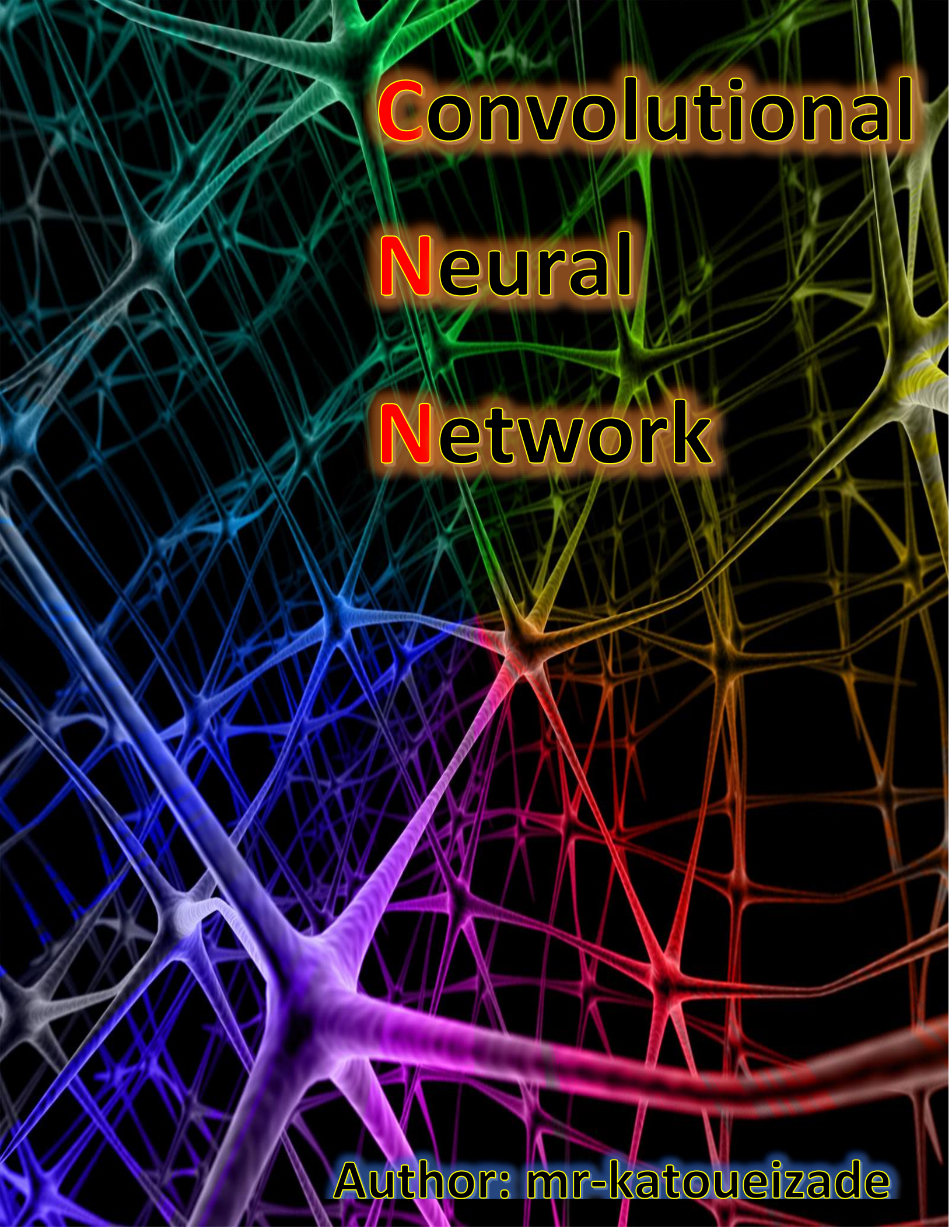

Convolutional
Neural
Network
Author: mr-katoueizade


شبکه های عصبی پیچشی
گردآوری: م. کتویی زاده

مقدمه

در این کتاب بصورت کاملا تخصصی در مورد شبکه های عصبی پیچشی (کانولوشن) با زبان ساده و روان توضیح می دهیم. پیش از شروع توجه شما را به نکات زیر جلب می کنم.

منابع کتاب به صورت کامل در انتها آورده شده است.

برای فهم بهتر موضوع در ابتدا شبکه های عصبی مصنوعی را توضیح می دهیم.

در قسمت توضیحات شبکه های عصبی کانولوشن، برای فهم بهتر مبحث مثال های واقعی و عینی و همچنین تمامی فرمول ها و راه حل ها با ارائه نمونه واقعی آورده شده است، همچنین در برخی قسمت ها به زبان غیر رسمی برای آسان تر شدن مفاهیم صحبت شده است.

تیترهای بارنگ قرمز تیترهای اصلی و تیترهای با رنگ بنفش زیرموضوع یا مفاهیم فرعی و یا نکات می باشند.

از لطف و اهتمام شما در انتشار این اثر و سایر آثار مولف کمال سپاسگزاری را دارم.

محمدرضا کتویی زاده

شبکه عصبی مصنوعی

شبکه‌های عصبی مصنوعی (Artificial Neural Networks - ANN) یا به زبان ساده‌تر شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوین برای یادگیری ماشینی، نمایش دانش و در انتها اعمال دانش به دست آمده در جهت بیش‌بینی پاسخ‌های خروجی از سامانه‌های پیچیده هستند. ایده اصلی این گونه شبکه‌ها تا حدودی الهام‌گرفته از شیوه کارکرد سیستم عصبی زیستی برای پردازش داده‌ها و اطلاعات به منظور یادگیری و ایجاد دانش می‌باشد. عنصر کلیدی این ایده، ایجاد ساختارهایی جدید برای سامانه پردازش اطلاعات است.

این سیستم از شمار زیادی عناصر پردازشی فوق‌العاده بهم‌پیوسته با نام نورون تشکیل شده که برای حل یک مسئله با هم هماهنگ عمل می‌کنند و توسط سیناپس‌ها (ارتباطات الکترومغناطیسی) اطلاعات را منتقل می‌کنند. در این شبکه‌ها اگر یک سلول آسیب ببیند بقیه سلول‌ها می‌توانند نبود آن را جبران کرده، و نیز در بازسازی آن سهیم باشند. این شبکه‌ها قادر به یادگیری‌اند؛ مثلاً با اعمال سوزش به سلول‌های عصبی لامسه، سلول‌ها یادمی‌گیرند که به طرف جسم داغ نروند و با این الگوریتم سیستم می‌آموزد که خطای خود را اصلاح کند. یادگیری در این سیستم‌ها به صورت تطبیقی صورت می‌گیرد، یعنی با استفاده از مثال‌ها وزن سیناپس‌ها به گونه‌ای تغییر می‌کند که در صورت دادن ورودی‌های جدید، سیستم پاسخ درستی تولید کند.

زمینه

فلسفه اصلی شبکه عصبی مصنوعی، مدل کردن ویژگی‌های پردازشی مغز انسان برای تقریب زدن روش‌های معمول محاسباتی با روش پردازش زیستی است. به بیان دیگر، شبکه عصبی مصنوعی روشی است که دانش ارتباط بین چند مجموعه داده را از طریق آموزش فراگرفته و برای استفاده در موارد مشابه ذخیره می‌کند. این پردازنده از دو جهت مشابه مغز انسان عمل می‌کند:

- یادگیری شبکه عصبی از طریق آموزش صورت می‌گیرد.
- وزن‌دهی مشابه با سیستم ذخیره‌سازی اطلاعات، در شبکه عصبی مغز انسان انجام می‌گیرد.

تعریف

یک شبکه عصبی مصنوعی، از سه لایه ورودی، خروجی و پردازش تشکیل می‌شود. هر لایه شامل گروهی از سلول‌های عصبی (نورون) است که عموماً با کلیه نورون‌های لایه‌های دیگر در ارتباط

هستند، مگر این که کاربر ارتباط بین نورون‌ها را محدود کند؛ ولی نورون‌های هر لایه با سایر نورون‌های همان لایه، ارتباطی ندارند.

نورون کوچک‌ترین واحد پردازشگر اطلاعات است که اساس عملکرد شبکه‌های عصبی را تشکیل می‌دهد. یک شبکه عصبی مجموعه‌ای از نورون‌هاست که با قرار گرفتن در لایه‌های مختلف، معماری خاصی را بر مبنای ارتباطات بین نورون‌ها در لایه‌های مختلف تشکیل می‌دهند. نورون می‌تواند یک تابع ریاضی غیرخطی باشد، در نتیجه یک شبکه عصبی که از اجتماع این نورون‌ها تشکیل می‌شود، نیز می‌تواند یک سامانه کاملاً پیچیده و غیرخطی باشد. در شبکه عصبی هر نورون به‌طور مستقل عمل می‌کند و رفتار کلی شبکه، برآیند رفتار نورون‌های متعدد است. به عبارت دیگر، نورون‌ها در یک روند همکاری، یکدیگر را تصحیح می‌کنند.

کاربرد

با استفاده از دانش برنامه‌نویسی رایانه می‌توان ساختار داده‌ای طراحی کرد که همانند یک نورون عمل نماید. سپس با ایجاد شبکه‌ای از این نورون‌های مصنوعی به هم پیوسته، ایجاد یک الگوریتم آموزشی برای شبکه و اعمال این الگوریتم به شبکه آن را آموزش داد.

این شبکه‌ها برای تخمین و تقریب، کارایی بسیار بالایی از خود نشان داده‌اند. گستره کاربرد این مدل‌های ریاضی بر گرفته از عملکرد مغز انسان، بسیار وسیع می‌باشد که به عنوان چند نمونه کوچک می‌توان استفاده از این ابزار ریاضی در پردازش سیگنال‌های بیولوژیکی، مخابراتی و الکترونیکی تا کمک در نجوم و فضانوردی را نام برد.

اگر یک شبکه را هم‌ارز با یک گراف بدانیم، فرایند آموزش شبکه تعیین نمودن وزن هر یال و base اولیه خواهد بود.

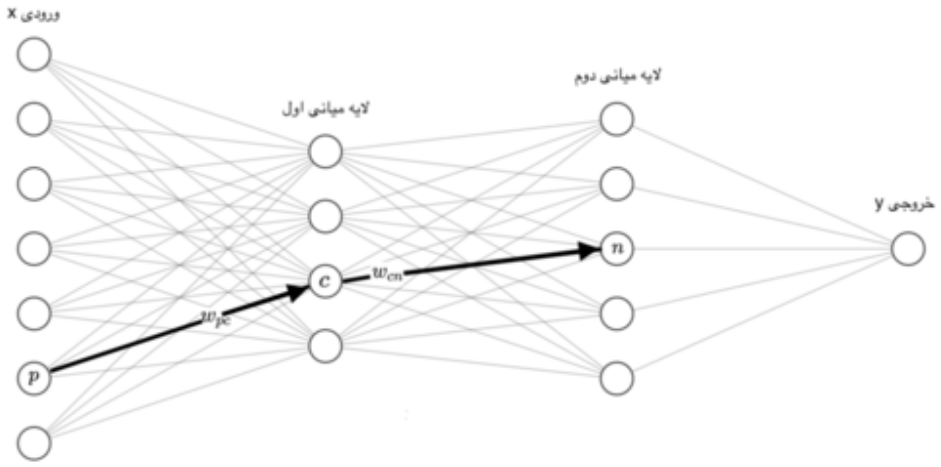
یادگیری

یادگیری ماشینی با نظارت (supervised learning) به دنبال تابعی از میان یک سری توابع هست که تابع هزینه (loss function) داده‌ها را بهینه سازد. به عنوان مثال در مسئله رگرسیون تابع هزینه می‌تواند اختلاف بین پیش‌بینی و مقدار واقعی خروجی به توان دو باشد، یا در مسئله طبقه‌بندی ضرر منفی لگاریتم احتمال خروجی باشد. مشکلی که در یادگیری شبکه‌های عصبی وجود دارد این است که این مسئله بهینه‌سازی دیگر محدب (convex) نیست. ازین رو با مشکل کمینه‌های محلی روبرو هستیم. یکی از روش‌های متداول حل مسئله بهینه‌سازی در شبکه‌های

عصبی بازگشت به عقب یا همان back propagation است. روش بازگشت به عقب گرادیانِ تابع هزینه را برای تمام وزن‌های شبکه عصبی محاسبه می‌کند و بعد از روش‌های گرادیان کاهشی (gradient descent) برای پیدا کردن مجموعه وزن‌های بهینه استفاده می‌کند. روش‌های گرادیان کاهشی سعی می‌کنند به صورت متناوب در خلاف جهت گرادیان حرکت کنند و با این کار تابع هزینه را به حداقل برسانند. پیدا کردن گرادیانِ لایه آخر ساده است و با استفاده از مشتق جزئی بدست می‌آید. گرادیانِ لایه‌های میانی اما به صورت مستقیم بدست نمی‌آید و باید از روش‌هایی مانند قاعده زنجیری در مشتق‌گیری استفاده کرد .روش بازگشت به عقب از قاعده زنجیری برای محاسبه گرادیان‌ها استفاده می‌کند و همان‌طور که در پایین خواهیم دید، این روش به صورت متناوب گرادیان‌ها را از بالاترین لایه شروع کرده آن‌ها را در لایه‌های پایینتر «پخش» می‌کند.

بازگشت به عقب(Backpropagation) ، روشی برای محاسبه گرادیانها

برای سلول عصبی Cورودیی که از سلول عصبیp به این سلول وارد می‌شود را با b_{pc} نشان می‌دهیم. وزن این ورودی W_{pc} است و مجموع ضرب ورودی‌ها با وزنهایشان را با a_c نمایش می‌دهیم. که در ادامه بیشتر با آن آشنا می شویم.



تصویری از یک شبکه عصبی با دو لایه پنهان، گرادیان c وابسته به گرادیان لایه‌های بالاتر است که به آنها متصل است.

تاریخچه شبکه های عصبی مصنوعی

از قرن نوزدهم به‌طور همزمان اما جداگانه، از سویی نوروفیزیولوژیست‌ها سعی کردند سیستم یادگیری و تجزیه و تحلیل مغز را کشف کنند، و از سوی دیگر ریاضیدانان تلاش کردند مدل ریاضی

ای بسازند که قابلیت فراگیری و تجزیه و تحلیل عمومی مسائل را دارا باشد. اولین کوشش‌ها در شبیه‌سازی با استفاده از یک مدل منطقی در اوایل دهه ۱۹۴۰ توسط وارن مک‌کالک و والتر پیتز انجام شد که امروزه بلوک اصلی سازنده اکثر شبکه‌های عصبی مصنوعی است. عملکرد این مدل مبتنی بر جمع ورودی‌ها و ایجاد خروجی با استفاده از شبکه‌ای از نورون‌ها است. اگر حاصل جمع ورودی‌ها از مقدار آستانه بیشتر باشد، اصطلاحاً نورون برانگیخته می‌شود. نتیجه این مدل اجرای ترکیبی از توابع منطقی بود.

در سال ۱۹۴۹ دونالد هب قانون یادگیری را برای شبکه‌های عصبی طراحی کرد. در سال ۱۹۵۸ شبکه پرسپترون توسط روزنبلات معرفی گردید. این شبکه نظیر واحدهای مدل شده قبلی بود. پرسپترون دارای سه لایه است که شامل لایه ورودی، لایه خروجی و لایه میانی می‌شود. این سیستم می‌تواند یاد بگیرد که با روشی تکرارشونده وزن‌ها را به گونه‌ای تنظیم کند که شبکه توان بازتولید جفت‌های ورودی و خروجی را داشته‌باشد. روش دیگر، مدل خطی تطبیقی نورون است که در سال ۱۹۶۰ توسط برنارد ویدرو و مارسیان هاف در دانشگاه استنفورد (به وجود آمد که اولین شبکه‌های عصبی به کار گرفته شده در مسائل واقعی بودند. آدالاین یک دستگاه الکترونیکی بود که از اجزای ساده‌ای تشکیل شده بود، روشی که برای آموزش استفاده می‌شد با پرسپترون فرق داشت.

در سال ۱۹۶۹ میسکی و پاپرت کتابی نوشتند که محدودیت‌های سیستم‌های تک لایه و چند لایه پرسپترون را تشریح کردند. نتیجه این کتاب پیش داوری و قطع سرمایه‌گذاری برای تحقیقات در زمینه شبیه‌سازی شبکه‌های عصبی بود. آن‌ها با طرح اینکه طرح پرسپترون قادر به حل هیچ مسئله جالبی نمی‌باشد، تحقیقات در این زمینه را برای مدت چندین سال متوقف کردند.

با وجود این‌که اشتیاق عمومی و سرمایه‌گذاری‌های موجود به حداقل خود رسیده بود، برخی محققان تحقیقات خود را برای ساخت ماشین‌هایی که توانایی حل مسائلی از قبیل تشخیص الگو را داشته باشند، ادامه دادند. از جمله گراسبگ که شبکه‌ای تحت عنوان Avalanch را برای تشخیص صحبت پیوسته و کنترل دست ربات مطرح کرد. همچنین او با همکاری کارپنتر شبکه‌های نظریه تشدید انطباقی را بنا نهادند که با مدل‌های طبیعی تفاوت داشت. اندرسون و کوهونن نیز از اشخاصی بودند که تکنیک‌هایی برای یادگیری ایجاد کردند. ورباس در سال ۱۹۷۴ شیوه آموزش پس انتشار خطا را ایجاد کرد که یک شبکه پرسپترون چندلایه البته با قوانین نیرومندتر آموزشی بود.

پیشرفت‌هایی که در سال ۱۹۷۰ تا ۱۹۸۰ به‌دست آمد، برای جلب توجه به شبکه‌های عصبی بسیار مهم بود. برخی فاکتورها نیز در تشدید این مسئله دخالت داشتند، از جمله کتاب‌ها و کنفرانس‌های

وسیعی که برای مردم در رشته‌های متنوع ارائه شد. امروز نیز تحولات زیادی در تکنولوژی ANN ایجاد شده‌است.

شبکه عصبی پیچشی

شبکه‌های عصبی پیچشی یا همگشتی به انگلیسی convolutional neural network :

مخفف *CNN* : یا *ConvNet* رده‌ای از شبکه‌های عصبی ژرف هستند که معمولاً برای انجام تحلیل‌های تصویری یا گفتاری در یادگیری ماشین استفاده می‌شوند.

شبکه‌های عصبی پیچشی به منظور کمینه کردن پیش‌پردازش‌ها از گونه‌ای از پرسپترون‌های چندلایه استفاده می‌کنند. به جای شبکه عصبی پیچشی گاهی از این شبکه‌ها با نام شبکه‌های عصبی تغییرناپذیر با انتقال (shift invariant) یا تغییرناپذیر با فضا (space invariant) هم یاد می‌شود. این نام‌گذاری بر مبنای ساختار این شبکه است که در ادامه به آن اشاره خواهیم کرد .

ساختار شبکه‌های پیچشی از فرایندهای زیستی قشر بینایی گربه الهام گرفته‌شده‌است. این ساختار به گونه‌ای است که تک‌نورون‌ها تنها در یک ناحیه محدود به تحریک پاسخ می‌دهند که به آن ناحیه پذیرش گفته می‌شود. نواحی پذیرش نورون‌های مختلف به صورت جزئی با هم همپوشانی دارند به گونه ای که کل میدان دید را پوشش می‌دهند.

شبکه‌های عصبی پیچشی نسبت به بقیه رویکردهای دسته‌بندی تصاویر به میزان کمتری از پیش‌پردازش استفاده می‌کنند. این امر به معنی آن است که شبکه معیارهایی را فرامی‌گیرد که در رویکردهای قبلی به صورت دستی فراگرفته می‌شدند. این استقلال از دانش پیشین و دستکاری‌های انسانی در شبکه‌های عصبی پیچشی یک مزیت اساسی است.

تاکنون کاربردهای مختلفی برای شبکه‌های عصبی از جمله در بینایی کامپیوتر، سیستم‌های پیشنهاددهنده و پردازش زبان طبیعی پیشنهاد شده‌اند .

طراحی

یک شبکه عصبی پیچشی از یک لایه ورودی، یک لایه خروجی و تعدادی لایه پنهان تشکیل شده‌است. لایه‌های پنهان یا پیچشی هستند، یا تجمعی یا کامل.

لایه‌های پیچشی

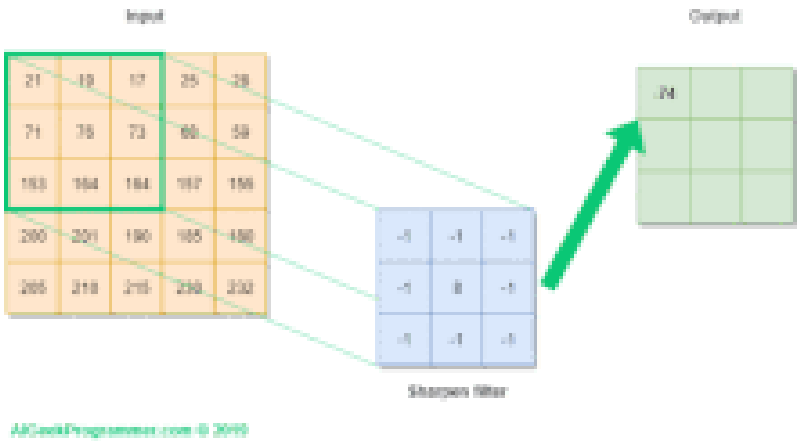
لایه‌های پیچشی یک عمل پیچش را روی ورودی اعمال می‌کنند، سپس نتیجه را به لایه بعدی می‌دهند. این پیچش در واقع پاسخ یک تک‌نورون را به یک تحریک دیداری شبیه‌سازی می‌کند.

هر نورون پیچشی داده‌ها را تنها برای ناحیه پذیرش خودش پردازش می‌کند. مشبک‌کردن به شبکه‌های پیچشی این اجازه را می‌دهد که انتقال، دوران یا اعوجاج ورودی را تصحیح کنند.

اگرچه شبکه‌های عصبی پیش‌خور کاملاً همبند می‌توانند برای یادگیری ویژگی‌ها و طبقه‌بندی داده به کار روند، این معماری در کاربرد برای تصاویر به کار نمی‌رود. در این حالت حتی برای یک شبکه کم‌عمق تعداد بسیار زیادی نورون لازم است. عمل پیچش یک راه‌حل برای این شرایط است که تعداد پارامترهای آزاد را به عمیق‌تر کردن شبکه کاهش می‌دهد.

لایه‌های پیچشی نسبت به شبکه‌های عصبی اولیه تفاوت‌های زیادی دارند که از آن‌ها می‌توان به موارد زیر اشاره کرد:

- تعامل پراکنده (sparse interaction): در این لایه به جای اینکه به ازای هریک از ورودی‌های لایه یک‌وزن تعریف شده باشد و هر ورودی با تمام خروجی‌های لایه در ارتباط باشد، یک هسته (Kernel) مشخص می‌شود که ماتریسی است که ابعاد آن نسبت به ورودی بسیار کوچک است. برای مثال برای پردازش تصویری که میلیون‌ها پیکسل دارد، یک ماتریس هسته با اندازه‌ای در حدود صد تا هزار پیکسل در نظر گرفته می‌شود که به کمک آن می‌توان تمام لبه‌هایی که در تصویر موجود است را مشخص کرد. این به این معناست که اطلاعات کمتری نسبت به ورودی نیاز داریم تا شبکه را آموزش دهیم.
- اشتراک‌گذاری پارامترها (Parameter Sharing): در یک شبکه عصبی هر یک از درایه‌های ماتریس وزن دقیقاً یکبار در محاسبه خروجی لایه استفاده می‌شود. اما در لایه‌های پیچشی، هریک از درایه‌های ماتریس هسته درتمام موقعیت‌های ورودی استفاده می‌شود. درواقع هربار این ماتریس هسته روی ماتریس ورودی می‌لغزد و تمامی درایه‌های هسته در آن بخش از ورودی ضرب می‌شود. اینکار باعث می‌شود برای تمام لایه تنها نیاز به یادگیری یک ماتریس هسته باشد که در کاهش حجم مورد نیاز کمک می‌کند.



نحوه لغزیدن ماتریس هسته روی ورودی و به دست آوردن ماتریس خروجی

لایه‌های ادغام

لایه‌های ادغام (pooling layer): **شبکه‌های عصبی پیچشی ممکن است شامل لایه‌های ادغام محلی یا سراسری باشند که خروجی‌های خوشه‌های نورونی در یک لایه را در یک تک‌نورون در لایه بعدی ترکیب می‌کند. به عنوان مثال روش حداکثر تجمع** (max pooling) **حداکثر مقدار بین خوشه‌های نورونی در لایه پیشین استفاده می‌کند. مثال دیگر میانگین تجمع** (average pooling) **است که از مقدار میانگین خوشه‌های نورونی در لایه پیشین استفاده می‌کند.**

کاملاً همبند

لایه‌های کاملاً همبند، هر نورون در یک لایه را به هر نورون در لایه دیگر متصل می‌کنند. این رویکرد در اصل مشابه کاری است که در شبکه عصبی پرسپترون چند لایه (MLP) **انجام می‌شود.**

وزن‌ها

شبکه‌های عصبی پیچشی وزن‌ها را در لایه‌های پیچشی به اشتراک می‌گذارند که باعث می‌شود حداقل حافظه و بیشترین کارایی به‌دست بیاید.

شبکه های عصبی متاخر

برخی شبکه‌های عصبی متأخر از معماری مشابهی استفاده می‌کنند، مخصوصاً آن‌هایی که برای تشخیص تصویر یا طبقه‌بندی استفاده می‌شوند.

یادگیری

یادگیری ماشینی با نظارت (supervised learning) به دنبال تابعی از میان یک سری توابع هست که تابع هزینه (loss function) داده‌ها را بهینه سازد. به عنوان مثال در مسئله رگرسیون تابع هزینه می‌تواند اختلاف بین پیش‌بینی و مقدار واقعی خروجی به توان دو باشد، یا در مسئله طبقه‌بندی ضرر منفی لگاریتم احتمال خروجی باشد. مشکلی که در یادگیری شبکه‌های عصبی وجود دارد این است که این مسئله بهینه‌سازی دیگر محدب (convex) نیست ازین رو با مشکل کمینه‌های محلی روبرو هستیم. یکی از روش‌های متداول حل مسئله بهینه‌سازی در شبکه‌های عصبی بازگشت به عقب یا همان back propagation است روش بازگشت به عقب گرادیانِ تابع هزینه را برای تمام وزن‌های شبکه عصبی محاسبه می‌کند و بعد از روش‌های گرادیان کاهشی (gradient descent) برای پیدا کردن مجموعه وزن‌های بهینه استفاده می‌کند روش‌های گرادیان کاهشی سعی می‌کنند به صورت متناوب در خلاف جهت گرادیان حرکت کنند و با این کار تابع هزینه را به حداقل برسانند .پیدا کردن گرادیانِ لایه آخر ساده است و با استفاده از مشتق جزئی به‌دست می‌آید. گرادیانِ لایه‌های میانی اما به صورت مستقیم به‌دست نمی‌آید و باید از روش‌هایی مانند قاعده زنجیری در مشتق‌گیری استفاده کرد .روش بازگشت به عقب از قاعده زنجیری برای محاسبه گرادیان‌ها استفاده می‌کند و همان‌طور که در پایین خواهیم دید، این روش به صورت متناوب گرادیان‌ها را از بالاترین لایه شروع کرده آن‌ها را در لایه‌های پایینتر «پخش» می‌کند.

نکته: بازگشت به عقب(Backpropagation) ، روشی برا محاسبه گرادیانهاست که در ادامه بیشتر با آن آشنا خواهیم شد.

تاریخچه شبکه‌های عصبی پیچشی

شبکه‌های عصبی پیچشی مشابه سیستم پردازش دیداری در موجودات زنده عمل می‌کنند. از قرن نوزدهم به‌طور همزمان اما جداگانه از سویی نوروفیزیولوژیست‌ها سعی کردند سیستم یادگیری و تجزیه و تحلیل مغز را کشف کنند، و از سوی دیگر ریاضیدانان تلاش کردند مدل ریاضی ای بسازند که قابلیت فراگیری و تجزیه و تحلیل عمومی مسائل را دارا باشد. اولین کوشش‌ها در شبیه‌سازی با استفاده از یک مدل منطقی در اوایل دهه ۱۹۴۰ توسط وارن مک‌کالک و والتر پیتز انجام شد که امروزه بلوک اصلی سازنده اکثر شبکه‌های عصبی مصنوعی است. عملکرد این مدل مبتنی بر جمع ورودی‌ها و ایجاد خروجی با استفاده از شبکه‌ای از نورون‌ها است. اگر حاصل جمع ورودی‌ها از

مقدار آستانه بیشتر باشد، اصطلاحاً نورون برانگیخته می‌شود. نتیجه این مدل اجرای ترکیبی از توابع منطقی بود.

