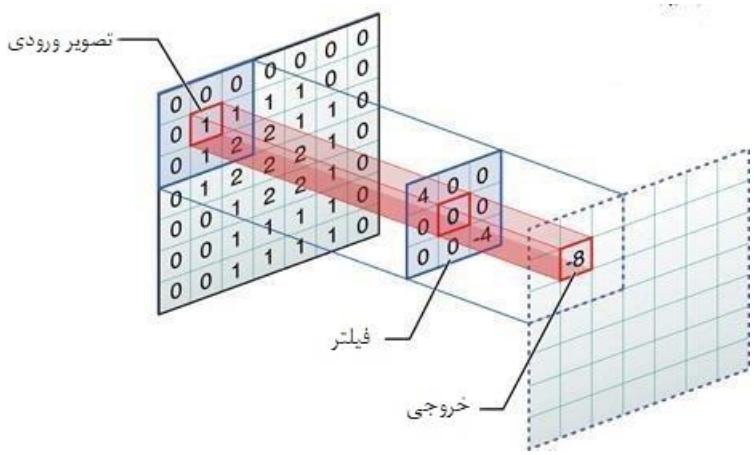
پروژه اول – شبکه عصبی کانولوشنی گام به گام

در این پروژه تصمیم داریم شبکه عصبی کانولوشنی را از اولین لایه به صورت مرحله به مرحله بسازیم و به جزییات آن بپردازیم.

پیاده سازی لایه کانولوشنی

شبکه عصبی کانولوشنی برای تشخیص اینکه چه ویژگی هایی مانند لبه ها در سراسر تصویر وجود دارد، از فیلترها (همچنین به عنوان هسته یا kernel نیز شناخته می شود) استفاده می کند. فیلترها ماتریسی از مقادیر هستند که این مقادیر وزن نامیده می شود و برای شناسایی ویژگی های خاص آموزش داده می شوند. فیلتر بر روی هر قسمت از تصویر حرکت می کند تا بررسی کند آیا ویژگی مورد نظر برای تشخیص وجود دارد یا خیر. برای محاسبه مقداری که نشان دهنده میزان اطمینان از وجود ویژگی خاص باشد ، فیلتر یک عملیات کانولوشن را انجام می دهد که یک ترکیبی از ضرب عناصر در هم و جمع ماتریس هاست.



وقتی این ویژگی در بخشی از تصویر وجود داشته باشد ، عمل کانولوشن بین فیلتر و آن قسمت از تصویر منجر به یک عدد بزرگ می شود. اگر آن ویژگی موجود نباشد ، مقدار عدد بسیار کم است. در مثال زیر ، فیلتری که وظیفه بررسی منحنی های سمت راست را دارد از بخشی از تصویر عبور می کند. از آنجا که آن قسمت از تصویر حاوی همان منحنی است که فیلتر به دنبال آن است ، نتیجه عملیات کانولوشن عدد زیادی است (۶۶۰۰).

تصویر

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

ماتریس متناظر تصویر

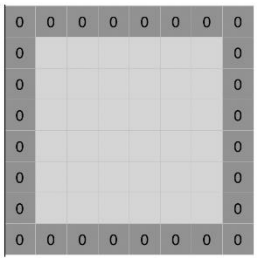
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

فیلتر

نحوه محاسبه کانولوشن دو ماتریس = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600

از مشکلات محاسبه کانولوشن این است که به هنگام حرکت و ضرب فیلتر کانولوشن روی تصویر، گوشه های تصویر یک بار در عمل کانولوشن دخالت میکنند در حالیکه عناصر وسط ماتریس بارها در محاسبه کانولوشن مورد توجه قرار میگیرند. برای حل این مشکل از Padding استفاده میشود، در واقع همانند شکل زیر ردیفی از عناصر صفر دور تا دور تصویر قرار میگیرند تا همه پیکسل های تصویر به صورت یکسان در محاسبه کانولوشن ها شرکت کنند:



تابع Zero_pad را پیاده سازی کنید. از تابع np.pad استفاده کنید.

