دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

گزارش سمینار درس داده کاوی

توسعه شبکه های عصبی پیچشی و کاربرد آن در طبقه بندی تصویر

نگارش

امیرحسین باقری

آرش حریرپوش

عرفان افشار

استاد

دکتر مزلقانی

بهمن ۹۹


بسم الله الرحمن الرحيم

چکیده

شبکه های عصبی مصنوعی[1] در کار های مربوط به تصاویر دارای یک سری مشکلات بودند، ابتدا اینکه این شبکه ها قاعده محل[2] را رعایت نمی کردند، بدین صورت که چون تمام نورون ها به یک دیگر متصل شده بودند، بنابراین ترتیب قرارگیری پیکسل ها کنار هم در آن ها در نظر گرفته نمی شد، و این برای کار های مربوط به تصاویر مطلوب نمی باشد به این دلیل که زمانی که یک شی در یک تصویر وجود دارد تمام اجزای آن شی در یک محدوده از تصویر قرار دارند، برای مثال هنگام مشاهده یک گربه در یک تصویر صورت و گوش گربه از نظر موقعیت پیکسل به یک دیگر نزدیک می باشند؛ اما شبکه های عصبی مصنوعی این قاعده را رعایت نمی کردند، و مشکل دیگری که این شبکه ها داشتند زیاد بودن پارامتر های قابل یادگیری در این شبکه ها می باشد، و هرچه ابعاد تصویر بزرگ تر بشود، این پارامتر ها نیز افزایش می یابد (در صورتی که ابعاد تصویر ورودی $3 \times 1000 \times 1000$ باشد به ۳ میلیون نورون در لایه اول برای شبکه های مصنوعی نیاز داریم)؛ برای حل مشکلات عنوان شده شبکه های عصبی کانولوشنی[3]معرفی شدند؛ این شبکه ها با عبور دادن کرنل از روی تصویر اصل محل را رعایت می کنند، بدین صورت که در این حالت هر درایه از ماتریس ورودی تنها با مضربی از درایه های همسایه خود ترکیب می شود، و با توجه به اینکه وزن های قرار گرفته در کرنل برای کل ماتریس استفاده می شود، بنابراین تعداد پارامتر های شبکه کاهش یافته است. پس از گذشت زمان با توسعه سخت افزار و افزایش داده های موجود برای آموزش شبکه های عصبی کانولوشنی، استفاده از این شبکه ها بیش از پیش افزایش یافت؛ اما یکی از مشکلاتی که برای توسعه این شبکه ها وجود دارد تنظیم هایپر پارامتر[4] های این شبکه ها می باشد، که عبارتند از: ابعاد کرنل های کانولوشن، مقدار Padding، مقدار Stride، ترتیب قرار گرفتن لایه های کانولوشن با کرنل با ابعاد مختلف یا Pooling، عمق شبکه و .... یکی از راه حل های حل این مشکل بررسی شبکه های عصبی کانولوشنی معروف می باشد، چرا که علاوه بر اینکه با مشاهده این شبکه ها می توانیم از آن ها برای طراحی شبکه های مورد نیاز خود ایده بگیریم، بلکه در خیلی از موارد نیز می توانیم از آن ها مجددا استفاده کنیم (با استفاده از یادگیری انتقالی[5] می توانیم دوباره از شبکه های عصبی کانولوشنی که قبل آموزش داده شده اند استفاده کنیم)؛ برای درک بهتر این شبکه ابتدا مقدمه ای از شبکه های عصبی کانولوشنی عنوان می شود، سپس عملیات های مختلفی که در

[1] Artificial Neural Network
[2] Locality
[3] Convolutional Neural Network
[4] Hyper Parameter
[5] Transfer Learninig

شبکه های عصبی کانولوشنال استفاده می شود را معرفی و بررسی می کنیم، و در نهایت به معرفی و بررسی پنج شبکه عصبی کانولوشنال معروف LeNet، AlexNet، VGG، Inception و ResNet که در سال های گذشته طراحی شده اند، پرداخته می شود.

واژه های کلیدی: شبکه عصبی مصنوعی، شبکه عصبی پیچشی، کلاس بندی تصاویر، کرنل، یادگیری انتقالی، LeNet، AlexNet، VGG، Inception و ResNet

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل اول

مقدمه

۱ - ۱ مقدمه

در سال های اخیر استفاده از شبکه های عصبی پیچشی در بخش های مختلف بینایی ماشین بسیار فراگیر شده و این دسته از شبکه ها عملکرد بسیار بهتری نسبت به روش های سنتی داشته اند[1].

یکی از مهمترین کاربرد های این شبکه ها در زمینه طبقه بندی تصاویر است. شبکه های عصبی پیچشی مورد استفاده در این زمینه، در سایر زمینه های بینایی ماشین مانند تشخیص اشیا،تقسیم بندی تصاویر و ... نیز مورد استفاده قرار میگیرند[1].

ما در ابتدا به تشریح ویژگی های عمومی شبکه های CNN می پردازیم، سپس معماری های مهم در روند توسعه CNN ها را مورد بررسی قرار می دهیم[1].

۱ - ۲ تاریخچه

هوبل و ویزل برای اولین بار مشاهده کردند که سلولهای عصبی در قشر بینایی به لبه متحرک حساس هستند، فوکوشیما و میاکه بر پایه همین مشاهدات شبکه "neocognitron" را عرضه کردند که می توان از آن به اولین پیاده سازی یک شبکه CNN یاد کرد، هرچند نبود الگوریتم یادگیری مناسب در آن زمان مانع گسترش آن شد. پس از معرفی الگوریتم های backpropagation ، لیکان از این الگوریتم برای یادگیری شبکه CNN خود استفاده کرد و شبکه LeNet را معرفی کرد[1].

اما این شبکه ها با افزایش عمق به مشکلات زیادی مثل بهینه محلی، کوچک شدن گرادیان و بیش برازش بر میخوردند تا اینکه هینتون نشان داد شبکه های عصبی با چند لایه پنهانی ، قابلیت یادگیری بسیار خوبی دارند و این آغازی بر یادگیری عمیق با شبکه های CNN بود[1].

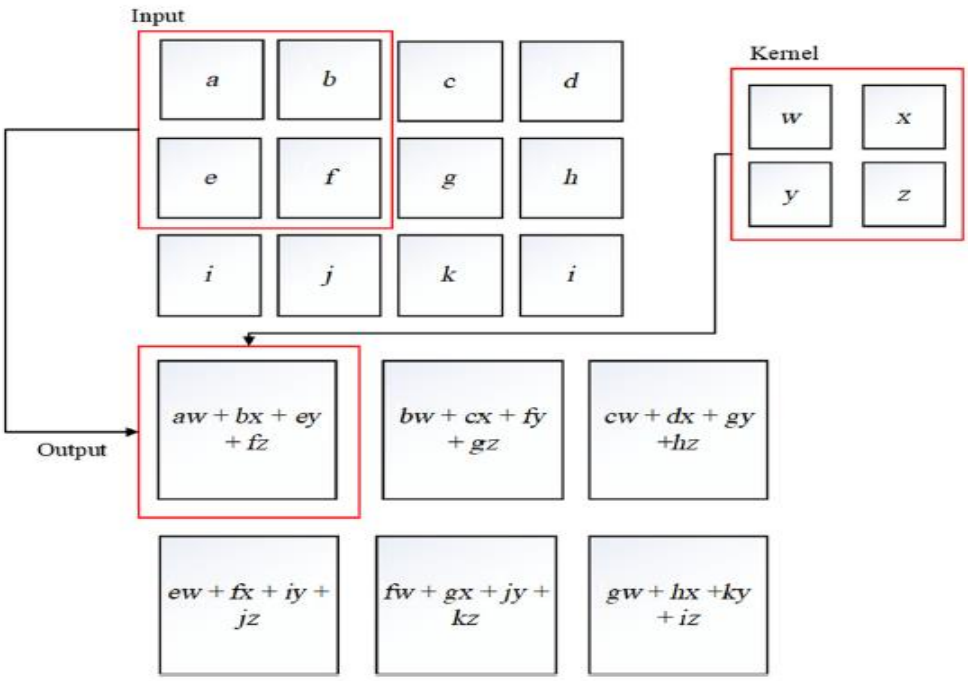
فصل دوم

تشریح شبکه های عصبی پیچشی

معماری شبکه های CNN به طور کلی از لایه پیچشی[1]، لایه تجمعی[2] و لایه کاملا متصل[3] تشکیل شده است.

۲ - ۱ لایه پیچش

در این لایه ابتدا یک کرنل پیچش مشخص می شود سپس عمل پیچش را مطابق شکل ۱ بر روی ورودی انجام می دهیم.



شکل ۱. نحوه محاسبه convolution

عمل پیچش بیشترین هزینه محاسباتی را در شبکه های CNN را دارد.

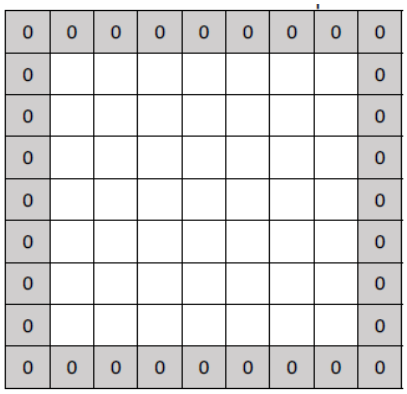
پارامتر های قابل یادگیری این لایه ضرایب کرنل هستند و ابعاد کرنل نیز یک هایپر پارامتر قابل تعیین می باشد.

[1] convolution
[2] pooling
[3] fully connected

از دیگر متغیر های قابل تنظیم این لایه طول گام[1] و استفاده/عدم استفاده از حاشیه[2] می باشد.

با افزایش طول گام می توانیم ابعاد خروجی را به سرعت کاهش دهیم و با استفاده از حاشیه می توانیم کاهش ابعاد ورودی حاصل از عمل پیچش را جبران کنیم و از مقادیر مرزی ورودی نیز بیشتر استفاده کنیم[1].

نمونه استفاده از حاشیه صفر به طول یک را در شکل ۲ قابل مشاهده است.



شکل ۲. نمونه zero padding

در این لایه نیز مشابه شبکه های عصبی ساده یک مقدار ثابت به عنوان bias در نظر میگیریم که خروجی حاصل از عمل پیچش را با آن جمع می کنیم.

استفاده از کرنل و عمل پیچش این اجازه را به ما میدهند تا ورودی با ابعاد مختلف را پردازش کنیم بدین ترتیب که لایه های اولیه ویژگی های سطح پایین تر را استخراج می کنند و لایه های عمیق تر ویژگی های کلی تر و سطح بالاتر[1].

عمل پیچش دارای دو ویژگی وابستگی تنک[3] و وزن های مشترک[4] هستند که باعث کاهش تعداد پارامتر ها و محاسبات نسبت به شبکه های عصبی معمولی می شوند[1].

منظور از وابستگی تنک این است که هر خانه در خروجی فقط به تعدادی خاصی خانه ورودی(به اندازه کرنل) وابسته است برخلاف شبکه های عصبی ساده که از تمامی خانه های ورودی اثر می پذیرند[1].

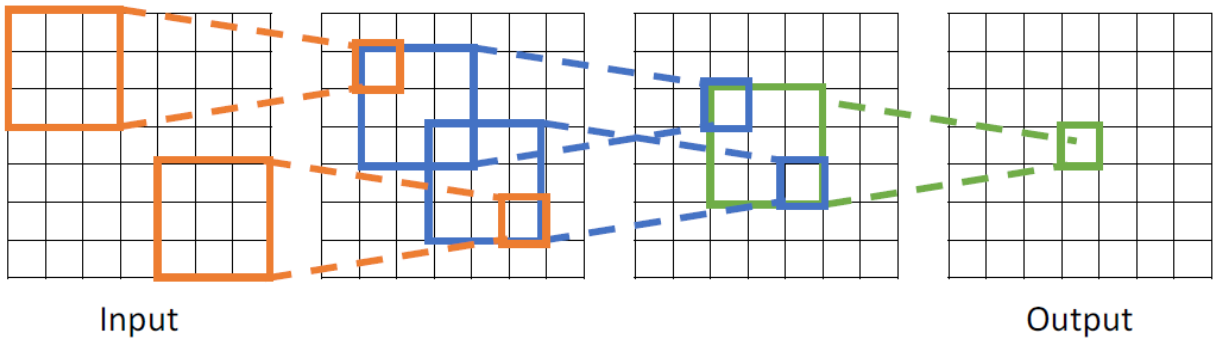
[1] stride
[2] padding
[3] sparse interaction
[4] weight sharing

و منظور از وزن های مشترک این است که با لغزاندن کرنل روی ورودی وزن های کرنل ثابت می مانند بر خلاف شبکه های عصبی ساده که هر خانه وزن متناظر خودش را دارد[1].

