect to center

(Blur : (ب)

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

blur (ب)

فیلتر blur با اختصاص دادن وزن به تمامی پیکسل های مجاور پیکسل هدف نوعی میانگین گیری را انجام میدهد منتها وزن پیکسل متناظر (هدف) نسبت به پیسکل های مجاور و پیکسل های مجاور مجاور (با فاصله دو پیکسل) (یا به نوعی مجاور اریب) به ترتیب با پله ها ½ به گونه ای کاهش می یابد که جمع عناصر کرنل ها ۱ شود (normalized kernel) و intensity تصویر در نهایت دچار تغییر شدیدی نشود.

Figure 1.3: Blur Filter Output Image

اینجا هم می توانیم ببینیم که اگر عنصر وسط را ۱ واحد افزایش دهیم همان مسئله ای که در بند قبل مطرح شد اتفاق می افتد و روشنایی تصویر مقدار قابل توجهی بیشتر خواهد شد.

0.0675	0.125	0.0675
0.125	1.25	0.125
0.0675	0.125	0.0675

Figure 1.4: Blur Filter Output Image with additional center weight:

Outline : (ج)

-1	-1	-1
-1	8	-1
-1	-1	-1

outline (ج)

همانطور که در انتهای قسمت " آ " توضیح داده شد، در حالتی که جمع وزن ها صفر باشد و خانه های مجاور وزن 1- داشته باشند ، مشتق گیری خالص صورت گرفته و وزن پیکسل هدف در محتوای پیکسل متناظر تاثیر گذار نخواهد بود. در فیلتر outline مشتق گیری در جهات اریب هم انجام می شود. با مقایسه تصویر زیر با Figure 1.2 در می یابیم که مشتق گیری در جهت اریب تاثیر بسیار بیشتری در یافتن لبه های جزئی و خطوط شکسته دارد. هر چند ممکن است کمی وضوح تصویر را مخدوش کند.

Figure 1.5: Outline Filter Output Image

در اینجا هم اگر وزن عنصر وسط را به 9 تغییر دهیم فیلتر outline عملا به sharpening فیلتر تبدیل می شود، با این تفاوت که این بار sharpening درجهت اریب هم صورت میگیرد و مشتق گیری با پیکسل هایی که فاصله بیشتری دارند هم انجام می شود، در صورتی که در بخش الف sharpening تنها درجهات افقی و عمودی انجام می شد.

-1	-1	-1
-1	9	-1
-1	-1	-1

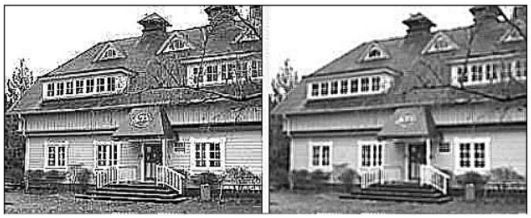

Figure 1.6: Vertical/Horizontal/Diagonal Sharpen Filter Output Image

(Gauss : (د)

0.0113	0.0838	0.0113
0.0838	0.6193	0.0838
0.0113	0.1111	0.0113

gauss (د)

فیلتر gauss نیز مانند فیلتر blur بر مبنای یک میانگین گیری وزن دار است. با این تفاوت که وزن هایی که استفاده می کند از توزیع گوسی (نرمال) بدست آمده. به همین دلیل به آن Gaussian Blur هم گفته می شود. نتیجه اعمال فیلتر gauss بر روی تصویر house در شکل زیر قابل مشاهده است:

Figure 1.7: Gaussian Blur Filter Output Image

وزن های فیلتر gauss چطور محاسبه می شوند؟

با توجه به شکل زیر برای محاسبه وزن های فیلتر gauss یک توزیع نرمال اولیه نیاز است که میانگین آن مهم نیست و می توان صفر فرض کرد اما میزان انحراف معیار در تعیین وزن ها عامل موثر اصلی است چرا که وزن ها با توجه میانگین pdf در بازه هایی که با توجه به سایز کرنل و اندازه st_dev تعریف می شود. یعنی در اینجا که کرنل های ۳ در ۳ داریم با میانگین گیری از pdf توزیع نرمال در بازه های مذکور عملا وزن توزیع نرمال در هر بخش را در مستطیل هایی که به آن fit کرده ایم جمع می کنیم. تصویر زیر شیوه تولید وزن را در کرنل 3 در 3 داده شده را نشان می دهد