در سال ۱۹۴۹ دونالد هب قانون یادگیری را برای شبکه‌های عصبی طراحی کرد. در سال ۱۹۵۸ شبکه پرسپترون توسط روزنبلات معرفی گردید. این شبکه نظیر واحدهای مدل شده قبلی بود. پرسپترون دارای سه لایه است که شامل لایه ورودی، لایه خروجی و لایه میانی می‌شود. این سیستم می‌تواند یاد بگیرد که با روشی تکرارشونده وزن‌ها را به گونه‌ای تنظیم کند که شبکه توان بازتولید جفت‌های ورودی و خروجی را داشته‌باشد. روش دیگر، مدل خطی تطبیقی نورون است که در سال ۱۹۶۰ توسط برنارد ویدرو و مارسیان هاف در دانشگاه استنفورد (به وجود آمد که اولین شبکه‌های عصبی به کار گرفته شده در مسائل واقعی بودند. آدالاین یک دستگاه الکترونیکی بود که از اجزای ساده‌ای تشکیل شده بود، روشی که برای آموزش استفاده می‌شد با پرسپترون فرق داشت.)

در سال ۱۹۶۹ میسکی و پاپرت کتابی نوشتند که محدودیت‌های سیستم‌های تک لایه و چند لایه پرسپترون را تشریح کردند. نتیجه این کتاب پیش داوری و قطع سرمایه‌گذاری برای تحقیقات در زمینه شبیه‌سازی شبکه‌های عصبی بود. آن‌ها با طرح اینکه طرح پرسپترون قادر به حل هیچ مسئله جالبی نمی‌باشد، تحقیقات در این زمینه را برای مدت چندین سال متوقف کردند.

با وجود اینکه اشتیاق عمومی و سرمایه‌گذاری‌های موجود به حداقل خود رسیده بود، برخی محققان تحقیقات خود را برای ساخت ماشین‌هایی که توانایی حل مسائلی از قبیل تشخیص الگو را داشته باشند، ادامه دادند. از جمله گراسبگ که شبکه‌ای تحت عنوان Avalanch را برای تشخیص صحبت پیوسته و کنترل دست ربات مطرح کرد. همچنین او با همکاری کارپنتر شبکه‌های نظریه تشدید انطباقی را بنا نهادند که با مدل‌های طبیعی تفاوت داشت. اندرسون و کوهونن نیز از اشخاصی بودند که تکنیک‌هایی برای یادگیری ایجاد کردند. ورباس در سال ۱۹۷۴ شیوه آموزش پس انتشار خطا را ایجاد کرد که یک شبکه پرسپترون چندلایه البته با قوانین نیرومندتر آموزشی بود.

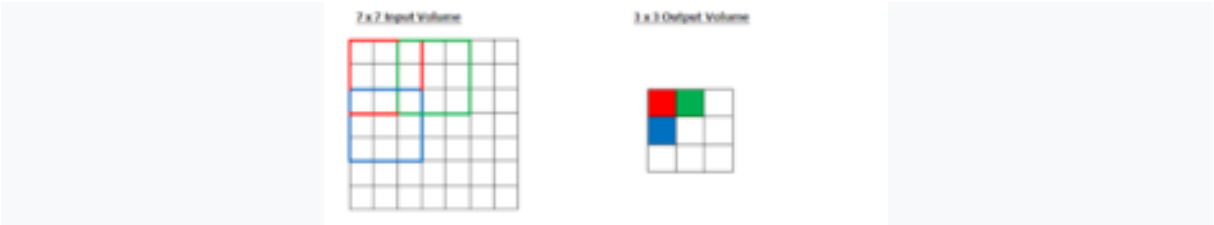
پیشرفت‌هایی که در سال ۱۹۷۰ تا ۱۹۸۰ به‌دست آمد برای جلب توجه به شبکه‌های عصبی بسیار مهم بود. برخی فاکتورها نیز در تشدید این مسئله دخالت داشتند، از جمله کتاب‌ها و کنفرانس‌های وسیعی که برای مردم در رشته‌های متنوع ارائه شد. امروز نیز تحولات زیادی در تکنولوژی ANN ایجاد شده‌است.

ابرپارامترها

ابرپارامترها نقش مهمی در عملکرد شبکه‌های عصبی دارند. با تنظیم این ابرپارامترها با استفاده از داده اعتبار سنجی (validation) **می‌توان عملکرد مدل را بهبود داد.**

نکته: تعدادی از ابرپارامترها در شبکه‌های عصبی پیچشی، مشابه با ابرپارامترهای سایر شبکه‌های عصبی است. برای مثال می‌توان به نرخ یادگیری(learning rate) ، تعداد ایپاک‌ها (epoch) اشاره کرد. اما این نوع شبکه‌ها نسبت به سایر شبکه‌ها تعداد ابرپاراکتر بیشتری دارند که به بررسی آن‌ها پرداخته می‌شود.

- **اندازه هسته :(kernel size) هر لایه پیچشی یا ادغامی دارای یک هسته است که عمق آن به اندازه تعداد فیلترهای ورودی است. اما اندازه طول و عرض آن قابل تنظیم است و معمولاً اندازه‌ای کوچک دارد.**
- **گام :(stride) همان‌طور که در ابتدا بیان شد، هسته (کرنل) در روی ماتریس ورودی می‌لغزد و با هربار لغزش، ماتریس کرنل و آن بخش از ورودی که منطبق بر کرنل است در یکدیگر ضرب می‌شوند. لغزیدن ماتریس هسته بر روی ورودی می‌تواند با گام‌های مختلف انجام شود.**

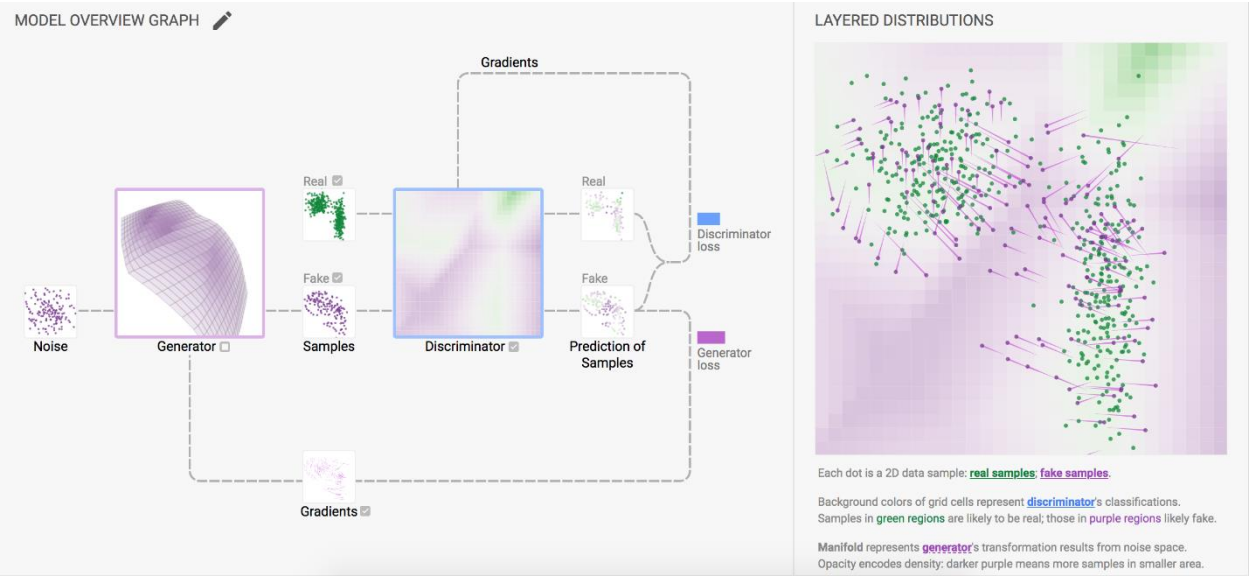


قرار دادن گام به اندازه ۲ باعث می‌شود که هربار کرنل به اندازه ۲ پیکسل به جلو لغزد. در نتیجه با داشتن کرنل به اندازه ۳*۳ و قرار دادن طول گام به اندازه ۲، خروجی نهایی ۳*۳ خواهد بود.

- **حاشیه‌گذاری :(padding) حاشیه گذاری به معنای اضافه کردن پیکسل‌هایی (معمولا با مقدار صفر) به حاشیه تصویر است. اینکار به دلایل متفاوتی انجام می‌شود. یکی از دلایل، آن است که با تغییر مقدار گام** (stride) **از ۱ به اعداد دیگر، ممکن است با لغزاندن ماتریس هسته روی ورودی، مقداری از این ماتریس خارج از ورودی بیفتد که منجر به بروز خطا در مدل می‌شود. در نتیجه با اضافه کردن حاشیه به تصویر از این مورد جلوگیری می‌شود. یکی دیگر از دلایل اضافه کردن حاشیه به تصویر، برابر کردن اهمیت پیکسل‌ها در ورودی است. درصورتی که حاشیه گذاری وجود نداشته باشد، پیکسل‌هایی که در گوشه تصویر هستند، تنها در یکی از عملیات‌های ضرب پیچشی قرار ی‌گیرند و در نتیجه اهمیت آن‌ها در خروجی کم‌رنگ‌تر است. با اضافه**

کردن حاشیه مناسب، تعداد بارهایی که هر پیکسل در عملیات ضرب پیچشی میآید برابر با تمام پیکسل‌های دیگر است.

- تعداد فیلترها: همان‌طور که پیشتر نیز ذکر شد، عمق هر ماتریس هسته به اندازه عمق ورودی است و از ضرب پیچشی هر ماتریس هسته در ورودی، یک ماتریس با عمق یک به وجود می‌آید. برای آنکه بتوان تعداد بیشتری ویژگی را از ورودی استخراج کرد، ورودی را با چند ماتریس هسته متفاوت با مقادیر مختلف ضرب پیچشی می‌کنند و در نتیجه عمق خروجی نیز به اندازه تعداد هسته‌هایی است که ورودی را در آن ضرب می‌کنند. در نتیجه تعداد فیلترها همان عمق خروجی را مشخص می‌کند.



مدل شبکه های عصبی پیچشی

در بخش اول در مورد کلیات شبکه های عصبی و شبکه های عصبی کانولوشن توضیح داده شد در این بخش به صورت تخصصی در مورد شبکه های عصبی کانولوشنی صحبت می کنیم و در بخش های بعدی وارد جزییات بیشتر خواهیم شد.

تعریف شبکه های کانولوشن

شبکه های عصبی کانولوشن تا حد بسیار زیادی شبیه شبکه های عصبی مصنوعی هستند که در بخش قبلی در مورد آنها توضیح داده شد. این نوع شبکه ها متشکل از نورونهایی با وزنها و بایاسهای قابل یادگیری (تنظیم) هستند. هر نورون تعدادی ورودی دریافت کرده و سپس حاصل ضرب وزنها در

ورودی ها را محاسبه کرده و در انتها با استفاده از یک یک تابع تبدیل(فعال سازی) غیرخطی نتیجه ای را ارائه دهد. کل شبکه همچنان یک تابع امتیاز (Score function) مشتق پذیر(differentiable) را ارائه میکند، که در یک طرف آن پیکسل های خام تصویر ورودی و در طرف دیگر آن امتیازات مربوط به هر دسته قرار دارد. این نوع شبکه ها هنوز یک تابع هزینه (Loss function) (مثل Softmax،SVM) در لایه آخر (تماما مرتبط یا fully connected) دارند و تمامی نکات مطرحی در مورد شبکه های عصبی معمولی در اینجا هم صادق است.

با توجه به مطالب گفته شده ،تفاوت شبکه عصبی کانولوشن با شبکه عصبی مصنوعی در چه چیزی میتواند باشد ؟ معماری های شبکه های عصبی کانولوشن بصورت صریح فرض میکنند که ورودی های آنها تصاویر هستند ، با این فرض ما میتوانیم ویژگی های مشخصی را درون معماری تعبیه (encode) کنیم .با این عمل تابع پیشرو (forward function) را میتوان بصورت بهینه تر پیاده سازی کرد و همینطور با این کار میزان پارامترهای شبکه نیز بشدت کاهش پیدا میکند.

خلاصه معماری

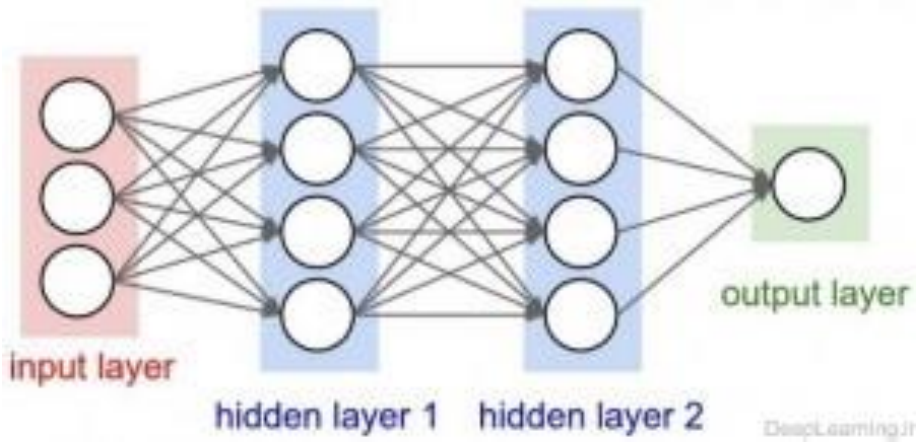
همانطور که ما میدانیم، شبکه های عصبی یک ورودی دریافت میکنند (در قالب یک بردار) و سپس آنرا از تعدادی لایه مخفی (Hidden layer) عبور میدهند و نهایتا یک خروجی که نتیجه پردازش لایه های مخفی است در لایه خروجی شبکه ظاهر میشود. هر لایه مخفی از تعدادی نورون تشکیل شده که این نورون ها به تمام نورون های لایه قبل از خود متصل میشوند. نورونهای هر لایه بصورت مستقل عمل کرده و هیچ ارتباطی با یکدیگر ندارند. آخرین لایه تماما متصل (fully connected layer) به لایه خروجی (output layer) معروف است و معمولا نقش نمایش دهنده امتیاز هر دسته(class) را ایفا میکند.

شبکه های عصبی معمولی برای تصاویر معمول (full images) بخوبی مقیاس پذیر نیستند. بعنوان مثال تصاویر موجود در دیتاست CIFAR-10 اندازه ای برابر با 32x32x3 دارند (۳۲ پیکسل عرض، ۳۲ پیکسل ارتفاع و ۳ کانال رنگ) . بنابر این یک نورون با اتصال کامل (fullyconnected) در لایه مخفی اول یک شبکه عصبی معمولی 32x32=3072 وزن خواهد داشت . این مقدار شاید در نظر اول مقدار قابل توجهی بنظر نیاید اما بطور واضح مشخص است که این معماری تماما مرتبط قابل استفاده برای تصاویر بزرگتر نخواهد بود. برای مثال یک تصویر با اندازه متعارف تر مثل 200 x 200 x 3 باعث میشود که یک نورون 000،200x200x3=120 وزن داشته باشد!. علاوه بر این ما قطعا خواهان تعداد بیشتری از این

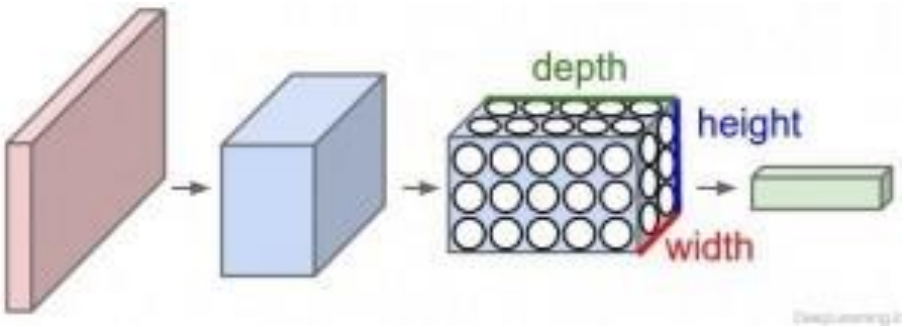
نورون ها خواهیم بود ، پس تعداد پارامترها بسرعت افزایش پیدا میکند . مشخص است این اتصال کامل (Full connectivity) باعث اتلاف (wasteful) بوده و تعداد بسیار زیاد پارامترها هم بسرعت باعث overfitting خواهد شد.

توده های سه بعدی از نورونها!(3d volumes of neurons)

شبکه های عصبی کانولوشن از این واقعیت که ورودی شامل تصاویر است استفاده کرده و معماری شبکه را به روش معقولی محدود کردند. بطور خاص، برخلاف یک شبکه عصبی معمولی، لایه های یک شبکه عصبی کانولوشن (به اختصار ConvNet) شامل نورونهایی است که در سه بعد عرض، ارتفاع و عمق قرار گرفته اند(مرتب شده اند).(دقت کنید که کلمه عمق در اینجا اشاره به بُعد سوم یک توده فعال سازی (activation volume) دارد و به معنای عمق یک شبکه عصبی کامل که به معنای تعداد لایه های موجود در آن است نمیباشد.) بعنوان یک مثال، تصاویر ورودی از دیتاست CIFAR-10 ،هر کدام یک توده ورودی حاوی مقادیر فعال سازی an Input volume of activations)) هستند که دارای ابعاد 32x32x3 (عرض، ارتفاع و عمق) هستند. همانطور که جلوتر خواهیم دید، هر نورون در هر لایه بجای اتصال با تمام نورون ها در لایه قبل تنها به ناحیه کوچکی از لایه قبل از خود متصل است. علاوه بر آن، لایه خروجی نهایی برای تصاویر رقابت CIFAR-10 دارای 1x1x10 بُعد خواهد بود. چرا که همگام با رسیدن به انتهای معماری شبکه ConvNet ما اندازه تصویر را کاهش میدهیم بگونه ای که در انتها تصویر کامل ورودی ما به یک بردار حاوی امتیاز دسته ها (کلاسها) کاهش پیدا میکند و ما با یک بردار که حاوی امتیاز هر دسته است مواجه خواهیم بود. این امتیازات در امتداد بعد عمق(depth dimension) مرتب شده اند. نمایشی از این عمل را در زیر میتوانید مشاهده کنید:



یک شبکه عصبی معمولی با ۳ لایه



یک شبکه عصبی کانولوشن

همانطور که در تصویر بالا میبینید در هر لایه، یک شبکه عصبی کانولوشن (ConvNet) نورون های خود را در ۳ بعد مرتب میکند (عرض ، ارتفاع و عمق) هر لایه یک شبکه ConvNet ورودی را در قالب یک توده سه بعدی به یک توده سه بعدی خروجی از مقادیر فعال سازی نورونها تبدیل میکند. در این مثال لایه ورودی قرمز رنگ حاوی تصویر است(مقادیر پیکسل های تصویر) بنابر این عرض و ارتفاع آن ابعاد تصویر خواهند بود و عمق آن هم برابر با ۳ خواهد بود (کانال های قرمز، سبز و آبی مربوط به تصویر)

یک شبکه ConvNet از چند لایه تشکیل میشود و هر لایه شیوه کار ساده ای دارد. که در آن یک توده سه بعدی ورودی دریافت کرده و آن را با استفاده از توابعی مشتق پذیر (differentiable function) که ممکن است با پارامتر یا بدون پارامتر باشند به یک توده سه بعدی خروجی تبدیل میکند.

نکات:

- **از آنجایی که مقادیر مربوط به این پارامترها ی مراحلی بصورت خودکار تنظیم میشود، ما از آن به یادگیری یاد میکنیم ، چرا که شبکه عصبی گام بگام با یادگیری این پارامترها قادر به انجام وظیفه شناسایی محول شده به آن میشود.**
- **یکی از دیتاست های معروف که جهت رقابتهای جهانی پردازش تصویر مورد استفاده قرار میگیرد. این دیتاست شامل ۶۰ هزار تصویر رنگی با اندازه ۳۲×۳۲ پیکسل در ۱۰ دسته مختلف است.** (CIFAR-10 and CIFAR-100 datasets،2012)
- **توده فعال سازی یا** Activation volume **به یک توده سه بعدی حاوی مقادیر عددی گفته میشود که بعنوان ورودی به تابع فعال سازی ارسال میشوند، برای همین به آنها توده فعال سازی گفته میشود. مقادیر موجود در این توده ها ممکن است مقادیر متناظر به**

پیکسلهای خام تصاویر باشند (توده فعال سازی ورودی) و یا نتیجه پردازش های انجام شده تا لایه خاصی در شبکه باشند (بعنوان مثال توده فعال سازی در لایه دوم یعنی مقادیر عددی در لایه دوم که نتیجه عملیاتهای لایه های قبل تا لایه فعلی است (ضرب وزنها در خروجی حاصل از لایه قبل و...)

- در اینجا مقادیر فعال سازی چیزی جز مقادیر مربوط به پیکسل های خام تصاویر ورودی نیستند.

لایه های مورد نیاز برای ایجاد یک شبکه ConvNet

همانطور که در بالا اشاره کردیم، هر لایه شبکه کانولوشن یک توده فعال سازی را از طریق یک تابع مشتق پذیر به توده فعال سازی دیگر تبدیل میکند. ما از سه نوع اصلی لایه ها برای ساخت یک معماری شبکه کانولوشن استفاده میکنیم . این لایه ها عبارتند از : لایه کانولوشن، لایه Pooling و لایه تماما متصل (Fully connected layer) که دقیقا همانند همان که در شبکه های عصبی معمولی میبینیم است . ما این لایه ها را روی هم قرار میدهیم تا یک معماری کامل از شبکه کانولوشن ایجاد کنیم .

برای روشن تر شدن بیشتر مباحث بالا یک شبکه کانولوشن ساده برای دسته بندی دیتاست CIFAR-10 ایجاد میکنیم . برای اینکار ما میتوانیم یک معماری با لایه های لایه ورودی ، لایه کانولوشن، لایه RELU ، لایه POOL ، لایه FC داشته باشیم .

لایه ورودی (Input layer) شامل مقادیر پیکسل های خام تصویر ورودی ما هستند . یعنی در اینجا ما یک تصویر با عرض ۳۲ ، ارتفاع ۳۲ و ۳ کانال رنگ قرمز،سبز ، آبی خواهیم داشت.

لایه کانولوشن (CONV layer) این لایه خروجی نورونهایی که به نواحی محلی در ورودی متصل هستند را محاسبه میکند. عمل محاسبه هم از طریق ضرب نقطه ای بین وزن های هر نورون و ناحیه ای که آنها به آن(توده فعال سازی ورودی) متصل هستند صورت میگیرد. نتیجه این عمل یک توده با اندازه 32x32x12 میشود.

لایه RELU بر روی تک تک نورون ها یک تابع فعال سازی مثل max(0,x) که آستانه گذاری را بر روی ۰ انجام میدهد یعنی مقادیر منفی را صفر در نظر میگیرد) را اعمال میکند. این کار تغییری در اندازه توده از مرحله قبل نمیدهد بنابر این نتیجه همچنان یک توده با اندازه 32x32x12 خواهد بود .

لایه Pooling عملیات downsampling را در امتداد ابعاد مکانی (عرض و ارتفاع) انجام می دهد

که نتیجه این کار یک توده با اندازه 16x16x12 خواهد بود . همانطور که دقت کردید در لایه pooling ما ابعاد توده ورودی (تصویر) را کاهش میدهیم.و در اصل از طریق عملیات این لایه است که در انتهای شبکه کانولوشن ما به یک بردار امتیاز دست پیدا میکنیم .

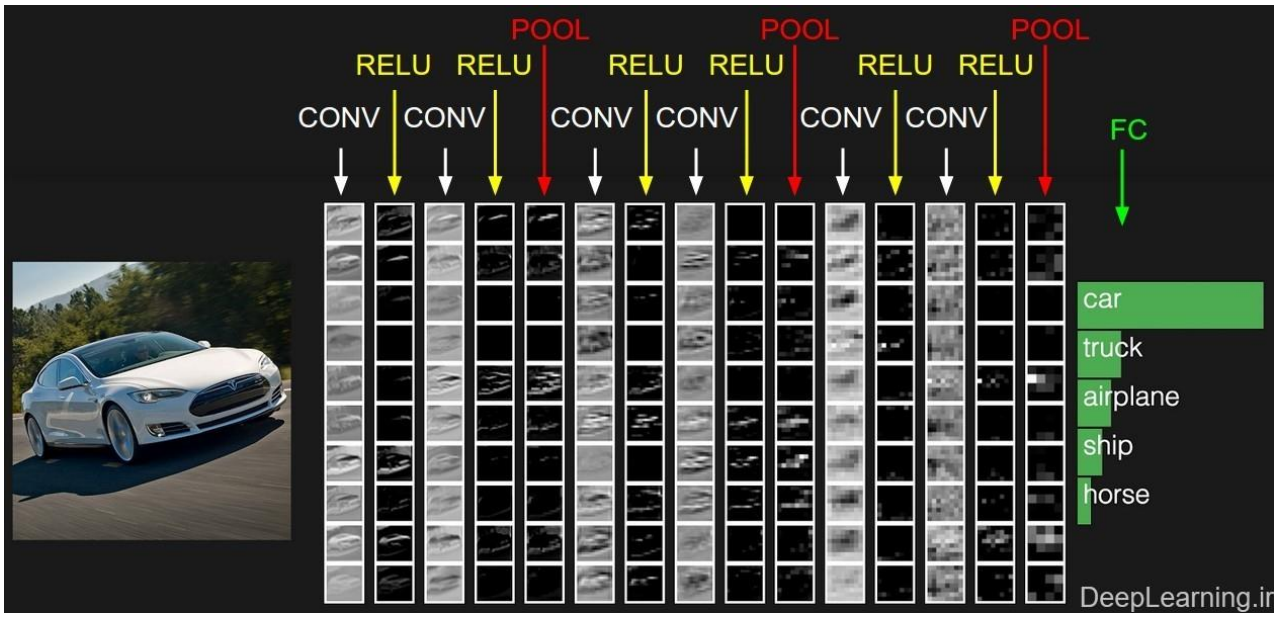
لایه FC یا تماما مرتبط وظیفه محاسبه امتیاز دسته ها (class) را دارد. نتیجه کار این لایه یک توده با اندازه 1x1x10 است که هر کدام از این ۱۰ عدد نمایانگر یک امتیاز برای یک دسته بخصوص (مثل یکی از ۱۰ دسته تصویر موجود در دیتاست CIFAR-10) است. مثل شبکه های عصبی معمولی و همانطور که از اسم این لایه بر می آید هر نورون در این شبکه با تمام نورون ها در توده قبل از خود ارتباط دارد.

با این روش، شبکه کانولوشن مقادیر پیکسل های خام تصویر اصلی را لایه به لایه به امتیاز دسته ها در انتهای شبکه تبدیل میکند.. دقت کنید که بعضی از لایه ها پارامتر داشته و بعضی دیگر فاقد پارامتر اند. بطور خاص لایه های Conv/FC لایه هایی هستند که تبدیلاتی را انجام میدهند که نه تنها تابعی از فعال سازی ها(مقادیر موجود) در توده ورودی اند بلکه تابع پارامترهایی نظیر وزن و بایاس نورون ها هم هستند .از طرف دیگر لایه های RELU/POOL تنها یک تابع ثابت را پیاده سازی میکنند. پارامترهای موجود در لایه های Conv/FC توسط روش gradient descent آموزش میبینند تا امتیازات دسته ها (class)یی که شبکه کانولوشن حساب میکند با برچسب های هر تصویر در مجموعه آموزشی سازگار باشد.(همخوانی داشته باشد).

بطور خلاصه :

- یک معماری شبکه کانولوشن لیستی از لایه هاست که توده متشکل از تصویر ورودی را به یک توده خروجی (مثل توده ای که امتیازات دسته ها را در خود دارد) تبدیل میکند.
- تعداد کمی از انواع لایه ها برای شبکه کانولوشن وجود دارد (بعنوان مثال CONV/FC/RELU/POOL متداول ترین لایه ها در شبکه کانولوشن هستند)
- هر لایه یک ورودی سه بعدی را دریافت کرده و انرا از طریق یک تابع مشتق پذیر به یک توده سه بعدی خروجی تبدیل میکند .
- هر لایه ممکن است دارا یا فاقد پارامتر باشد (بعنوان مثال لایه های CONV/FC دارای پارامتر و لایه های RELU/POOL فاقد پارامتر هستند)
- هر لایه ممکن است دارا یا فاقد فرا پارامتر(hyperparameter) اضافی باشد. (مثلا لایه های

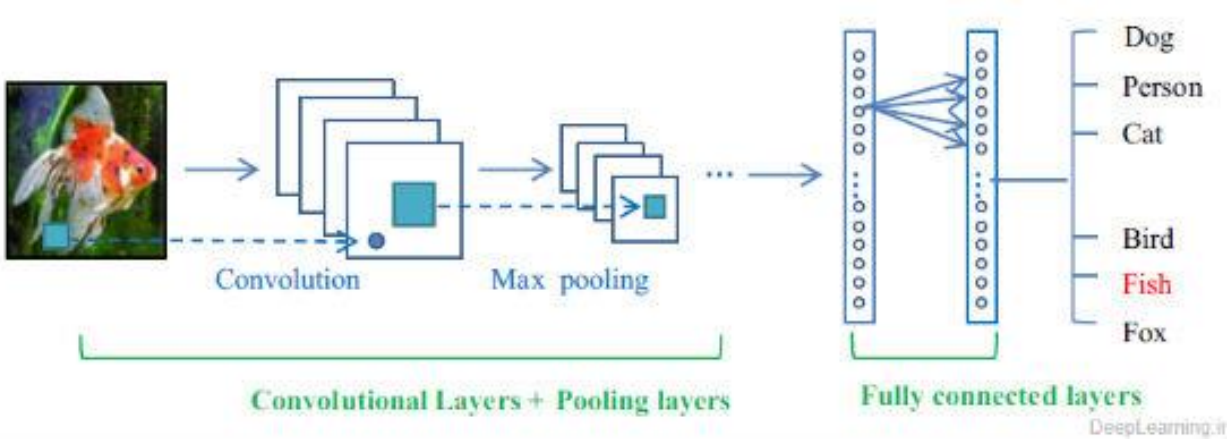
CONV/FC/POOL دارای این نوع پارامترها هستند در حالی که لایه RELU فاقد این نوع پارامتر است)



نمونه ای فعال سازی های در یک نمونه از معماری شبکه کانولوشن

توده ابتدایی پیکسل های خام تصویر را در خود ذخیره میکند(تصویر اتومبیل در سمت چپ) و آخرین توده امتیاز دسته ها را در خود جای میدهد(لایه تماما متصل یا به اختصار FC). هر توده فعال سازی در طی مسیر پردازش در قالب یک ستون نمایش داده شده است. از آنجایی که نمایش توده های ۳بعدی مشکل است ما برشهای (قاچهای) هر توده را بصورت سطری مرتب کردیم. توده مربوط به آخرین لایه حاوی امتیازات مربوط به هر دسته (class) است. اما در اینجا ما تنها ۵ امتیاز بالاتر را نمایش دادیم . معماری نمایش داده شده در اینجا یک شبکه VGGNet کوچک است که جلوتر در مورد آن توضیح داده ایم .