با توجه به این که عمل پیچش یک عمل خطی است پس به یک عملگر غیر خطی بین لایه های پیچش نیاز داریم، برای این امر می توانیم از توابع sigmoid یا ReLu استفاده کنیم[1].

در این لایه یک مفهوم به نام receptive fields تعریف می شود به این معنی که هر خانه در خروجی از چه خانه هایی در ورودی اثر می پذیرد[1].

همان طور که در شکل مشاهده می شود با افزایش تعداد لایه ها، receptive fields بزرگتر می شوند.



شکل ۳. receptive fields

۲ - ۲ لایه تجمعی

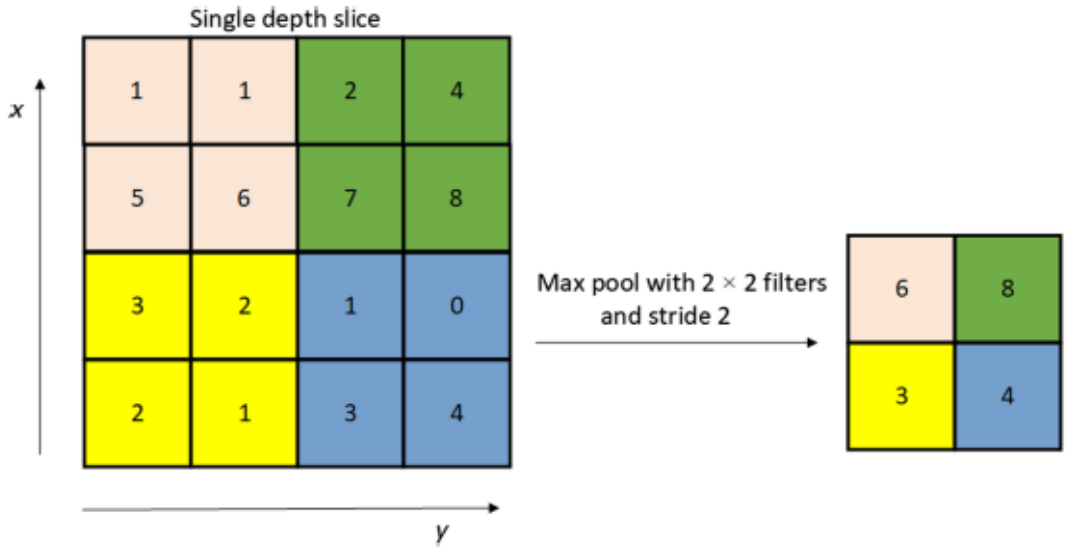
معمولا بعد از چند لایه پیچش از یک لایه تجمعی استفاده میکنیم.هدف این لایه کاهش ابعاد ورودی است تا هم هزینه محاسباتی به شکل قابل ملاحظه ای کاهش پیدا کند و هم بتوانیم ویژگی های کلی تری به دست بیاوریم.

برای این کار معمولا از دو روش max pooling و average pooling استفاده میکنیم .

در max pooling بزرگترین مقدار را در خروجی قرار می دهیم و در average pooling میانگین خانه های قرار گرفته روی کرنل را قرار می دهیم[1].

این که لایه پارامتر قابل یادگیری ندارد و هزینه محاسباتی بسیار کمی نسبت به لایه پیچش دارد که در عمل از آن صرف نظر می شود.

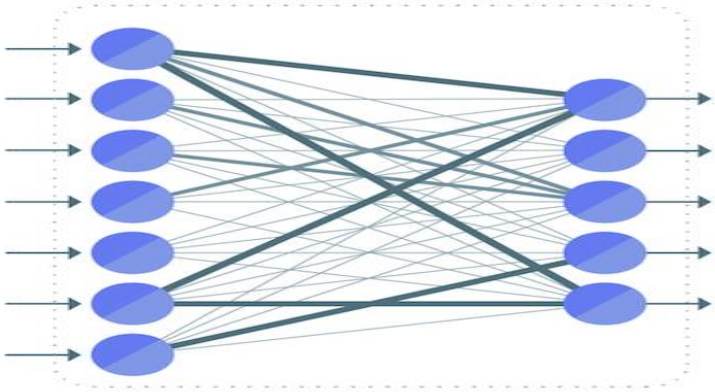
نمونه ای از عمل max pooling در شکل ۴ قابل مشاهده است.



شکل ۴. نحوه محاسبه max pooling

۲ - ۳ لایه کاملا متصل

آخرین لایه موجود در شبکه های CNN یک شبکه عصبی کاملا متصل است که اطلاعات محلی استخراج شده توسط لایه های قبل را با هم ترکیب و یکپارچه میکند و در نهایت با استفاده از یک لایه softmax خروجی نهایی را تولید میکند. این لایه از شبکه های CNN بیشترین تعداد پارامتر را دارد. و نمونه ای از آن در شکل ۵ قابل مشاهده است[1].



شکل ۵. شبکه کاملا متصل

فصل سوم

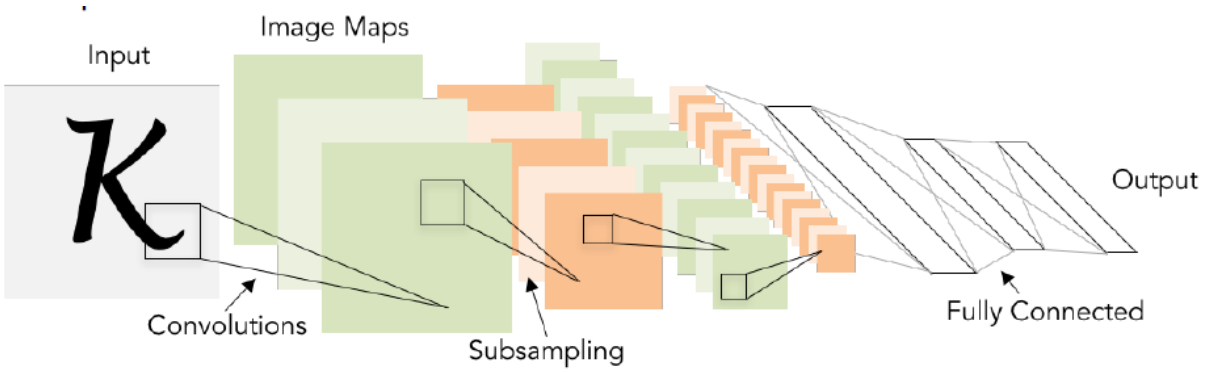
معماری شبکه های کلاسیک

بعد از معرفی ساختار CNN ها به بررسی سیر تکامل معماری های معروف آن می پردازیم.

۳ - ۱ شبکه LeNet

این شبکه یکی از معماری های کلاسیک CNN هست که برخی از ویژگی هایش مثل نحوه ترکیب لایه های مختلف، کاهش ابعاد و افزایش عمق (تعداد کانال ها) در طول شبکه در بسیاری از معماری های بعدی نیز به چشم می خورند[1].

طبق شکل ۶ این معماری از ۵ لایه تشکیل شده است. به ترتیب conv-pool-conv-pool-fc



شکل ۶. معماری شبکه LeNet

جزییات این شبکه در جدول ۱ قابل مشاهده است.

جدول ۱. معماری شبکه LeNet

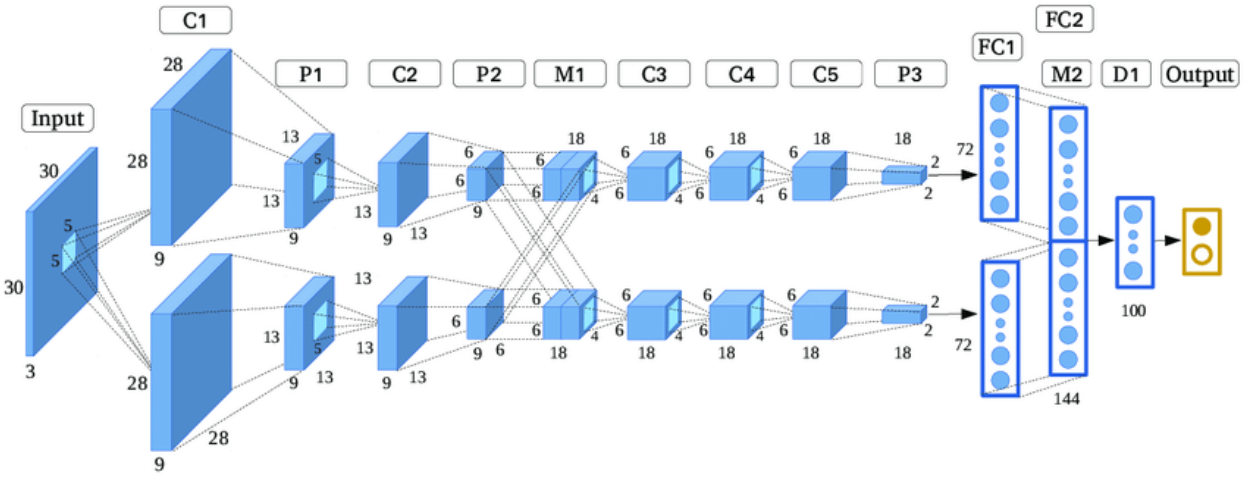
Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10

۳ - ۲ شبکه AlexNet

با توجه به عدم وجود داده ها[1] و سخت افزار[2] مناسب، بعد از معرفی شبکه LeNet-5 توجه زیادی را به خود جلب نکرد. چرا که در آن سال ها به دلیل محدودیت در انجام محاسبات و کم بودن داده ها شبکه ها به صورت کم عمق[3] استفاده می شدند. با توسعه سخت افزار ها و افزایش مقدار داده های موجود برای آموزش شبکه های عصبی، شبکه AlexNet توانست مسابقه ILSVRC را در سال ۲۰۱۲ با اختلاف زیاد نسبت به جایگاه دوم برنده

[1] Data
[2] Hardware
[3] Shallow

شود. بنابراین شبکه های عصبی عمیق[1] توجه بقیه را به خود جلب کرد. ساختار این شبکه در شکل ۷ نمایش داده شده است[1].



شکل ۷. این شکل شبکه AlexNet را نمایش می دهد که به دو قسمت تقسیم شده است.

۳ – ۲ – ۱ مقایسه شبکه AlexNet با LeNet-5

بهبود های شبکه AlexNet در مقایسه با LeNet-5 به صورت زیر می باشد:

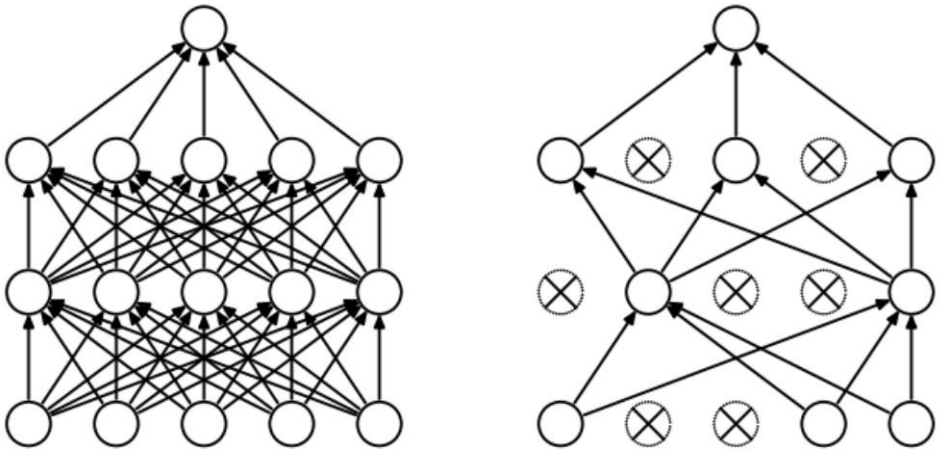
۱. **استفاده از تابع فعالیت[2] ReLU**: با استفاده از این تابع فعالیت ویژگی غیر خطی[3] و sparsity به شبکه اضافه می شود. ویژگی sparsity می تواند نورون ها را به صورت انتخابی یا به صورت توزیع شده فعال کند. و همچنین می تواند ویژگی های sparse را یاد بگیرد و به صورت خودکار گسستگی[4] ایجاد کند[1].

۲. **انجام Data Augmentation**: این شبکه دو راه برای کم کردن مشکل بیش برازش[5] استفاده کرده است؛ اولین راه برای کم کردن این مشکل افزایش داده های آموزشی می باشد، که در این معماری قبل از آموزش شبکه با استفاده از یک تبدیل با حفظ برچسب[6] مجموعه داده ها را بزرگ کرده است. انواع

[1] Deep Neural Network
[2] Activation Function
[3] Non-linearity
[4] Dissociation
[5] Over fitting
[6] Label-preserving

این Augmentation ها شامل تولید image translation ها، بازتاب های افقی[1]، و تغییر در شدت[2] کانال های RGB تصاویر آموزش می شده است [1, 2].