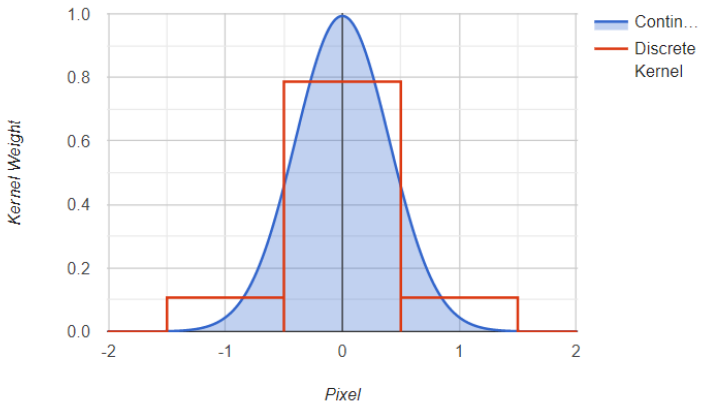

Figure 1.8: Producing Kernel Weights (In this Case: Sigma = 0.40165, Kernel Size = 3)

. با این حساب اگر kernel size = n داریم، کل بازه مورد بررسی از [-n/2 , +n/2] خواهد بود و تعداد بازه ها هم به اندازه kernel size خواهد بود. (طول بازه ها همیشه واحد است). برای مقادیر کرنل gauss آورده شده در صورت پروژه با آزمون و خطای sigma های مختلف در یافتم gauss داده شده با انحراف معیاری در حدود sigma = 0.40165 تولید شده است

بدین ترتیب از میانگین گیری در هر بازه عدد بدست می آید که از کنار هم قرار دادن اعداد حاصله در یک بردار با تعداد (kernel size) عنصر، کرنل یک بعدی ما ساخته می شود. حال از ضرب کردن ترانهاده این کرنل یک بعدی که ستونی است $(\text{kernel size} \times 1)$ در خودش که ردیفی است یک ماتریس دو بعدی که همان کرنل نهایی است به دست می آید. که اعداد متناظر با کرنل استفاده شده در این تمرین در تصویر زیر قابل مشاهده اند.

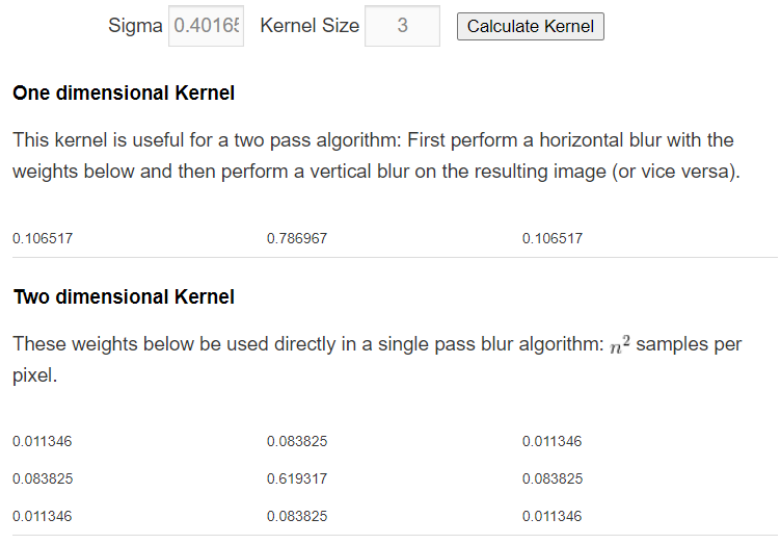

Figure 1.9: Building 2D kernel weights

می توان به این نتیجه نیز رسید که اگر میزان انحراف معیار توزیع نرمالی که این وزن ها را تولید می کند افزایش دهیم. وزن بیشتری به عنصر وسط (متناظر با پیکسل هدف در کانولوشن) و وزن کمتری به همسایگان داده خواهد شد و بدین ترتیب، شدت blur کمتر خواهد شد و از این نکته می توان برای طراحی این فیلتر در حالتی که مثل نرم افزار های ادیت عکس مثل فتوشاپ شدت آن قابل تنظیم است استفاده کرد.

0.1111	0.1111	0.1111
0.1111	0.1111	0.1111
0.1111	0.1111	0.1111

avg moving (٥)

(*Avg Moving*: (٥)

در این فیلتر به تمام عناصر کرنل وزن یکسان اختصاص داده شده (همچنین normalize شده تا intensity تصویر در مجموع زیاد و کم نشود. در اینجا بر خلاف میانگین وزن داری که در blur داشتیم، که وزن های مختلف به عنصر وسط و همسایه ها اختصاص داده می شد، تاثیر پیکسل هدف (که متناظر با عنصر وسط کرنل است)، در خروجی مانند پیکسل های مجاور خود است و میانگین گیری به نوعی عادلانه انجام می شود. بدین ترتیب میزان مشتق (گرادیان) در میان پیکسل ها کاهش می یابد و تغییر نور شدید در پیکسل های مجاور نرم می شود. خروجی این فیلتر در تصویر زیر قابل مشاهده است.