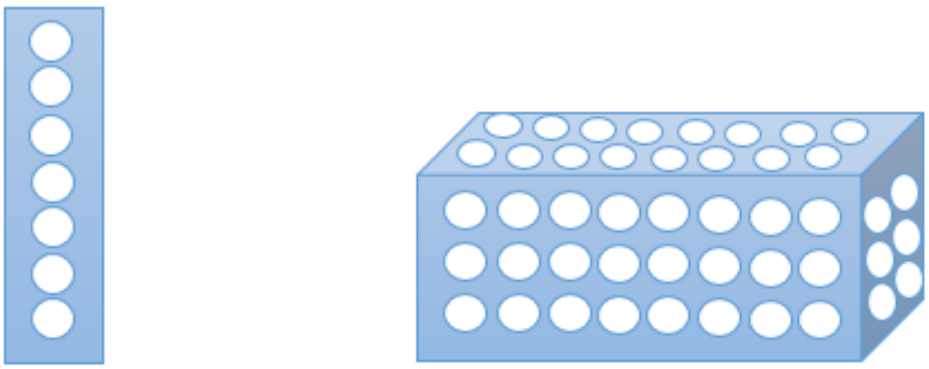
ما حالا می توانیم بعد از این آشنایی اولیه، به جزییات بیشتر در رابطه با این لایه ها بپردازیم . در بخش بعد لایه های مختلف و جزییات مربوط به فراپارامترها و اتصالات آنها شرح داده می شوند.



یادگیری عمیق (deep learning)

لایه Convolutional

لایه کانولوشن هسته اصلی تشکیل دهنده شبکه عصبی کانولوشن است، و توده خروجی آن را میتوان بصورت یک توده سه بعدی از نورون ها تفسیر کرد. به زبان ساده تر یعنی اینکه خروجی این لایه یک توده سه بعدی است . برای درک بهتر این مسئله شبکه های عصبی معمولی را در نظر بگیرید . در شبکه های عصبی معمولی هر لایه چیزی جز لیستی (یک بعدی همانند یک مستطیل!) از نورون ها نبود که هر نورون خروجی خاص خود را تولید میکرد و نهایتا یک لیست از خروجی ها که متناظر با هر نورون بود حاصل میشد . اما در شبکه عصبی کانولوشن بجای یک لیست ساده ما با یک لیست سه بعدی (یک مکعب !) مواجه هستیم که نورونها در سه بعد آن مرتب شده اند. در نتیجه خروجی این مکعب نیز یک توده سه بعدی خواهد بود . تصاویر زیر این مفهوم و تفاوت بین این دو مفهوم را بهتر بیان میکند:



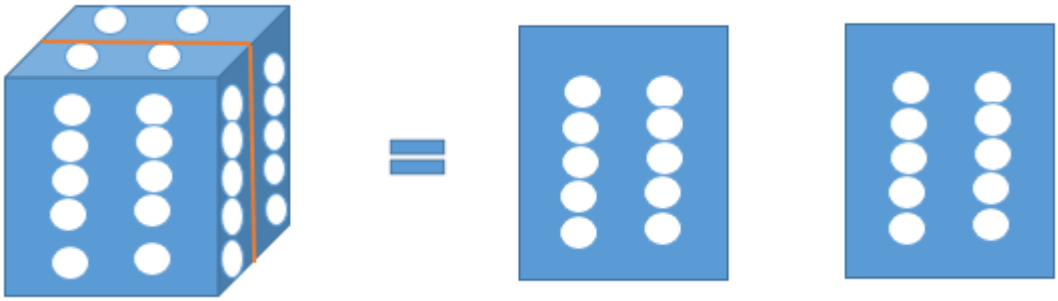
یک لایه شبکه عصبی معمولی (توده یک بعدی)　　یک لایه شبکه کانولوشن (توده سه بعدی)

دایره های سفید رنگ نمایانگر نورون ها هستند

در اینجا ما در مورد جزییات اتصالات نورون ها و نحوه قرارگیری آنها در فضا و طرح اشتراک پارامتر آنها صحبت می کنیم .

خلاصه و درک شهودی مسئله

پارامترهای لایه کانولوشن شامل مجموعه ای از فیلترهای قابل یادگیری هستند. هر فیلتر از لحاظ مکانی کوچک بوده اما در امتداد عمق توده ورودی ادامه پیدا میکند. به زبان ساده تر میتوان اینطور گفت که ما با یک توده سه بعدی مواجه هستیم . این توده سه بعدی دارای یک طول و عرض و یک عمق است. اگر فرض کنیم عمق ما برابر با X باشد به این معناست که ما X برش (قاچ) از توده خواهیم داشت. بعنوان مثال اگر عمق برابر با ۱ باشد ما با یک ماتریس ساده (آرایه دوبعدی) مواجه هستیم که دارای طول و عرض است . حال اگر عمق برابر با ۲ باشد یعنی ما دارای دو ماتریس هستیم . یعنی در هر عمق در اصل یک ماتریس وجود دارد. بنابر این زمانی که ما میگوییم فیلتر مورد نظر در امتداد یا راستای عمق ادامه می یابد به این معناست که این فیلتر بر روی تمامی ماتریس ها (برش های توده ورودی) اعمال می شود.



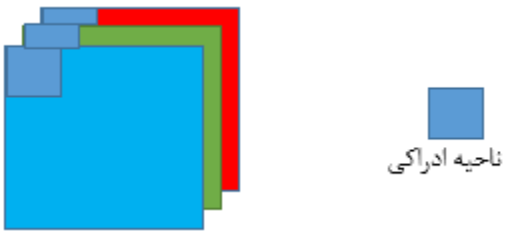
یک توده سه بعدی با عمق برابر با ۲ که بصورت عمودی برش داده شده است .

در زمان forward pass ما هر فیلتر را در امتداد پهنا (width) و ارتفاع (height) ورودی می لغزانیم (در اصل convolve می کنیم) با این کار ما یک نگاشت فعال سازی (Activation Map) دو بعدی برای آن فیلتر ایجاد می کنیم. همانطور که ما فیلتر را در عرض ورودی می لغزانیم (slide می دهیم) ضرب نقطه ای بین ورودی های فیلتر با ورودی را هم انجام می دهیم . بصورت شهودی، شبکه، فیلترهایی که در زمان مشاهده برخی از ویژگی های خاص در بعضی موقعیتهای مکانی در ورودی فعال میشوند را یاد می گیرد . با انباشته کردن این نگاشتهای فعال سازی (Activation Maps) برای تمامی فیلترها در راستای بُعد عمق ،توده خروجی کامل بدست می آید . هر مدخل در این توده خروجی را می توان بعنوان خروجی یک نورون که تنها به ناحیه کوچکی در ورودی نگاه

میکند در نظر گرفت که پارامترهای مشترک با بقیه نورون ها در همان نگاشت فعال سازی (Activation Map) دارد (چرا که این اعداد همه نتیجه اعمال یک فیلتر یکسان هستند).

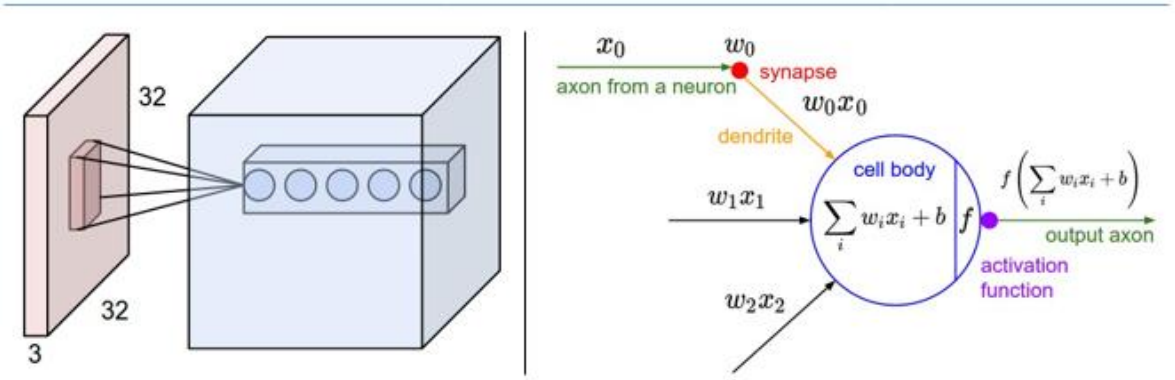
اتصال محلی (Local Connectivity)

زمانی که ما با ورودی هایی با ابعاد بالا(high dimensional) مثل تصاویر مواجه هستیم، همانطور که در بالا دیدیم اتصال نورون ها به تمام نورونهای قبل از خود (در توده قبلی) غیر عملی است. بنابر این بجای اینکار ما هر نورون را تنها به ناحیه کوچکی از توده ورودی متصل میکنیم .میزان وسعت این ناحیه کوچک برای اتصال یک فراپارامتر است که به آن receptive field و یا ناحیه ادراکی گفته میشود . وسعت اتصال در محور عمق همیشه مساوی با عمق توده ورودی است .توجه به این عدم تقارن در نحوه برخورد ما با ابعاد مکانی (طول و عرض تصویر و یا همان عرض و ارتفاع) و بعد عمق حائز اهمیت است. ارتباطات در مکان محلی هستند یعنی در راستای عرض و ارتفاع تصویر هستند اما همیشه تمام عمق توده ورودی را در بر میگیرند. اجازه دهید با یک مثال توضیح بیشتری بدهیم . فرض کنید توده ورودی ما دارای اندازه ای برابر با 3x32x32 باشد (مثلا یک عکس رنگی (RGB) از دیتاست CIFAR-10). حال اگر ناحیه ادراکی (Receptive Field) اندازه ای برابر با ۵×۵ داشته باشد این به این معناست که هر نورون در لایه کانولوشن برای ناحیه ای در توده ورودی با اندازه x5x3۵ تعداد وزنی برابر5x5x3=75 خواهد داشت. لطفاً دقت کنید که میزان وسعت اتصال در امتداد محور عمق باید مساوی ۳ باشد چرا که عمق توده ورودی ما ۳ است (تصویر ورودی ما دارای ۳ کانال رنگ است). پس همانطور که دقت کردید اتصالات فقط در راستای عرض و ارتفاع تصویر ورودی نیستند بلکه در محور عمق هم امتداد پیدا میکنند. و این به زبان ساده تر به این معناست که اگر تصویر رنگی ورودی خود با ۳ کانال رنگ را ۳ ماتریس که پشت سر هم قرار گرفته اند در نظر بگیریم . توضیحات بالا به این معناست که ما ناحیه ای به اندازه ۵×۵ را بر روی تمام این سه ماتریس اعمال میکنیم . تصویر زیر این مطلب را بهتر نشان میدهد.



نمایشی از ارتباط در یک توده که در راستای عرض و ارتفاع در امتداد عمق ادامه یافته است.

مثال دوم را در نظر بگیرد. فرض کنید توده ورودی اندازه ای برابر با 16x16x20 داشته باشد. بنابر این با استفاده از ناحیه ادراکی (Receptive field) با اندازه ۳×۳ هر نورون در لایه کانولوشن تعداد 3x3x20=180 اتصال به توده ورودی خواهد داشت. توجه کنید که اتصال در فضا محلی است (مثلا در اینجا ۳×۳ است) اما تمام عمق را در بر گرفته است (۲۰).



تصویر سمت چپ یک توده ورودی را نشان میدهد که با رنگ قرمز مشخص شده است (مثلا یک عکس با اندازه 3x32x32 از دیتا ست CIFAR-10). همینطور شما میتوانید یک توده از نورون ها را در لایه کانولوشن که به رنگ آبی نمایش داده شده است مشاهده کنید. هر نورون در لایه کانولوشن تنها به یک ناحیه محلی از لحاظ مختصات مکانی(طول و عرض) در توده ورودی متصل است اما این ارتباط در عمق بصورت کامل امتداد می یابد (یعنی تمام کانال های رنگ را در بر میگیرد). توجه کنید که چندین نورون (در این مثال ۵ نورون) در راستای عمق وجود دارند که همگی به یک ناحیه در ورودی نگاه میکنند(توضیحات بیشتر در ادامه داده می شود).

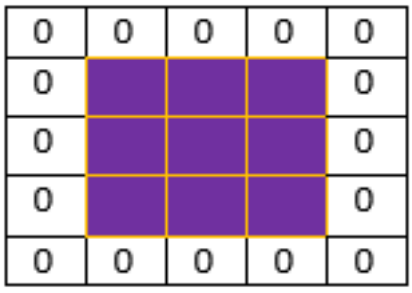
تصویر سمت راست ساختار یک نورون را نشان میدهد که دقیقا همانند چیزی است که در شبکه های عصبی معمولی مورد استفاده قرار می گیرد بعبارت دیگر نورون ها هیچ تفاوتی نسبت به قبل نکرده اند. آنها هنوز ضرب نقطه ای بین وزنها و ورودی ها را انجام داده و نتیجه را از یک تابع غیرخطی عبور می دهند و خروجی را تولید می کنند. تنها تفاوت در اینجا این است که اتصال دستخوش محدودیت شده است به این معنا که اتصالات آنها باید از لحاظ مکانی محلی باشد.

ما تا به اینجا در مورد اتصال هر نورون به توده ورودی در لایه کانولوشن صحبت کردیم اما چیزی در مورد اینکه چه تعداد نورون در توده خروجی بایستی وجود داشته باشند و یا اینکه ترتیب قرار گیری آنها به چه صورت باید باشد صحبتی نکردیم.۳ فراپارامتر (Hyper parameter) اندازه توده خروجی را کنترل میکنند. این سه پارامتر عمق (depth) ،گام (stride) و لایه گذاری با صفر (zero-padding) هستند. در زیر به توضیح بیشتر این پارامترها می پردازیم.

عمق توده خروجی پارامتری است که ما میتوانیم خود انتخاب کنیم . این پارامتر تعداد نورون هایی که در لایه کانولوشن به یک ناحیه در توده ورودی متصل میشوند را کنترل میکند. این پارامتر همانند حالتی در شبکه های عصبی معمولی است که ما در یک لایه مخفی چندین نورون داشتیم که همه به یک ورودی متصل بودند. همانطور که جلوتر خواهیم دید تمام این نورون ها یاد می گیرند که برای ویژگی های مختلف موجود در ورودی فعال شوند. بعنوان مثال اگر لایه کانولوشن اول یک تصویر خام را بعنوان ورودی دریافت کند، نورون هایی که در امتداد بعد عمق قرار دارند ممکن است با مواجهه با لبه های جهتدار (oriented edges) و یا لکه های رنگ (blobs of color) فعال شوند. ما به مجموعه نورون هایی که همه به یک ناحیه یکسان از ورودی نگاه میکنند یک ستون عمقی یا depth column می گوییم.

ما باید گام (stride) را که بوسیله آن ستون های عمقی (depth column) را حول ابعاد مکانی (عرض و ارتفاع) معین می کنیم مشخص کنیم. زمانی که stride برابر با ۱ باشد ما یک ستون عمقی (depth column) جدید از نورون ها را به مختصات مکانی با فاصله تنها ۱ واحد مکانی از هم، اختصاص می دهیم. این باعث بوجود آمدن نواحی ادراکی دارای اشتراک زیاد بین ستونها و همچنین توده های خروجی بزرگ میشود. برعکس اگر ما گام ها (stride) را بزرگتر بگیریم نواحی ادراکی اشتراک کمتری داشته و توده خروجی نیز از لحاظ ابعاد مکانی کوچکتر می شود.

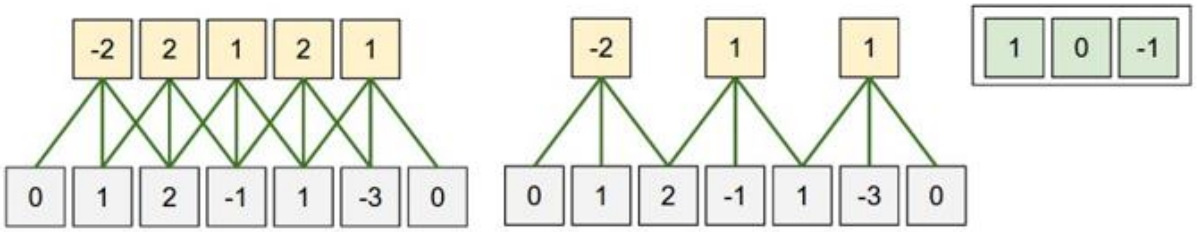
همانطور که بزودی خواهیم دید بعضی اوقات راحت تر است که مرز توده ورودی را با صفر بپوشانید(zero-pad) . به عبارت دیگر یعنی دور تصویر ورودی را با صفر پر کنیم . مثل اینکه یک سطر ۰ و یک ستون ۰ به ابتدا و انتهای تصویر اضافه کنیم اینطور تصویر ما در داخل قابی از صفر قرار خواهد گرفت مثل شکل زیر:



یک تصویر که zero-pad شده

اندازه این Zero-padding یک فراپارامتر(hyper parameter) است . ویژگی خوب این پارامتر این است که بما اجازه کنترل کردن اندازه توده خروجی را میدهد. بطور خاص ما بعضی اوقات میخواهیم که دقیقا اندازه مکانی توده ورودی حفظ شود.

ما برای محاسبه اندازه (مکانی) توده خروجی میتوانیم از اندازه توده ورودی (W) اندازه ناحیه ادراکی نورون های لایه کانولوشن (F) ، اندازه گام (stride) و مقدار zero padding که روی مرزهای توده ورودی اعمال شده استفاده کنیم . ما میتوانیم از فرمول: $\frac{W-F+2P}{S}+1$ **برای محاسبه اینکه چه تعداد نورون مناسب است استفاده کنیم (با تست این فرمول میتوانید به درستی آن پی ببرید) اگر نتیجه این فرمول یک عدد اعشاری باشد این به این معناست که مقدار استفاده شده برای stride نادرست بوده و نورون ها را نمیتوان با این مقدار طوری کنار یکدیگر مرتب کرد که بخوبی در سرتاسر توده ورودی بصورت متقارن جای شوند. برای درک این فرمول مثال زیر را در نظر بگیرید .**



تصویری از ترتیب (قرارگیری) مکانی.

در این مثال تنها یک بعد مکانی (محور x) ، یک نورون با ناحیه ادراکی با اندازه F = 3 و یک ورودی با اندازه W = 5 وجود دارد(۲،۱،-۱،۱،-۳). Zero padding هم برابر با P = 1 است. (یک صفر به دو طرف ورودی اضافه شده است)

در تصویر سمت چپ stride با اندازه S = 1 اعمال شده است که باعث اندازه خروجی : $\frac{5-3+2}{1}+1=5$ **شده است. در تصویر سمت راست stride با اندازه S = 2 باعث اندازه خروجی :** $\frac{5-3+2}{2}+1=3$ **شده است. توجه کنید که stride با اندازه S = 3 امکان پذیر نمیباشد چرا که با این مقدار نمیتوان نورون ها را بخوبی در سرتاسر توده جای داد. اگر از لحاظ فرمولی بخواهیم به این مسئله نگاه کنیم میبینیم که مقدار ۴=۳+۲-۵ بر ۳ قابل تقسیم نیست (بخش پذیر نیست). وزن نورونها در این مثال [۰،۱،-۱] (همانطور که در منتها الیه تصویر سمت راست نمایش داده شده است**

) و بایاس آن برابر با ۰ میباشد.. این وزنها بین تمام نورون های زرد رنگ مشترک است (بخش اشتراک پارامترها را در زیر بخوانید)

استفاده از Zero-padding

در مثال قبل (تصویر سمت چپ) توجه کنید که بُعد ورودی برابر ۵ و بعد خروجی نیز مساوی ۵ بود. به این خاطر این اتفاق رخ داد چرا که ناحیه ادراکی برابر با ۳ بود و ما از zero padding مساوی با ۱ استفاده کردیم . اگر zero padding یی وجود نداشت ، توده خروجی تنها بعدی برابر با ۳ میداشت. چرا که تنها این تعداد نورون در سرتاسر ورودی اصلی میتوانستند جا بگیرند. بطور کلی تنظیم مقدار Zero padding بر اساس فرمول: $P = \frac{F-1}{2}$ در زمانی که Stride برابر با S = 1 باشد اطمینان حاصل میکند که توده ورودی و توده خروجی هر دو از لحاظ مکانی دارای اندازه یکسانی خواهند بود. استفاده از Zero padding به اینصورت بسیار رایج بوده و ما توضیحات تکمیلی را جلوتر زمانی که در مورد معماری شبکه های کانولوشن بیشتر صحبت می کنیم خواهیم داد.

محدودیت های Stride

توجه کنید که ترتیب مکانی فراپارامترها (Hyper paramters) محدودیت های دوطرفه دارند. بعنوان مثال زمانی که اندازه ورودی برابر با ۱۰ بوده و هیچ zero padding استفاده نشده باشد (P=0) و اندازه فیلتر برابر با F=3 باشد بنابر این استفاده از stride یی با اندازه S = 2 غیر ممکن خواهد بود. چرا که: $\frac{W-F+2P}{S} + 1 = \frac{10-3+0}{2} + 1 = 4.5$ که همانطور که مشاهده میکنید نتیجه یک عدد صحیح نبوده و فلذا نورون ها نمیتوانند بصورت متقارن و بخوبی در سرتاسر ورودی جای(قرار) بگیرند. بنابراین این مقادیر این فراپارامترها نامعتبر در نظر گرفته میشود و اگر شما سعی کنید یک شبکه عصبی کانولوشن با این مقادیر در یک کتابخانه مربوط به شبکه عصبی کانولوشن ایجاد کنید به احتمال بسیار زیاد با یک Exception مواجه خواهید شد. همانطور که جلوتر در بخش معماری های شبکه کانولوشن خواهیم دید، تنظیم اندازه ها در شبکه کانولوشن بصورت مناسب بطوری که تمامی ابعاد آن بدرستی کار کنند کار واقعا طاقت فرسایی میتواند باشد که البته با استفاده از Zero-padding و تعداد دیگری از راهکارهای طراحی تا حد بسیار زیادی میتوان از سختی آن کاهید.

- یعنی تنها بخشی از عرض و ارتفاع را شامل میشود. (مثلا اگر عرض و ارتفاع تصویر به ترتیب برابر ۳۲ و ۳۲ باشد. فیلتر ممکن است اندازه برابر با ۴×۴ داشته باشد.
- به متغیرهای مختلف که در نتیجه مسئله ای دخیل هستند در Machine learning بُعد گفته میشود. در اینجا هر پیکسل یک تصویر یک متغییر بحساب می آید (چرا که بصورت یک ورودی مجزا به شبکه ارائه میشود و مقدار آن در کارکرد شبکه موثر است) . بنابر این یک تصویر با 20x20x3=1200 بعد خواهد داشت. از این رو در اینجا ابعاد بالای تصاویر به معنای بزرگتر بودن اندازه تصاویر است.

نمونه واقعی از شبکه های عصبی کانولوشن

معماری Krizhevsky et al که برنده رقابت ImageNet در سال ۲۰۱۲ شد تصاویری با اندازه 227x227x3 را بعنوان ورودی دریافت میکرد. در لایه کانولوشن اول این شبکه از نورون هایی با ناحیه ادراکی با اندازه Stride،F=11 با مقدار S=4 و zero padding P = 0 استفاده میشد. از آنجایی که ۵۵=۴+۱/(۱۱-۲۲۷) و عمق لایه کانولوشن برابر با K = 96 بود توده خروجی لایه کانولوشن اندازه ای برابر با 55x55x96 داشت. هر کدام از 55x55x96 نورون در این توده به ناحیه ای از ورودی با اندازه ۱۱x11x3 متصل بودند. علاوه بر آن تمام ۹۶ نورون در هر ستون عمقی (depth column) هم به یک ناحیه از ورودی با اندازه 11x11x3 متصل بودند البته با وزنهای متفاوت.

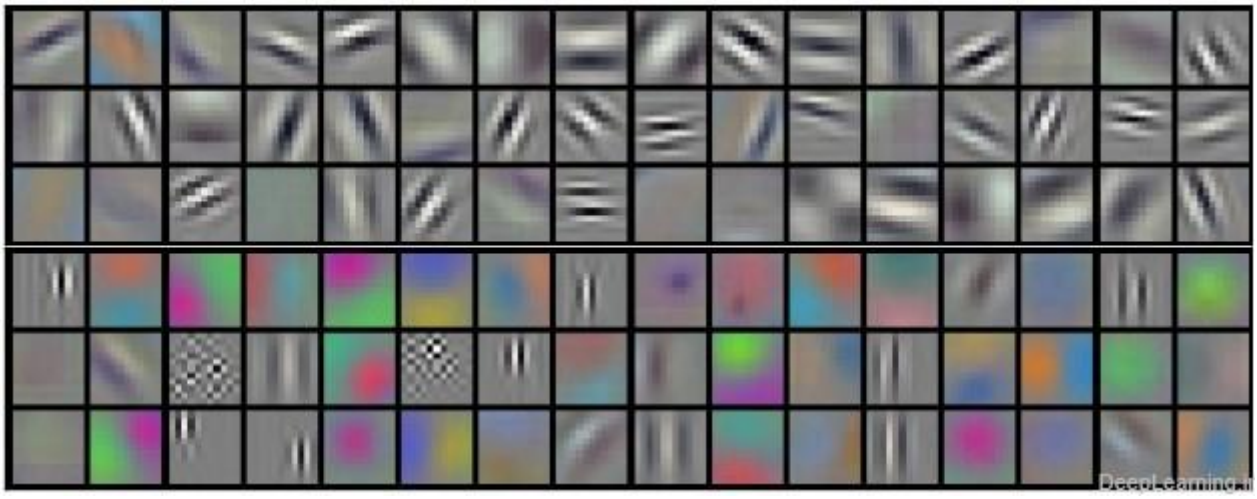
اشتراک پارامتر

طرح اشتراک پارامتر در لایه های کانولوشن به منظور کنترل تعداد پارامترها مورد استفاده قرار گرفت. با استفاده از نمونه عینی که در بالا داده شد، ما میبینیم که 55x55x96=290، 400 نورون در لایه کانولوشن اول وجود دارد و هر کدام از آنها 11x11x3=363 وزن و ۱ بایاس دارند. همه اینها با هم تشکیل ۲۹۰۴۰۰×۳۶۴=۱۰۵،۷۰۵،۶۰۰ پارامتر را تنها برای لایه اول شبکه کانولوشن می دهند. کاملا پیداست که این عدد بسیار بزرگی است.

ما می توانیم با یک فرض منطقی تعداد بسیار زیادی ازاین پارامترها را کاهش دهیم. و آن فرض این است که اگر یک Patch feature برای محاسبه در یک موقعیت مکانی (y،x) مفید باشد پس باید برای محاسبه همان feature در موقعیت متفاوت (y2،x2) هم مفید باشد. به عبارت دیگر اگر یک برش دو بعدی از عمق را یک برش عمقی بنامیم (بعنوان مثال یک توده با اندازه 55x55x96

دارای ۹۶ برش عمقی است که هر کدام دارای اندازه ۵۵×۵۵ می باشند) ما هر نورون در این برش عمقی را محدود به استفاده از یک مجموعه وزن و بایاس می کنیم . با این طرح اشتراک پارامتر، لایه کانولوشن اول در مثال ما حالا فقط ۹۶ مجموعه وزن یکتا خواهد داشت (یک مجموعه وزن برای هر برش عمقی) که در کل برابر 96x11x11x3 = 34، 848 وزن یکتا یا ۳۴،۹۴۴ پارامتر (به اضافه ۹۶ بایاس) خواهد بود . متناوبا تمام ۵۵×۵۵ نورون موجود در هر برش عمقی حالا از یک مجموعه پارامتر استفاده میکنند. در عمل در حین Back propagation هر نورون موجود در توده gradient را برای وزنهایش حساب میکند اما این gradient ها درسرتاسر هر برش عمقی جمع شده و تنها یک مجموعه وزن در هر برش را بروز می کند.