۳. **استفاده از Dropout**: دومین راهی که در این معماری برای کم کردن مشکل بیش برازش انجام شده است؛ استفاده از dropout در لایه های FC می باشد، بدین صورت که با توجه به یک درصد مشخص تعدادی از نورون ها در یک لایه غیر فعال می شوند؛ این کار باعث کم شدن پارامتر های شبکه می شود، بنابراین با ساده تر شدن مدل در لایه های FC[3] احتمال رخداد بیش برازش کاهش می یابد[1]. شکل ۸ بخش a یک شبکه عصبی مصنوعی را قبل از انجام عملیات dropout نمایش می دهد، و شکل b همان شبکه را پس از انجام عملیات dropout در لایه های مختلف آن شبکه را نشان می دهد، برای مثال در لایه اول این شبکه یک dropout با مقدار ۰,۵ انجام شده است یعنی نصف نورون های لایه اول این شبکه غیر فعال شده است؛ همانطور که مشاهده می شود با انجام این کار تعداد پارامتر های شبکه عصبی مصنوعی کاهش پیدا می کند و باعث ساده تر شدن مدل می شود، که این کار سبب می شود تا احتمال رخداد بیش برازش کاهش یابد.



شکل ۸. این شکل عملیات Dropout را نمایش می دهد.

۴. **آموزش بر روی دو GPU به نام NVIDIA GTX 580 3GB**: یک GPU به نام GTX 580 تنها 3GB حافظه دارد، که این مقدار باعث ایجاد محدودیت در سایز شبکه ای می شد که قرار بود به وسیله آن آموزش داده شود. و مشخص شد که ۱,۲ میلیون نمونه آموزشی برای آموزش شبکه های عصبی

[1] Horizontal reflection
[2] Intensity
[3] Fully Connected

کافی می باشد اما برای یک GPU مقدار زیادی می باشد. بنابراین تصمیم گرفته شد تا شبکه را بین دو GPU تقسیم کنند. با توجه به اینکه GPU ها در آن زمان می توانستند به صورت موازی با یک دیگر کار کنند، بدین صورت که از روی حافظه یک دیگر بخوانند یا بر روی حافظه یک دیگر بنویسند، بدون اینکه از حافظه ماشین میزبان[1] عبور کند. برای انجام موازی سازی مد نظر خود، آن ها نصف هر کدام ازکرنل ها را در هر یک از GPU ها قرار دادند، و GPU ها در برخی از لایه ها با یک دیگر ارتباط برقرار می کردن[1] .

شکل ۷ این موازی سازی را نمایش می دهد، همانطور که مشاهده می شود از ابتدا عملیات به دو بخش تقسیم شده است که هر یک از این بخش ها در یک GPU انجام می شود؛ و علاوه بر عملیات Conv و Pool یک عملیات Merge هم استفاده شده است که نتایج دو GPU را با یک دیگر ادغام می کند و سپس دوباره عملیات های مورد نظر را بر روی feature map های ایجاد شده انجام می دهد. (به هر یک از خروجی هایی که توسط عملیات convolution با یک کرنل بدست می آید، feature map یا activation map گفته می شود)

۵. **نرمال سازی پاسخ محلی**[2] **(LRN)**: داده های نزدیک به یک دیگر برای انجام عملیات نرمال سازی استفاده می شوند، که این کار باعث بهبود عملیات دسته بندی[3] می شود[1].

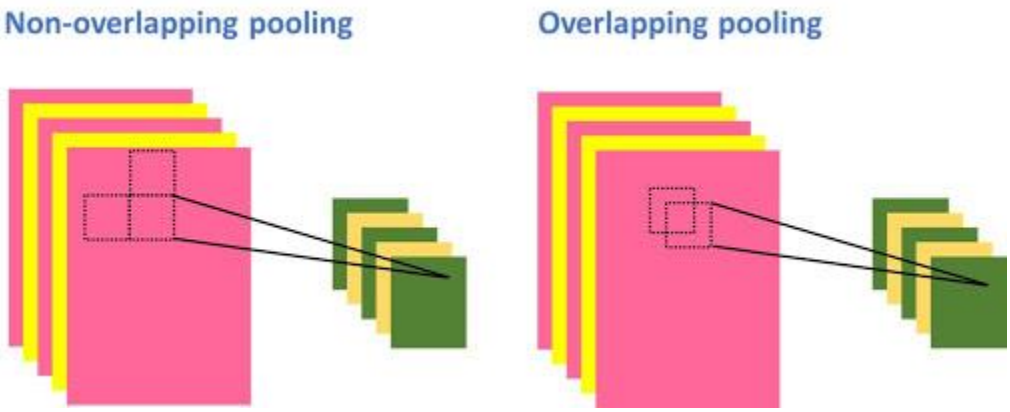
پس از معرفی روش Batch Normalization در سال های بعد این روش نرمال سازی دیگر استفاده نمی شود و بجای آن از Dropout یا Batch Normalization استفاده می شود.

۶. **Overlapping pooling**: لایه های pooling در شبکه های CNN داده ها را بر اساس همسایه هایشان خلاصه سازی می کنند. در شبکه LeNet-5 به دلیل برابر بودن مقدار stride با سایز kernel، هنگامی که کرنل را بر روی feature map جابجا می کردیم، مکان هایی که هر بار کرنل قرار می گرفت با مکان های قبلی اشتراکی نداشت. در شبکه AlexNet سایز stride را عملیات pooling کمتر از سایز کرنل قرار می دهند، که به آن overlapping pooling می گویند، به این دلیل که زمانی که در این عملیات کرنل را بر روی feature map ها جابجا می کنیم، مکان هایی که کرنل هر بار در آن قرار می گیرد با بخشی از مکان هایی که قبلا در آن بوده است اشتراک دارد؛

[1] Host machine
[2] Local response normalization
[3] Classification

همچنین با انجام این کار مشاهده شد که امکان رخداد بیش برازش در این حالت بسیار کاهش پیدا می کند[1, 2].

شکل ۹ این عملیات را نمایش می دهد، همانطور که در شکل ۹ مشاهده می شود در عملیات non-Overlapping pooling از هر درایه از ماتریس ورودی تنها یک بار کرنل قرار می گیرد، در حالی که در عملیات Overlapping pooling از هر درایه از ماتریس ورودی ممکن است چند بار کرنل قرار بگیرد، که این بدین معنی است که ممکن است هر درایه از ماتریس ورودی در بیش از یک درایه از ماتریس خروجی تاثیر داشته باشد.



شکل ۹. شکل سمت چپ عملیات pooling ساده و شکل سمت راست عملیات Overlapping pooling را نمایش می دهد.

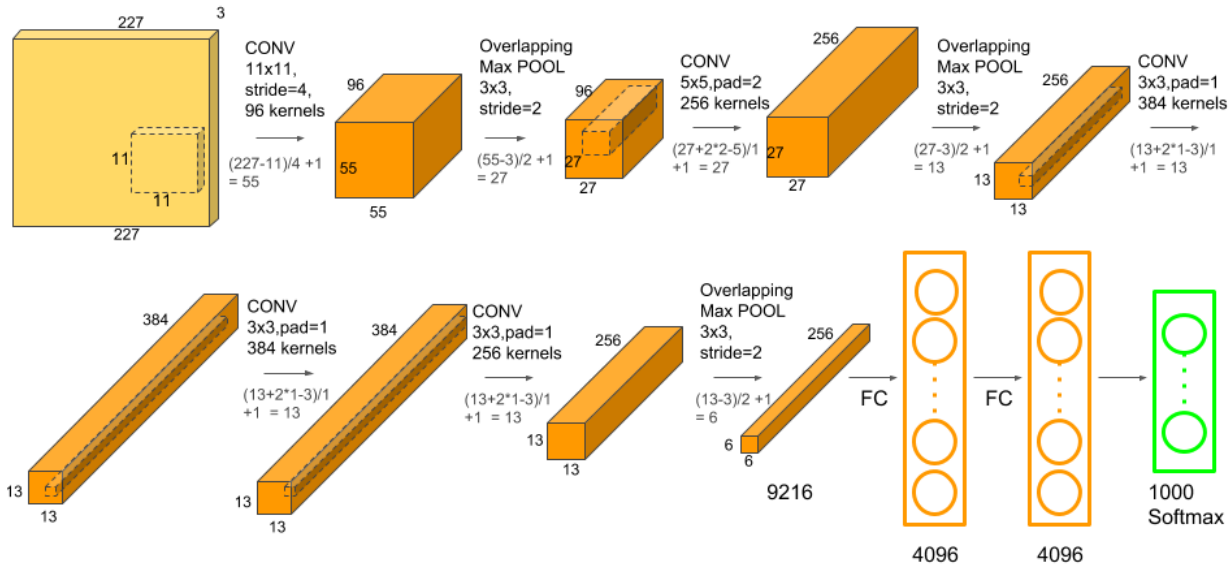
۳ – ۲ - ۲ آموزش شبکه AlexNet

این شبکه بر روی مجموعه داده[1] ImageNet که شامل ۱۵ میلیون عکس با کیفیت که به 22000 کلاس تعلق داشتند آموزش داده شده است. این مجموعه داده از عکس های با رزولوشن[2] متفاوت تشکیل شده است، در حالی که برای آموزش این شبکه به تصاویر با ابعاد ثابت نیاز داشته اند، به همین دلیل عکس های موجود را به ابعاد ثابت ۳ × 224 × 224 تبدیل کرده اند؛ بنابراین همانطور که عنوان شد این شبکه ابتدا عکس های با ابعاد ۳ × 224 × 224 را دریافت می کند، سپس یک عملیات Conv[3] با ابعاد فیلتر 11 × 11 و stride برابر ۴ و تعداد فیلتر ۹۶ تا بر روی آن انجام می دهد که این کار باعث ایجاد تغییر ابعاد ماتریس ورودی به ابعاد ۹۶ × ۵۵ × ۵۵ می شود، و سپس یک عملیات Overlapping Max POOL با کرنل به ابعاد 3 × 3 stride برابر

[1] Data set
[2] Resolution
[3] Convolution

۲ انجام می دهد که این کار ابعاد ماتریس خروجی را به ۹۶ × ۲۷ × ۲۷ تغییر می دهد، این تغییر ابعاد زیاد در یک لایه از شبکه (لایه pooling به دلیل عدم وجود پارامتری برای یادگیری جزو لایه های شبکه محاسبه نمی شوند) مطلوب نمی باشد، به این دلیل که ممکن است باعث از بین رفتن اطلاعاتی از داده شود، اما احتمالا با توجه به محدودیت های محاسباتی موجود در آن سال این تغییر ابعاد ناگهانی در یک لایه انجام شده است.

سپس همانطور که در شکل ۱۰ مشاهده می شود تعدادی عملیات Conv و Pool در لایه های بعدی انجام شده است، و در نهایت خروجی لایه ۶ این شبکه که ابعاد برابر ۲۵۶ × ۶ × ۶ را مسطح[1] می کند و به دو لایه FC که هر کدام ۴۰۹۶ نورون دارد می دهد، و در نهایت خروجی آخرین لایه FC را به یک لایه softmax می دهد، تا احتمال تعلق به هر یک از کلاس ها برای داده ورودی مشخص شود.

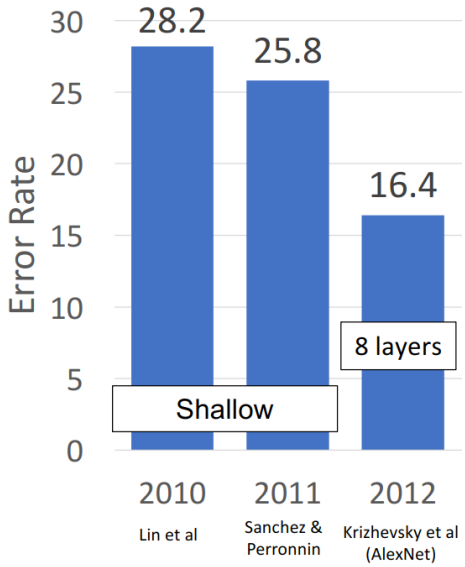


شکل ۱۰. ترتیب عملیات انجام شده در شبکه AlexNet را نمایش می دهد.

۳ – ۲ - ۳ نتایج بدست آمده توسط شبکه AlexNet و تعداد پارامتر ها

همانطور که در شکل ۱۱ مشاهده می شود، نرخ خطای Top5 در شبکه AlexNet برابر ۱۶,۴٪ می باشد، که در مقایسه با ۲۵,۸٪ که در سال ۲۰۱۱ بهترین نتیجه بوده است، تفاوت قابل توجهی داشته است.