Figure 1.10: Average Moving Filter Output Image

(و ، ز) : *(line_H, line_V)*

-1	2	-1
-1	2	-1
-1	2	-1

line V (ز)

-1	-1	-1
2	2	2
-1	-1	-1

line H (و)

این دو فیلتر، فیلتر های مورد استفاده در فار اول تمرین هستند. که در واقع line_H با مشتق گیری نسبت به راستای عمودی، خطوط افقی برجسته خواهند شد و در line_V هم با مشتق گیری نسبت به راستای افقی، خطوط عمودی برجسته می شوند. فرق این مشتق گیری با مشتق گیری انجام شده در فیلتر outline مشتق گیری همرمان نسبت به راستای افقی و عمودی برای پیکسل هدف صورت میگیرد، در خالی که در این دو فیلتر برای برجسته شدن خطوط عمدی و افقی مشتق گیری فقط نسبت به راستای دیگر صورت می گیرد.

خروجی فیلتر های line_H و line_V در شکل های زیر قابل مشاهده اند.

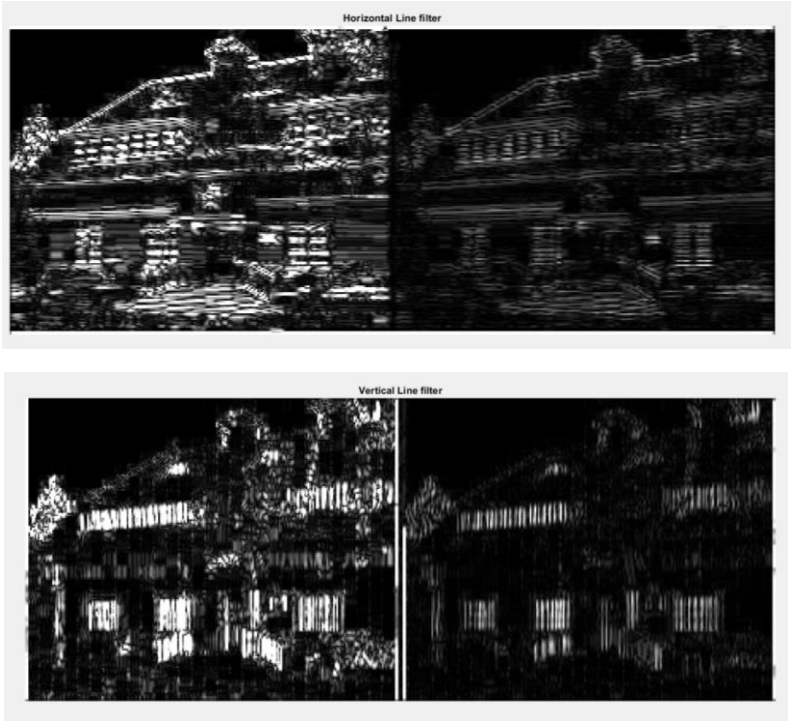

Figure 1.11: line_H, line_V Filter Output Image

(*Identity*: (ح)

0	0	0
0	1	0
0	0	0

identity (ح)

کرنل فیلتر identity در واقع همان ماتریس identity است که هیچ تاثیری بر پیکسل هدف نمی گذارد و در نتیجه تصویر خروجی همان تصویر اولیه است.

تصویری خروجی همان تصویر اولیه است :

Figure 1.12: Identity Filter Output Image

می توان از حاصلضرب اعداد در این کرنل برای تغییر سطح روشنایی استفاده کرد. به عنوان مثال در تصویر سطح روشنایی تمام پیکسل ها ۲ برابر شده است.

0	0	0
0	2	0
0	0	0

Figure 1.13: (2 × Identity) Filter Output Image

۱.۲ (*Image Scaling* :

در تصویر داده شده (kobe.jpeg) ابتدا با استفاده از تابع imresize و وارد کردن 0.2 به عنوان ضریب گسترش، یک پنجم پیکسل ها حذف می شوند بنابراین یا باید سایز عکس خروجی ثابت بماند و **رزولوشن نمایش به یک پنجم مقدار اولیه کاهش یابد** و یا بالعکس (که انتخاب متلب گزینه اول است)

1.2.1: Down sampling:

در تصویر زیر مشاهده می شود که بدلیل حذف ۸۰ درصد پیکسل ها تصویری که از کنار هم قرار دادن پیکسل های باقی مانده تشکیل می شود، اگر در همان ابعاد نشان داده شود، بدلیل تراکم پیکسلی (رزولوشن کمتر (۰.۲)) دارای وضوح و کیفیت کمتری خواهد بود..