توجه کنید که اگر تمام نورون ها در یک برش عمقی از یک بردار وزن یکسان استفاده کنند بنابر این عملیات forward pass لایه کانولوشن میتواند در هر برش عمقی بصورت ضرب (convolution) بین وزنهای نورون با توده ورودی محاسبه شود (برای همین به آن لایه کانولوشن گفته میشود) بنابر این رایج است که به مجموعه وزن ها بعنوان یک فیلتر نگاه شود(یا یک کرنل). که با ورودی ضرب (convolve) شده است. نتیجه این عمل (Convolution) یک نگاشت فعالسازی (Activation map) است (مثلا با اندازه ۵۵×۵۵) و مجموعه این Activation map ها برای هر فیلتر مختلف در راستای بعد عمق بر روی هم قرار گرفته تا توده خروجی را ایجاد کنند(مثلا با اندازه 55x55x96)



فیلترهای نمونه که توسط Krizhevsky et al یادگرفته شده اند. هر کدام از این ۹۶ فیلتر نمایش داده شده در اینجا اندازه ای برابر با 11x11x3 دارند و هر کدام از انها با ۵۵×۵۵ نورون در یک برش عمقی به اشتراک گذاشته شده اند. دقت کنید که فرض اشتراک پارامترها نسبتا منطقی هم

هست. اگر کشف یک لبه افقی در جایی از تصویر مهم باشد، قائدتا برای سایر مکانها در تصویر هم باید بواسطه ساختار translationally-invariant تصاویر مفید باشد. بنابر این دیگر نیازی به یادگیری دوباره کشف لبه افقی برای تک تک ۵۵×۵۵ مکان مشخص در توده خروجی لایه کانولوشن نیست. یعنی لازم نیست تا این عمل یادگیری کشف لبه افقی برای تک تک مکان ها تکرار شود.

توجه کنید که بعضی اوقات فرض اشتراک پارامتر ممکن است منطقی بنظر نیاید. خصوصا زمانی که تصاویر ورودی به یک شبکه کانولوشن دارای ساختارهای وسط چین (specific centered structure**) خاص باشند، که ما باید انتظار داشته باشیم ، بعنوان مثال ویژگی های مختلفی از یک طرف تصویر نسبت به طرف دیگر یاد گرفته شود. یک نمونه عملی از این مورد زمانی است که تصاویر صورت وسط چین شده اند. ممکن است ما انتظار داشته باشیم که ویژگی های مختلف مخصوص مو (**hair specific**) و یا مخصوص چشم (**eye-specific**) بتوانند (و باید) در موقعیتهای مکانی مختلف یاد گرفته شوند. در این حالت معمولا رایج است که طرح اشتراک پارامتر را کنار گذاشته و بجای آن خیلی ساده به آن لایه بصورت محلی متصل (** Locally connected layer**) گفته شود.**

برای اینکه مطالبی که تا بحال بیان شد را بهتر درک کنیم در اینجا سعی می کنیم مفاهیم بیان شده را در قالب کد بیان کنیم . برای اینکار از زبان پایتون استفاده میکنیم که امروزه یکی از زبانهای بسیار پرکاربرد خصوصا در زمینه Deep learning و شبکه های کانولوشن است.

ما برای راحتی هرچه بیشتر از کتابخانه Numpy استفاده میکنیم که یکی از کتابخانه های مشهور پایتون در زمینه کار با عملیاتهای برداری است. سینتکس این کتابخانه هم شبیه به متلب بوده و کسانی که با متلب آشنایی دارند براحتی دستورات معادل را درک می کنند.

فرض کنید توده ورودی ما یک آرایه با نام X باشد. در اینصورت :

برای بدست آوردن یک depth column در موقعیت (y،x) از دستور `X[x,y,:]` استفاده می کنیم.

برای بدست آوردن یک برش عمقی (depth slice) یا بطور مساوی همان نگاشت فعال سازی(activation map) در عمق d از دستور `X[:,:,d]` استفاده میکنیم.

فرض کنید توده ورودی X دارای ابعاد X.shape: (11,11,4) **باشد این دستور یعنی توده ای با اندازه** 11x11x4 **ایجاد شود که به معنای عرض و ارتفاع ۱۱ و عمق ۴ میباشد. حال فرض کنید ما از** Zero padding **استفاده نکنیم (یعنی** P=0 **) و اندازه فیلتر (یا همان ناحیه ادراکی) برابر با** F=5 **و** stride **هم برابر** S=2 **باشد. با توجه به این اطلاعات ما میتوانیم اندازه مکانی (یعنی عرض و ارتفاع) توده خروجی را بدست بیاوریم . اندازه توده خروجی برابر ۴=۲+۱/(۵-۱۱) خواهد بود که به معنای توده ای با عرض و ارتفاع ۴ است . نگاشت فعال سازی** (Activation map) **نیز در توده خروجی (که از این به بعد به آن** V **میگوییم) بصورت زیر خواهد بود (دقت کنید ما تنها بعضی از محاسبات را در مثال زیر قید میکنیم)**

- V[0,0,0] = np.sum(X[:5,:5,:] * W0) + b0
- V[1,0,0] = np.sum(X[2:7,:5,:] * W0) + b0
- V[2,0,0] = np.sum(X[4:9,:5,:] * W0) + b0
- V[3,0,0] = np.sum(X[6:11,:5,:] * W0) + b0

ذکر این نکته ضروری است که در numpy **عملیات ضرب (*) که در بالا شاهد آن هستید عمل ضرب خانه به خانه** (element wise) **را در آرایه انجام میدهد. نکته دیگر آنکه بردار وزن** W0 **بردار وزن مربوط به نورون** 0 **بوده و** b0 **هم بایاس مربوط به آن میباشد. در اینجا فرض کردیم** W0 **ابعادی بصورت** 5x5x4 **دارد .** w0.shape:(5,5,4) **، چرا که اندازه فیلتر (یا همان ناحیه ادراکی) برابر با ۵ بوده و عمق توده ورودی نیز برابر با ۴ است. توجه کنید که در اینجا ما همانند شبکه های عصبی معمولی ضرب نقطه ای را انجام می دهیم. همچنین در اینجا مشاهده میکنید که ما در حال استفاده از یک وزن و بایاس هستیم (بواسطه اشتراک پارامتر) و ابعاد در راستای عرض در گام های ۲ تایی (مقدار** Stride**) افزایش پیدا میکنند. به منظور ساختن نگاشت فعال سازی**(Activation map) **دوم در توده خروجی بصورت زیر عمل می کنیم :**

- V[0,0,1] = np.sum(X[:5,:5,:] * W1) + b1
- V[1,0,1] = np.sum(X[2:7,:5,:] * W1) + b1
- V[2,0,1] = np.sum(X[4:9,:5,:] * W1) + b1
- V[3,0,1] = np.sum(X[6:11,:5,:] * W1) + b1

نمونه ای از حرکت در راستای y**:**

- V[0,1,1] = np.sum(X[:5,2:7,:] * W1) + b1

نمونه ای از حرکت در هر دو راستای x **و**y**:**

- V[2,3,1] = np.sum(X[4:9,6:11,:] * W1) + b1

جایی که مشاهده می کنید ما در حال اندیس گذاری در بُعد عمقی دوم در V (اندیس ۱) هستیم به این خاطر است که ما در حال محاسبه نگاشت فعال سازی دوم (Activation map) هستیم و به همین دلیل مجموعه متفاوتی از پارامترها (W1) باید استفاده شود. در مثال بالا ما بخاطر خلاصه سازی از نوشتن تمامی عملیات هایی که در لایه کانولوشن برای پر کردن باقی بخش های آرایه خروجی V صورت میگیرد پرهیز کردیم. علاوه بر آن بیاد داشته باشید که این نگاشتهای فعال سازی (Activation maps) اغلب بصورت خانه به خانه توسط یک تابع فعال سازی مثل ReLU مورد پردازش قرار میگیرند (تا خروجی نهایی تشکیل شود) اما این مسئله در اینجا نشان داده نشده است.

خلاصه این بخش

در اینجا بصورت خلاصه مطالبی که در بالا به آن پرداختیم را مرور می کنیم:

یک شبکه کانولوشن

یک توده با اندازه $W_1 \times H_1 \times D_1$ را بعنوان ورودی دریافت میکند که W نشانگر عرض ، H نشانگر ارتفاع و D نشانگر عمق آن میباشد.

نیازمند ۴ فراپارامتر(Hyper parameter) است :

- تعداد فیلتر ها K
- اندازه وسعت مکانی(اندازه x،y) فیلترها)اندازه ناحیه ادراکی) F
- اندازه گام یا Stride S
- مقدار Zero padding P

تولید یک توده خروجی با اندازه $W_2 \times H_2 \times D_2$ که :

$$W2=(W1-F+2P)/S+1$$

$H2=(H1-F+2P)/S+1$ (یعنی عرض و ارتفاع هر دو بطور مساوی بصورت متقارن محاسبه میشوند)

$$D2=K$$

با اشتراک پارامتر، دارای F.F.D۱ وزن به ازای هر فیلتر بوده که در کل (F.F.D1).K وزن و K بایاس میباشد.

برش dام (با اندازه W2 X H2) ،در توده خروجی نتیجه انجام یک ضرب (convolution) معتبر بین فیلتر dام با توده ورودی با stride S بوده که سپس با بایاس d ام جمع شده است.

یک نمونه از مقادیر رایج برای فراپارامترهای توضیح داده شده در بالا بصورت P=1،S=1،F=3 است. البته شیوه ها و قوانین متداولی وجود دارند که باعث رسیدن به این مقادیر میشوند. (بخش معماری شبکه کانولوشن در ادامه را برای اطلاعات بیشتر ببینید)

در این قسمت ما نحوه فعالیت صورت گرفته در یک لایه کانولوشن را گام بگام توسط تصویر نشان میدهیم . بخاطر اینکه نمایش حجم های سه بعدی کمی مشکل است تمام توده های سه بعدی (توده ورودی (با رنگ آبی) ، توده وزن ها (با رنگ قرمز) و توده خروجی (با رنگ سبز)) بصورت برش هایی نمایش داده شده اند. اندازه توده ورودی برابر با W۱ = ۵، H۱ = ۵،D۱= ۳ میباشد(به ترتیب معرف عرض، ارتفاع و عمق) ، پارامترهای لایه کانولوشن نیز به ترتیب برابر با K=2، F = 3 ، S = 2 و P =1 میباشند که به معنای آن است که ما دو فیلتر با اندازه ۳×۳ داریم که با گام stride S=2 بر روی توده ورودی اعمال میشوند.

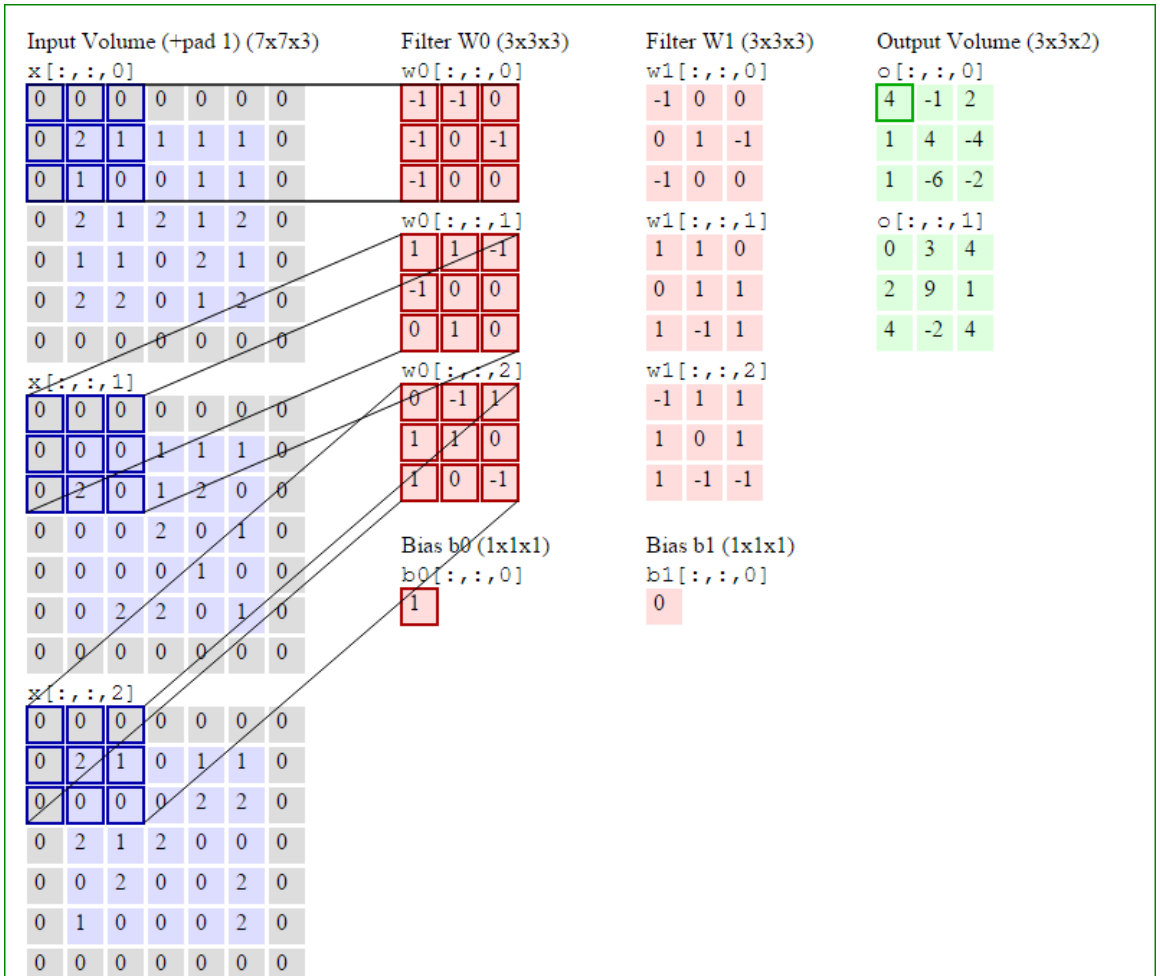
بنابر این اندازه توده خروجی ما هم برابر با ۳=۱+ ۲/(۳+۲–۵) خواهد بود. علاوه بر این ، توجه کنید که عمل Padding با مقدار P=1 بر روی توده ورودی اعمال شده که این عمل باعث صفر شدن مرز های بیرونی توده ورودی شده است. در تصاویر زیر میبینید که تعداد تکرار عملیات به اندازه تعداد عناصر موجود در توده خروجی(به رنگ سبز) است در تصویر مشاهده میکنید که هر عنصر در توده خروجی از ضرب عنصر به عنصر توده وزن (ماتریس وزن برنگ قرمز) با توده ورودی (برنگ آبی) و سپس جمع تمامی عناصر با هم و نهایتا افزودن بایاس به نتیجه نهایی بدست می آید.(Convolution) بعنوان مثال برای عنصر اول توده خروجی داریم :

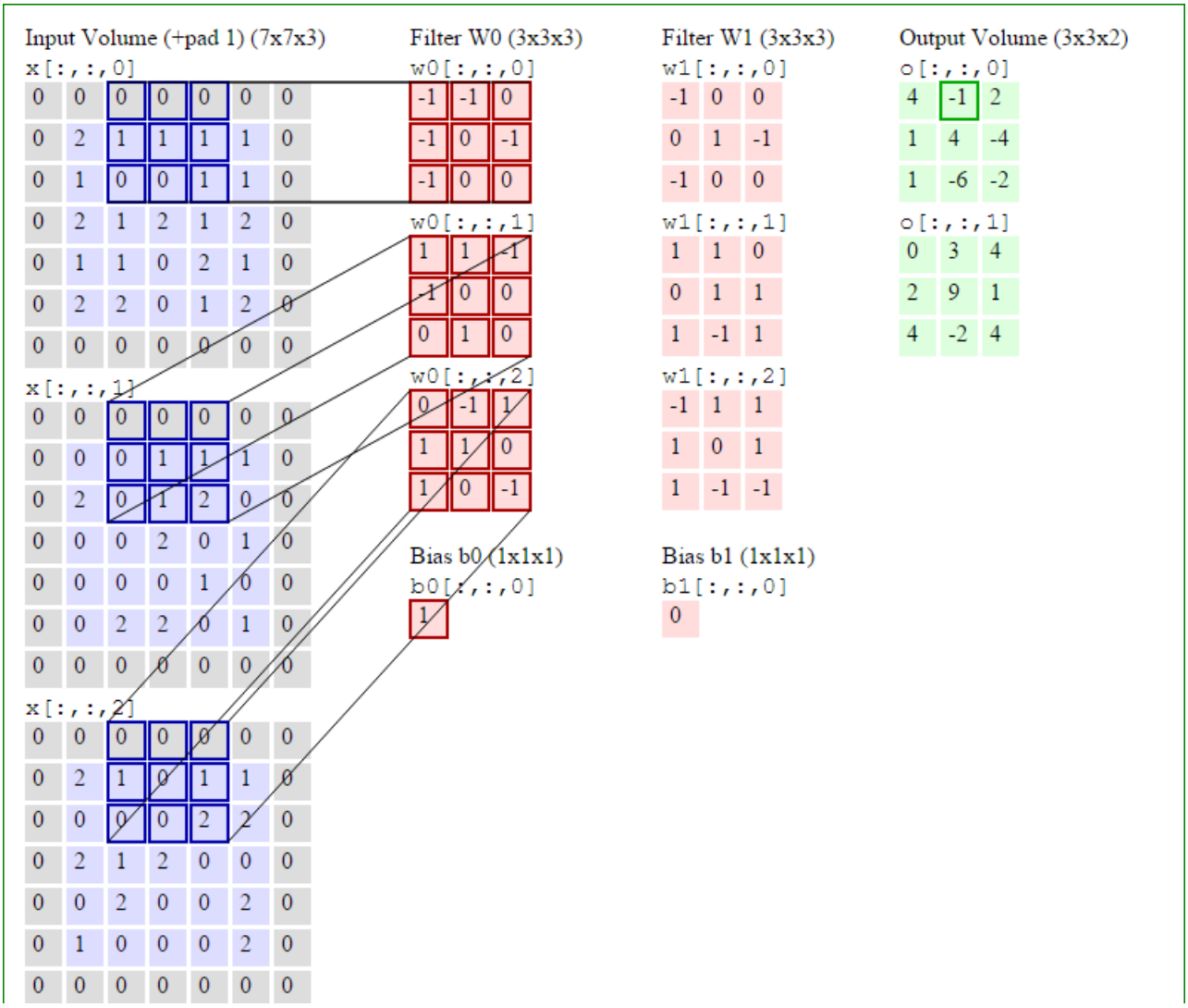
((۰*-۱)+(۰*-۱)+(۰*۰)+(۰*-۱)+(۲*۰)+(۱*-۱)+(۰*-۱)+(۱*۰)+

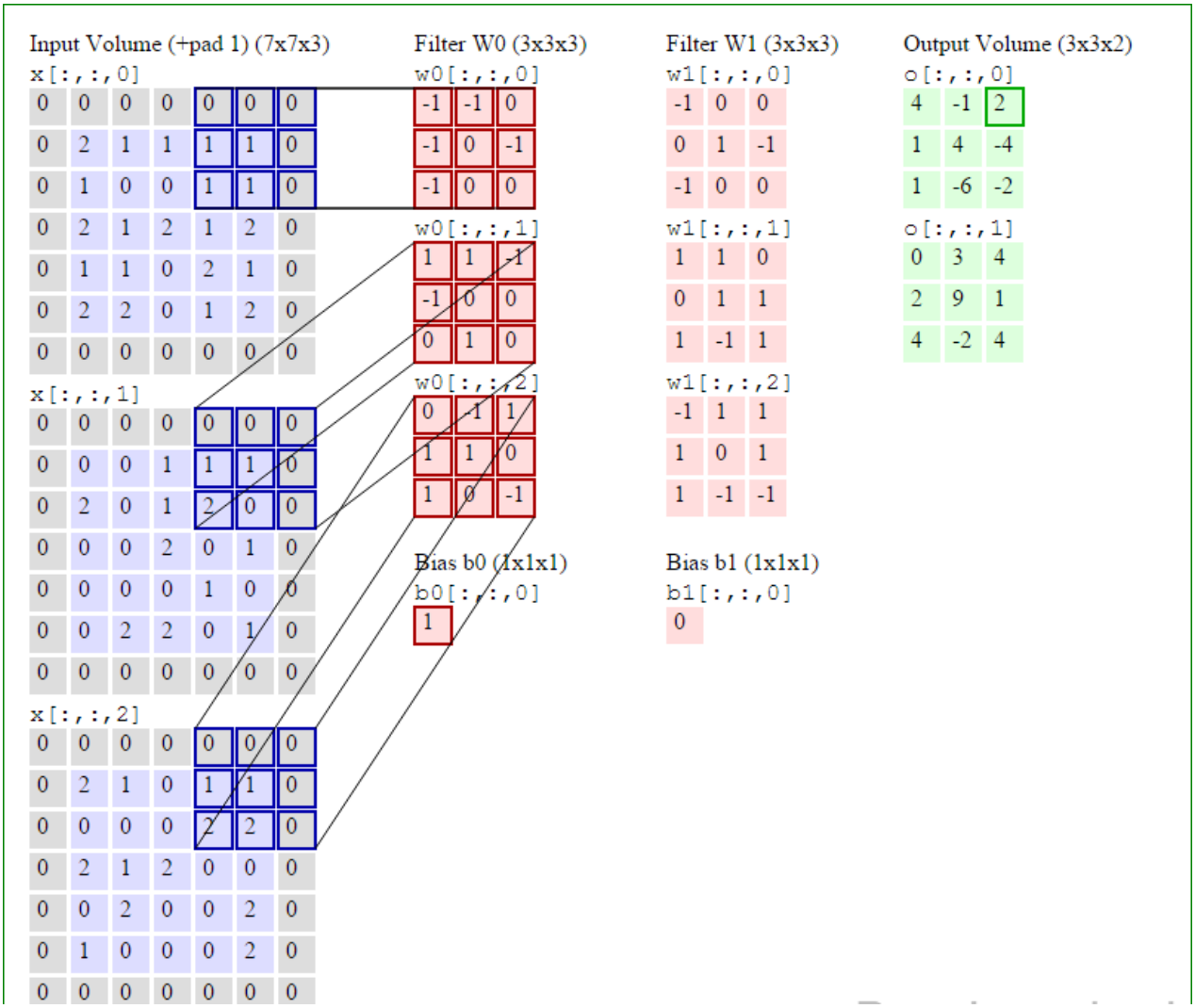
(۰*۰))+((۰*۱)+(۰*۱)+(۰*-۱)+(۰*-۱)+(۰*۰)+(۰*۰)+(۰*۰)+(۲*۱)+

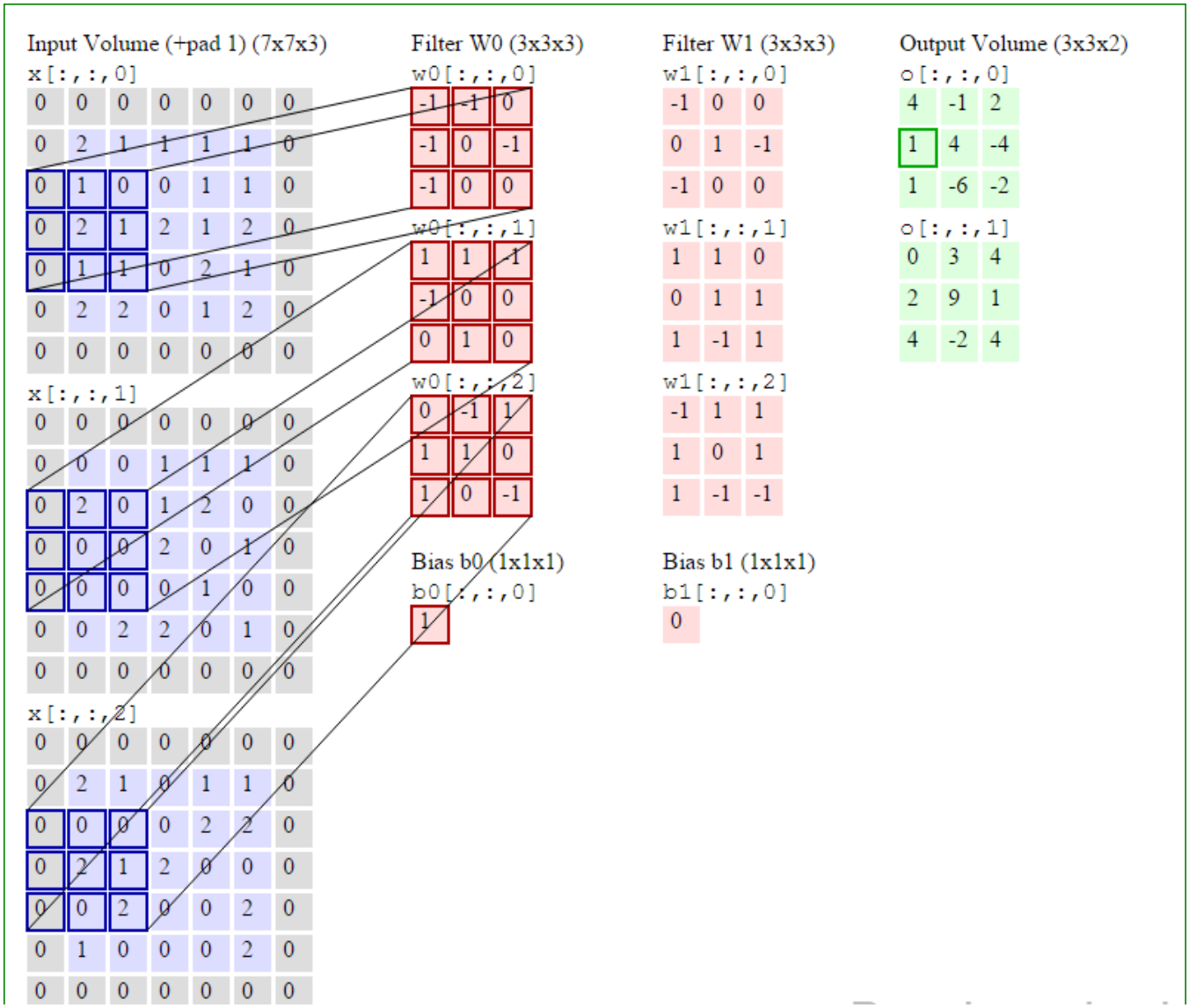
(۰*۰))+((۰*۰)+(۰*-۱)+(۰*۱)+(۰*۱)+(۲*۱)+(۱*۰)+(۰*۱)+(۰*۰)+