[1] Flatten



شکل ۱۱. نرخ خطای Top5 بدست آمده توسط شبکه های مختلف تا سال ۲۰۱۲

با توجه به جدول ۲ در این شبکه حدود۶۲ میلیون پارامتر وجود دارد، که بیشتر آن ها متعلق به لایه های FC می باشد.(حدود ۵۸ میلیون از ۶۲ میلیون پارامتر این شبکه مربوط به لایه های FC این شبکه می باشد)

جدول ۲. تعداد پارامتر های آموزش داده شده در شبکه AlexNet

AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							**62,378,344**

۳ - ۳ شبکه VGG

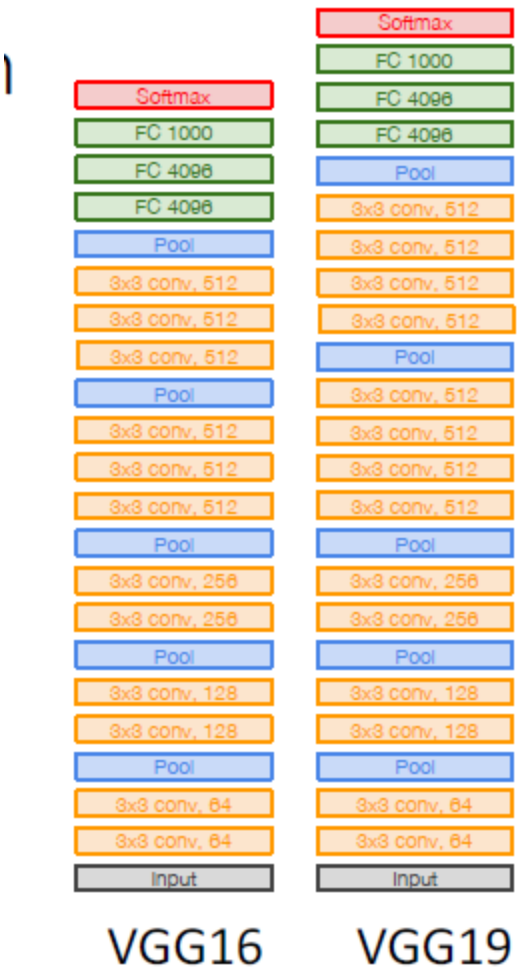
شبکه های با عمق کم برای کاربردهای بینایی ماشین بزرگ مقیاس با محدودیت رو به رو می شوند.

پس محققان برای بهبود عملکرد شبکه های CNN، افزایش عمق شبکه را مد نظر قرار دادند.

یکی از این شبکه ها شبکه VGG بود که توانست با افزایش عمق شبکه به ۱۹/۱۶ لایه و به تبع آن هزینه محاسباتی بسیار بالا به بهبود قابل توجهی را به دست بیاورد[1].

علاوه بر این VGG با نوآوری های خود یک سری اصول طراحی برای شبکه های CNN پدید آورد که معماری های بعدی نیز از این اصول بهره برده اند از جمله:

- استفاده از کرنل های ۳x۳ به جای کرنل با ابعاد مختلف. همانطور که در بخش ۲ - ۱ مشاهده کردیم میتوان یک کرنل ۵x۵ را با استفاده از دو لایه با کرنل های ۳x۳ شبیه سازی کنیم.با این کار علاوه بر کاهش هزینه محاسباتی و پارامتر ها، با توجه به این که می توانیم از دو تابع غیر خطی استفاده کنیم ،میتوانیم توابع پیچیده تری را اعمال کنیم[1].
- بعد از هر لایه تجمعی که ابعاد نصف می شوند ،تعداد کانال ها دو برابر می شوند تا هزینه محاسباتی در طول شبکه ثابت بماند[1].
- معماری متشکل از ساختار های بلوکی. همانطور که در تصویر زیر مشاهده می شود VGG از تکرار بلوک های conv-conv-pool ساخته شده است[1].



شکل ۱۲. معماری شبکه VGG

همانطور که در شکل ۱۲ مشاهده می شود VGG شامل ۵ بخش پیچش است.

Stage 1: conv-conv-pool

Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

Stage 5: conv-conv-conv-[conv]-pool

فصل چهارم

معماری شبکه

GoogLeNet/Inception

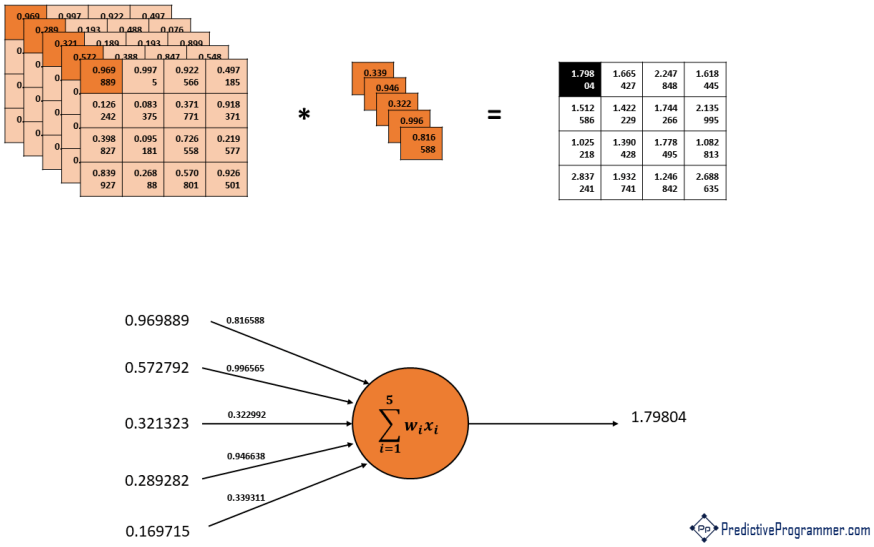
VGG با استفاده از کرنل های با ابعاد کوچک برای کانولوشن توانست شبکه عمیق تری را مدل کند، و GoogLeNet با الهام از network in network ساختار شبکه خود را گسترش داد و توانست inception module را پیشنهاد کند. استفاده از inception module در شبکه های عصبی باعث می شود تا نه تنها عمق و ابعاد مدل شبکه افزایش یابد، بلکه آن مدل بهتر بتواند محتوای داده های ورودی را توصیف کند[1].

۴ - ۱ نوآوری های ایجاد شده در شبکه GoogLeNet

نوآوری هایی که در این شبکه انجام شده است، عبارتند از:

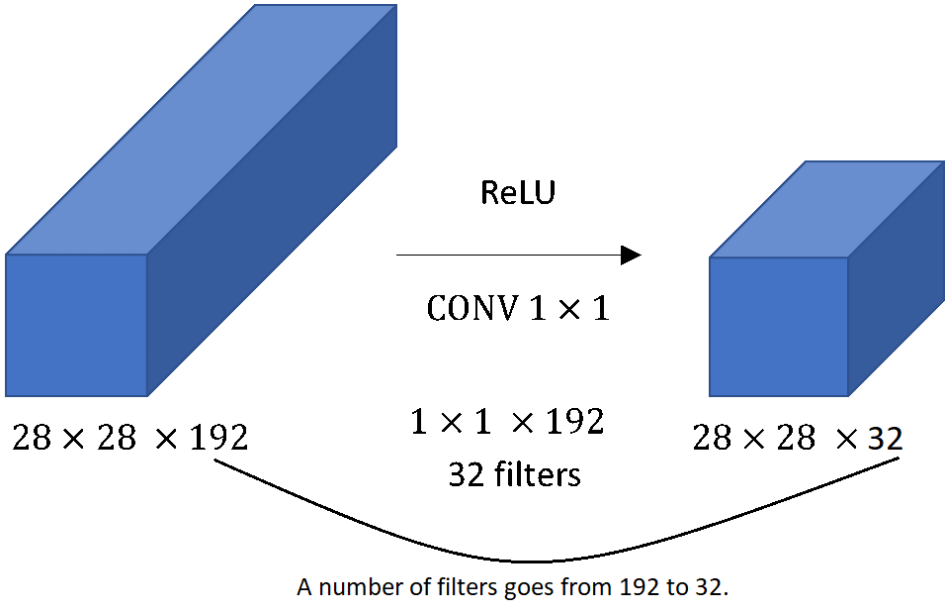
۱. **استفاده از کانولولشن با ابعاد کرنل 1 × 1**: شبکه GoogLeNet اولین شبکه ای می باشد که از این عملیات استفاده کرده است، و تا قبل از آن به دلیل عدم آشنایی با مزایای استفاده از آن این عملیات مورد استفاده قرار نگرفت؛ این عملیات بدین صورت انجام می شود که برای هر feature map یک وزن که همان کرنل می باشد در نظر گرفته می شود و یک درایه با مکان یکسان از همه feature map ها با وزن های متناظر با آن feature map ضرب می شود و در نهایت این اعداد با یک دیگر جمع می شود، یعنی بدین صورت عمل می کند که هر یک از درایه های feature map ها با در نظر گرفتن وزن های مشخص با یک دیگر جمع می کند و همچنین می توان در ابتدا حاصل جمع را با یک بایاس جمع کرد، و برای اضافه کردن خاصیت غیر خطی هم می تواند حاصل این جمع را به عنوان ورودی به یک تابع فعالیت[1] دهیم و خروجی تابع فعالیت را در ماتریس خروجی قرار دهیم؛ شکل ۱۳ یک نمونه از انجام محاسبات برای بدست آوردن یک درایه از ماتریس خروجی بدست آمده از طریق این عملیات را نشان می دهد. با انجام این عملیات می توان عمق ماتریس های خروجی را تغییر داد، بنابراین می توان عمق ماتریس های خروجی را تغییر داد و باعث کاهش هزینه های محاسباتی در لایه های بعدی و کاهش مقدار پارامتر های مدل شد.

[1] Activation Function



شکل ۱۳. نحوه محاسبه خروجی کانولوشن با ابعاد کرنل 1 × 1

شکل ۱۴ یک نمونه از این عملیات را نمایش می دهد که همانطور که مشاهده می شود این عملیات طول و عرض ماتریس را تغییر نمی دهد، اما می تواند عمق ماتریس خروجی را که برابر تعداد کرنل های استفاده شده در کانولوشن می باشد تغییر دهد، که در این شکل عمق تصویر ورودی از ۱۹۲ به ۳۲ کاهش یافته است.



شکل ۱۴. یک عملیات کانولوشن با ابعاد کرنل 1 × 1 را نشان می دهد.

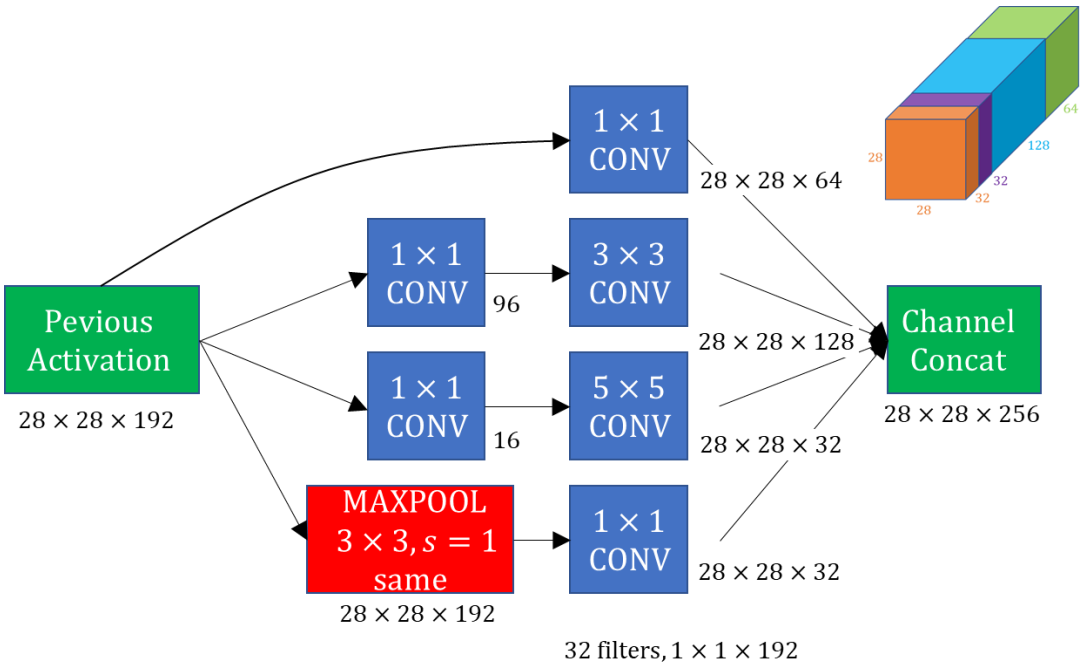
۲. **Inception Module**: یکی از کارهایی که در لایه های کانولوشن انجام می شود کاهش یا افزایش ابعاد با استفاده از مشخص کردن تعداد کرنل های آن لایه می باشد. در نسخه اول شبکه Inception با استفاده از کانولوشن های 1 × 1 ابعاد داده کم شده است، که این کار می تواند باعث کاهش تعداد پارامتر های شبکه و feature map ها شود. عملیات کانولوشن 1 × 1 همانند تبدیل مقیاس[1] عکس اصلی می باشد، با این تفاوت که اندازه تصویر تغییر نکند، که این کار می تواند دقت دسته بندی تصاویر را بهبود دهد[1].