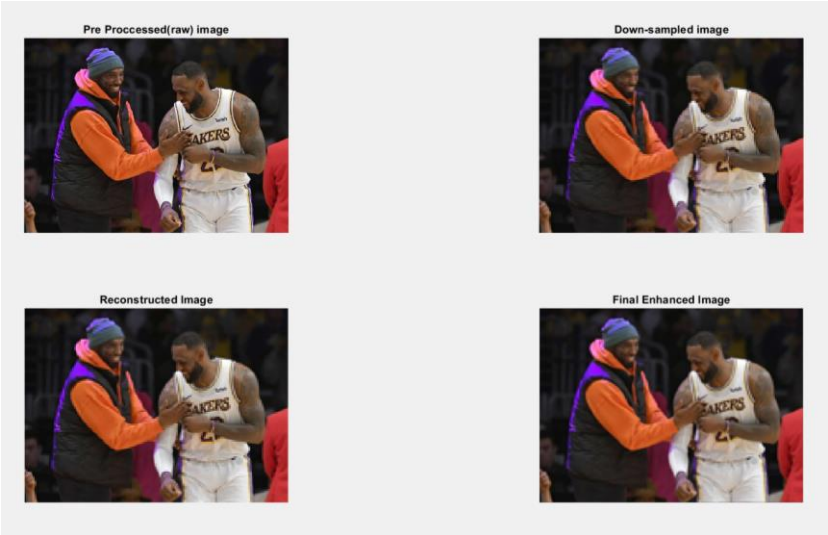

Figure 1.14: kobe.jpeg output at each processing state

down sapling یک تصویر هم مانند سیگنال های یک بعدی بدین صورت انجام می شود که هر ۵ پیکسل در میان مقدار نگه داشته می شود و محتوای سایر پیکسل ها به ۰ تغییر پیدا می کند. اگر فرایند در اینجا متوقف شود تصویر نهایی پر از پیکسل های سیاه (نماینده rgb = [0,0,0]) خواهد بود. به همین دلیل این پیکسل ها از ماتریس تصویر برداشته می شوند. و پیکسل های باقی مانده کنار هم قرار داده می شوند.

1.2.2: Up-Sampling (Reconstruction)

در این مرحله با توجه به انتخاب اسکیل 5 برای upsample کردن تصویر پیکسل های تصویر down sample شده به اندازه ۵ پیکسل از هم فاصله می گیرند و فضای بین آن ها را پیکسل های مصنوعی (تولیدی) با interpolation (درون یابی) با متد های مختلف پر می کنند که در اینجا متد مورد استفاده nearest است که یعنی پیکسل های تولیدی همان مقدار rgb نزدیک ترین پیکسل واقعی به خودشان را میگیرند. این شیوه درون یابی (بازیابی) باعث ایجاد پیکسل هایی با rgb یکسان در فضا هایی که نزدیک ترین پیکسل واقعی آن ها مشترک باشد می شود و به همین دلیل شیب (مشتق، گرادیان) تصویر در برخی نقاط سفر است که موجب ماهش وضوح در لبه ها با جهش رنگی بالا می شود.

این موضوع خصوصا در قسمت هایی که تغییر رنگ های ناگهانی زیاد است مانند صورت دو بازیکن و لبه های کاپشن و کلاه Kobe Bryant دیده می شود (همچنین تراکم پیکسلی (resolution) تصویر ۵ برابر شده اما استفاده از همین پیکسل های مصنوعی باعث می شود انتظاری که از بهبود کیفیت تصویر تا حد کیفیت اولیه آن داریم بر آورده نشود. و این نشان می دهد image compression اگر بر

مبنای downsampling باشد، در حالت کلی حتی پس از بازیابی کیفیت اولیه را نخواهد داشت و ۸۰ درصد داده های واقعی حذف شده اند.
(این اثر در scale factor های کوچکتر طبعا کمتر دیده خواهد شد.)

1.2.3: Final Enhancement (using moving avg):

استفاده از فیلتر avg_moving که در قسمت اول شیوه عملکرد آن شرح داده شد، باعث می شود نا پیوستگی های ناگهانی ناشی از انتخاب nearest neighbor (real pixel) های متفاوت در پیکسل های مجاور کمی اصلاح شود و تغییر رنگ تدریجی تر رخ بدهد، اما sharpening لارم برای تغییر رنگ های ناگهانی هم به ناچار کمی کاهش می یابد.