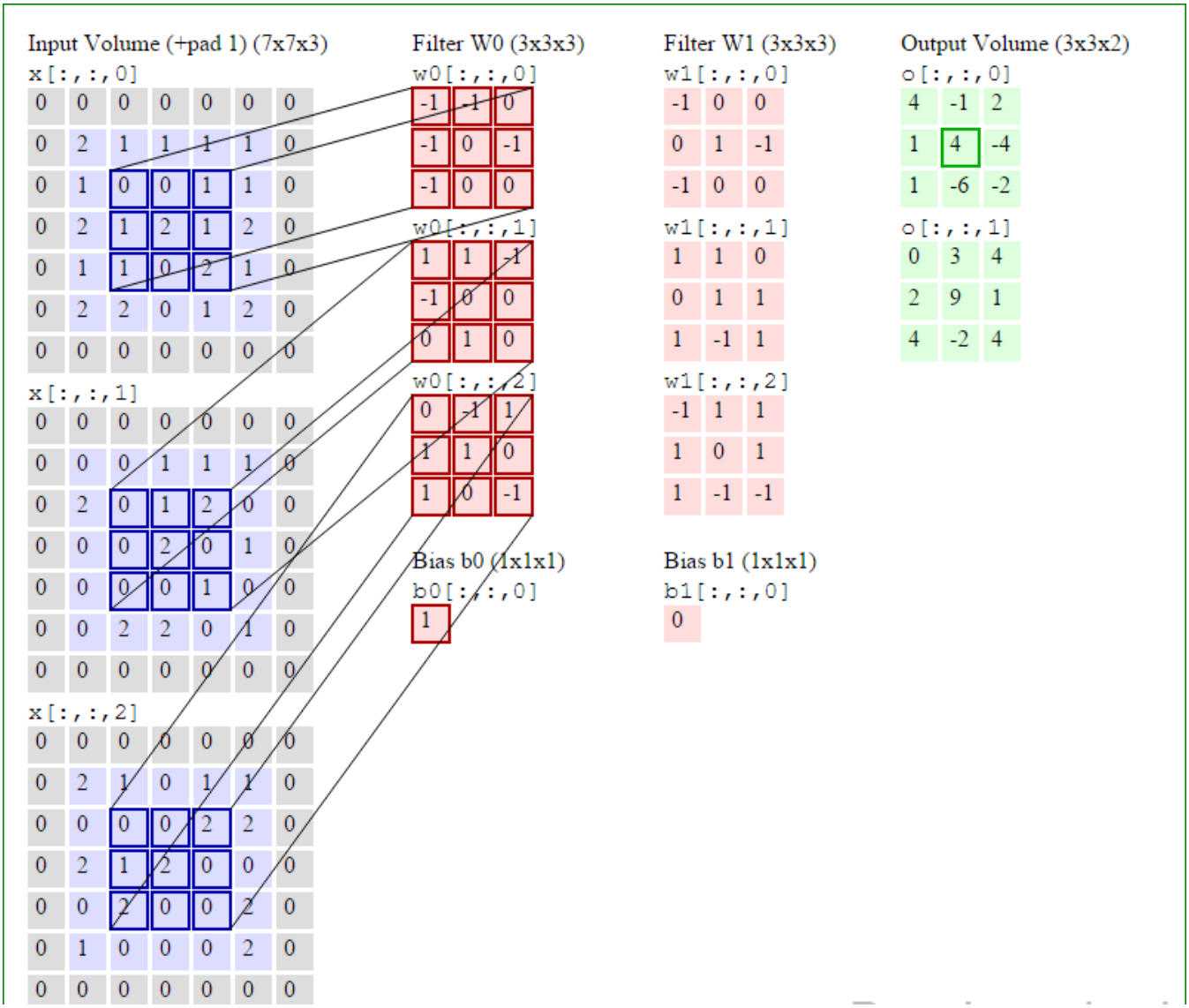
(۰*-۱))+۱=۴

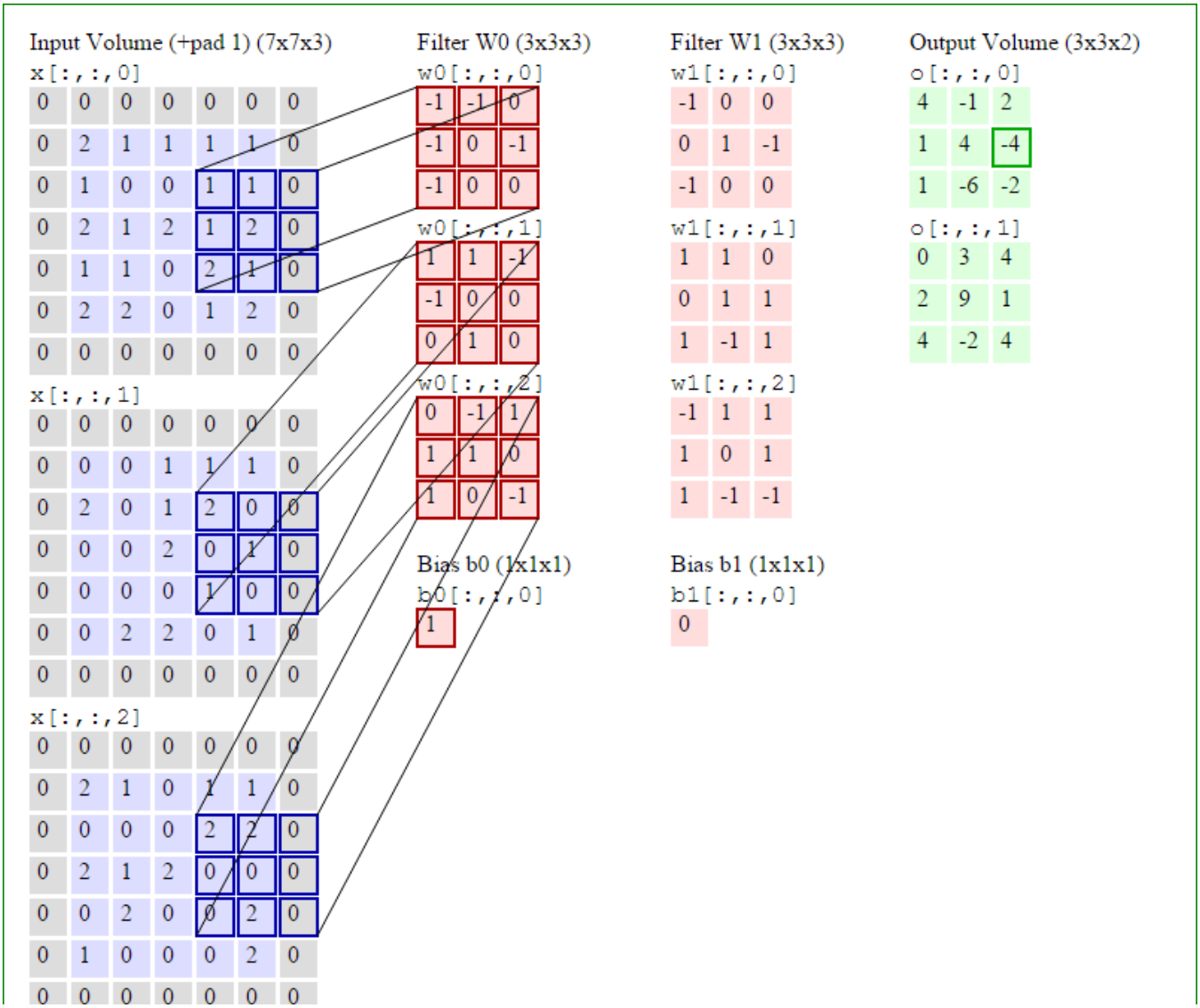

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

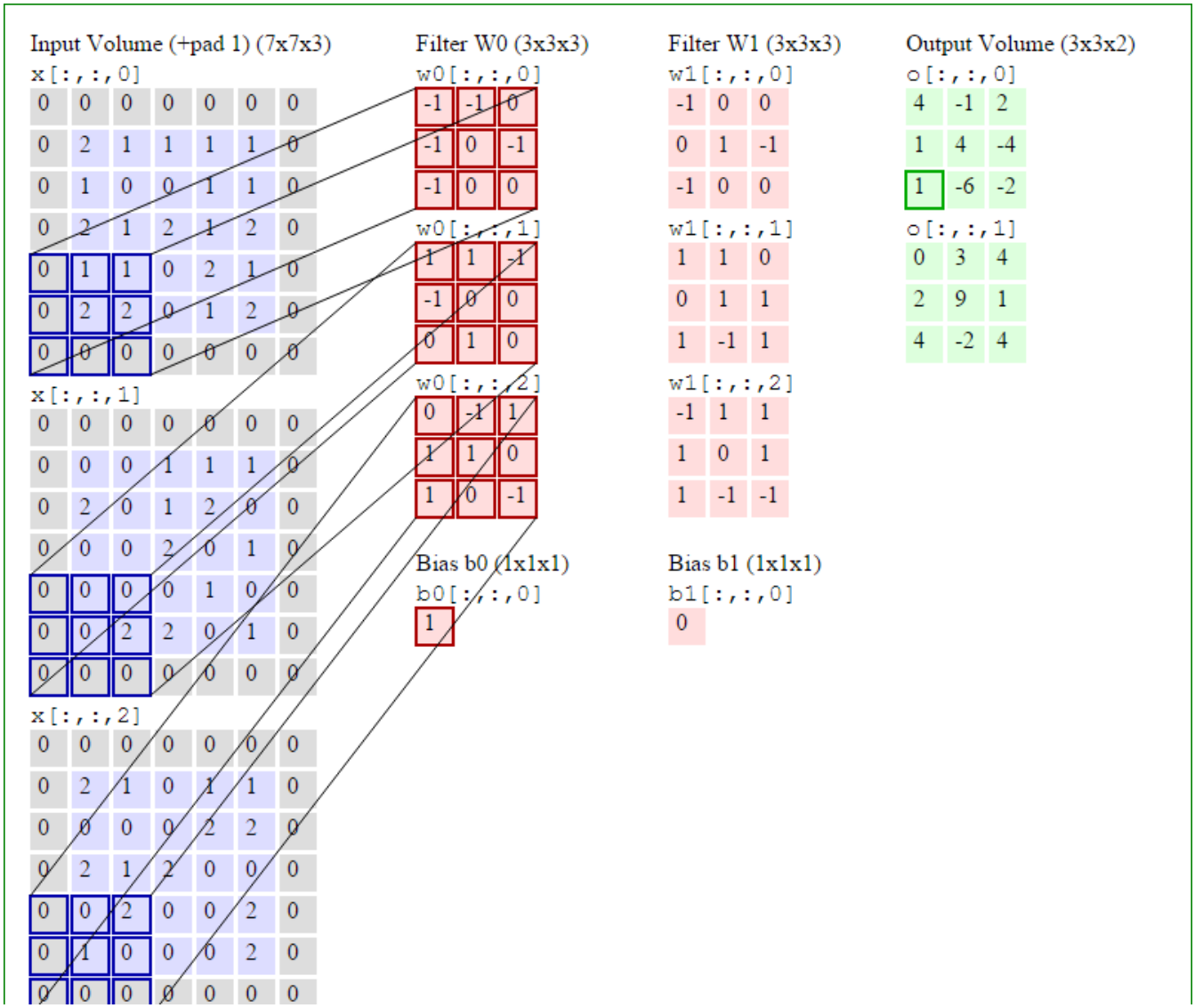

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

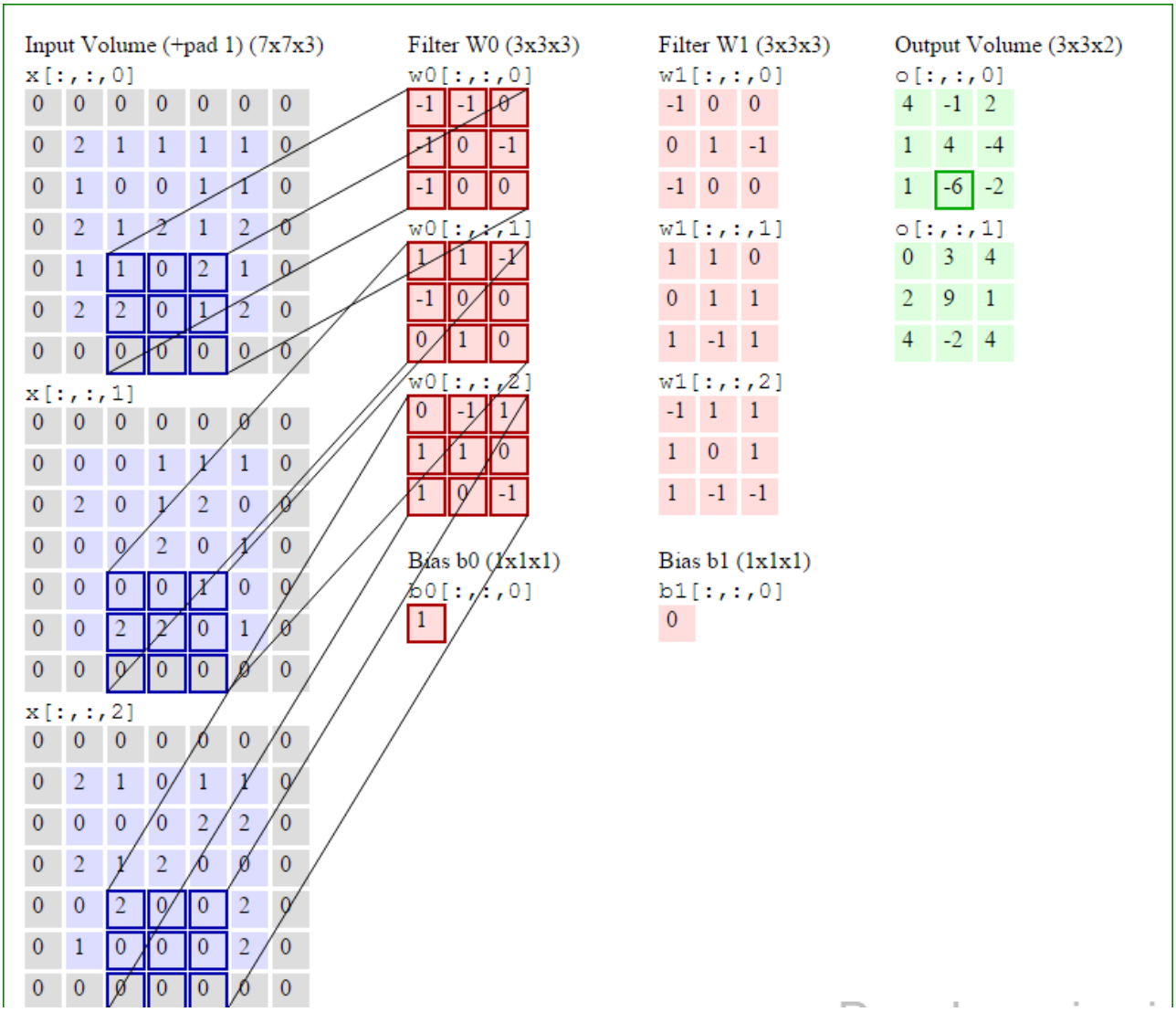

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

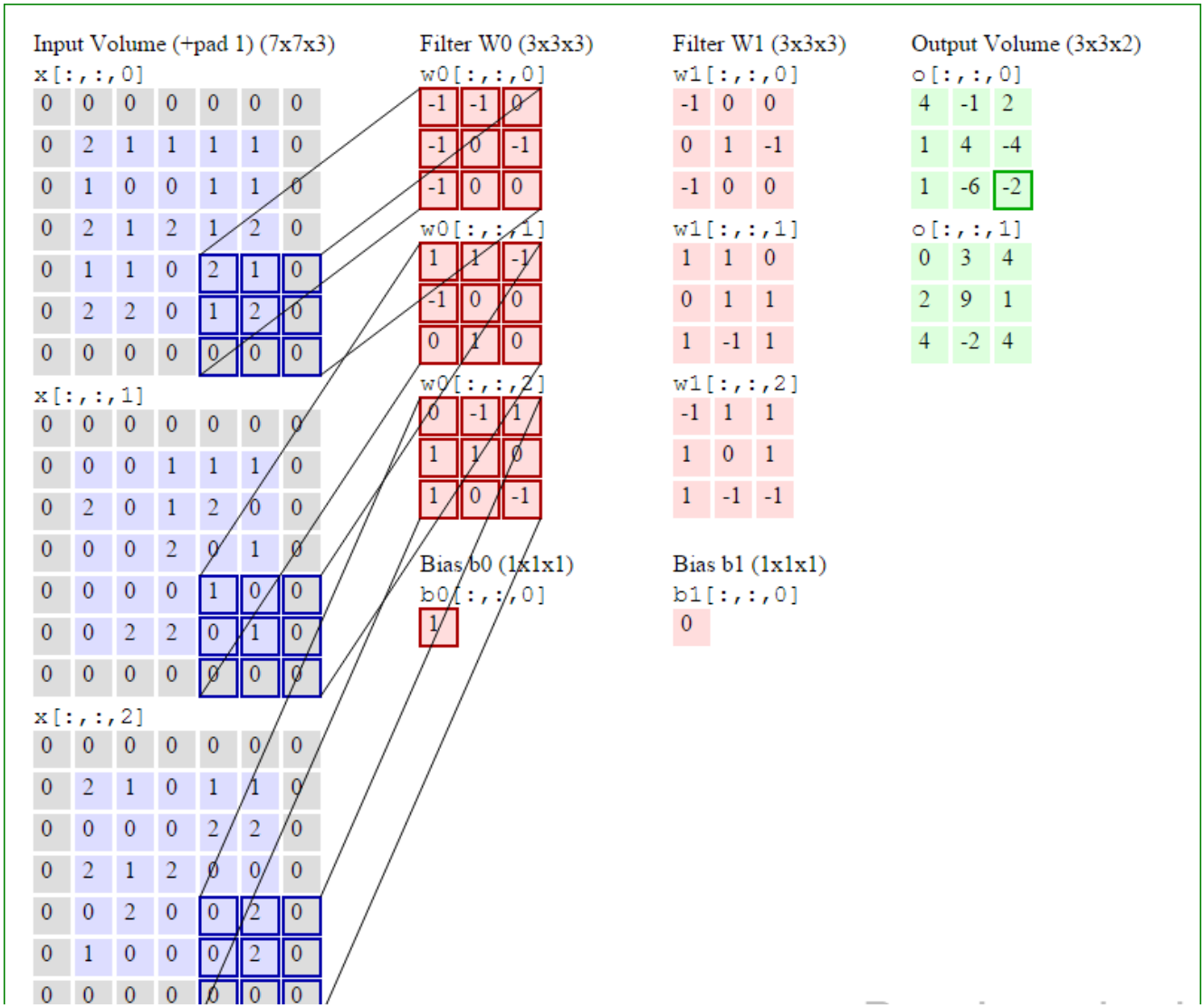

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

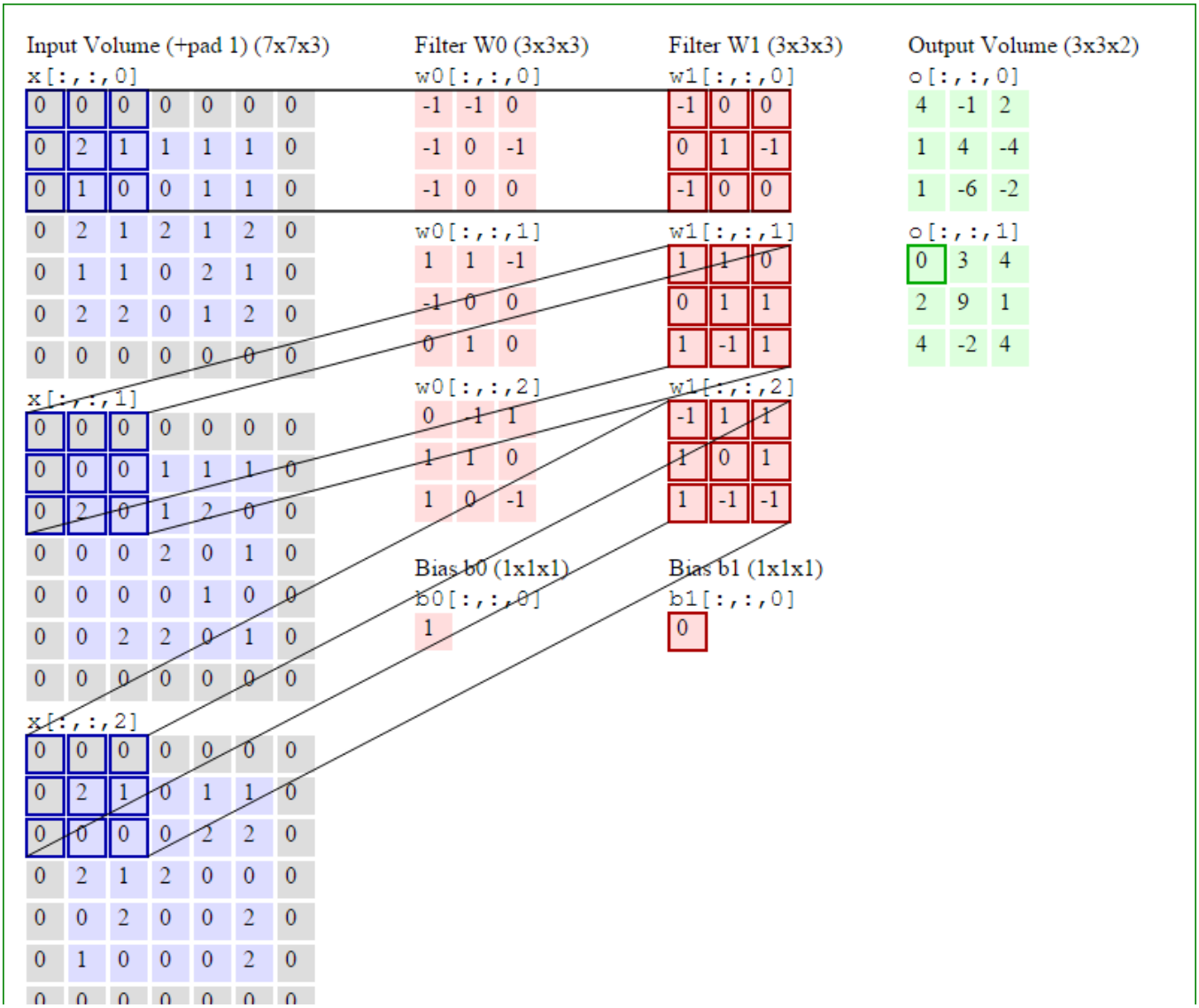

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

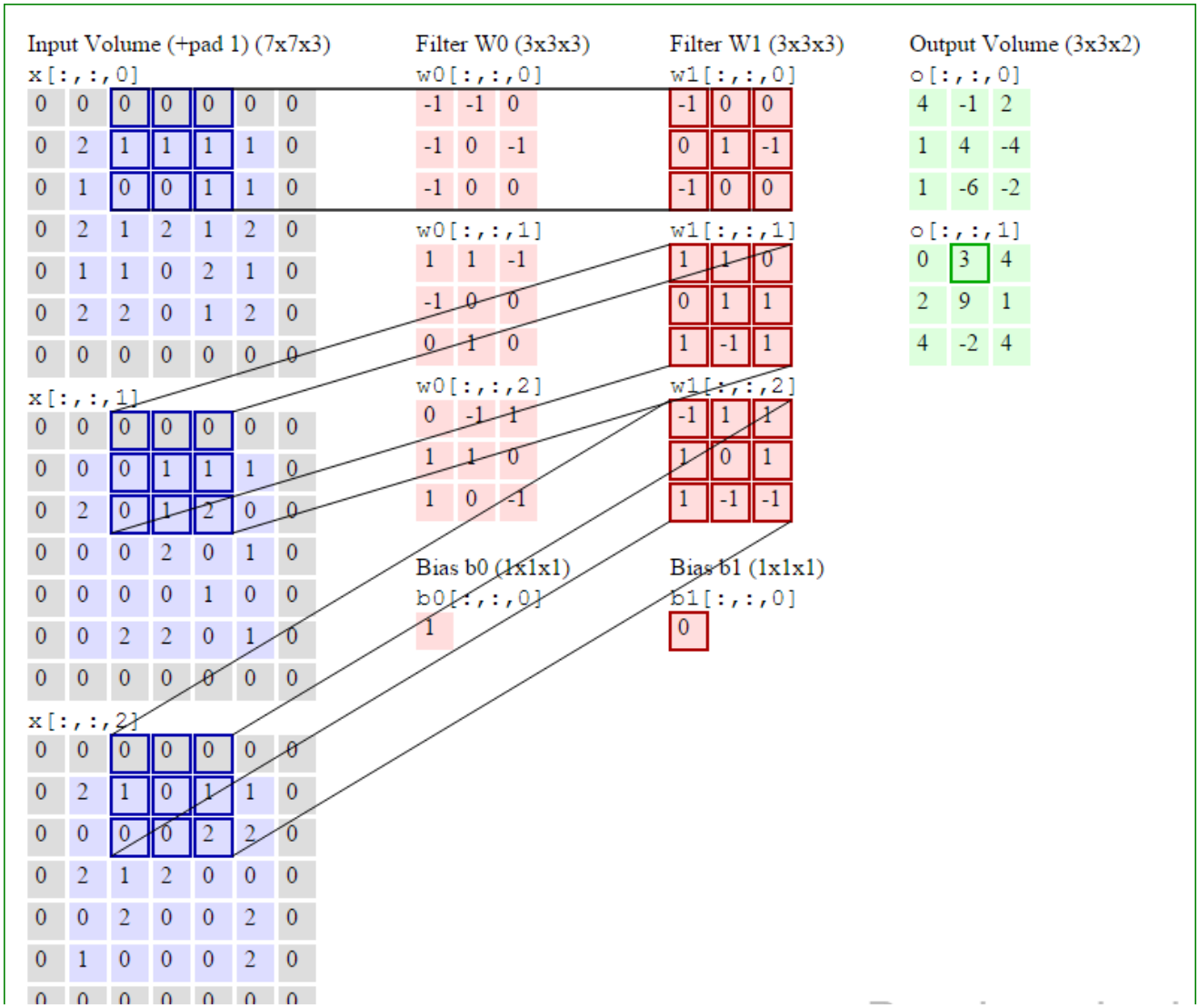

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

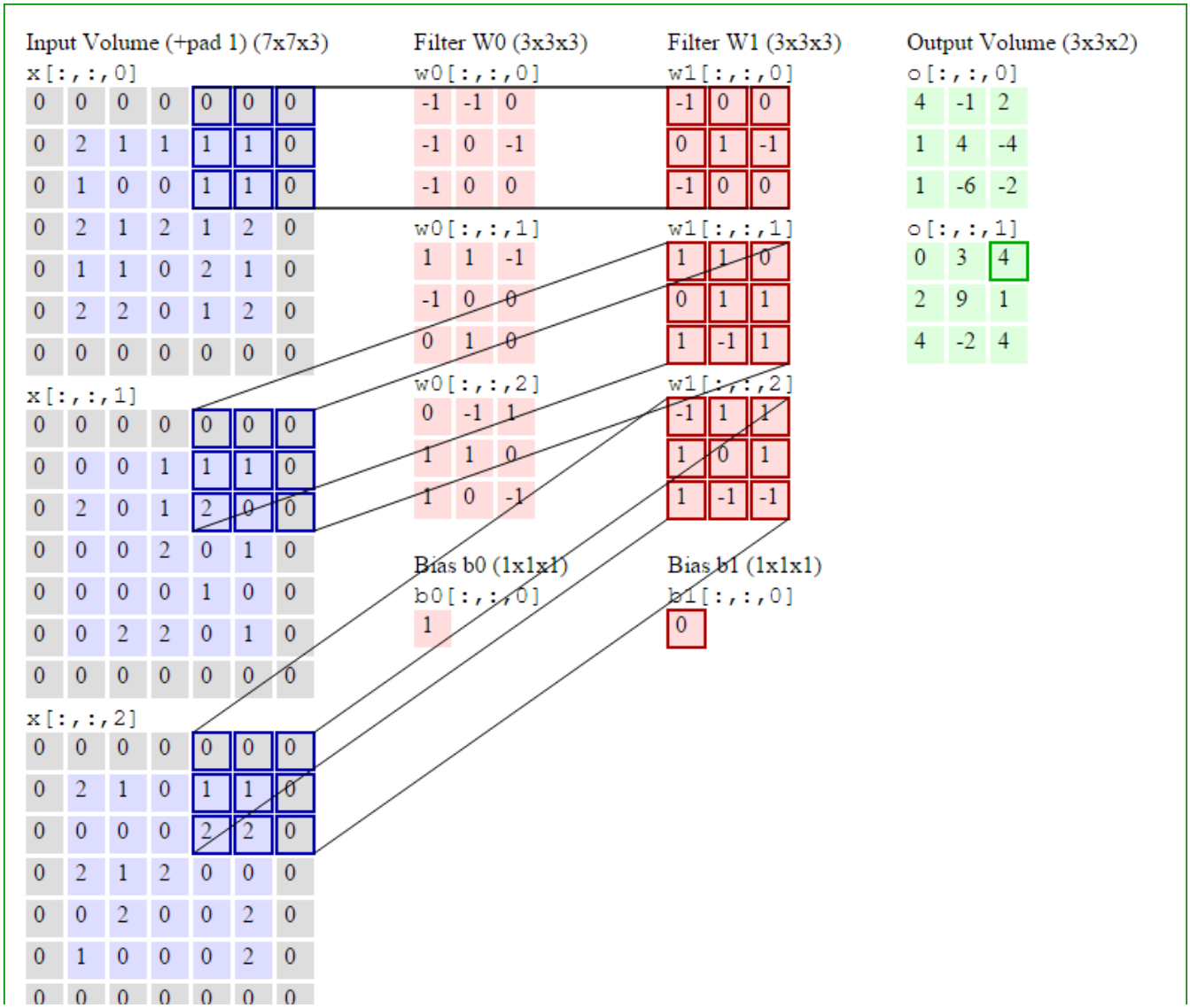

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

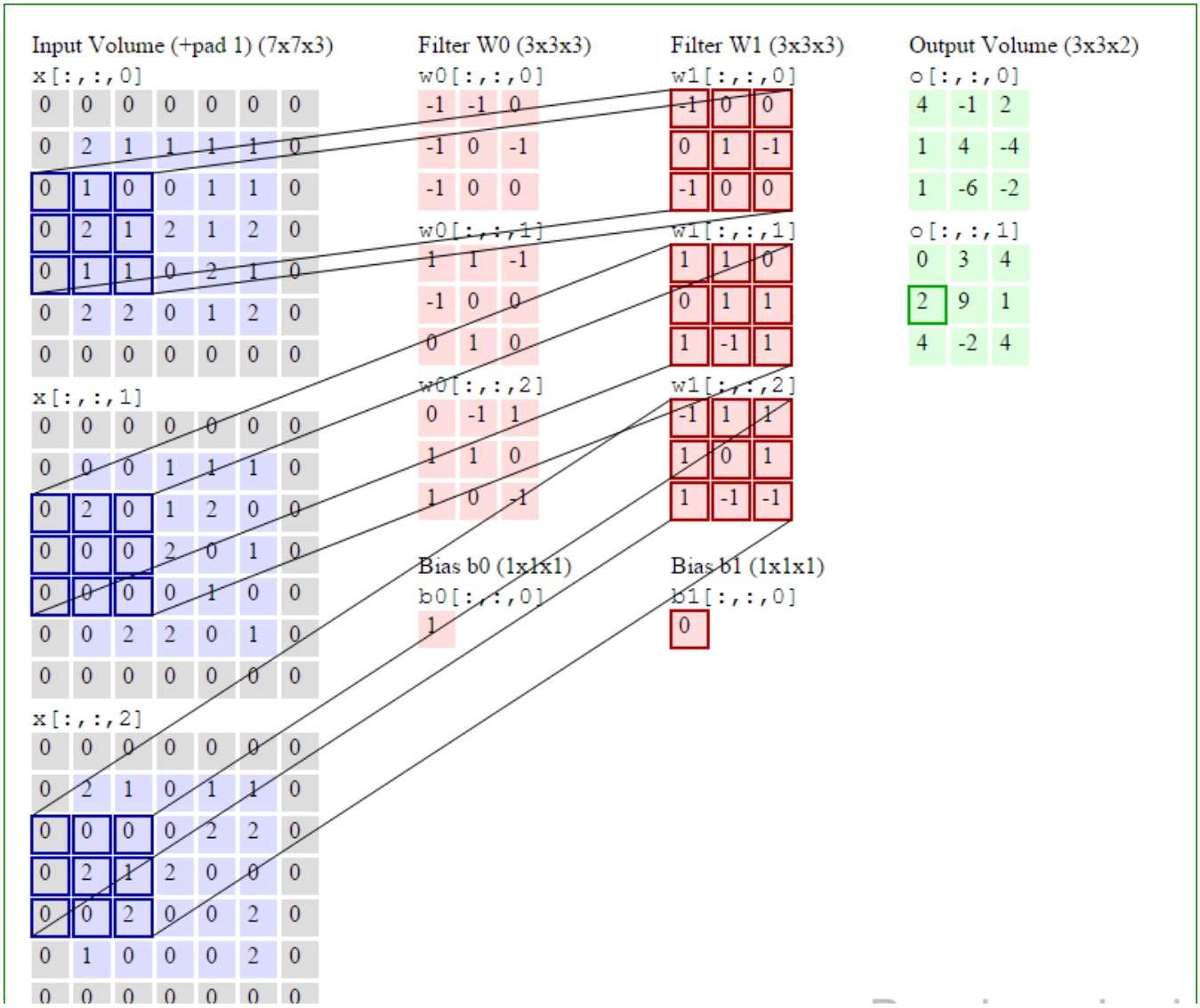

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

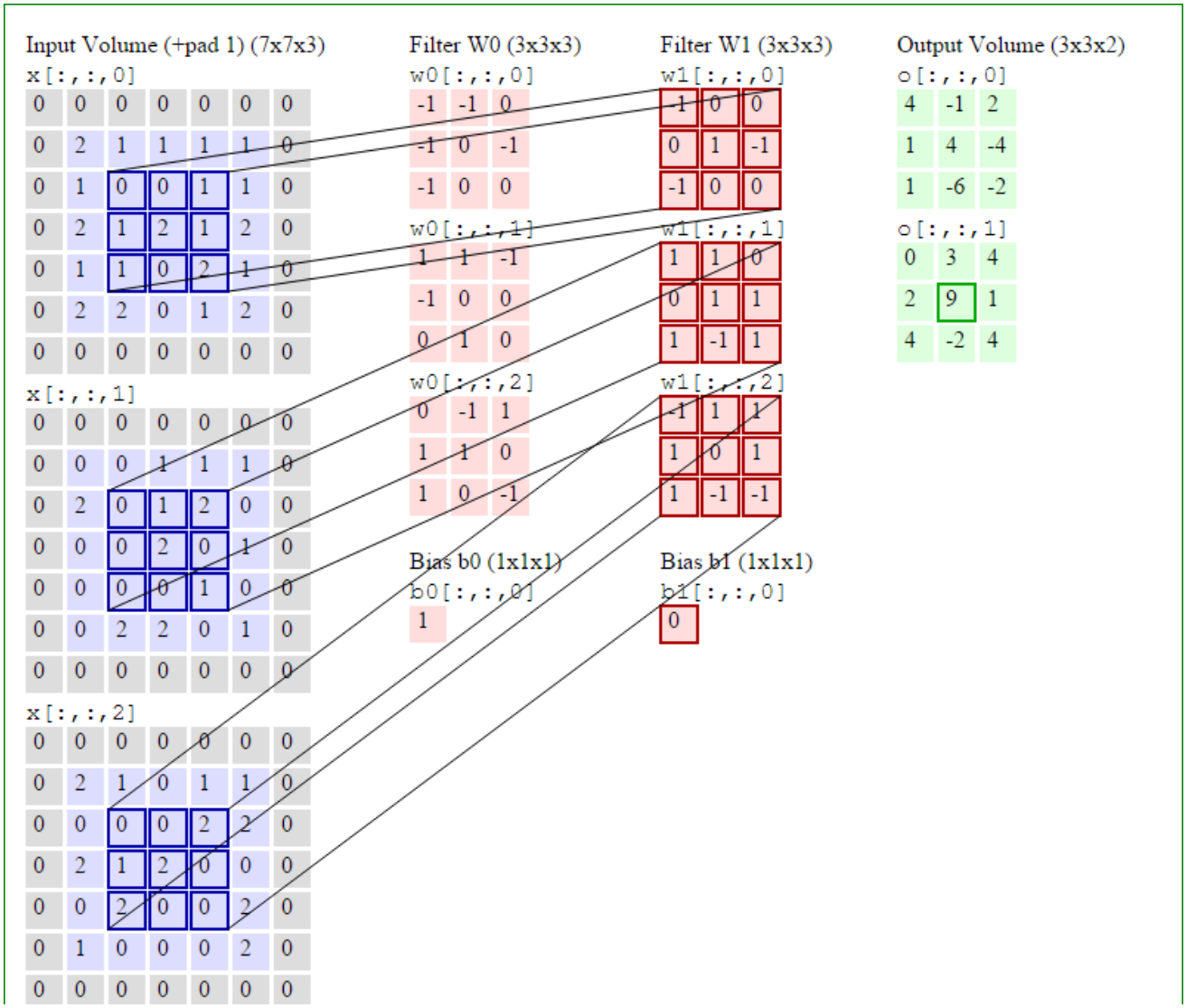

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

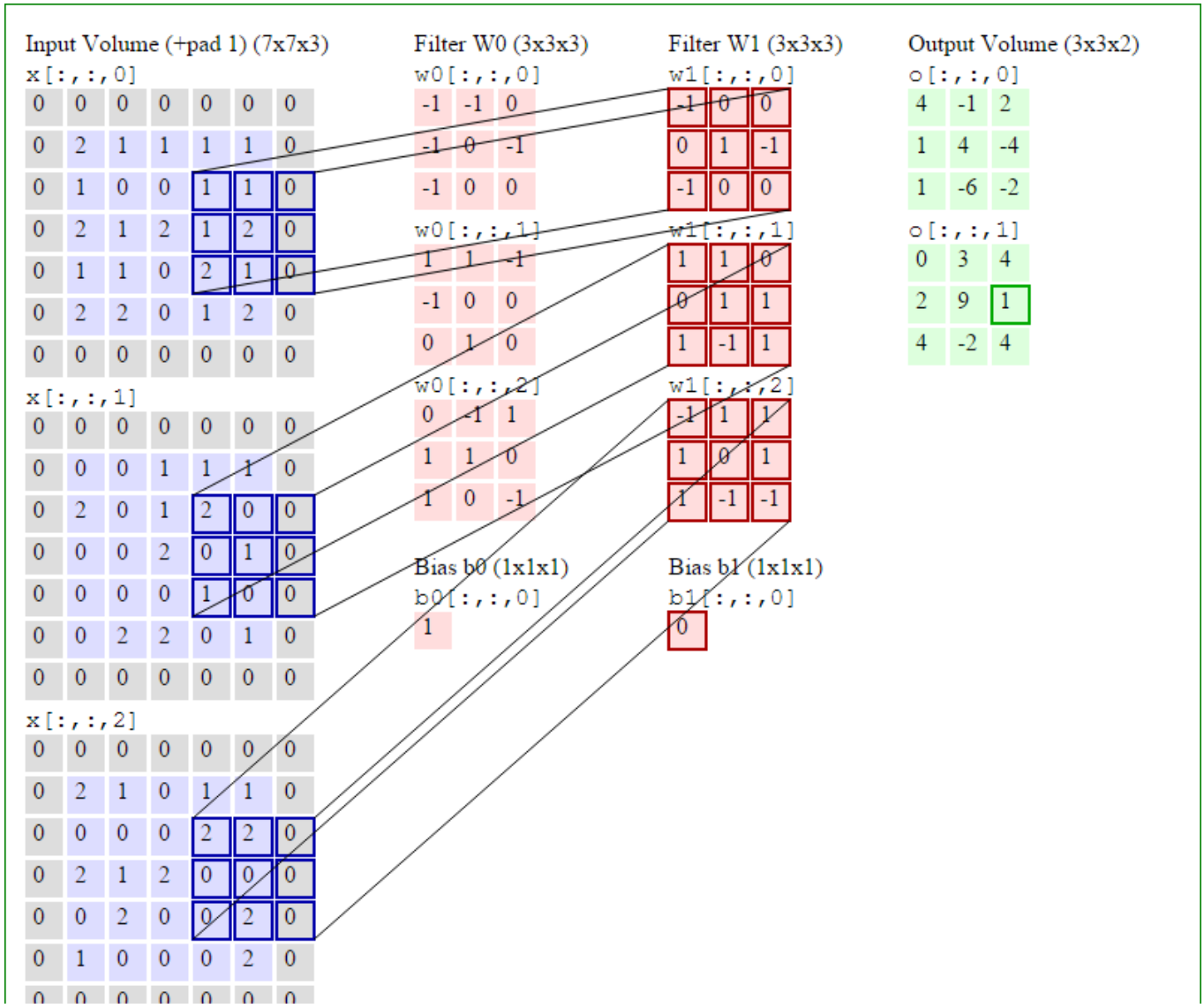

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

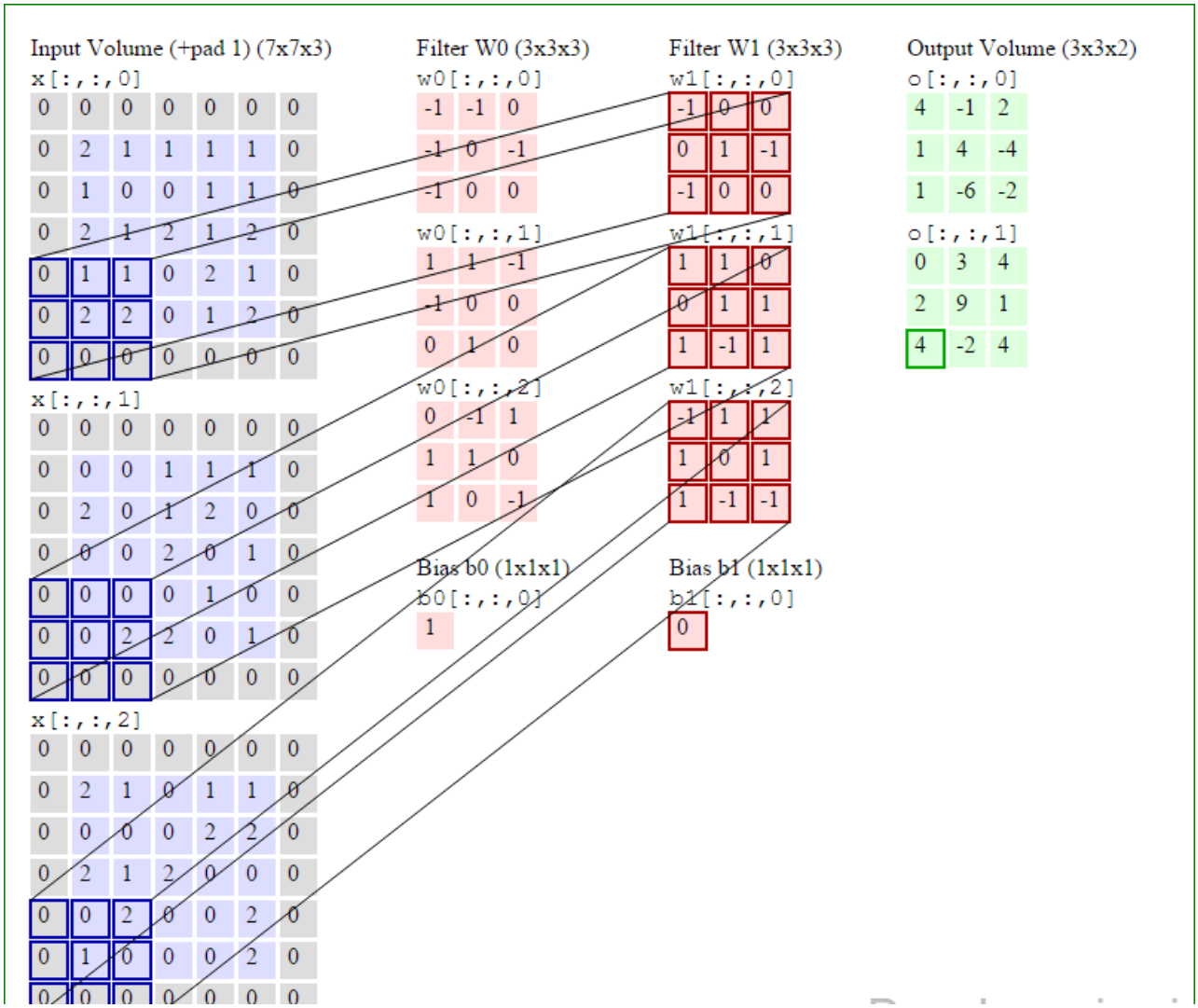

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

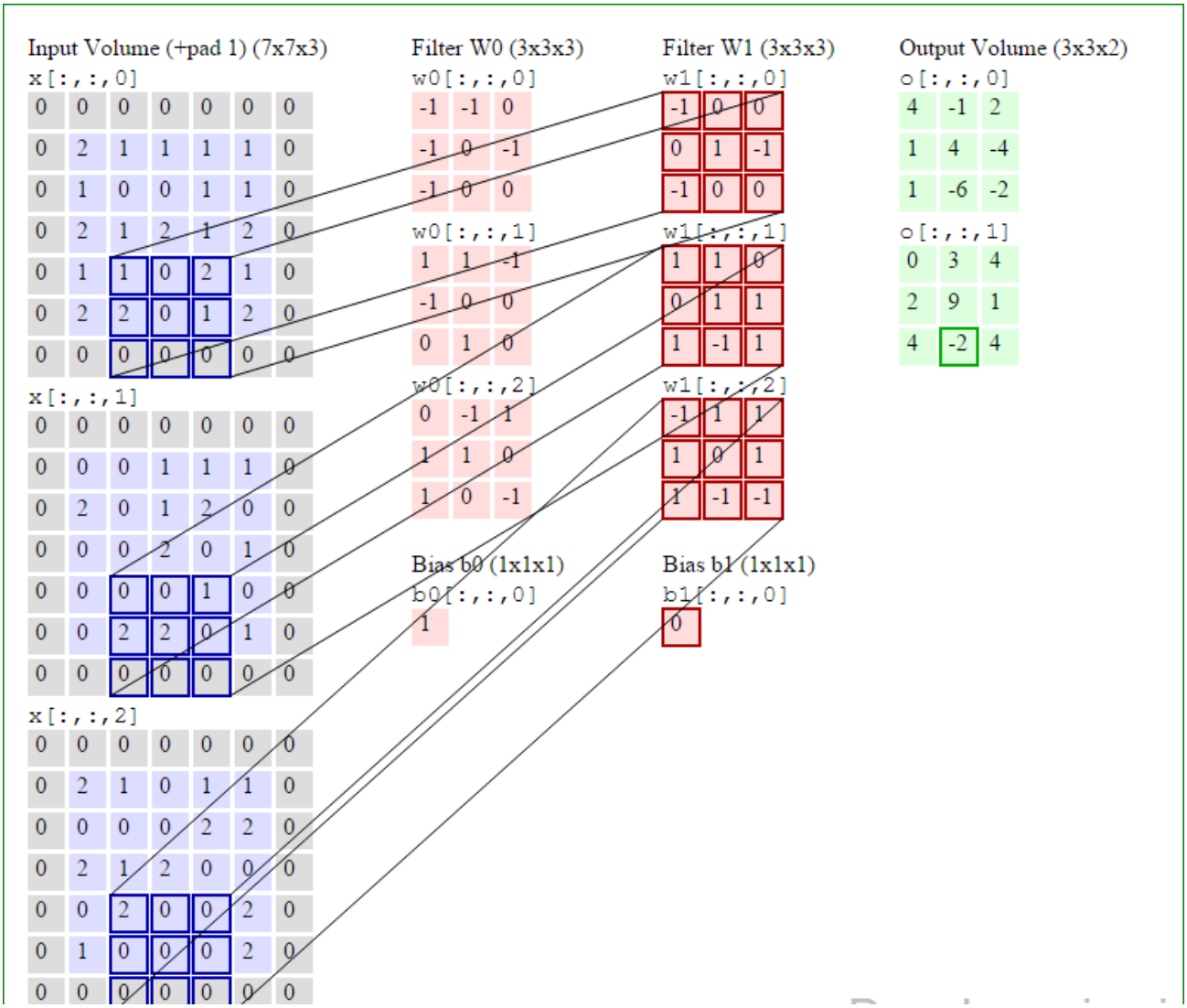

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 1 1 1 0
0 1 0 0 1 1 0
0 2 1 2 1 2 0
0 1 1 0 2 1 0
0 2 2 0 1 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 0 1 1 1 0
0 2 0 1 2 0 0
0 0 0 2 0 1 0
0 0 0 0 1 0 0
0 0 2 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 0 1 1 0
0 0 0 0 2 2 0
0 2 1 2 0 0 0
0 0 2 0 0 2 0
0 1 0 0 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 -1 0
-1 0 -1
-1 0 0
w0[:,:,1]
1 1 -1
-1 0 0
0 1 0
w0[:,:,2]
0 -1 1
1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
0 1 -1
-1 0 0
w1[:,:,1]
1 1 0
0 1 1
1 -1 1
w1[:,:,2]
-1 1 1
1 0 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
4 -1 2
1 4 -4
1 -6 -2
o[:,:,1]
0 3 4
2 9 1
4 -2 4

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	1	1	1	1	0
0	1	0	0	1	1	0
0	2	1	2	1	2	0
0	1	1	0	2	1	0
0	2	2	0	1	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	2	0	1	2	0	0
0	0	0	2	0	1	0
0	0	0	0	1	0	0
0	0	2	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	0	0	0	2	2	0
0	2	1	2	0	0	0
0	0	2	0	0	2	0
0	1	0	0	0	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	-1	0
-1	0	-1
-1	0	0

w0[:,:,1]

1	1	-1
-1	0	0
0	1	0

w0[:,:,2]

0	-1	1
1	1	0
1	0	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	0	0
0	1	-1
-1	0	0

w1[:,:,1]

1	1	0
0	1	1
1	-1	1

w1[:,:,2]

-1	1	1
1	0	1
1	-1	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	-1	2
1	4	-4
1	-6	-2

o[:,:,1]

0	3	4
2	9	1
4	-2	4

پیاده سازی بصورت ضرب ماتریسی

توجه کنید که عملیاتی که در بالا انجام دادیم به کانولوشن معروف است (یا به فارسی ضرب پیچشی (در اصل به همین دلیل است که به این لایه لایه کانولوشن گفته می شود!) این عملیات ضرب نقطه ای(یعنی هر عنصر را با عنصر نظیر خود ضرب کردن) –به آن ضرب مولفه به مولفه هم می گویند– را بین ناحیه محلی ورودی با فیلتر انجام میدهد . یک روش رایج در پیاده سازی لایه کانولوشن بهره بردن از همین اصل است بدین صورت که عملیات forward pass یک لایه کانولوشن را بصورت یک ضرب ماتریسی که شرحش در زیر رفته است به انجام برسانیم و اینگونه پیاده سازی ای بسیار سریع و کارا ارائه کنیم(بجای ضرب ترتیبی عناصر، همه عناصر با هم در یک زمان ضرب شوند و نتیجه حاصل شود)

ابتدا هر کدام از نواحی محلی در تصویر ورودی را به ستون تبدیل میکنیم (این عملیات به im2col **معروف است). بعنوان مثال اگر ورودی ما حجمی بصورت** 227x227x3 **داشته باشد و بخواهیم آنرا با فیلترهایی با اندازه** 11x11x3 **به همراه** stride s=4 **ضرب پیچشی** (convolve) **کنیم(نکته در انتهای متن و کامنت رو ببینید)، بلوکهایی با اندازه** 11x11x3 **پیکسل را به یک بردار ستونی با اندازه** 11x11x3=363 **تبدیل میکنیم . با تکرار این عمل بر روی ورودی با گام** stride S=4 **ناحیه در طول عرض و ارتفاع بدست میدهد که نهایتا باعث ایجاد ماتریس خروجی** X_col **با اندازه** [۳۶۳×۳۰۲۵] **خواهد شد که هر ستون این ماتریس یک ناحیه ادراکی** (receptive field) **بوده و تعداد** ۵۵×۵۵=۳۰۲۵ **نمونه از آنها در کل وجود خواهد داشت. دقت کنید از انجایی که نواحی ادراکی با هم اشتراک دارند** (overlap) **هر عدد در توده خروجی ممکن است چندین بار در ستونهای مختلف تکرار شود.**