با توجه تنوع بسیار زیاد در موقعیت های اطلاعات، انتخاب ابعاد مناسب کرنل برای انجام عملیات کانولوشن سخت می باشد، کرنل های با ابعاد بزرگ برای اطلاعاتی که به صورت جامع تر[2] در توزیع شده اند مناسب می باشد، و کرنل های با ابعاد کوچک تر برای اطلاعاتی که به صورت محلی تر[3] توزیع شده اند مناسب می باشد[4].

بنابراین ایده اصلی پشت inception module این است که اطلاعات کلیدی قرار گرفته در تصویر در سطوح مختلفی از جزئیات قرار گرفته اند. اگر از کرنل های با ابعاد بزرگ استفاده کنیم، می توان اطلاعات را از محدوده بزرگ تری که شامل تغییرات محدود می باشد بدست آورد، و اگر از کرنل های با ابعاد کوچک استفاده کنیم، می توان اطلاعاتی را که در یک محدوده کوچک قرار گرفته اند بدست آورد. سپس برای اینکه بتوان اطلاعاتی را که در سطوح مختلف قرار دارند در نظر گرفت، inception module سه ابعاد متفاوت 1 × 1، 3 × 3 و 5 × 5 را برای کرنل ها را به صورت موازی در نظر می گیرد. و با توجه به اینکه کرنل های قرار گرفته در inception module قابلیت یادگیری دارند، شبکه عصبی می تواند تصمیم بگیرد که کدام یک از کرنل ها تاثیر بیشتری بر روی خروجی دارد[3].

همانطور که در شکل ۱۵ مشاهده می شود، inception module نسخه اول دارای چهار شاخه محاسباتی می باشد که هر کدام کرنل های با ابعاد متفاوت را بر روی ماتریس های ورودی انجام می دهد و در نهایت خروجی همه شاخه ها کنار یک دیگر قرار می گیرد (خروجی هر یک از شاخه ها به یک دیگر متصل[4] می شود) و به عنوان ورودی به لایه بعدی داده می شود. همچنین تمام عملیاتی که در inception module ها انجام می شود Same می باشد بدین صورت که با انجام padding به اندازه مناسب باعث عدم کاهش طول و عرض ماتریس ورودی می شود.

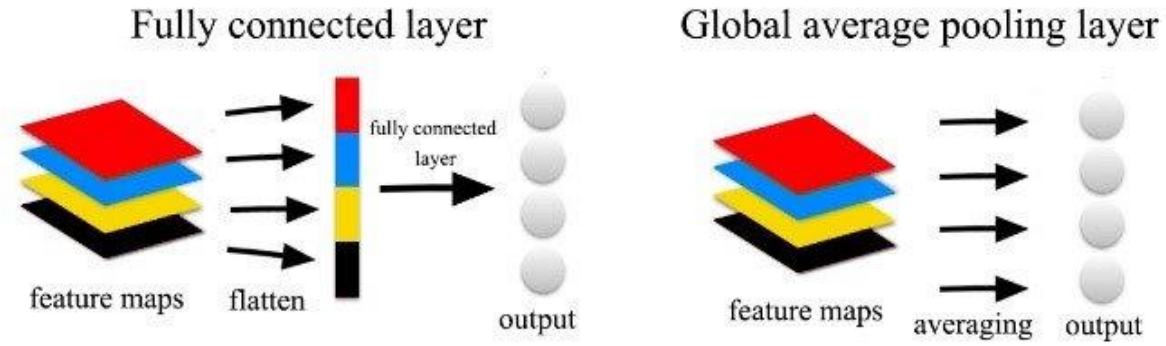
[1] Scale transformation
[2] Globally
[3] Locally
[4] Concatenate



شکل ۱۵. Inception Module نسخه اول را نشان می دهد.

۳. **Global Average Pooling**: در شبکه های قبلی پس از رسیدن به آخرین لایه های کانولوشن یا pooling ماتریس آن لایه Flatten می شد یعنی به یک ماتریس با طول و عرض یک تبدیل می شد و در نهایت به شبکه های FC داده می شد تا با ترکیب ویژگی های بدست آمده خروجی نهایی محاسبه گردد؛ اما یکی از مشکلاتی که این کار داشت زیاد بودن تعداد پارامتر ها در لایه های FC بود که باعث افزایش میزان محاسبات مورد نیاز برای آموزش شبکه می شد؛ در شبکه های Inception برای حل این مشکل و کاهش تعداد پارامتر ها تعداد لایه های FC کاهش پیدا می کند و بجای آن از Global Average Pooling استفاده می شود؛ بدین صورت که پس از آخرین لایه Inception Module یک Average Pooling با سایز کرنل برابر طول و عرض feature map های ورودی انجام می دهد، با انجام این کار به اندازه عمق ماتریس ورودی feature map های جدید با طول و عرض یک ایجاد می شود که هر یک برابر میانگین ویژگی های موجود در feature map ورودی این لایه می باشد؛ در نتیجه با انجام این کار توانسته ویژگی های قبلی را با یک دیگر ترکیب کند و ویژگی های جدیدی ایجاد کند و سپس این ویژگی ها را به یک لایه FC و سپس Softmax بدهد تا خروجی نهایی مشخص گردد؛ شکل ۱۶ سمت چپ عملیات هایی که در شبکه های قبلی برای بدست آوردن خروجی انجام می شده است نشان داده شده است، که همانطور که مشاهده می شود ابتدا feature map ها

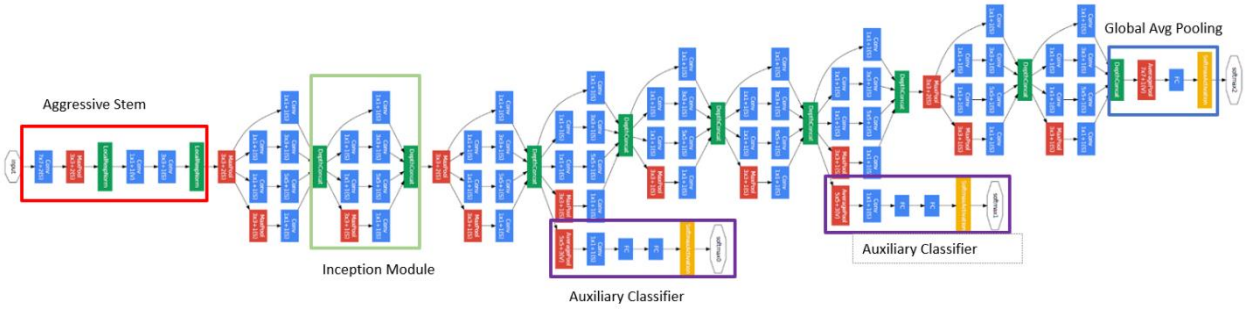
Flatten می شوند و سپس با ترکیب آن ها توسط لایه های FC ویژگی های جدیدی ایجاد می شود که به کمک آن خروجی نهایی شبکه مشخص می شده است، و در سمت راست عملیات Global Average Pooling مشاهده می شود که انجام عملیات Average Pooling توانسته ویژگی های جدیدی ایجاد کند و سپس به کمک آن ها خروجی شبکه را مشخص کند.



شکل۱۶. تفاوت عملیات Global Average Pooling با لایه FC را نشان می دهد.

۴ - ۲ معماری شبکه GoogLeNet

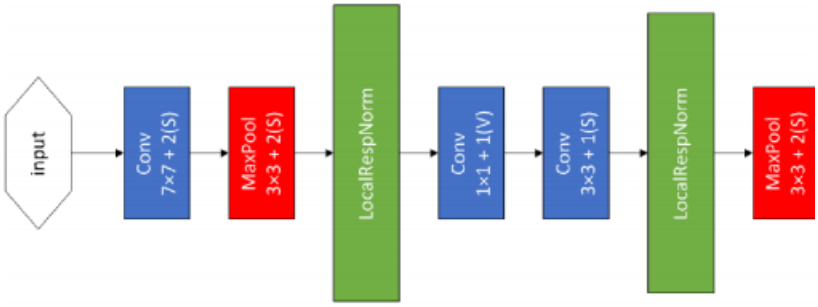
شکل ۱۷ معماری این شبکه را نمایش می دهد، همانطور که مشاهده می شود این شبکه از بخش های متنوعی ایجاد شده است که عبارتند از:



شکل ۱۷. معماری شبکه GoogLeNet را نشان می دهد.

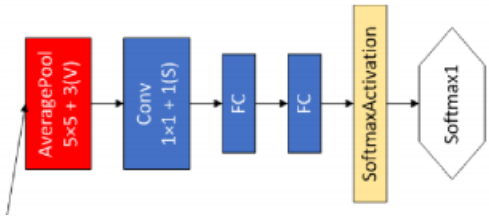
۱. **Aggressive Stem**: در ابتدا این شبکه یک Aggressive Stem قرار دارد، که در آن مجموعه ای از عملیات Conv و Pool انجام می شود تا برای کاهش محاسبات مورد نیاز در

لایه های بعدی ابعاد ماتریس ورودی را کاهش دهد و عمق آن را افزایش دهد، شکل ۱۸ این بخش از شبکه را نمایش می دهد ؛ که همانطور که مشاهده می شود در خروجی برخی از لایه ابتدا نرمال می شود و سپس به عنوان ورودی به لایه بعدی داده می شود.



شکل ۱۸. بخش Aggressive Stem شبکه GoogLeNet را نشان می دهد.

۲. **Auxiliary Classifier**: با توجه به عمیق بودن این شبکه امکان رخ دادن مشکل ناپدید شدن گرادیان[1] وجود داشت، بدین صورت که مقدار گرادیان بخاطر تابع فعالیت یا ضرایب خیلی کوچک در لایه آخر به شدت کاهش پیدا کند و به همین دلیل یادگیری به خوبی انجام نشود، در این شبکه برای حل این مشکل دو خروجی دیگر نیز از درون این شبکه در لایه های وسط ایجاد کردند تا در صورتی که مقدار گرادیان در لایه های آخر به شدت کاهش پیدا کردن و نتیجه مطلوبی از آخرین لایه بدست نیامد بتوانند از دو خروجی دیگر استفاده کنند که با توجه به اینکه آن خروجی ها در لایه های بالاتری نسبت به خروجی نهایی قرار دارند بنابراین احتمال رخ دادن مشکل ناپدید شدن گرادیان کمتر است؛ شکل ۱۹ این Classifier ها را نمایش می دهد؛ لازم به ذکر است که این بخش از این شبکه در آینده با معرفی لایه Batch Normalization می تواند از شبکه حذف گردد.



[1] Gradient Vanishing

شکل ۱۹. بخش Auxiliary Classifier شبکه GoogLeNet را نشان می دهد.

همانطور که در جدول ۳ مشاهده می گردد، در این شبکه ابتدا برای کاهش هزینه های محاسباتی یک down sampling توسط Aggressive Stem بر روی تصویر انجام می شود تا طول و عرض آن کاهش و تعداد ویژگی های بدست آمده که در اینجا عمق ماتریس می باشد افزایش یابد، که این کار باعث می شود تا ابعاد ماتریس ورودی از ۳ × ۲۲۴ × ۲۲۴ (در اینجا به دلیل اینکه بر روی تصاویر محاسبات انجام می شود و یک تصویر یک ماتریس سه بعدی می باشد که دارای سه کانال RGB می باشد به عمق ماتریس های بدست آمده در خروجی هر لایه تعداد کانال[1] هم گفته می شود) به ۱۹۲ × ۲۸ × ۲۸ تبدیل شود، پس از آن لایه های Inception Module و Max Pooling چندین بار تکرار می شود تا در نهایت ابعاد ماتریس به ۱۰۲۴ × ۷ × ۷ تبدیل شود، سپس یک عملیات Average Pooling با کرنل به ابعاد ۷ × ۷ که به Global Average Pooling معروف است (به این دلیل که ابعاد کرنل این عملیات با ابعاد ماتریس ورودی برابر است به این نام معروف شده است و در صورت تغییر ابعاد ماتریس ورودی ابعاد کرنل این ماتریس نیز تغییر می کند) انجام می شود، که این کار ابعاد ماتریس را به ۱۰۲۴ × ۱ × ۱ تبدیل می کند و در نهایت این ماتریس را به عنوان ورودی به یک لایه FC و سپس Softmax می دهیم تا خروجی نهایی محاسبه شود.