۱.۲ (*Page Detection* :

شرح الگوریتم تشخیص لبه های صفحه:

در ابتدا پس از خواندن فایل page.jpg آن را به یک تصویر gray تبدیل می کنیم تا پردازش های پیش رو را انجام دهیم.

در مرحله اول با توجه به عملکر فیلتر های line_H و line_V که در قسمت اول شرح داده شد، لبه های افقی و عمودی تصویر را هایلایت (برجسته) می کنیم. تصویر خروجی این فیلتر ها بر روی فایل page.jpg به صورت زیر خواهد بود.

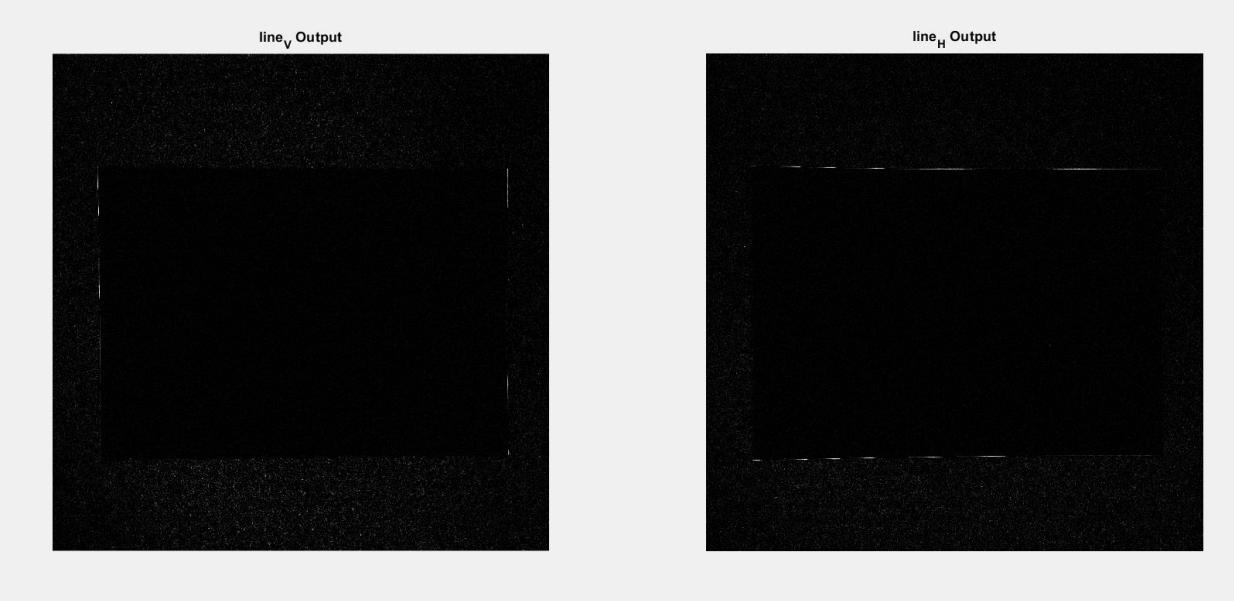

Figure 1.15: line_V and line_H output on page.jpg

در مرحله بعد روی عناصر هر یک از تصاویر خروجی بالا حمع می بندیم. روی خروجی line_H به صورت افقی (ردیفی) جمع می بندیم تا لبه هایی که سفید مانده اند هم را تقویت کنند و برای line_V هم عمودی (ستونی) جمع می بندیم و خروجی این دو جمع را در دو بردار (ماتریس) تحت عنوان vsum , hsum ذخیره می کنیم. توجه به این نکته که خروجی vsum به صورت ردیفی (ماتریس 2976 ×1) و خروجی hsum به صورت ستونی (ماتریس 1 × 2976) است ضروری است.

```
%%
% Part 3 : Page Detection

page = imread("page.jpg");
page_gray = rgb2gray(page);

% Highliting vertical and horizantal edges using line_v and line_H:

v_page = imfilter(page_gray,line_V);
h_page = imfilter(page_gray,line_H);
imwrite(v_page,"vertical.jpg");
imwrite(h_page,"horizantal.jpg");

figure
subplot(1,2,1), imshow(v_page);
title("line_V Output")
subplot(1,2,2), imshow(h_page);
title("line_H Output")

% Detecting corners for page detection:
vsum = sum(v_page,1);
hsum = sum(h_page,2);
```

Figure 1.15: First part of code for page detection

در ادامه باید گوشه های تصویر را پیدا کنیم. ایده آن است که مکان پیک های موجود در ماتریس هایvsum,hsum محل لبه هارا مشخص کند، منتها نویز های موجود در تصویر خروجی فیلتر باعث می شوند. ماکزیمم های تشخیص داده شده ب وسیله تابع maxk,2 به اشتباه نویز های اطراف بلند ترین پیک را به عنوان پیک دوم برگردانند و لبه کم نور تر به اشتباه تشخیص داده نشود. به همین دلیل باید از یک LPF برای نرم کردن تغییرات و حذف نویز استفاده کرد که انتخاب ما دوبار استفاده از تابع smooth است.

