به نام هستی بخش

سیگنال و سیستم ها

تمرین سوم از سری تمارین کامپیوتری

دانشکده مهندسی و علوم کامپیوتر

مدرس: دکتر سلیمی بدر

لطفا به **حتما** نکات زیر برای حل سوالات دقت کنید:

۱. در این تمرین از MATLAB و Python می توانید برای پیاده سازی استفاده کنید . برای MATLAB حق استفاده از signal processing toolkit ها ندارید. برای Python فقط از کتابخانه numpy می توانید استفاده کنید.

۲. در نهایت باید یک zip شامل کد هایتان و عکس ها در درس افزار آپلود کنید. شیوه دقیق آپلود در ادامه توضیح داده شده است.

پیاده سازی کانولوشن دو بعدی روی تصاویر

همانطور که می دانیم کانولوشن دو تابع یکی از مهمترین فرآیند هایی است که در پردازش سیگنال از آن استفاده می کنیم. در حوزه هایی مانند گرافیک کامپیوتری و پردازش تصاویر به علت ذات گسسته بودن توابع (مانند تصاویر) با فرم گسسته کانولوشن سرکار داریم. از کاربرد های کانولوشن ها در این حوزه ها می توان به حذف نویز و استخراج ویژگی های بصری برای یادگیری عمیق (هوش مصنوعی) اشاره کرد. در این تمرین قرار است برخی از این کاربرد ها را پیاده سازی کنیم.

۱. تعریف عملگر کانولوشن

همانطور که می دانیم کانولوشن رو سیگنال fو g به صورت زیر تعریف می شود:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau.$$

عبارت معمولا $(f * g)(t)$ به عنوان مقدار فیلتر شده سیگنال f تحت کرنل g نامیده می شود. یکی از قضایای که می توان به آن اشاره کرد عبارت است از:

$$F\{f * g\} = kF\{f\}F\{g\}$$

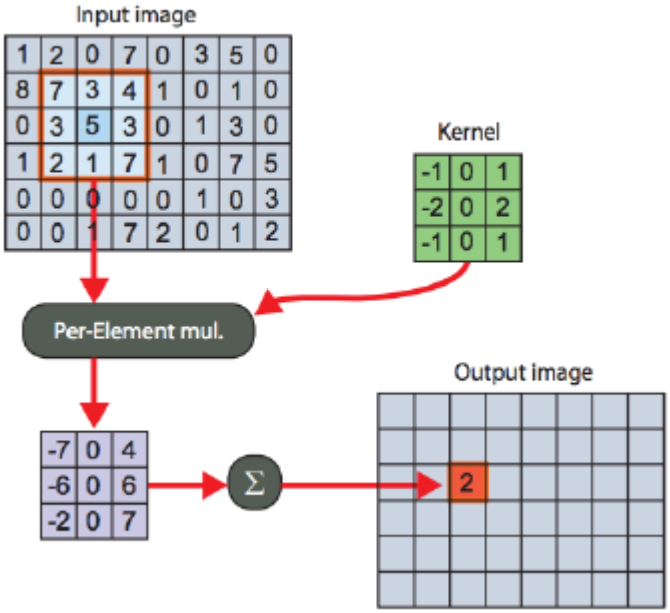
که F را تبدیل فوریه سیگنال مورد نظر می نامیم. در پردازش تصویر دیجیتال از این ویژگی برای برای شارپ(sharp) و بلور(blure) کردن تصاویر استفاده می کنند.

اگر یک تصویر به صورت یک سیگنال گسسته دو بعدی $y[,]$ نمایش دهیم آنگاه کانولوشن گسسته دو بعدی را به کمک کرنل $k[,]$ به صورت زیر تعریف می کنیم:

$$(y * k)[i, j] = \sum_n \sum_m y[i - n, j - m]k[n, m].$$

برای بهتر متوجه شدن نحوه ای محاسبه این رابطه فرض کنید یک عکس ورودی و یک کرنل ۳X۳ داریم. کرنل را در ابتدا روی بالاترین مکان ممکن روی عکس قرار می دهیم. درایه های کرنل را در نظیر به نظیر درایه های عکس ورودی ضرب می کنیم و حاصلضرب ها را با هم جمع می کنیم. حاصل جمع را در بالاترین درایه ماتریس خروجی قرار می دهیم. در مرحله بعد کرنل را یک واحد به راست برده و همین روال را تکرار می کنیم. شکل زیر یک اسنپ شات از زمانی است که کرنل روی محل مشخص شده قرار گرفته است.