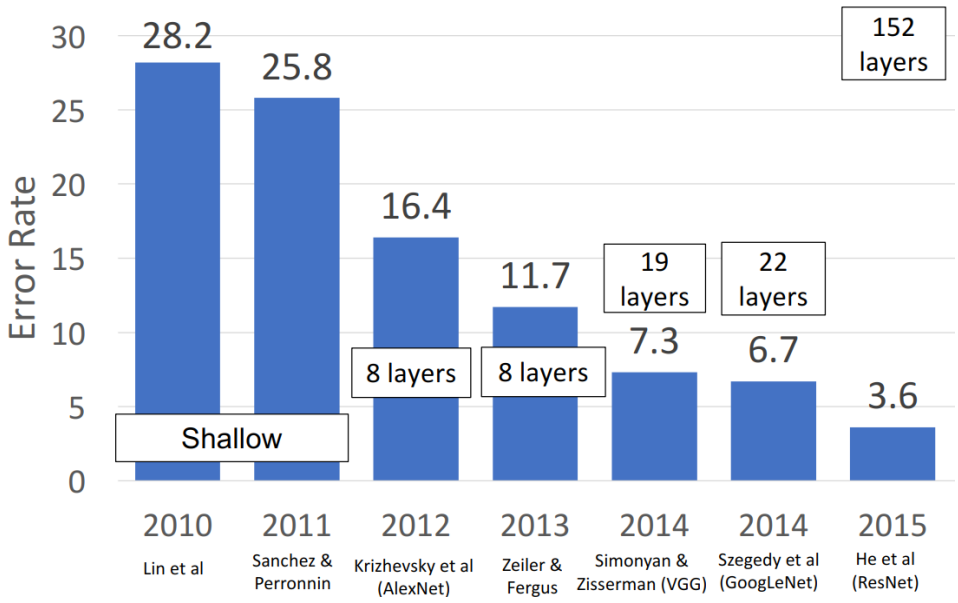
[1] Number of Channels

جدول ۳. تعداد پارامتر ها و ابعاد ماتریس های بدست آمده در هر لایه

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

۴ - ۳ نتایج بدست آمده توسط شبکه GoogLeNet

همانطور که در شکل ۲۰ مشاهده می شود، شبکه GoogLeNet با افزایش تعداد لایه های شبکه به ۲۲ تا و عمیق تر کردن شبکه ها و اضافه کردن برخی نوآوری ها و کاهش تعداد پارامتر های قابل یادگیری توسط شبکه توانسته است در سال ۲۰۱۴ نرخ خطای Top5 را ۵٪ نسبت به سال ۲۰۱۳ کاهش دهد و به نرخ خطای Top5 برابر ۶,۷٪ برسد و توانست جایگاه نخست بهترین مدل را در سال ۲۰۱۴ بدست آورد.



شکل ۲۰. نرخ خطای Top5 بدست آمده توسط شبکه های مختلف تا سال ۲۰۱۵

درحالی که هدف طراحان شبکه VGG استفاده حداکثری از منابع محاسباتی برای کاهش نرخ خطای شبکه بود، Google شبکه Inception را با هدف استفاده بهینه از منابع محاساباتی و کاهش تعداد پارامتر های مدل توسعه داده بود. همانطور که در جدول ۴ مشاهده می شود شبکه Inception تعداد پارامترهای قابل یادگیری بسیار کمتری نسبت به VGG دارد و هزینه محاسباتی آن نیز بسیار کمتر است؛ اما کمتر شدن تعداد پارامتر های قابل یادگیری لزوما باعث کاهش هزینه محاسبات مورد نیاز برای آن شبکه نمی شود، زیرا با توجه به اینکه شبکه Inception دارای ۶,۴ میلیون پارامتر قابل یادگیری می باشد که این مقدار بسیار کمتر است ۶۲ میلیون پارامتر قابل یادگیری شبکه AlexNet می باشد، اما هزینه محاسباتی شبکه Inception از شبکه AlexNet بیشتر می باشد.

جدول ۴. جدول مقایسه شبکه های CNN

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

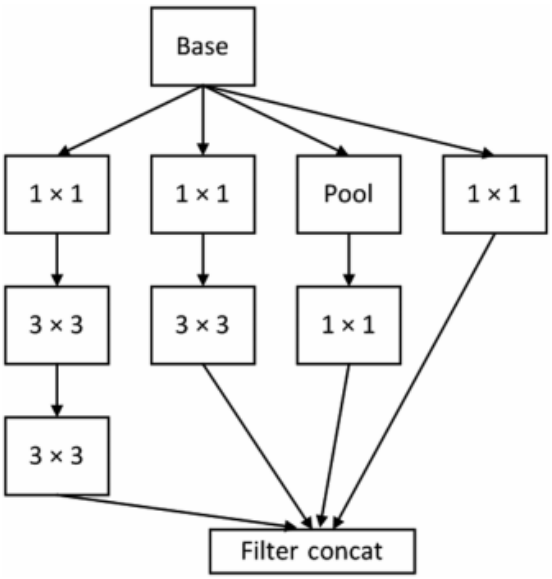
۴ - ۴ نسخه دوم شبکه Inception

شکل ۲۱ نسخه دوم Inception Module را نشان می دهد؛ تغییراتی که در این نسخه از شبکه Inception انجام شده است عبارت است از:

۱. لایه BN (Batch Normalization) برای نرمال سازی خروجی هر یک از لایه ها بر اساس توزیع گوسی[1] $N(0,1)$ اضافه شده است. که باعث می شود شبکه سریع تر همگرا شود و بتواند آزادانه تر مقدار دهی اولیه[2] شود. لایه BN نه تنها ویژگی robustness مدل را افزایش می دهد (مدل را نسبت به نویز[3] و داده های پرت[4] مقاوم تر می کند) بلکه استفاده از Dropout به عنوان یک روش منظم سازی[5] را کاهش می دهد[1].

۲. با توجه به شبکه VGG که بجای استفاده از یک کانولوشن با کرنل به ابعاد 5 × 5 از دو کانولوشن پشت سر هم با کرنل به ابعاد ۳ × ۳ استفاده می کند؛ در این نسخه از شبکه Inception نیز کانولوشن های با کرنل به ابعاد ۵ × ۵ در نسخه اول Inception Module با دو کانولوشن به ابعاد ۳ × ۳ تعویض می شود (به این دلیل که دارای receptive field های یکسان می باشند). که شکل ۲۱ نسخه دوم Inception Module را نمایش می دهد؛ این کار باعث می شود تعداد پارامتر های قابل یادگیری شبکه کاهش یابد، و عمق شبکه نیز افزایش یابد؛ که این کار سبب افزایش عمق کلی نسخه دوم این شبکه به اندازه ۹ لایه نسبت به نسخه قبلی شبکه شده است. همچنین نرخ خطای Top5 برای نسخه دوم این شبکه نیز نسبت به نسخه قبلی کاهش می یابد و برابر ۴,۹۴٪ می شود، همچنین توانست اولین شبکه ای باشد که در چالش تشخیص بصری[6] از عملکرد در سطح انسان یعنی ۵,۱٪ نیز بهتر باشد[1].

[1] Gaussian Distribution
[2] Initialized
[3] Noise
[4] Outlier
[5] Regularization
[6] Visual Recognition Challenge



شکل ۲۱. Inception Module نسخه دوم را نشان می دهد.

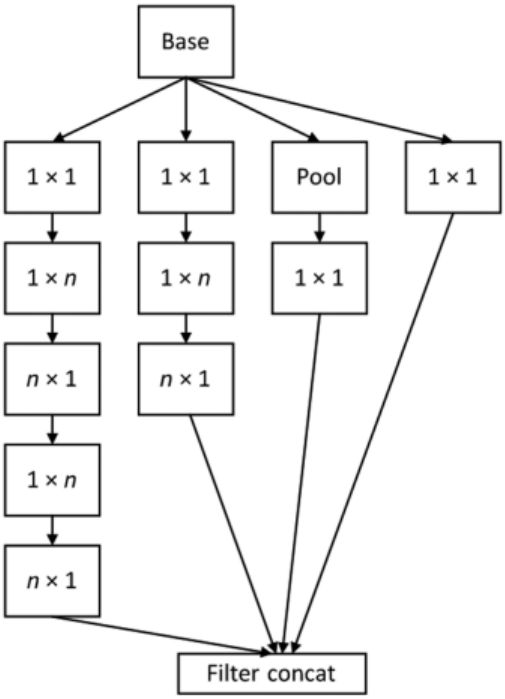
۴ - ۵ نسخه سوم شبکه Inception

شکل ۲۲ نسخه سوم Inception Module را نشان می دهد؛ تغییراتی که در این نسخه از شبکه Inception نسبت به نسخه قبلی انجام شده است عبارت است از:

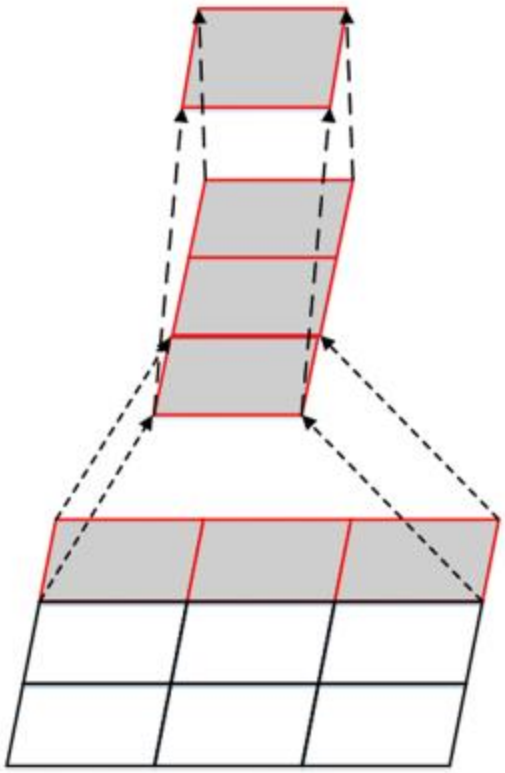
۱. Spatial factorization into asymmetric convolutions: با استفاده از این ویژگی که در شکل ۲۳ نیز نمایش داده شده است؛ نشان داده می شود که دو کانولوشن پشت سر هم به ترتیب با کرنل به ابعاد ۱ × ۳ و ۳ × ۱ دارای receptive field های یکسانی با یک کانولوشن با کرنل به ابعاد ۳ × ۳ می باشند، بنابراین کانولوشن های ۳ × ۳ در نسخه دوم Inception Module با دو کانولوشن متوالی با ابعاد ۱ × ۳ و ۳ × ۱ جایگزین می شود. این کار باعث افزایش عمق شبکه، کاهش پارامتر های قابل یادگیری و هزینه های محاسباتی می شود؛ اگر تعداد کرنل های ورودی و خروجی برابر باشد، با جایگزینی دو لایه عنوان شده پاسخ نسبت به یک لایه کانولوشن با کرنل با ابعاد ۳ × ۳ با $\frac{9-(3+3)}{9} = 33\%$ تعداد پارامتر کمتر و در نتیجه با هزینه کمتر محاسبه می شود. به طور تئوری، هر کانولوشن با کرنل با ابعاد n × n را می توان با استفاده از دو کانولوشن متوالی با کرنل به ابعاد n × 1 و 1 × n جایگزین کرد، که هر چه مقدار n افزایش یابد صرفه جویی در کاهش هزینه های محاسباتی به صورت چشم گیری افزایش پیدا می کند. (هزینه های محاسباتی به صورت چشم گیری کاهش پیدا می کند)[1].

۲. ابعاد تصویر ورودی از ۲۲۴ × ۲۲۴ به ۲۹۹ × ۲۹۹ افزایش یافته است.

۳. همچنین نرخ خطای Top5 برای نسخه سوم این شبکه نیز نسبت به نسخه های قبلی کاهش می یابد و برابر ۳,۵۸٪ می شود.



شکل ۲۲. Inception Module نسخه سوم را نشان می دهد.



شکل ۲۳. نمودار Spatial factorization into asymmetric convolutions را نمایش می دهد.