خروجی فیلتر smooth را رسم می کنیم تا پیک ها را ببینیم. برای vsum_sm پیک ها به صورت زیر است و در صورتی که دراین مرحله بخواهیم دو ماکسیمم محلی را با تابع maxk تشخیص دهیم دو عنصر اول و آخر به دلیل نویز بالا به اشتباه برگردانده خواهند شد. بنابراین عناصر موجود در index های زیر 200 و بالای 2800 را صفر می کنیم. برای hsum نیز به همین شیوه عمل می کنیم.

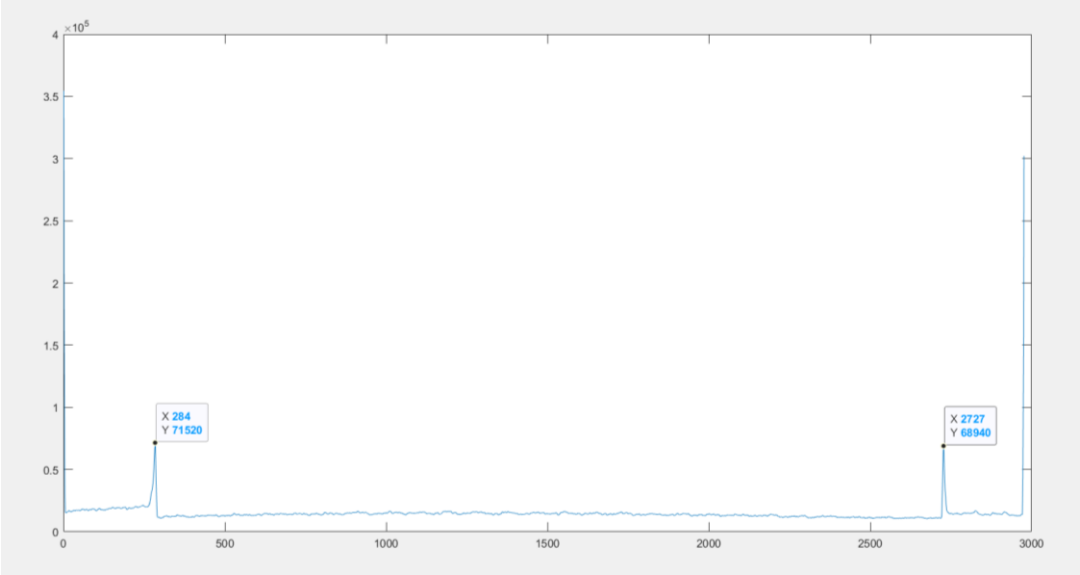

Figure 1.16: vsum_sm plot

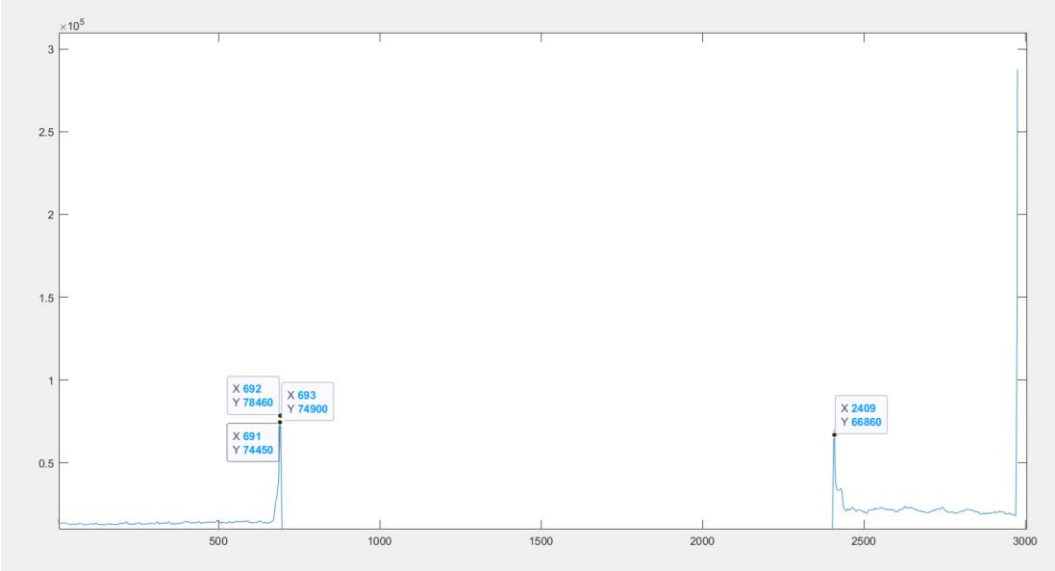

Figure 1.16: hsum_sm plot

مشاهده میکنیم که به دلیل نویز شدیدی که در خروجی hsum وجود دارد دو قله محلی دیگر نزدیک پیک اول اصلی مشخص شده اند. برای رفع این مشکل به جای دادن آرگومان 2 به تابع maxk به آن آرگومان 4 میدهیم تا 4 ماکزیمم اول ماتریس برگردانده شوند و سپس به صورت دستی ماکزیمم دوم و چهارم که لبه های اصلی هستند را نگه می دارم. البته مشخص است که دخالت طراح حین عملیات page detection در اپلیکیشن هایی مثل cam scanner ممکن نیست که از روش های دیگری برای پیدا کردن لبه های درست استفاده می شود.

4x1 double

	1	2
1	692	
2	693	
3	691	
4	2408	

```
vsum_sm = smooth(smooth(vsum));
%   figure
%   plot(vsum_sm);

for i=1:length(vsum_sm)
    if (i<200) || (i>2800)
       vsum_sm(i) = 0;
    end
end
[v_max,max_index_v] = maxk(vsum_sm,2);


hsum_sm = smooth(smooth(hsum));
%   figure
%   plot(hsum_sm);

for i=1:length(hsum_sm)
    if (i<200) || (i>2800)
       hsum_sm(i) = 0;
    end
end

% There were two noise( unreal ) peak values near the first peak
% I used maxk, 4 to find the second real peak an then

[h_max,max_index_h] = maxk(hsum_sm,4);
h_max = [h_max(2),h_max(4)];
max_index_h = [max_index_h(2),max_index_h(4)];