وزنهای شبکه کانولوشن هم به همین شکل به بردار های سطری تبدیل میشوند. بعنوان مثال اگر ۹۶ فیلتر(منظور مجموعه وزنها میباشد) با اندازه 11x11x3 **وجود داشته باشد (تعداد وزنها برابر با تعداد عناصر موجود در ناحیه ادراکی است یعنی اگر اندازه ناحیه ادراکی برابر با** 11x11x3 **است بنابر این به همین تعداد وزن نیز وجود دارد) این عمل باعث ایجاد ماتریس** W_row **با اندازه** [۳۶۳×۹۶] **خواهد شد.**

حالا نتیجه عملیات کانولوشن (ضرب نقطه به نقطه) برابر با اجرای یک ضرب ماتریسی بزرگ خواهد بود. (X_col،np.dot(W_row). **این عملیات نتیجه ضرب نقطه ای بین تمام فیلترها و تمام نقاط نواحی ادراکی را بما خواهد داد.**

در مثال ما خروجی این عملیات یک ماتریس با اندازه [96 x 3025] خواهد بود که نتیجه ضرب نقطه ای بین هر فیلتر در هر موقعیت را میدهد.

نهایتا نتیجه بالا بایستی دوباره به فرم صحیح آن بصورت [55x55x96] تغییر شکل داده شود.

نکته منفی این روش مصرف حافظه بالای آن است چرا که بعضی از مقادیر در توده ورودی چندین بار در X_col تکرار شده اند. اما از طرف دیگر نکته مثبتی که این روش دارد این است که پیاده سازی های بسیار بهینه ای از ضرب ماتریس ها در کتابخانه های مختلفی همانند BLAS وجود دارد که میتوان از آنها بهره برد و به کارایی و افزایش سرعت بالایی دست پیدا کرد . علاوه بر آن میتوان از این ایده در عملیات Pooling که در ادامه به آن خواهیم پرداخت نیز استفاده کرد .

عملیات BackPropagation

عملیات Backpropagation برای یک عملیات کانولوشن (هم برای داده و هم وزن ها) هم یک عملیات کانولوشن (اما با فیلترهای از لحاظ مکانی برعکس شده (spatially filipped filters) می باشد.

لایه Pooling

قرار دادن یک لایه Pooling بین چندین لایه کانولوشنی پشت سر هم در یک معماری کانولوشن امری رایج است . کارکرد این لایه کاهش اندازه مکانی (عرض و ارتفاع) تصویر (ورودی) بجهت کاهش تعداد پارامترها و محاسبات در داخل شبکه و بنابر این کنترل overfitting است .لایه Pooling بصورت مستقل بر روی هر برش عمقی از توده ورودی عمل کرده و آنرا با استفاده از عملیات MAX از لحاظ مکانی تغییر اندازه (resize) میدهد. رایجترین شکل استفاده از این لایه به صورت استفاده این لایه با فیلترهایی با اندازه ۲×۲ به همراه stride S=2(گام) است که هر برش عمقی در ورودی را با حذف ۲ عنصر از عرض و ۲ عنصر از ارتفاع کاهش داده و باعث حذف ٪۷۵ مقادیر موجود در آن برش عمقی میشود. هر عملیات MAX در اینجا ماکسیمم بین ۴ عدد (یک ناحیه ۲×۲ در برش عمقی) را بدست میدهد. در اینجا بُعد عمق بدون تغییر باقی میماند.

بطور کلی لایه Pooling یک توده با اندازه W۱ x H۱ x D۱ را بعنوان ورودی دریافت میکند که W نشانگر عرض ، H نشانگر ارتفاع و D نشانگر عمق آن میباشد.

نیازمند ۲ فراپارامتر است :

- **اندازه وسعت مکانی(اندازه y،x) فیلترها)اندازه ناحیه ادراکی) F**
- **اندازه گام یا Stride S**

یک توده خروجی با اندازه ۲W ۲x H ۲x D تولید میکند که :

(یعنی عرض و ارتفاع هر دو بطور مساوی بصورت متقارن محاسبه میشوند)

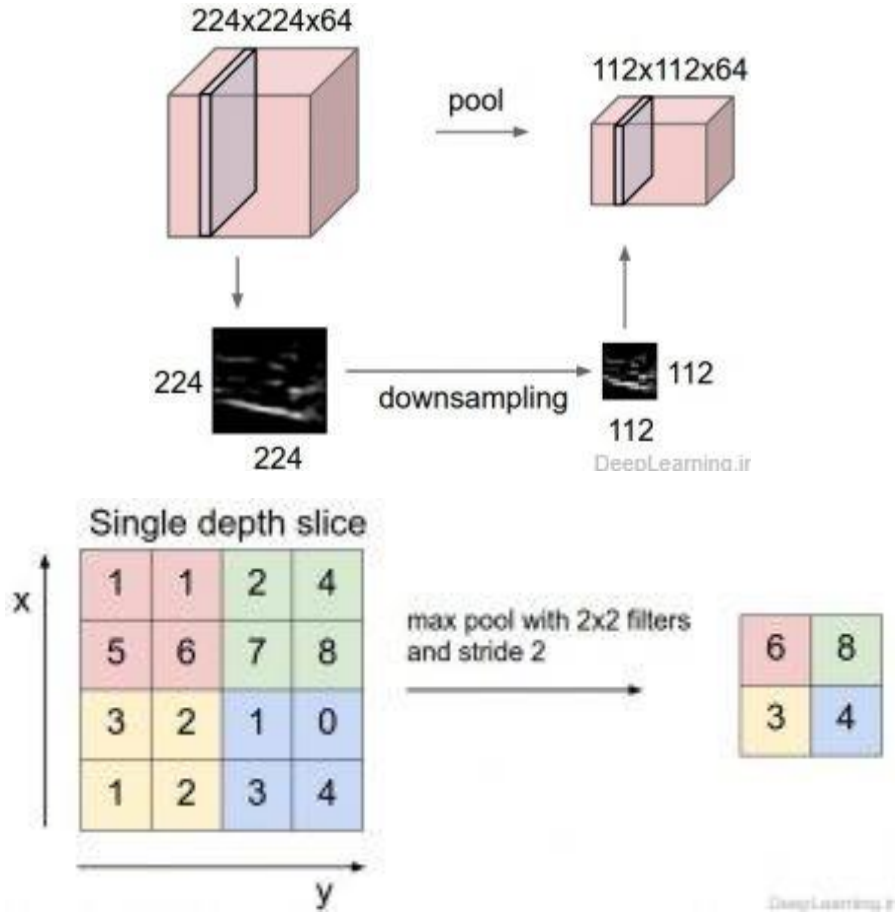
بواسطه اینکه این لایه یک تابع ثابت از ورودی را محاسبه میکند هیچ پارامتری به شبکه اضافه نمی کند.

دقت کنید که استفاده از zero padding در لایه pooling عمومیت ندارد و این کار صورت نمی گیرد.

این نکته حائز اهمیت است که تا بحال در عمل لایه max pooling به دو صورت ،بیشتر رایج بوده و مورد استفاده قرار گرفته است . صورت اول آن با F=3 و S=2 (که به overlapping pooling معروف است) طراحی شده و صورت دوم که رایج تر است با F=2 و S=2 طراحی میشود. عملیات Pooling با نواحی ادراکی بزرگتر بیش از حد مخرب بوده و به همین دلیل در عمل معمولا یکی از دو حالت فوق مورد استفاده قرار میگیرند.

General Pooling

علاوه بر max pooling واحدهای pooling قادر به اجرای توابع دیگری نظیری average pooling و یا حتی L2-norm pooling نیز هستند. Average pooling در ابتدا اغلب مورد استفاده قرار میگرفت تا اینکه اخیرا در قیاس با max pooling که در عمل، عملکرد بهتری از خود ارائه داده است گرایش به آن از بین رفته است.