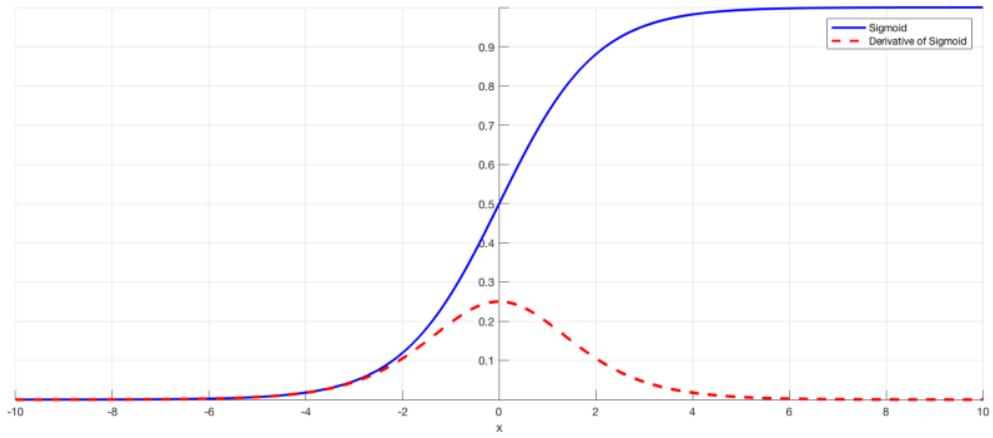
فصل پنجم

معماری شبکه Residual Network

شبکه های عصبی پیچشی عمیق منجر به پیشرفت هایی برای طبقه بندی تصاویر شده اند. شبکه های عمیق به طور طبیعی ویژگی ها و طبقه بندی کننده های سطح پایین / متوسط / سطح بالا را به صورت چند لایه انتها به انتها ادغام می کنند و "سطح" ویژگی ها را می توان با تعداد لایه های انباشته (عمق) بالا برد. شواهد اخیر نشان می دهد که عمق شبکه از اهمیت حیاتی برخوردار است. همچنین بسیاری دیگر از کارهای غیر بصری نیز از مدل های بسیار عمیق بسیار بهره مند شده اند. با توجه به اهمیت عمق این سوال پیش می آید که آیا بهبود شبکه صرفا به سادگی انباشته کردن لایه های بیشتر است ؟ در عمل دیده شده است که اگر تعداد لایه ها از حدی زیاد تر شوند یکی از دو مشکل به نام های ناپدید شدن گرادیان[1] و انفجار گرادیان[2] پیش می آید[5].

۵ – ۱ مشکل ناپدید شدن گرادیان

هر قدر لایه های بیشتری به شبکه عصبی اضافه می شوند. مقدار گرادیان تابع هزینه به صفر نزدیک می شود. و این رخداد باعث می شود که آموزش شبکه بسیار سخت شود. علت این اتفاق این است که برخی توابع فعال سازی مانند تابع سیگموید[3] بازه بزرگی از ورودی را به بازه کوچکی بین ۰ و ۱ تبدیل می کنند. در نتیجه یک تغییر بزرگ در ورودی باعث تغییر کوچکی در خروجی می شود. در نتیجه مقدار مشتق بسیار کوچک می شود[6].



شکل ۲۴. تابع سیگموید و مشتق آن

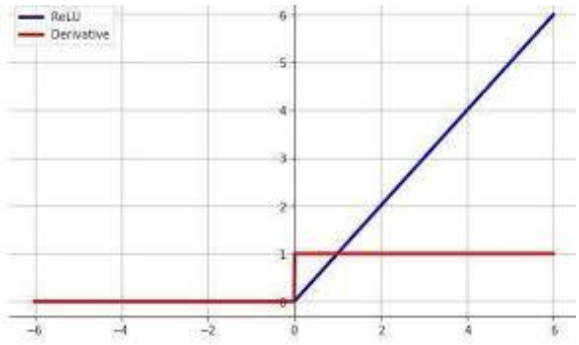
[1] Vanishing gradient
[2] Exploding gradient
[3] sigmoid

به عنوان مثال در شکل ۲۴ تابع سیگموید و مشتق آن دیده می شوند. همان طور که مشخص است زمانی که ورودی تابع سیگموید بزرگ و یا کوچک می شود، مقدار تابع مشتق به صفر نزدیک می شود[6].

برای شبکه های کم عمق این اتفاق مشکل بزرگی نیست. اما زمانی که عمق شبکه افزایش یابد باعث می شود مقدار گرادیان آنقدر کوچک شود که نتواند تغییری در وزن های شبکه ایجاد کند. زمانی که یک شبکه عمیق به عمق n با تابع فعال سازی ای مانند تابع سیگموید داشته باشیم، n مقدار کوچک در یکدیگر ضرب شده و باعث می شوند مقدار گرادیان به طور نمایی کاهش یابد. در نتیجه به علت بسیار کوچک بودن مقدار نهایی گرادیانی که به لایه های اولیه برمیگردد، این لایه ها آموزش درستی نخواهند داشت. برای حل این مشکل راه حل های زیر ارائه شده اند[6].

۵ – ۱ – ۱ استفاده از توابع فعال سازی مناسب

ساده ترین راه برای حل این مشکل استفاده از توابع فعال سازی دیگر مانند تابع ReLU می باشد[6].

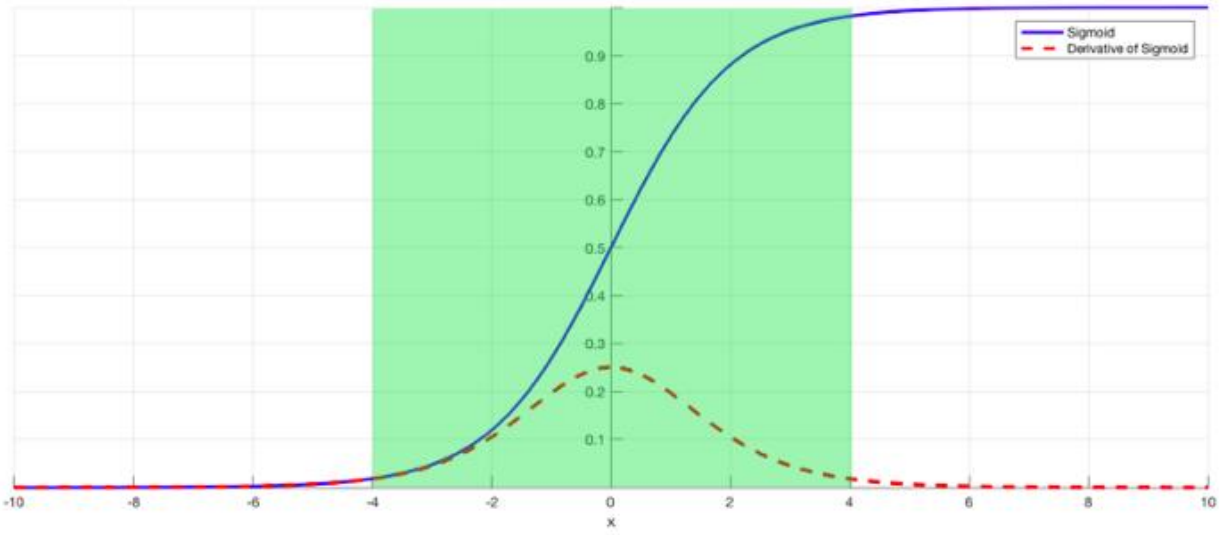


شکل ۲۵. تابع ReLU و مشتق آن

همان طور که در شکل ۲۵ دیده می شود مشتق تابع ReLU در تمامی نقاط مثبت برابر با ۱ می باشد. در نتیجه مشکل کوچک شدن گرادیان در این حالت رخ نخواهد داد[6].

۵ – ۱ – ۲ استفاده از روش نرمال سازی دسته ای[1]

همان طور که پیش از این بیان شد علت اصلی مشکل نگاشت بخش بزرگی از ورودی به بخش کوچکی از خروجی می باشد. کاری که نرمال سازی دسته ای انجام می دهد این است که ورودی تابع سیگموید را همواره طوری تنظیم می کند تا به دو انتهای بازه ورودی نزدیک نشود. بنابراین دیگر مشتق نزدیک صفر نخواهد شد و مشکل قبلی را نخواهیم داشت[6].



شکل ۲۶. تابع سیگموید و ناحیه مناسب ورودی آن

یعنی به صورت کلی مقدار ورودی را در بازه سبز در شکل ۲۶ قرار می دهد.

۵ – ۱ – ۳ استفاده از شبکه های باقیمانده [2]

در این شبکه ها مقدار ورودی هر بلاک مستقیما به مقدار خروجی(باقیمانده) اضافه می شود. و تابع فعال سازی سپس بر روی این جمع انجام می شود در نتیجه مشکل قبلی به وجود نخواهد آمد[6].

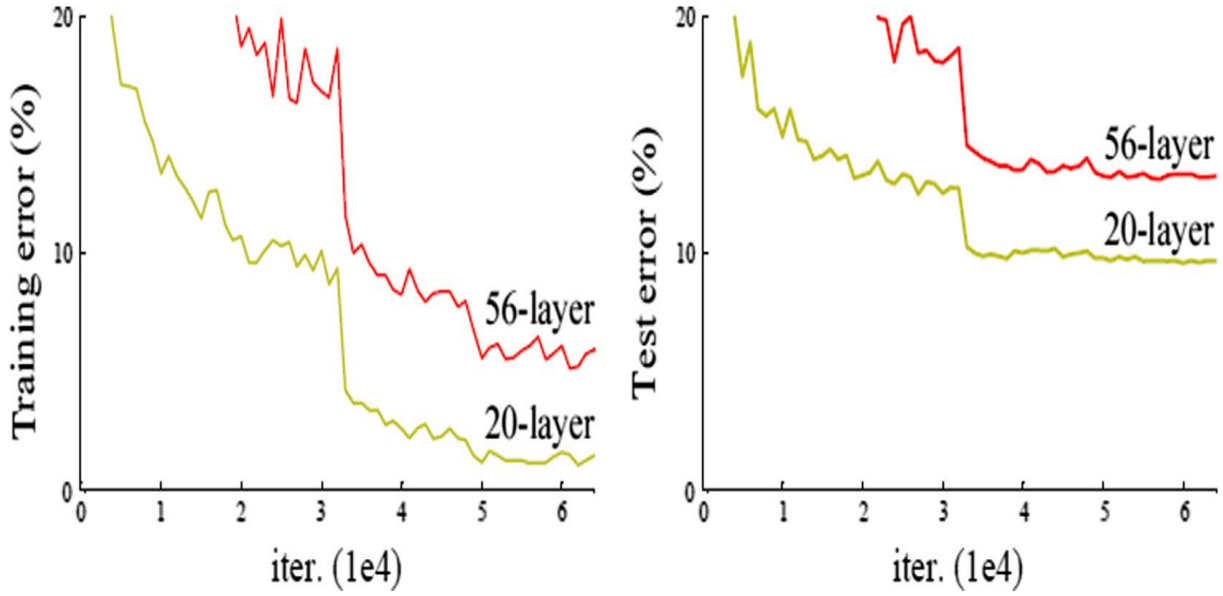
[1] batch normalization
[2] Residual networks

۵ – ۲ مشکل انفجار گرادیان

این مشکل نیز به صورت کلی مانند مشکل ناپدید شدن گرادیان می باشد با این تفاوت که در این حالت مقادیر گرادیان به نسبت مقادیر بزرگی هستند و با ضرب شدن تعداد زیادی مقادیر بزرگ، مقدار گرادیانی که به لایه های ابتدایی بر می گردد بسیار بزرگ خواهد شد. در نتیجه مقدار وزن های لایه های اولیه تغییرات بسیار بزرگی خواهند داشت که باعث می شود شبکه بی ثبات شده و آموزش آن بسیار سخت شود[7].

۵ – ۳ مشکل تنزل

پس از حل شدن مشکل ناپدید شدن گرادیان در شبکه های عمیق مشکل تنزل[1] به وجود آمد. این مشکل به این صورت می باشد که با افزایش عمق شبکه، دقت انباشته شده و افزایش نمی یابد. همچنین علاوه بر اینکه دقت افزایش نمی یابد به سرعت شروع به کاهش نیز می کند. این مشکل مشخصا مشکل بیش برازش نیست چون در بیش برازش دقت باید بر روی داده های تمرین بالا باشد و بر روی داده های تست پایین اما در اینجا دقت بر روی داده های تمرین نیز پایین می آید[5].