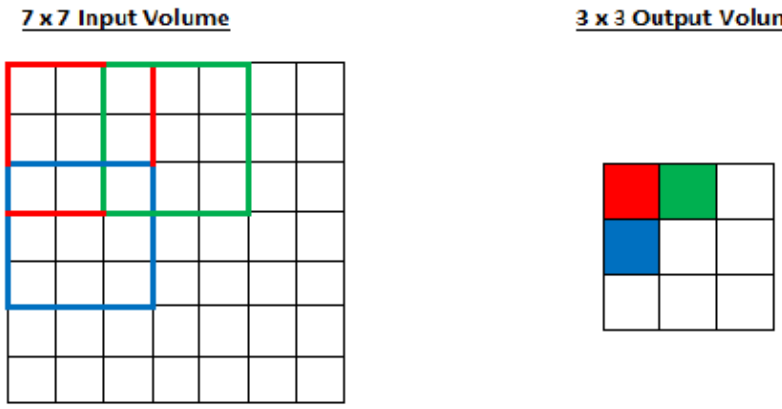
توضیحات همراه با gif را میتوانید از اینجا ببینید.



۲. پارامتر های فیلتر(کرنل) ها:

الف) **kernel size** : اینکه کرنل چند در چند باشد. به طور مثال در شکل صفحه قبل ۳ بود.

ب) **stride** : اینکه گام هایی که برای انتقال به مکان بعدی برمیداریم را **stride** می نامند. در مثال صفحه قبل **stride** برابر یک و در شکل پایین برابر ۲ است:



۳. انواع فیلترینگ:

همانطور که مشخص است هرچه سایز کرنل بزرگتر باشد سایز عکس خروجی کوچکتر است:

input image	*	filter	=	output image
(n x n)		(f x f)		(n-f-1)x(n-f-1)

در حوزه پردازش تصویر در یادگیری عمیق این موضوع یک مشکل جدی است. به دو دلیل:

الف) کوچک شدن سایز ورودی شبکه های عصبی به مرور زمان. (اگر علاقه داشتید که بدانید این چرا مشکله می تونید به بنده پیام دهید بپرسید)

ب) حذف اطلاعات از گوشه های تصویر

برای اینکه این مشکل را بر طرف کنند از تکنیکی به اسم zero padding استفاده می کنند. به این صورت که قبل از عمل کانوولوشن به دور تا دور ماتریس عکس ورودی مقادیر صفر اضافه می کنیم که در نهایت سایز خروجی برابر سایز عکس واقعی شود:

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

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

114	328	-26	470	158
53	266	-61	-30	344

برای همین می توان رابطه زیر را نوشت: (دقت شود که سایز کرنل را اکثر مواقع مقدار فردی در نظر می گیرند.)

$$p = (f - 1) / 2$$

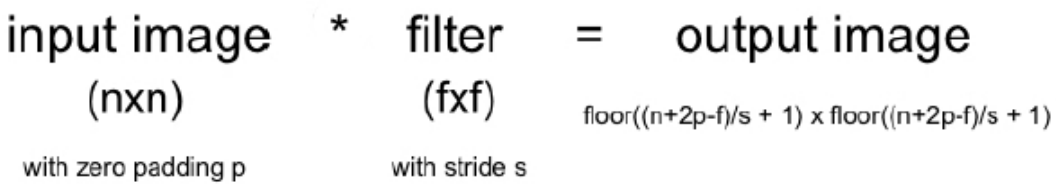
p : zero padding size
k : kernel size

اگر zero padding داشته باشیم انگاه نوع فیلترینگ از نوع **same** است و اگر نداشته باشیم (حالت معمولی) **valid** است.

پس برای فیلترینگ same داریم

input image	*	filter	=	output image
(n+2p)x(n+2p)		(f x f)		(n+2p-f+1)x(n+2p-f+1)

اگر stride بزرگتر از یک باشد آنگاه می توان فرم کلی را به صورت زیر نوشت:



۳. فیلتر های معروف:

هر ماتریس kxk ای می تواند یک فیلتر باشد. اما در اینجا به چند تا از مهمترین و معروف ترین فیلتر ها که کاربرد های متنوعی دارند اشاره می کنیم:

الف) Sharpness filter:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

ب) Horizontal Edge Detector:

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

ج) Embossing filter:

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

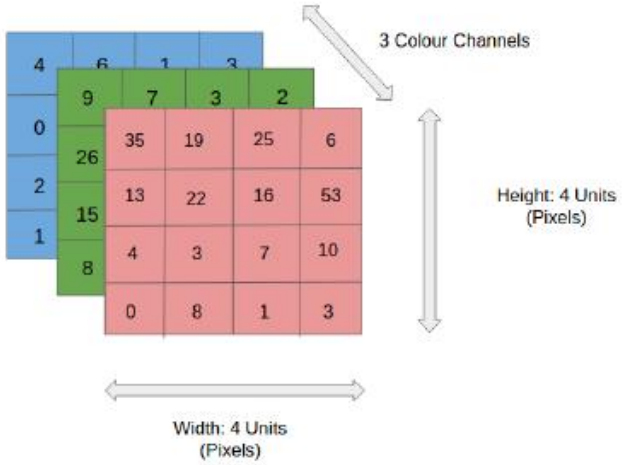
د) Gaussian filter:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