نمایش لایه max-pooling و چگونگی کاهش ابعاد

لایه Pooling توده ورودی را در هر برش عمقی بصورت مستقل(یعنی هر برش بدون توجه به برش دیگر) از لحاظ مکانی کاهش میدهد (اصطلاحا downsample میکند). تصویر سمت چپ: در این مثال توده ورودی با اندازه ای برابر با [224x224x64] را با اندازه فیلتر F=2 و Stride S=2 به توده خروجی با اندازه [112x112x64] کاهش داده شده است. توجه کنید که عمق توده بدون تغییر حفظ شده است . تصویر سمت راست : رایجترین عملیات downsampling عملیات max است که به همین علت به این لایه ،لایه max pooling گفته میشود اینجا از stride با اندازه ۲ استفاده شده است. به عبارت ساده تر این به این معناست که ماکسیمم ۴ عدد گرفته شده است (یک مربع ۲×۲ کوچک.)

دستاوردهای جدید در این حوزه

Fractional Max Pooling روشی را جهت انجام عملیات pooling پیشنهاد میکند که در آن از فیلترهای کوچکتر از اندازه ۲×۲ استفاده میشود. این روش با تولید تصادفی نواحی pooling همراه با فیلترهایی با اندازه ۱×۱ ، ۲×۱ ، ۱×۲ و یا ۲×۲ به منظور پوشش سطح نگاشت فعال سازی انجام میشود. این Grid ها بصورت تصادفی در هر forward pass ایجاد میشوند و در زمان آزمایش (Test time) میتوان با میانگین گیری از چندین grid به تخمینی در این زمینه دست پیدا کرد (state of the art دیتاست Cifar10 با این روش بدست آمده است!).

Striving for Simplicity: The All Convolutional Net پیشنهاد حذف لایه pooing در ازای استفاده از معماری ای که در آن تنها از لایه های تکراری کانولوشن استفاده شده است را میدهد. به منظور کاهش اندازه تصویر، آنها استفاده از گام های بزرگتر در لایه کانولوشن را بصورت هر از چندگاهی، پیشنهاد می کنند.

بواسطه کاهش شدید در اندازه تصویر[۱] (که تنها برای دیتاست های کوچک برای کنترل با overfitting سودمند است) ، گرایش مقاله ها به سمت حذف لایه Pooling در شبکه های کانولوشن جدید است.

در زیر لیستی از لایه های جدید این حوزه رو می بینید:

لایه Normalization

تعداد زیادی لایه Normalization جهت استفاده در معماری های شبکه های کانولوشن پیشنهاد شده اند که بعضی اوقات بقصد پیاده سازی طرحهای بازداری که در مغز انسان مشاهده شده است بوده اند. اما این لایه ها اخیرا از مد افتاده اند چرا که در عمل میزان اثربخشی آنها حتی در صورت وجود بسیار ناچیز بوده است.

Lateral inhibitiondh یا همون بازداری جانبی، در بینایی برای افزایش شفافیت سیگنالهای ورودی به مغز مورد استفاده قرار میگیرن. در انتها هم این بحث بیشتر توضیح داده شده.

inhibition Lateral

در نوروبایولوژی (عصب شناسی) ما مفهومی داریم بنام inhibition Lateral که به فارسی شاید گفت منع جانبی . یعنی چی؟ این مفهوم به ظرفیت یک نورون برانگیخته شده برای بی اثر کردن(یا کاهش اثر)

همسایه هاش اشاره داره. اساسا ما به یه افزایش قابل توجه نیاز داریم تا یه شکلی از ماکسیمای محلی داشته باشیم . این مسئله باعث ایجاد کنتراست در اون ناحیه میشه و بنابر این بلطبع باعث افزایش درک حسی(یا sensory perception) میشه. افزایش درک حسی چیز خوبیه برای همین هم در ابتدا در شبکه کانولوشن سعی در پیاده سازیش کردن .

این لایه این مفهوم رو که تازه صحبتش رو کردیم سعی میکنه پیاده کنه . این لایه زمانی که ما با نورونهای ReLU سرو کار داریم مفید هست دلیلش هم اینه که نورونهای ReLU فعالسازی های نامحدود (unbounded) دارن و ما به LNR نیاز داریم تا اوناو نرمالیزه کنیم . ما میخوایم ویژگی های فرکانس بالا با پاسخهای بزرگ رو کشف کنیم. اگه ما عملیات نرمالسازی رو در همسایگی محلی نورون برانگیخته انجام بدیم حساسیت بنسبت به همسایه هاش باز بیشتر میشه. در همین زمان این کار باعث تعدیل پاسخ هایی که بصورت یکنواخت بزرگ در هر همسایگی هستن میشه. اگه تمامی مقادیر بزرگ باشن نرمالسازی اونها باعث کاهش همه اونا میشه بنابر این ما میخوایم نوعی از inhibition یا بازداری رو ایجاد کنیم و نرونهای با برانگیختگی بزرگ رو تقویت کنیم. این مساله توضیحش تو بخش ۳٫۳ مقاله کریژوسکی اومده.

لایه تماما متصل (Fully Connected layer)

نورونهایی که در یک لایه تماما متصل قرار دارند با تمام نورون های موجود در لایه قبلی ارتباط دارند دقیقا همانند همان چیزی که در شبکه های عصبی معمولی دیده میشود. بنابراین میتوان از ضرب ماتریسی و سپس جمع نتیجه حاصله با بایاس جهت محاسبه خروجی تمامی نورون ها (فعالسازی ها) در یک لحظه استفاده کرد . بطور خلاصه تمامی قوانین مطرحی در شبکه های عصبی معمولی در این بخش صادق است .

لایه Batch-Normalization:

لایه ELU:

لایه PReLU:

تبدیل لایه های تماما متصل به لایه های کانولوشنی Converting Fully connected layers (to convolutional layers)

ذکر این نکته حائز اهمیت است که تنها تفاوت بین یک لایه تماما متصل و کانولوشنی این است که نورون ها در لایه کانولوشن تنها به یک ناحیه محلی از ورودی متصل هستند و پارامترها را با یکدیگر

به اشتراک میگذارند. البته نورونها در هر دو لایه ضرب نقطه ای را انجام میدهند بنابر این شکل عملکردی (functional form) آنها یکسان بوده و همانطور که در ادامه خواهید دید تبدیل این دولایه به یکدیگر کاملا ممکن و عملی است.

به ازای هر لایه کانولوشن لایه تماما متصلی وجود دارد که دقیقا همان تابع forward را پیاده سازی میکند. ماتریس وزن در این حالت ماتریس بزرگی خواهد بود که تمام درایه های آن به غیر از بلوکهای خاصی (بواسطه اتصال محلی) که وزنها در بسیاری از آنها با یکدیگر برابر اند (بواسطه اشتراک پارامتر) صفر است.

به همین شکل میتوان هر لایه تماما متصل را به یک لایه کانولوشن تبدیل کرد . به عنوان مثال یک لایه تماما متصل با K=4096 فیلتر که به توده ورودی ای با اندازه 7x7x512 نگاه میکند را میتوان بصورت یک لایه کانولوشنی با ناحیه ادراکی با اندازه F=7، zero padding P=0، S=1 گام یا stride و K=4096 فیلتر ایجاد کرد . به عبارت دیگر در اینجا ما اندازه فیلتر(ناحیه ادراکی) را دقیقا برابر با اندازه توده ورودی قرار میدهیم و اینطور اندازه خروجی برابر با 1x1x4096 میشود چرا که تنها یک ستون عمقی را میتوان در توده ورودی جای داد که این دقیقا باعث نتایج یکسان با لایه تماما متصل اولیه می شود.

تبدیل لایه تمام متصل به لایه کانولوشنی (CONV conversion<FC)

از بین این دو تبدیل، تبدیل لایه تماما متصل به لایه کانولشنی در عمل مفیدتر است . بعنوان مثال یک معماری شبکه کانولوشن را در نظر بگیرید که یک تصویر با اندازه 224x224x3 را بعنوان ورودی دریافت میکند و سپس از مجموعه ای از لایه های کانولوشنی و Pooling جهت کاهش اندازه تصویر به اندازه 7x7x512 استفاده میکند (در معماری Alexnet که در ادامه با آن آشنا میشوید، این عمل توسط ۵ لایه Pooling که اندازه ورودی را با فاکتور ۲ هر بار کاهش داده تا آنکه آنرا به اندازه ۲/۲/۲/۲/۲/۲۲۴ =۷ برساند، انجام میشود). در ادامه AlexNet از ۲ لایه تماما متصل با اندازه ۴۰۹۶ استفاده کرده و سپس در لایه تماما متصل آخر با ۱۰۰۰ نورون امتیاز دسته ها (class scores) را حساب میکند. ما با توجه به نکات گفته شده در بالا میتوانیم این شبکه های تماما متصل را به شبکه کانولوشنی معادل آنها تبدیل میکنیم . برای این کار بصورت زیر عمل میکنیم :

- **اولین لایه تماما متصل را که به توده ای با اندازه 7x7x512 نگاه میکند با یک لایه کانولوشنی که از فیلتر با اندازه F=7 استفاده میکند تعویض میکنیم تا توده خروجی با اندازه 1x1x4096 بدست بیاید.**
- **لایه تماما متصل دوم را با یک لایه کانولوشنی که از فیلتر با اندازه F=1 استفاده میکند عوض میکنیم تا توده خروجی با اندازه 1x1x4096 را بدست بیاوریم.**
- **نهایتا آخرین لایه کانولوشن را مثل مرحله قبل با یک لایه کانولوشنی که از فیلتر با اندازه F=1 استفاده میکند تعویض میکنیم تا خروجی نهایی 1x1x1000 حاصل شود.**

هر کدام از این تبدیلات لایه تماما متصل به لایه کانولوشنی در عمل ممکن است نیازمند تغییراتی (نظیر تغییر شکل / تغییر ابعاد) در ماتریس وزن W باشد. در ادامه مشخص میشود که این تبدیل به ما اجازه میدهد تا در یک forward pass بصورت بسیار بهینه ای موقعیت های مکانی زیادی را در تصاویر بزرگتر پیمایش کنیم.

به عنوان مثال اگر تصویری با اندازه ۲۲۴×۲۲۴ توده ای با اندازه 7x7x512 نتیجه دهد ، این به معنای کاهش ۳۲ برابری است ، بنابر این forwarding یک تصویر ۳۸۴×۳۸۴ پیکسلی در یک معماری تبدیل شده ، همان توده را با اندازه 12x12x512 نتیجه خواهد داد .چرا که ۳۲/۳۸۴=۱۲. با گذشت از ۳ لایه کانولوشنی بعدی که ما تازه از لایه تماما متصل تبدیل کرده ایم توده نهایی با اندازه 6x6x1000 نتیجه خواهد شد چرا که (۱۲ – ۷)/۱ + ۱ = ۶ میشود. توجه کنید که بجای یک بردار حاوی امتیاز کلاسها با اندازه 1x1x1000 ، ما حالا یک آرایه ۶×۶ کامل از امتیاز کلاسها از تصویر ۳۸۴×۳۸۴ پیکسلی را بدست می آوریم .

انجام عمل Forwarding در شبکه کانولوشن تبدیل شده برای یکبار بسیار بهینه تر از تکرار آن در شبکه کانولوشن اصلی در تمام ۳۶ مکان است چرا که این ۳۶ ارزیابی محاسبات مشترک دارند. این نکته اغلب در عمل برای بدست آوردن کارایی بهتر در زمانهایی که مثلا تغییر اندازه تصویر به تصویر بزرگتر رایج است مورد استفاده قرار میگیرد.نحوه استفاده به این صورت است که از یک شبکه کانولوشن تبدیل شده جهت ارزیابی امتیاز دسته ها در نقاط مکانی زیادی استفاده شده و سپس میانگین این امتیازات گرفته می شود.

سوال: اگر ما قصد این را داشتیم تا بطور بهینه ای شبکه کانولوشن اصلی را بر روی تصویر با stride کمتر از ۳۲ پیکسل اعمال کنیم آنوقت چطور میتوانستیم این عمل را به انجام برسانیم ؟

ما می توانستیم از چندیدن forward pass برای این منظور استفاده کنیم . بعنوان مثال دقت کنید اگر ما میخواستیم از stride با اندازه ۱۶ پیکسل استفاده کنیم ما میتوانستیم با ترکیب توده های دریافتی توسط عمل forwarding در شبکه کانولوشن تبدیل شده در دوبار این کار را انجام دهیم . بار اول آن را بر روی تصویر اصلی و بار دوم بر روی تصویری که ۱۶ پیکسل از لحاظ مکانی در راستای عرض و ارتفاع تغییر کرده است اعمال کنیم .

ConvNet Architectures

تا به اینجای کار ما دیدیم که شبکه های عصبی تنها از سه لایه Pool،Conv (بصورت پیشفرض ما Max Pool را در نظر میگیریم مگر اینکه خلاف آن عنوان شود) و لایه FC (یا تماما متصل) تشکیل میشوند. همچنین ما تابع فعال سازی RELU را در قالب یک لایه که تابع فعال سازی را بر روی تک تک عناصر اعمال میکند در نظر میگیریم . در این بخش ما خواهیم دید که چگونه این لایه ها با یکدیگر ترکیب شده و یک شبکه عصبی کانولوشن را تشکیل می دهند.

Layer Patterns

رایج ترین شکل یک معماری شبکه عصبی کانولوشن ترکیب چند لایه Conv-RELU است که بعد از آنها لایه های POOL قرار میگیرند و این قالب یا طرح انقدر تکرار میشود تا تصویر ورودی به اندازه دلخواه کوچک شود. معمولا در این زمان است که از لایه های تماما مرتبط استفاده میشود. آخرین لایه تماما متصل حاوی خروجی نظیر امتیاز دسته ها میباشد . به عبارت دیگر رایجترین معماری شبکه عصبی کانولوشن طرحی همانند زیر دارد :

INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

که در اینجا *نشانه تکرار و POOL? هم به معنای وجود یک لایه POOLing اختیاری است. معمولا مقادیر M،N و K بصورت N=>۰ (و معمولا N=<۳) ، M=>۰ ، K=>۰ (و معمولا K<۳) میباشند. در زیر شما میتوانید تعدادی از معماری های رایج شبکه کانولوشن را مشاهده کنید :

FC <- INPUT

معماری فوق یک کلاسیفایر خطی را پیاده سازی میکند. در اینجا مقادیر N = M = K همگی برابر با 0 هستند.

FC <- RELU <- CONV <- INPUT

FC <– RELU <– FC <– POOL]*2 <– RELU <– CONV] <– INPUT

در اینجا نیز شاهد هستیم که هر لایه کانولوشن بین ۲ لایه POOLing قرار گرفته است

FC <– RELU]*2 <– FC] <– POOL]*3 <– RELU <– CONV <– RELU <– CONV] <– INPUT

در اینجا هم میبینیم که دو لایه کانولوشن قبل از هر لایه Pooling قرار گرفته اند . بطور کلی این شیوه برای شبکه های بزرگ و عمیق ایده خوبی است چرا که چندین لایه کانولوشن ترکیب شده ویژگی های پیچیده تر بیشتری از توده ورودی قبل از اینکه توسط عملیات Pooling از بین بروند می تواند بدست بیاورد.

معمولا سعی کنید از ترکیب لایه کانولوشن با فیلتر کوچکتر نسبت به لایه کانولوشنی با اندازه ناحیه ادراکی(فیلتر) بزرگتر استفاده کنید. فرض کنید شما سه لایه کانولوشن با اندازه فیلتر ۳×۳ را با هم ترکیب میکنید (دقت کنید که بین این لایه ها هم لایه RELU وجود دارد) با این ترتیب هر نورون در لایه کانولوشن اول یک دید ۳×۳ از توده ورودی دارد . یک نورون که در لایه کانولوشن دوم قرار دارد یک دید ۳×۳ از لایه کانولوشن اول و به همین صورت یک دید ۵×۵ از توده ورودی دارد.به همین شکل یک نورون در لایه کانولوشن سوم یک دید ۳×۳ از لایه کانولوشن دوم دارد و از این جهت یک دید ۷×۷ هم از توده ورودی دارد. فرض کنید بجای این سه لایه کانولوشن ۳×۳ (منظور با اندازه فیلتر ۳×۳ است) ما فقط بخواهیم از یک لایه کانولوشن به همراه نواحی ادراکی با اندازه ۷×۷ استفاده کنیم . نورون های موجود در این لایه دارای اندازه ناحیه ادارکی خواهند بود که برابر با اندازه توده ورودی است (۷×۷) که البته این کار کاستی ها و مشکلاتی نیز به همراه دارد. مشکل اول این است که در این حالت، نورون ها یک تابع خطی از ورودی را محاسبه میکنند در حالی که ترکیب سه لایه کانولوشن در حالت قبل ،یک تابع غیر خطی از ورودی را که باعث بیان بهتر ویژگی های ورودی میشود محاسبه میکند. دومین مشکل این است که اگر فرض کنیم تمام توده ها C کانال داشته باشند بنابر این یک لایه کانولوشن ۷×۷ تعداد $C \times (7 \times 7 \times C) = 49C^2$ پارامتر خواهد داشت در حالی که سه لایه کانولوشن ۳×۳ تنها $3 \times (C \times (3 \times 3 \times C)) = 27C^2$ پارامتر خواهند داشت. همانطور که دیدیم ترکیب لایه های کانولوشن با اندازه فیلتر کوچک در مقابل استفاده از یک لایه کانولوشن با اندازه فیلتر بزرگتر به ما اجازه میدهد تا ویژگی های قدرتمندتر بیشتر را با پارامترهای کمتری از ورودی را بیان کنیم.

در حالت اول هم مشکلی که وجود دارد اگر قصد استفاده از backpropagation را داشته باشیم نیاز به حافظه بیشتری نسبت به حالت دوم است.

Layer Sizing Patterns

تا به اینجا ما صحبتی از فراپارامترهای رایجی که در هر لایه یک شبکه کانولوشن مورد استفاده قرار می گیرند نکردیم. در اینجا ما ابتدا قوانین کلی مرتبط با مدیریت اندازه معماری های شبکه کانولوشن را بیان کرده و سپس به این قوانین به همراه توضیحی از نمادها می پردازیم .

لایه ورودی (Input layer) (که حاوی تصویر ورودی است) باید چندین بار بر ۲ قابل تقسیم (پخش پذیر) باشد.

اعداد رایج در این زمینه شامل ۳۲ (مثلا تصاویر موجود در دیتاست CIFAR-10 (۹۶،۶۴) مثل تصاویر موجود در دیتاست (STL-10) و یا ۲۲۴ (مثل شبکه های ImageNet متداول) ،۳۸۴ و ۵۱۲ هستند.

لایه های کانولوشن باید از فیلتر هایی کوچک (مثل ۳×۳ و یا نهایتا ۵×۵) با Stride S =1 و خصوصا zero padding توده ورودی بصورتیکه لایه کانولوشن ابعاد مکانی (عرض و ارتفاع) توده ورودی را تغییر ندهد استفاده کنند. به عبارت ساده تر این به این معناست که زمانی که F=3 باشد استفاده از zero padding با مقدار P=1 ابعاد اصلی ورودی را حفظ خواهد کرد. به همین ترتیب زمانی که F=5 باشد zero padding با مقدار P=2 باعث حفظ ابعاد ورودی خواهد شد . برای یک مقدار F کلی ، میتوان نشان مشاهده کرد که باعث حفظ اندازه ورودی میشود. اگر به هر دلیلی نیازمند استفاده از فیلترهایی با اندازه های بزرگتر باشیم (مثلا فیلترهایی با اندازه ۷×۷) ، باید عنوان کنیم که تنها استفاده از این اندازه در لایه اول کانولوشن که مستقیما به تصویر ورودی نگاه میکند متداول است .

لایه های Pooling وظیفه downsample کردن یا کاهش دادن ابعاد مکانی (عرض و ارتفاع) ورودی را دارند. رایجترین تنظیمات برای این لایه استفاده از max pooling به همراه ناحیه ادراکی با اندازه ۲×۲ (یعنی F=2) و stride با مقدار S=2 میباشد. توجه کنید که این پیکربندی لایه pooling باعث نابودی ٪۷۵ مقادیر در توده ورودی میشود (بخاطر downsampling 2 عنصر هم در ارتفاع و هم درعرض). یک پیکربندی کمتر متداول دیگر نیز وجود دارد که در آن از نواحی ادراکی با اندازه ۳×۳ و stride با مقدار S=2 استفاده میشود. استفاده از نواحی ادراکی با

اندازه ای بزگتر از این مقدار در لایه pooling بسیار نادر است چرا که در این صورت این لایه بیش از حد پراتلاف خواهد شد(یعنی بیش از حد داده ها را حذف می کند) و معمولا این کار باعث بدتر شدن کارایی نیز می شود.

طرحی که در بالا مشاهده کردید از این جهت جذاب است که لایه های کانولوشن همگی اندازه مکانی ورودی خود را حفظ می کنند. در حالی که لایه های POOL به تنهایی مسئول کاهش اندازه توده ها هستند. در روش های دیگری که ما از stride هایی با مقادیر بیشتر از ۱ استفاده میکنیم و یا در لایه های کانولوشن از zero padding در ورودی استفاده نمی کنیم، لازم است که با دقت بسیار زیادی توده های ورودی در سرتاسر شبکه کانولوشن را تحت نظر داشته و اطمینان حاصل کنیم تمامی stride ها و فیلتر ها با همدیگر سازگار بوده و همخوانی داشته باشند و معماری شبکه کانولوشن بصورت متقارن و مناسبی بهم متصل باشد.

چرا باید از stride با مقدار ۱ دریک لایه کانولوشن استفاده کرد؟

در عمل مقادیر کوچک stride بهتر عمل میکنند. علاوه بر آن همانطور که قبلا گفته شد، stride با مقدار ۱ بما اجازه میدهد تا وظیفه downsampling را به عهده لایه Pooling بگذاریم و لایه های کانولوشن به وظیفه خود که تبدیل توده ورودی از بصورت عمقی است بپردازند.

چرا باید از padding استفاده کرد؟

علاوه بر فایده حفظ اندازه های مکانی (عرض و ارتفاع) بعد از لایه های کانولوشن که قبلا هم به آن اشاره شد، انجام این کار در واقع باعث افزایش کارایی میشود. اگر در لایه های کانولوشن در ورودی از zero padding استفاده نمیکردیم، و تنها عمل کانولوشن را انجام میدادیم ، در اینصورت اندازه توده ها بعد از هر لایه کانولوشن به میزان کمی کاهش می یافت و اطلاعات موجود در مرزها(ی بیرونی توده) بسیار سریع از بین می رفتند.

اعمال تغییرات با توجه به محدودیت های حافظه

در بعضی حالات، (خصوصا در ابتدای معماری های شبکه عصبی کانولوشن) میزان مصرف حافظه با قوانین ارائه شده در بالا بسرعت افزایش پیدا می کند. بعنوان مثال فیلتر کردن یک تصویر با اندازه 224x224x3 با سه لایه کانولوشن و ۶۴ فیلتر با اندازه ۳×۳ و zeropadding برابر P=1 باعث ایجاد یک توده با اندازه 224x224x64 خواهد شد. این میزان برابر با ۱۰ میلیون عدد و یا ۷۲

مگابایت حافظه (به ازای هر تصویر هم برای اعداد و هم گرادیانت آنها) است. از آنجایی که GPU های فعلی توسط حافظه bottleneck شده اند شاید نیاز به تغییراتی در این زمینه باشد. در عمل افراد ترجیح میدهند تا تنها تغییرات را در لایه کانلووشن اول شبکه اعمال کنند. بعنوان مثال، یک تغییر میتواند استفاده از فیلتر با اندازه ۷×۷ و stride S=2 در لایه کانولوشن اول باشد (همانطور که در شبکه ZF دیده می شود). نمونه دیگر AlexNet است که از فیلترهایی با اندازه ۱۱×۱۱ و Stride با مقدار S=4 در لایه کانولوشن اول خود استفاده کرد.

مطالعات موردی

در حوزه شبکه های کانولوشنی چندین معماری وجود دارد که نام اختصاصی دارند. معروف ترین آنها را در زیر مشاهده میکنید .

شبکه LeNet

اولین کاربردهای موفقیت امیز شبکه های عصبی کانولشن توسط Yann LeCun در سال ۱۹۹۰ توسعه داده شدند. از میان آنها، معماری LeNet معروفترین آنهاست که برای خواند کدهای پستی ، ارقام و... مورد استفاده قرار گرفته بود.

معماری AlexNet

اولین نمونه ای که شبکه های عصبی را در Computer Vision به محبوبیت رساند AlexNet بود. این معماری توسط Alex Krizhevsky،Ilya Sutskever و Geoff Hinton توسعه داده شد. AlexNet در رقابت ILSVRC در سال ۲۰۱۲ ارائه داده شد و توانست با اختلاف فاحشی نسبت به جایگاه دوم به پیروزی برسد. این شبکه اساس معماری شبیه LeNet داشت با این تفاوت که عمیقتر، بزرگتر بود و همچنین از ترکیب چندین لایه کانولوشن با هم استفاده میکرد (در گذشته یک لایه کانولوشن که بعدش یک لایه Pool قرار میگرفت رایج بود)

ZF

برنده رقابت ILSVRC 2013 شبکه کانولوشنی بود که توسط Matthew Zeiler و Rob Fergus توسعه پیدا کرده بود. بعدها این شبکه به نام ZF که مخفف نام توسعه دهندگان آن است معروف شد. این معماری ورژن بهینه شده ای از AlexNet بود که با تغییر فراپارامترهای معماری ، بطور خاص با افزایش اندازه لایه های کانولوشن میانی این بهینه سازی انجام شده بود.

GoogLeNet

برنده رقابت ILSVRC 2014 هم شبکه عصبی کانولوشنی بود که توسط Szegedy et al از طرف گوگل توسعه داده شده بود . ویژگی جدیدی که این معماری عرضه کرده بود توسعه یک ماجول مفهومی (Inception Module) بود که بشدت تعداد پارامترهای شبکه را کاهش میداد.

(۴ میلیون پارامتر را با ۶۰ میلیون پارامتر AlexNet مقایسه کنید!)

VGGNet

برنده جایگاه دوم در رقابت ILSVRC 2014 شبکه عصبی کانولوشنی بود که توسط Karen Simonyan و Andrew Zisserman توسعه داده شده بود که بعدها با نام VGGNet معروف شد.ویژگی جدیدی که این معماری عرضه کرده بود ، این بود که نشان داد عمق شبکه یک مولفه حیاتی برای کارایی خوب است. ورژن نهایی بهترین شبکه آنها شامل ۱۶ لایه CONV/FC و بصورت خوش آیندی یک معماری بشدت همگن (homogeneous) که تنها دارای فیلتر با اندازه ۳×۳ در لایه کانولوشن و فیلتر ۲×۲ در لایه pooling از ابتدا تا به انتها بود. بعدا مشخص شد که برخلاف قدرت کمتر دسته بندی نسبت به VGGNet،GoogLeNet در چندین وظیفه یادگیری انتقالی (Multiple transfer learning tasks) از GoogLeNet بهتر عمل میکند. بنابر این شبکه VGG در حال حاضر محبوبترین انتخاب برای Feature extraction از تصاویر است . بطور خاص مدل از پیش آموزش داده شده آنها برای استفاده در کتابخانه Caffe وجود دارد . یک کاستی این شبکه این است که حافظه مصرفی و تعداد پاراامتر بسیار زیادی دارد (۱۴۰ میلیون پارامتر)

توضیحات بیشتر VGGNET

VGGNet از لایه های کانولوشن با اندازه فیلتر ۳×۳ ، stride برابر ۱ و zero padding برابر P=1 و لایه های Pooling با اندازه فیلتر ۲×۲ با همراه stride برابر با S=2 و بدون هیچ zero paddingیی تشکیل شده است . در زیر ما اندازه هر تصویر در هر مرحله را ثبت میکنیم اینطور میتوانیم هم اندازه تصویر و تعداد کلی وزنها را تحت نظر داشته باشیم .