```

Figure 1.15: Second part of code for page detection

بدین ترتیب مختصات لبه های نهایی را در یک ماتریس تحت عنوان corners ذخیره میکنیم. و طول و عرض صفحه را (به پیکسل) در دو متغیر ذخیره می کنیم تا برای تابع rect که حین پلات دور صفحه تشخیص داده شود مستطیل قرمز fit می کند استفاده می کنیم.

Variables - corners

+2 | max_index | top

4x2 double

	1	2
1	284	693
2	2727	693
3	284	2408
4	2727	2408
5		

```
rect_length =

        2443


rect_width =

        1715
```

Figure 1.16: Corners(TL,TR,BL.BR), rectangle size

```
% Indicating the corner pixel positions:

 top_left = [max_index_v(1),max_index_h(1)];
 top_right = [max_index_v(2),max_index_h(1)];
 bottom_left = [max_index_v(1),max_index_h(2)];
 bottom_right = [max_index_v(2),max_index_h(2)];

corners = [top_left;top_right;bottom_left;bottom_right];

rect_length = max_index_v(2) - max_index_v(1)
rect_width = max_index_h(2) - max_index_h(1)

figure
imshow(page);
hold on;
rectangle('position',[top_left(1),top_left(2),rect_length,rect_width],'EdgeColor','r');
```

Figure 1.15: Last part of code for page detection

خروجی نهایی به شکل زیر است:

Figure 1.16: Page Detected.

2.Frequency domain filtering

۲_۱) تاثیر نویز گوسی بر اندازه و فاز تصویر:

Figure 2.1: Pre-processed image

در ابتدا یک تصویر را به تصادف انتخاب میکنیم که من s24\3 بود. پس از خواندن محتوای تصویر و نمایش آن که در شکل بالا مشخص است با کمک دستور mesh، رویه مروبط به تصویر فوق را رسم میکنیم. بعد سوم intensity یا همان محتوای پیکسل های ماتریس تصویر را نشان می دهدو دو بعد اول موقعیت هر پیکسل در تصویر را نشان می دهند