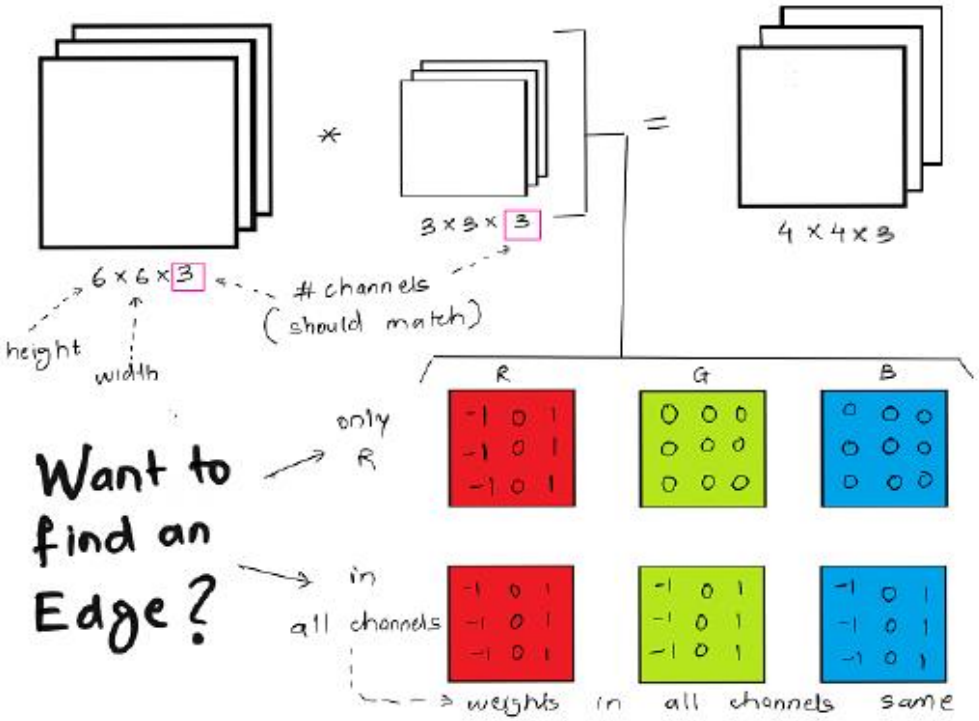
Figure 2: *Even small image convolution kernels can be powerful image processing operators. (A): The original image. (B): Sharpening filter. (C): Edge detection filter. (D): Embossing filter.*

۴. وجود چالش در عکس های رنگی نسبت به سیاه سفید:

عکس های سیاه سفید مقادیر درایه هایشان اعداد صحیحی بین ۰ تا ۲۵۵ است(۰ مشکی و ۲۵۵ سفید). تفاوت عکس های رنگی با سیاه سفید در این است که عکس های سیاه سفید تک کاناله هستند ولی عکس های رنگی ۳ کاناله(RGB) برای همین به طور مثال اگر عکسی ۱۲۸ در ۱۲۸ باشد آنگاه ابعاد ماتریس ورودی ما به صورت (128x128x3) خواهد بود.



برای عکس های رنگی می بایست فرآیند کانولوشن را برای هر ۳ کانال به صورت جداگانه انجام داد. و ماتریس های خروجی را روی هم قرار داد:



تمرین

در پوشه پروژه یک فولدر حاوی عکس های ورودی مشاهده می کنید.(input_images).

تابعی با پروتوتایپ زیر بنویسید که آدرس عکس ورودی و ماتریس کرنل و نوع فیلترینگ را بگیرد و عکس خروجی را با فرمت زیر در فولدر output_images ذخیره کند:

filename_filtername_valid/same_stride.jpg

Example : lecun_gaussian_same_2.jpg

```python
def conv2D(src="input_images/lecun.jpg", stride=2, is_same=True, filter=np.array([[0, -1, 0]
                                                                                  [-1, 5, -1]
                                                                                  [0, -1, 0]])):
```

```
function conv2D(img_path, stride, is_same, filter)
%CONV2D Summary of this function goes here
%   Detailed explanation goes here
```

در فولدر خروجی **به ازای هر عکس ورودی باید ۴ تا عکس با کرنل های زیر** جنریت کنید.

a) Sharpness filter (valid/stride = 1)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

b) Horizontal Edge filter (same/stride = 2)

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

c) Embossing filter (valid/stride = 3)

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

d) Gaussian filter (same/stride = 1)

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

دقت شود که ملاک نمره گیری شما درست بودن کد است(یعنی ما با فیلتر های دیگر کد شما را امتحان می کنیم). **پس اگر تابعی که نوشتید به هر دلیلی اجرا نشود و یا خروجی مطلوب را ندهد شما نمره آن قسمت را نخواهید گرفت.**

موفق باشید.