INPUT: [224x224x3] memory: ۲۲۴*۲۲۴*۳=۱۵۰K weights: 0

CONV3-64: [224x224x64] memory: ۲۲۴*۲۲۴*۶۴=۳,۲M weights: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: ۲۲۴*۲۲۴*۶۴=۳,۲M weights: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: ۱۱۲*۱۱۲*۶۴=۸۰۰K weights: 0

CONV3-128: [112x112x128] memory: ۱۱۲*۱۱۲*۱۲۸=۱,۶M weights: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: ۱۱۲*۱۱۲*۱۲۸=۱,۶M weights: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: ۵۶*۵۶*۱۲۸=۴۰۰K weights: 0

CONV3-256: [56x56x256] memory: ۵۶*۵۶*۲۵۶=۸۰۰K weights: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: ۵۶*۵۶*۲۵۶=۸۰۰K weights: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: ۵۶*۵۶*۲۵۶=۸۰۰K weights: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: ۲۸*۲۸*۲۵۶=۲۰۰K weights: 0

CONV3-512: [28x28x512] memory: ۲۸*۲۸*۵۱۲=۴۰۰K weights: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: ۲۸*۲۸*۵۱۲=۴۰۰K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: ۲۸*۲۸*۵۱۲=۴۰۰K weights: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: ۱۴*۱۴*۵۱۲=۱۰۰K weights: 0

CONV3-512: [14x14x512] memory: ۱۴*۱۴*۵۱۲=۱۰۰K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: ۱۴*۱۴*۵۱۲=۱۰۰K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: ۱۴*۱۴*۵۱۲=۱۰۰K weights: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: ۷*۷*۵۱۲=۲۵K weights: 0

FC: [1x1x4096] memory: ۴۰۹۶ weights: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: ۴۰۹۶ weights: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: ۱۰۰۰ weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)

همانطور که در شبکه های کانولوشنی متداول است ، می بینید که بیشترین حافظه مصرفی در لایه های کانولوشن ابتدایی مصرف شده اند و بیشترین تعداد پارامترها نیز در لایه های تماما متصل آخر وجود دارند. در این مورد خاص، اولین لایه تماما متصل دارای ۱۰۰ میلیون وزن از ۱۴۰ میلیون وزن موجود در سراسر شبکه میباشد.

ResNet:(برنده رقابت ایمیج نت در سال ۲۰۱۵ با ۱۵۲ لایه)NiN

ملاحظات محاسباتی :

بزرگترین گلوگاهی که باید در زمان ساخت یک معماری شکبه کانولوشن مد نظر قرار دهید حافظه است . بسیاری از GPU های جدید حافظه ای با اندازه ۳، ۴ و ۶ گیگابایت دارند و تنها مدلهای معدودی دارای حافظه های بیشتری نظیر ۱۲ گیگابایت هستند که میتوان در این بین به سری تایتان ایکس شرکت انویدیا اشاره کرد. مواردی که در زیر لیست شده اند مهمترین منابع حافظه ای هستند که باید زیرنظر گرفته شوند.

اندازه توده های لایه میانی

اینها تعداد اعداد در هر لایه کانولوشن و همچنین گرادیانت های(با اندازه مساوی) آنها هستند. معمولا بیشترین تعداد این مقادیر در لایه های ابتدایی شبکه کانولولشن (لایه های کانولوشن اول) وجود دارد. اینها بخاطر انکه برای backpropagarion مورد نیاز هستند نگهداری می شوند. اما یک پیاده سازی هوشمندانه که شبکه کانولوشن را تنها در زمان آزمایش اجرا می کنید، میتواند در اصل با ذخیره مقادیر فعلی در هر لایه و حذف مقادیر قبلی در لایه های پایینی بشدت این مقدار را کاهش دهد.

اندازه پارامترها

اینها همان اعدادی هستند که پارامترهای شبکه ، گرادیانتهای آنها در زمان backpropagation و معمولا هم یک مرحله cache درصورتی که از Adagrad،momentum ، و یا RMSProp استفاده میکنند را در خود نگه میدارند. بنابر این حافظه ای که برای ذخیره سازی بردار پارامتر استفاده میشود معمولا باید حداقل با فاکتور ۳ یا عددی در این حدود ضرب شود.

هر پیاده سازی شبکه کانولوشن نیازمند نگهداری حافظه متفرقه مثل image data batches و شاید ورژن افزایش یافته آنها و... باشد .

زمانی که شما یک تخمین کلی از تمام مقادیر (برای خروجی نرونها، گرادیانت آنها و موارد متفرقه) بدست آوردید . حال باید آنرا به گیگابایت تبدیل کنید . تعداد اعداد را گرفته و آنرا در ۴ ضرب کنید تا تعداد بایتهای مورد نیاز را بدست آورید (چون هر عدد اعشاری در ۴ بایت ذخیره میشود و یا اگر از دابل استفاده میکنید در ۸ ضرب کنید) و سپس چندیدن بار بر ۱۰۲۴ تقسیم کنید تا به

مقدار حافظه به کیلوبایت ، مگابایت و نهایتا گیگابایت برسید . اگر شبکه شما در حافظه جا نمیشود یک روش ابتکاری جهت جای دادن آن کاهش اندازه batch است چرا که بیشترین میزان حافظه معمولا توسط خروجی نرونها مصرف می شود.

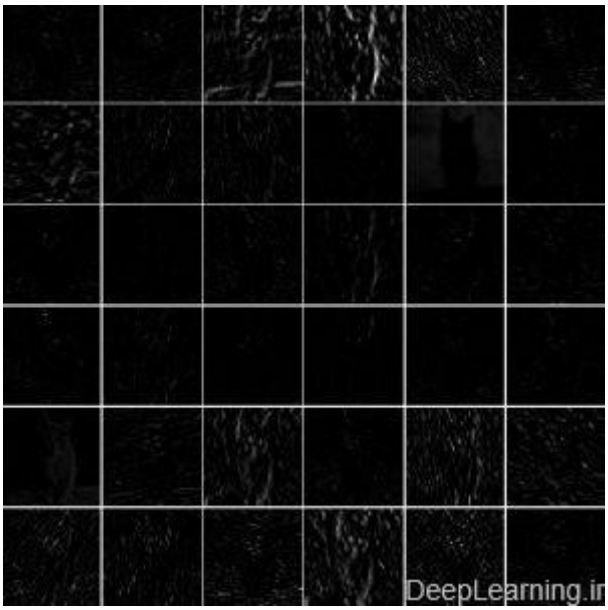
بصری سازی آنچه شبکه عصبی کانولوشن یاد می گیرد

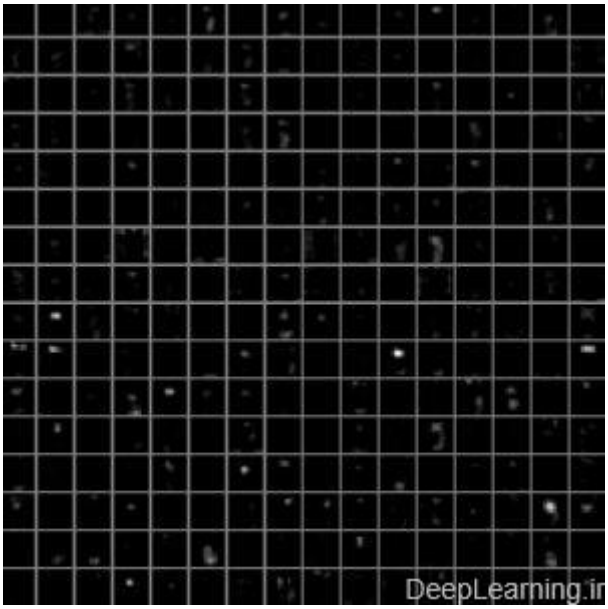
چندین روش برای فهمیدن و نمایش بصری اطلاعات درونی یک شبکه عصبی کانولوشن در تحقیقات سالهای اخیر توسعه و ارائه شده است که تا حدودی بعنوان پاسخی به انتقاد معمول " ویژگی های یاد گرفته شده در شبکه های عصبی قابلیت تفسیر ندارند" بودند. در این بخش ما بصورت خلاصه وار تعدادی از این روشها و کارهای مرتبط را مورد بررسی قرار می دهیم.

نمایش بصری (Visualizing) مقادیر فعال سازی و وزن های لایه اول

مقادیر فعال سازی در لایه ها

سرراست ترین روش نمایش بصری ، نمایش مقادیر فعال سازی شبکه در حین forward pass است. برای شبکه های ReLU ، فعال سازی ها معمولا ابتدا به شکل لکه های متراکم شروع شده و بعد با ادامه پیدا کردن آموزش (training) این فعال سازی ها پراکنده تر و محلی تر میشوند. یک نکته بسیار مهم که در اینجا با نمایش بصری قابل مشاهده است، این است که بعضی نگاشتهای فعال سازی ممکن است تماما برای تعداد زیادی ورودی صفر باشند. که این به معنای فیلترهای از کار افتاده (dead filter) بوده و میتواند نشانه ای از overfitting باشد.



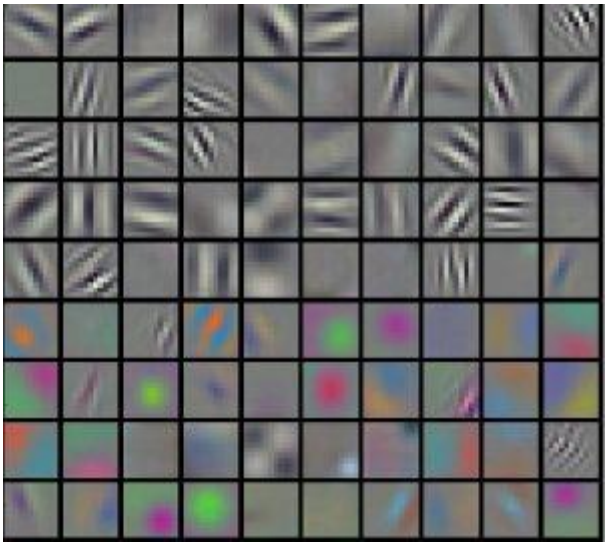


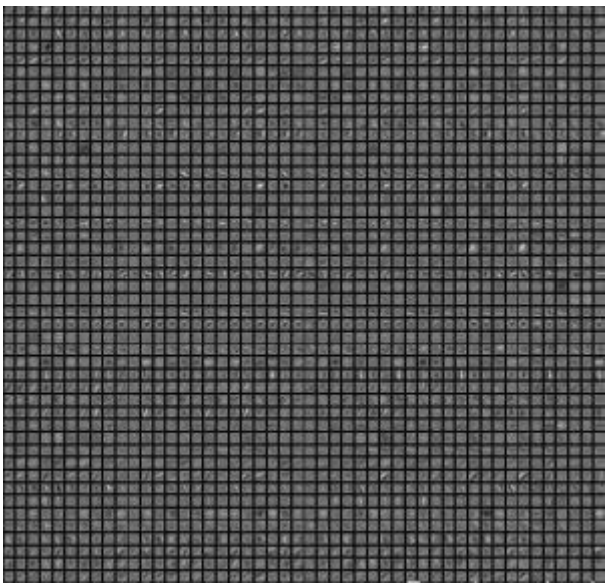
یک نمایش بصری از مقادیر فعال سازی در اولین لایه کانولوشن

تصویر اول: نمایش بصری مقادیر فعال سازی در اولین لایه کانولوشن تصویر دوم: نمایش بصری مقادیر فعال سازی در پنجمین لایه کانولوشن در یک شبکه AlexNet آموزش دیده که در حال نگاه به تصویر یک گربه است هر مربع نمایش داده شده در بالا، یک نگاشت فعال سازی متناظر با تعدادی فیلتر است. توجه کنید که مقادیر این فعال سازی ها پراکنده (بیشتر مقادیر صفر هستند که در این نمایش بصری بصورت سیاه نمایش داده شده اند) و محلی هستند.

فیلتر های لایه کانولوشن و تماما متصل (Conv/FC Filters)

روش دوم نمایش بصری وزن هاست. معمولا وزنها در لایه کانولوشن اول که مستقیما به مقادیر پیکسل های خام تصویر نگاه میکنند قابل تفسیرترین پارامتر هستند، اما نمایش بصری وزنهای عمیقتر(وزنها در لایه های عمیقتر شبکه) در شبکه هم ممکن است . نمایش بصری وزنها به این دلیل مفید است چون که شبکه هایی که بخوبی آموزش دیده اند معمولا فیلترهای خوب و یکنواختی را بدون هیچ قالب نویزداری (noisy patterns) نمایش می دهند. قالب های نویزدار میتوانند نشانه شبکه ای باشند که به اندازه کافی آموزش ندیده است و یا احتمالا نشانه regularization **ضعیفی باشد که باعث** overfitting **شده است .**





یک نمایش بصری از فیلترهای یادگرفته شده در لایه کانولوشن اول

تصویر اول : نمایش بصری فیلترها در لایه کانولوشن اول تصویر دوم : نمایش بصری فیلترها در لایه کانولوشن دوم یک شبکه AlexNet آموزش دیده. توجه کنید که وزن های لایه اول خیلی خوب و یکنواخت هستند که نشانه همگرایی خوب شبکه است. ویژگی های رنگی و سیاه و سفید (color/grayscale) به این دلیل با هم دسته بندی شده اند چون AlexNet دارای دو جریان پردازش جداگانه است که نتیجه آشکار این معماری این است که یک جریان ویژگی های سیاه و سفید فرکانس بالا را ایجاد کرده و جریان دیگر ویژگی های رنگی فرکانس پایین را ایجاد میکند. وزنهای لایه کانولوشن دوم به این اندازه(لایه اول) قابل تفسیر نیستند اما مشخص است که آنها هنوز هم یکنواخت، خوش فرم و عاری از هرگونه قالب نویزدار هستند.

بدست آوردن دوباره تصاویری که بصورت بیشنه ای یک نورون را فعال میکنند (Retrieving images that maximally activate a neuron)

یک روش دیگر نمایش این است که یک دیتاست عظیم از تصاویر را گرفته و شبکه را توسط این تصاویر تغذیه کنیم و در این حین ، کنترل کنیم کدام تصاویر بعضی از نورون ها را بصورت بیشینه ای فعال میکنند. . ما سپس میتوانیم برای بدست اوردن درکی از انچه که نورون دارد بدنبال آن در ناحیه ادراکی خود میگردد با نمایش این تصاویر برسیم . یکی از این نمایش های بصری (از میان سایر روشها)، روشی است که در مقاله " Rich feature hierarchies for accurate object detection and semantic segmentation " توسط Ross Grshick et al توضیح داده شده است.

تصاویری که نورون های لایه Pooling پنجم در یک شبکه AlexNet را به بیشترین میزان فعال کرده اند. مقادیر فعال سازی و ناحیه ادراکی یک نورون مشخص با رنگ سفید نمایش داده شده است. (بصورت خاص، توجه کنید که نورون های لایه Pool5 (پنجمین لایه Pooling) تابعی از یک قسمت نسبتا بزرگ از تصویر ورودی هستند). میتوان دید که بعضی از نورون ها نسبت به بالا تنه، متن، و یا نقاط درخشان حساس هستند.(واکنش میدهند).

یک اشکال این روش این است که نورون های لایه ReLU الزاما هیچ معنایی به تنهایی ندارند. بلکه مناسبتر این است که چندین نورون لایه ReLU را بعنوان بردارهای اولیه (basis vectors) بعضی فضاها که نمایانگر پچ های تصاویر(image patches) هستند در نظر بگیریم . به عبارت دیگر، نمایش بصری(visualization)، در حال نمایش patchهایی که در لبه توده ای از تصاویر(at the edge of cloud of representation) ، در راستای محورهایی (اختیاری) که متناظر با وزنهای فیلتر هستند، است. میتوان این مسئله را از این واقعیت مشاهده کرد که نورونها در

شبکه کانولوشن بصورت خطی در فضای ورودی عمل میکنند بنابر این چرخش اختیاری فضا ممکن نیست. این نکته در مقاله ای بنام Intriguing properties of neural networks **توسط** Szegedy et al **مورد بررسی بیشتر قرار گرفت که در آن آنها نمایش بصری مشابه ای را در راستای جهت های دلخواه در فضای نمایشی (تصویر) انجام دادند.**

Embedding the codes with t-SNE

شبکه های عصبی کانولوشن را میتوان بعنوان تبدیل تدریجی تصاویر به نمایش هایی که در آن دسته ها (کلاسها) توسط یک کلاسیفایر خطی جدا شده اند، در نظر گرفت. ما میتوانیم به ایده خوبی در باره توپولوژی این فضا با تعبیه تصاویر در دو بعد بصورتی که نمایش بُعد پایین دارای فاصله های نسبتا یکسان نسبت به نمایش بُعد بالای آنها است برسیم . روش های تعبیه سازی زیادی توسعه داده شده اند که از بصیرت (intuition) تعبیه بُردارهای با بعد بلند در فضای با بعد کوتاه در حالی که فاصله های متقابل از هم حفظ شده اند استفاده می کنند. در میان آنها t-SNE **یکی از معروفترین روشهایی است که نتایج از لحاظ بصری زیبایی را تولید می کند.**

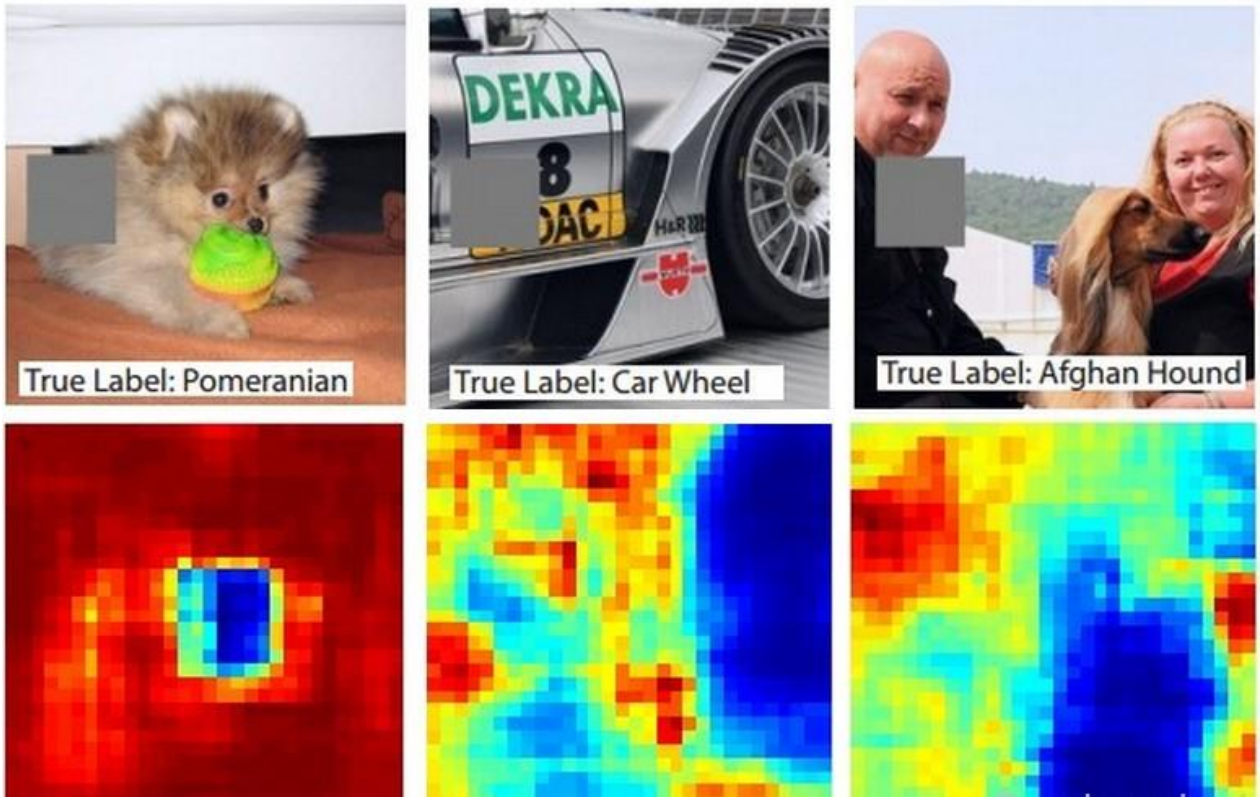
برای تولید یک تعبیه، ما میتوانیم مجموعه ای از تصاویر را گرفته و از شبکه عصبی کانولوشن برای استخراج کدهای CNN **استفاده کنیم. (مثلا در** AlexNet **بردار ۴۰۹۶ بعدی دقیقا قبل از کلاسیفایر (دسته بندی کننده)، و بطور حیاتی شامل غیرخطی** ReLU**). ما سپس اینها را وارد یک** t-SNE **کرده و یک بردار دو بعدی برای هر تصویر بدست میاوریم. تصاویر متناظر سپس می توانند در قالب یک** grid **به نمایش در آیند.**

تعبیه t-SNE مجموعه ای از تصاویر بر اساس کدهای CNN آنها

تصاویری که در کنار هم قرار دارند در فضای نمایش CNN هم بهم نزدیک هستند، که به معنای آن است که CNN آنها را بسیار شبیه به هم "میبیند". دقت کنید که شباهت اغلب مبتنی بر کلاس (دسته) و معنایی بوده تا مبتنی بر رنگ و پیکسل. برای اطلاعات بیشتر در چگونگی تولید این نمایش ، سورس کد آن و نمایش های مرتبط بیشتر در مقیاس های متفاوت به t-SNE visualization of CNN codes مراجعه کنید.

مسدود سازی بخشهایی از تصویر

فرض کنیدکه یک شبکه عصبی کانولوشن تصویری را بعنوان تصویر یک سگ دسته بندی میکند. حالا یک سوال مهم، ما چطور میتوانیم مطمئن باشیم که شبکه در اصل دارد سگ را انتخاب میکند و نه اطلاعات محتوایی از پس زمینه و یا اشیاء دیگری در تصویر را؟ به عبارت دیگر چطور مطمئن شویم که شبکه واقعا سگ را شناسایی کرده و صرفا جواب دسته بندی آن بخاطر وجود اطلاعات دیگری در تصویر نیست؟ یک راه برای فهمیدن اینکه پیشبینی شبکه از کدام بخش از تصویر دارد می آید این است که احتمال دسته دلخواه (در اینجا دسته سگ ها) را بعنوان یک تابع از مکان یک شی مسدود شده رسم کنیم . این یعنی اینکه ما بر روی نواحی مختلف تصویر حرکت کرده و یک پچ را بر روی تصویر قرار دهیم که تمام آن ناحیه را برابر صفر قرار دهد.و بعد به احتمال کلاس نگاه کنیم . ما می توانیم احتمال را بصورت یک نقشه گرمایی heat map بعدی نمایش دهیم . این روش در مقاله Mathew Zeiler با عنوان Visualizing and Understanding Convolutional Networks آمده است .



سه تصویر ورودی (در بالا) و heatmap های متناظر با هر کدام (در پایین)

دقت کنید که ناحیه مسدود کننده بصورت خاکستری نشان داده شده است. در همان حین که ما مسدود کننده را بر روی تصویر حرکت میدهیم (می لغزانیم/اسلاید می کنیم) احتمال کلاس صحیح را نیز ثبت کرده و سپس آنرا بصورت یک heatmap **(که در زیر هر تصویر نمایش داده شده است) نمایش میدهیم. بعنوان مثال در سمت چپ ترین تصویر ما میبینیم که احتمال تصویر سگ** Pomeranian **زمانی که مسدود کننده صورت سگ را میپوشاند کاهش پیدا میکند و اینطور ما بسطحی از اطمینان میرسیم که در اصل این صورت سگ است که باعث امتیاز بالای دسته بندی میشود. حال بطور برعکس ، میبینیم که صفر کردن بقیه نواحی تصویر تاثیر نسبتا ناچیزی در امتیاز کلاس دارد که باز موید نتیجه قبلی است . یعنی صورت سگ مسئول بیشترین امتیاز کسب شده تصویر برای دسته بندی بوده و مابقی اجزای تصویر تاثیر بسیار ناچیزی در امتیاز مورد نظر داشته اند.**

inhibition schemes

برای اطلاعات بیشتر در این زمینه میتوان به Cuda-convnet Library API **آقای** Alex Krizhevsky **مراجعه کرد.**

میتوانید نحوه انجام این تبدیل را در عمل در قالب کد (با استفاده از کتابخانه Caffe) در Net Surgery مشاهده کنید.

با ورود نسل جدید کارتهای گرافیکی انویدیا در معماری پاسکال ما شاهد افزایش حجمی حافظه و همینطور کارایی این کارتها در حوزه یادگیری عمیق و خصوصا محسابات مورد نیاز شبکه کانولوشن هستیم. در زمان بروز آوری این نگارش کارتهای GTX1080،GTX1070 با ۸ گیگابایت رم از سری جدید معرفی شده اند.

http://cs.stanford.edu/people/karpathy/cnnembed

توضیحات Lateral inhibition

در مورد local response normalization یا normalization که بالا توضیحش اومده بعنوان مثال تو دیتاست ImageNet طبق گزارش اومده در مقاله VGGNet تاثیر (چندانی) نداشته برای همین در مدل VGGNet بعنوان مثال استفاده نشده . (این مدل مقام دوم کلسیفیکیشن ایمیج نت و مقام اول دیتکشن رو کسب کرده در سال ۲۰۱۴)

بازداری بصری یا Visual Inhibition

lateral inhibition یا بازداری جانبی باعث افزایش کنتراست و شفافیت (sharpness) در پاسخ بصری میشود. این پدیده بعنوان مثال در شبکیه چشم پستاندارن رخ میدهد.در تاریکی یک محرک نوری کوچک باعث بهبود فوتو ریسپتورها (rod cells یا سلولهای میله ای)ی مختلف میشود. سلولهای میله ای در مرکز محرک نوری سیگنال "روشن" را به مغز منتقل میکنند در حالی که سلولهای میله ای دیگر در خارج محرک سیگنال "تاریک" را به مغز ارسال میکنند.این تفاوت بین روشنایی و تاریکی باعث ایجاد تصویری شفافتر(sharper) میشود. این مکانیزم همچنین باعث ایجاد اثر بصری باند ماخ (Mach Band) میشود.

بازداری جانبی بینایی (Visual Lateral inhibition) فرایندی است که در آن سلولهای فوتو ریسپتور به مغز در درک تفاوت موجود در یک تصویر کمک می کنند. نور الکترومغناطیسی وارد چشم شده و سپس از قرنیه مردم و سپس لنز چشم عبور کرده و سپس از سلولهای bipolar،amacrine،ganglion و سلولهای افقی گذشته تا به سلولهای میله ای فوتو ریسپتور که نور را جذب میکنند برسد. این سلولهای میله ای بوسیله انرژی موجود در نور تحریک شده و یک سیگنال عصبی محرک را به سمت سلول های افقی آزاد می کند .

سیگنال محرک اما تنها از طریق سلولهای میله ای در مرکز ناحیه ادراکی سلول Ganglion قابل انتقال است چرا که سلولهای افقی با ارسال یک سیگنال بازداری به سلولهای میله ای همجوار برای ایجاد توازنی که به پستاندارن اجازه درک تصاویر واضح تر را میدهد پاسخ می دهند.
سلولهای ganglion شبکیه کماکان یک پاسخ عصبی محرک را تنها از سلول میله ای مرکزی دریافت خواهند کرد. سلول میله ای مرکزی پاسخ عصبی محرک خود را مستقیما به سلولهای bipolar ارسال میکند این سلولها هم سپس این سیگنال را به سمت سلولهای ganglion باز ارسال میکنند._سلولهای bipolar نقش رله را ایفا می کنند.
بازداری ایجاد شده توسط سلولهای افقی باعث ایجاد سیگنال متوازن تر و متمرکز تری برای سلولهای ganglion شبیکه میشود که متعاقبا به کورتکس سلبرال از طریق عصب بینایی ارسال میشود.

توضیحات در مورد Convolve

یک توضیح مهم اینه که convolve به معنای ضرب نقطه به نقطه نیست .در اصل به معنای ضرب پیچشی هست (با تشکر از سرکار خانوم سلیمیان که معادل فارسی رو لطف کردن). عملیات کانولوشن یعنی ضرب مولفه به مولفه یا همون عنصر به عنصر دوماتریس با هم (در اینجا ناحیه ادراکی از ورودی با ماتریس وزن که ما بهش میگیم فیلتر) نکته ای که باید دقت کنید اینه که در تعریف کانولوشن یکی از دو ماتریس جای سطر و ستونهاش برعکس میشه و بعد بین ماتریس حاصل و ماتریس اول ضرب مولفه به مولفه انجام میشه) و سپس جمع همه عناصر ماتریس حاصل با هم . یعنی وقتی دو ماتریس با هم ضرب شدن بصورت مولفه به مولفه یک ماتریس سوم حاصل می شود. حالا درایه های ماتریس سوم رو با هم جمع باید کرد این میشه عملیات کانولوشن که تو درس computer vision بچه ها میخونن و ازش تو کارهای مختلف استفاده می کنند.

نکته مهم بعدی این که شما در آموزش بالا و همینطور احتمالا ۱۰۰ درصد بقیه اموزشهای یادگیری عمیق میبنید که خبری از برعکس کردن یا بهتر بگم چرخش (flip) سطر و ستون نمی بینید! هرچه هست خیلی معمولی تو ماتریس رو مولفه به مولفه ضرب و بعد هم درایه ها رو با هم جمع میکنند و بعنوان خروجی ارائه می کنند. دلیل این مساله اینه که کاری که در اصل اینجا اتفاق دارد می افتد کانولوشن نیست! correlation هست. کاری که ما داریم میکنیم اینجا discrete correlation بهش میگن که تو شبکه های عمیق نتیجه معادل discrete convolution رو به ما می دهد . برای همین تقریبا همه ی کتابخونه ها و چارچوب هایی که تو این حوزه دارند کار میکنن کانولوشن را

بصورت correlation پیاده سازی می کنند.

(برای درک بهتر کانولوشن در حالت کلی میتونید اینجا و اینجا رو هم ببینید)

منابع و مآخذ:

مقاله و ترجمه سید حسین حسین پور متی کلایی - خانم سلیمیان

www.Estanford.edu

www.Deeplearning.ir

www.Wikipedia.org