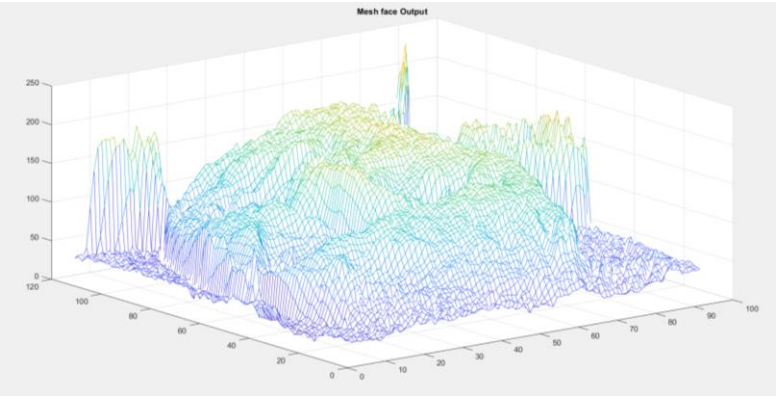

Figure 2.2: Face Mesh Output

در ادامه با استفاده از fft2 , fftshift تصویر را به حوزه فرکانس می بریم و اندازه و فاز آن را جدا می کنیم. با تابع awgn به تولید نویز سفید گوسی در snr های 0.25,0.5,1 برای اندازه و فاز می پردازیم. سپس با توجه به رابطه اندازه و فاز یک سیگنال که رد شکل زیر آمده یکبار اندازه سالم سیگنال را با فاز مخدوش همراه می کنیم و سپس بالعکس. در انتها حاصل هر دو را به حوزه تصویر می بریم تا تصاویر را رسم کنیم و تاثیر نویز را ببینیم.

$$A = z\, e^{j\theta},$$

Figure 2.3: Complex Exponential Representation of magnitude and phase

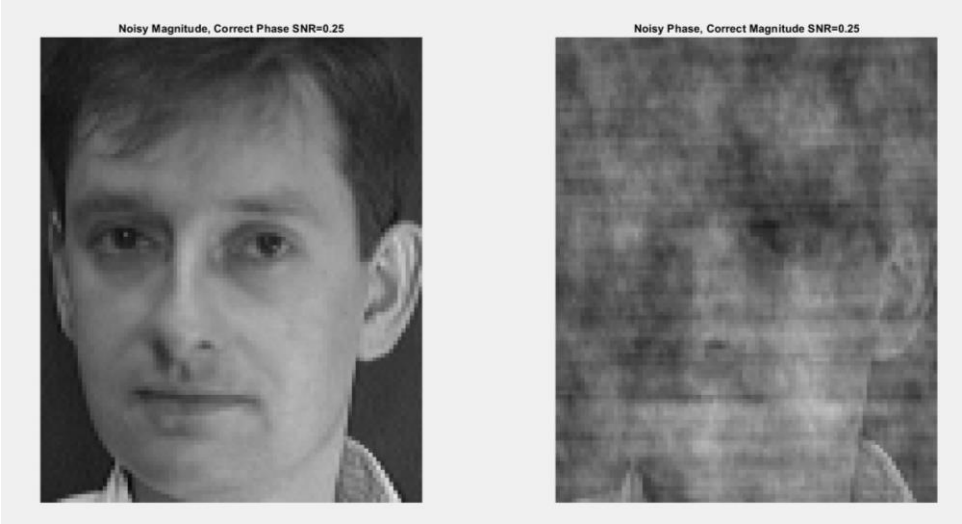

Figure 2.4: Noise Effect on Magnitude and Phase (SNR = 0.25)

Figure 2.5: Noise Effect on Magnitude and Phase (SNR = 0.5)

Figure 2.6: Noise Effect on Magnitude and Phase (SNR = 1)

با توجه به نتایج تصاویر بالا مشاهده می کنیم مخدوش کردن فاز توسط یک نویز گوسی لطمه بسیار بیشتری نسبت به حالتی که اندازه را مخدوش می کنیم، وارد می کند.

با توجه به همین نکته انتظار داریم با عوض کردن فاز تصویر دو نفر در قسمت بعد تصویر آن ها اسیب جدی ببینند.

۲_۲) تاثیر تعویض فاز تصویر صورت افراد:

برای این قسمت از صورت های s20/8 و s4/2 استفاده کرده ام.

Figure 2.7: Pre-processed face images of Person 1, Person 2

مشاهده می شود که پس از تعویض فاز تصاویر، تصویر اول که شامل اندازه تصویر person 1 و فاز تصویر person 2 است بیشتر به person 2 شبیه است و بالعکس. که همین موضوع نشان می دهد بیشتر اطلاعات تصویر این افراد در فاز آن ها قرار دارد که پس از جابجایی فاز قسمت زیادی از این اطلاعات به تصویر صورت نفر دیگر منتقل شده است. و اندازه نقش جزئی تری دارد.

Figure 2.8: Face images of Person 1, Person 2 after phase swapping

References)

1) Gaussian Blur weight calculation:

http://dev.theomader.com/gaussian-kernel-calculator/

2) Kernel Visualization:

https://setosa.io/ev/image-kernels/