برای محاسبه کانولوشن در ابتدا کانولوشن یک مرحله ای را پیاده سازی میکنیم، یعنی محاسبه کانولوشن به ازای اعمال شدن آن به یک نقطه از تصویر. با استفاده از ضرب و جمع تابع conv_single_step را پیاده سازی کنید. مقدار b یک مقدار از نوع float و نشان دهنده بایاس است.

یک فیلتر می تواند با استفاده از پارامتر stride، روی تصویر ورودی کشیده شود. ابعاد خروجی با در نظر گرفتن ابعاد ورودی و اندازه فیلتر (f) و مقادیر pad و stride به صورت زیر محاسبه می شود:

$$n_H = \left\lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \right\rfloor + 1$$
$$n_W = \left\lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \right\rfloor + 1$$

مقدار nc برابر با تعداد فیلترهای استفاده شده در کانولوشن است.

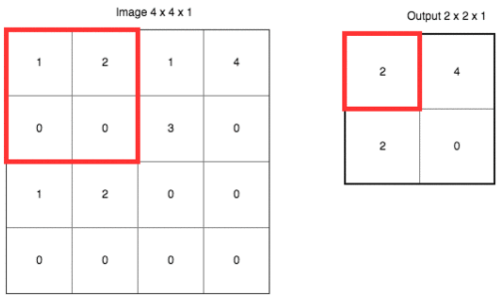
با استفاده از Numpy ، به راحتی می توانیم عمل کانولوشن را پیاده سازی کنیم. در هر مرحله از کانولوشن، فیلتر با عملگر ضرب عناصر (*) با بخشی از تصویر ورودی ضرب می شود. در نهایت نتیجه این ضرب با یک مقدار بایاس جمع میشود.

کد تابع کانولوشن را با استفاده از Numpy و به کمک حلقه های for با در نظر گرفتن فرمول بالا پیاده سازی کنید.

پیاده سازی لایه Pooling

بعد از یک یا دولایه کانولوشنی لایه ای برای کاهش ابعاد مورد استفاده قرار میگیرد که به آن لایه Pooling میگویند.

برای سرعت بخشیدن به روند آموزش و کاهش میزان حافظه مصرفی توسط شبکه ، سعی می کنیم افزونگی موجود در ویژگی ورودی را کاهش دهیم. در زیر نمونه ای از لایه MaxPooling را با فیلتری به ابعاد ۲*۲ مشاهده می کنید.



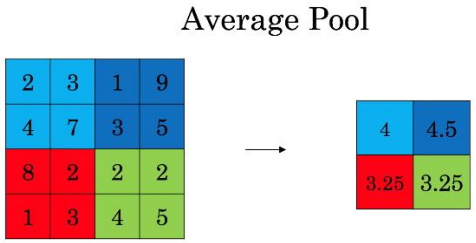
همانند قبل اندازه ماتریس خروجی پس از لایه Pooling با رابطه زیر محاسبه می شود:

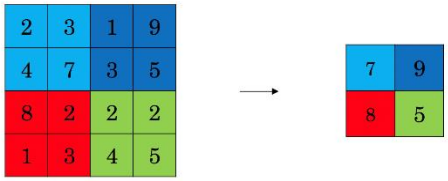
$$n_H = \left\lfloor \frac{n_{H_{prev}} - f}{stride} \right\rfloor + 1$$

$$n_W = \left\lfloor \frac{n_{W_{prev}} - f}{stride} \right\rfloor + 1$$

$$n_C = n_{C_{prev}}$$

لایه Pooling را با روش MaxPooling و AveragePooling پیاده سازی کنید.





در فریم ورک های یادگیری عمیق مدرن ، شما فقط باید حرکت رو به جلو را پیاده سازی کنید ، و فریم ورک حرکت رو به عقب را به عهده میگیرد، بنابراین اکثر مهندسین یادگیری عمیق نیازی به یادگیری جزئیات حرکت رو به عقب را ندارند. حرکت رو به عقب برای شبکه های کانولوشنی پیچیده است. در صورت تمایل می توانید در مورد جزییات آن مطالعه کنید.