شکل ۲۷. درصد خطای شبکه plain ۲۰ و ۵۶ لایه بر روی CIFAR-10

[1] degradation

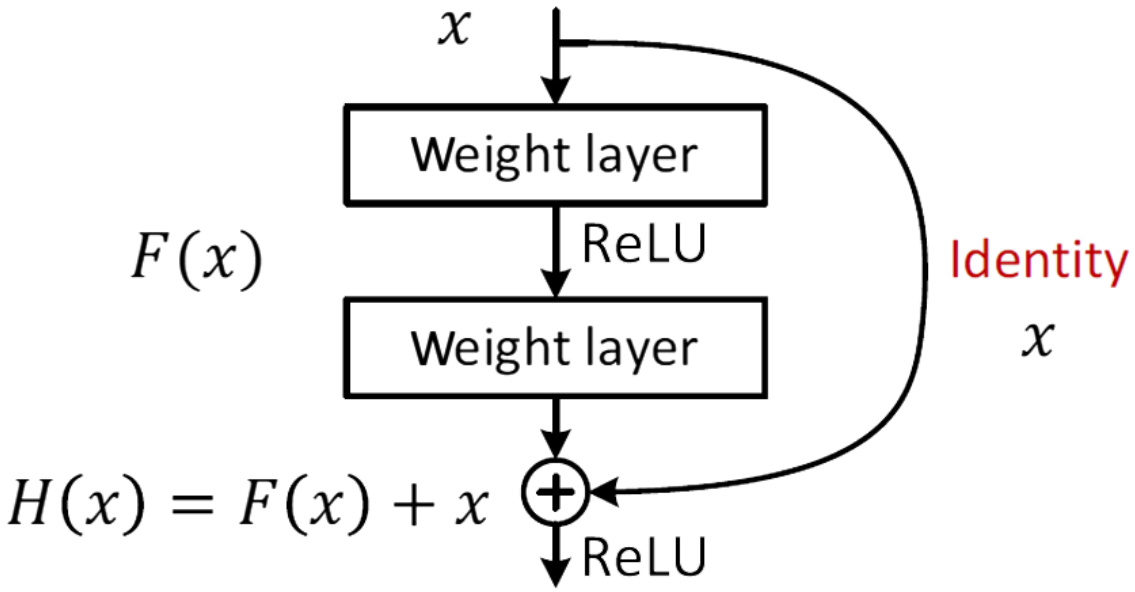
این مشکل در شکل ۲۷ به خوبی قابل مشاهده است. همان طور که دیده می شود زمانی که تعداد لایه ها از ۲۰ به ۵۶ تا رسیده اند، نه تنها دقت بر روی داده های تست پایین تر آمده است بلکه دقت بر روی داده های تمرین نیز کاهش یافته است[5].

ResNet ۴ – ۵

مشکل تنزل نشان می دهد که تمامی سیستم ها به راحتی قابل بهینه سازی و آموزش نیستند. اگر یک شبکه کم عمق و یک شبکه عمیق را در نظر بگیریم. منطقا شبکه عمیق باید بتواند حداقل عملکرد یکسانی با شبکه کم عمق بر روی داده های تمرین داشته باشد. دلیل این امر این است که اگر شبکه عمیق را این طور در نظر بگیریم که لایه های ابتدایی اش دقیقا معادل لایه های شبکه کم عمق باشند و لایه های اضافه اش همگی تابع همانی[1] باشند، در نتیجه این دو شبکه یکسان خواهند بود و عملکرد یکسان نیز خواهند داشت. اما آزمایش ها نشان می دهند که سیستم های عمیق قادر به یافتن این تابع همانی نیستند و در نتیجه عملکرد بدتری دارند[5].

شبکه های عمیق باقیمانده[2] برای حل مشکل تنزل پیشنهاد شده اند. در این روش به جای آنکه هر لایه سعی در آموختن توابع کاملا جدید داشته باشد، هر لایه صرفا باید مقدار باقیمانده را بیاموزد و مقدار اولیه به طور مستقیم به آن داده می شود[5].

[1] identity
[2] Deep residual learning network



شکل ۲۸. ساختار یک بلاک از شبکه ResNet

در شکل ۲۸ یک بلاک از این نوع شبکه ها دیده می شود. کل شبکه از به هم پیوستن تعدادی از این بلاک ها ساخته می شود. در این شبکه در حقیقت به دنبال پیدا کردن مقدار تابع باقی مانده یا همان F(x) در هر بلاک هستیم. سپس در انتهای هر بلاک مقدار ورودی (x) را مستقیما با مقدار باقی مانده جمع می کنیم. تا تابع نهایی که همان H(x) است بدست آید. بهینه کردن و آموزش این شبکه از شبکه عادی راحت تر است و مشکل تنزل نیز به وجود نمی آید. به طور دقیق اگر در بلاکی تابع همانی بهینه باشد برای شبکه راحت تر است که مقدار باقی مانده را به صفر نزدیک کند به نسبت اینکه در شبکه های اصلی بخواهد تابع همانی را بدست آورد[5].

مقداری که از ابتدا بلاک به انتهای آن اضافه می شود که به آن اتصال میانبر[1] نیز می گویند، می توانند دو یا چند لایه را رد کند. در ResNet دقیقا دو لایه رد می شود. همچنین مقداری که به خروجی اضافه می شود نیز دقیقا معادل ورودی است که این نیز می تواند متفاوت باشد. عمل اضافه کردن ورودی به خروجی هیچ پارامتر اضافه ای به سیستم اضافه نمی کند و پیچیدگی محاسباتی را نیز افزایش نمی دهد[5].

شبکه ResNet به راحتی قابل بهینه سازی است. در مقابل آن شبکه plain قرار دارد که صرفا لایه ها را پشت سر یکدیگر قرار داده است. شبکه plain عمیق دقت کمتری بر روی داده های تمرین به نسبت حالت کم عمق

[1] shortcut connection

دارد. در شبکه ResNet دقت می تواند به راحتی و با افزایش تعداد لایه ها بهبود یابد. در شبکه ResNet تعداد لایه ها می تواند به صد و یا حتی به هزار برسد[5].

۵ – ۴ – ۱ یادگیری باقیمانده[1]

اگر شبکه پیچشی بتواند به طور متناوب به مقدار تابع H(x) نزدیک شود، به طور مشابه این شبکه می تواند یاد بگیرد که به طور متناوب به تابع F(x) یا همان تابع باقیمانده نزدیک شود. در این حالت زمانی که تابع همانی بهینه باشد برای دستیابی به آن تنها کافیست مقدار F(x) به صفر نزدیک بشود. در حالت واقعی بسیار بعید است که تابع همانی بهینه باشد، با این وجود اضافه کردن این مقدار اولیه باعث شده است که بتوانیم بهینه سازی راحت تری داشته باشیم[5].

۵ – ۴ – ۲ نگاشت تابع همانی با استفاده از میانبر ها

هر بلاک را به صورت زیر تعریف می کنیم

$$y = F(x, \{Wi\}) + x$$

در این فرمول x و y به ترتیب ورودی و خروجی بلاک می باشند و تابع $F(x, \{Wi\})$ همان تابع باقیمانده که باید آموخته شود، می باشد. عمل F + x با استفاده از اتصال میانبر و جمع عضو به عضو انجام می گردد[1].
همچنین تابع فعالسازی دوم بعد از انجام شدن این جمع، اعمال می شود. برای انجام این جمع باید ابعاد تابع F و ورودی x یکسان باشد. زمانی که این طور نباشد مجبوریم که x را به فضای با ابعاد مناسب نگاشت کنیم در نتیجه فرمول زیر را خواهیم داشت[5].

$$y = F(x, \{Wi\}) + Ws.x$$

۵ – ۴ – ۳ معماری شبکه

ابتدا به بررسی معماری شبکه plain و سپس به معماری شبکه residual می پردازیم.

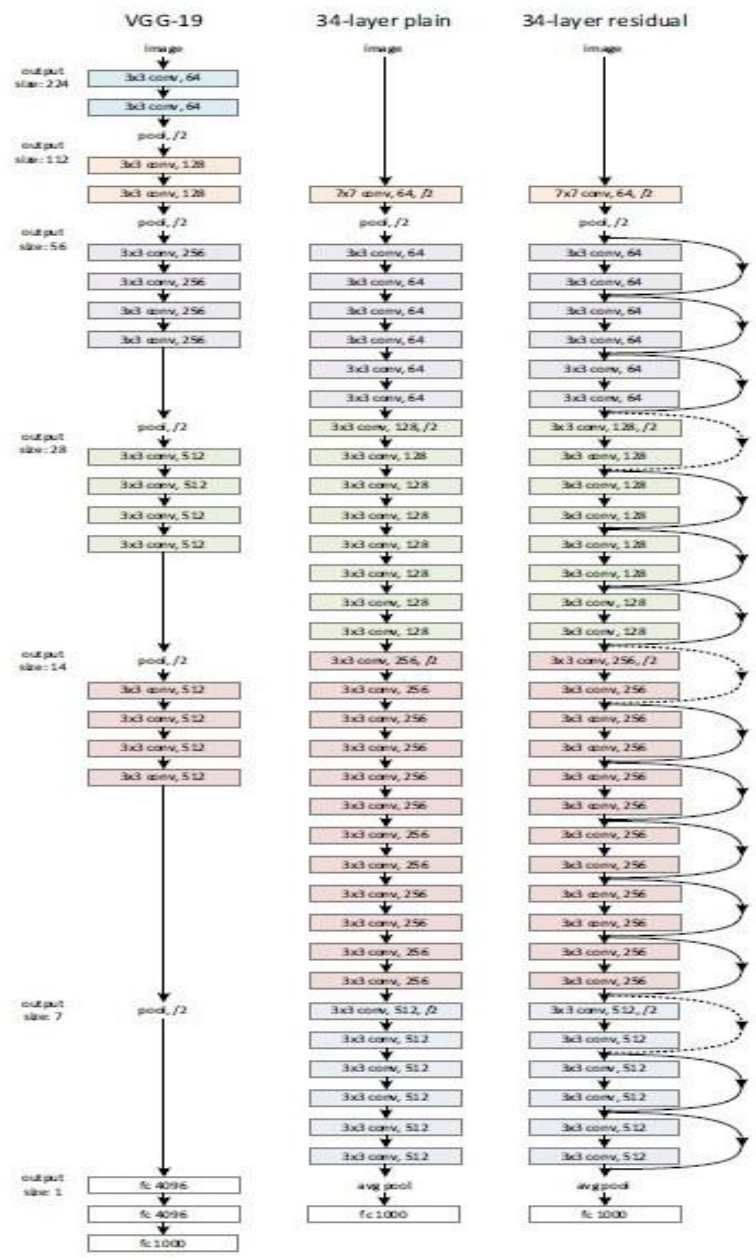
[1] Residual Learning

در شبکه plain که بسیار شبیه به VGG می باشد. تمامی فیلتر های کانولوشن از سایز ۳ در ۳ می باشند. در این شبکه دو قانون اصلی در طراحی شبکه وجود دارد. اول اینکه در صورتی که ابعاد ورودی و خروجی هر بلاک یکسان باشند، تعداد لایه های فیلتر ها برابر خواهد بود. دوم اینکه در صورتی که ابعاد خروجی نصف ابعاد ورودی باشد، تعداد فیلتر ها دو برابر خواهد شد. و با این کار پیچیدگی در تمامی لایه ها یکسان خواهد ماند. کاهش ابعاد با استفاده از لایه های کانولوشن با گام های به اندازه دو انجام می شود. شبکه با یک لایه شراکت میانگین[1] و سپس یک لایه کاملا متصل[2] هزار راهه[3] به همراه تابع فعال سازی softmax پایان می یابد[5].

شبکه residual کاملا شبیه به شبکه plain می باشد با این تفاوت که در آن اتصالات میانبر اضافه شده اند. زمانی که ورودی و خروجی دارای سایز یکسان هستند، مقدار ورودی بلاک می تواند مستقیما با مقدار خروجی بلاک جمع شود. زمانی که ابعاد ورودی و خروجی متفاوت اند دو راه داریم. راه اول این است که به جای تمامی داده هایی که باید در ابعاد جدید اضافه شوند، مقدار صفر را بگذاریم. و راه دوم این است که با استفاده از کانولوشن های ۱ در ۱ این تغییر ابعاد را انجام دهیم. در هر دو حالت زمانی که ابعاد ورودی و خروجی متفاوت هستند از کانولوشن با طول گام ۲ استفاده می کنیم[5].

معماری کلی شبکه های بیان شده در شکل ۲۹ قابل مشاهده است.

[1] Average pooling
[2] Fully-connected
[3] 1000-way



شکل ۲۹. معماری کلی شبکه VGG-19 و شبکه هایplain و ResNet با ۳۴ لایه

فصل ششم

جمع بندی و نتیجه گیری

از ابتدای ظهور شبکه های عصبی اولیه مانند AlexNet تا ظهور شبکه های عمیق تر مانند VGGNet شواهد نشان می دهند که عمق شبکه های عصبی اهمیت بالایی در کارایی آنها دارد. پس از آن طراحی مبتکرانه ماژول inception اهمیت بالای معماری شبکه های عصبی را به همه نشان داد. معماری ResNet نیز بار دیگر اهمیت بالای عمق شبکه را نشان داد که نقش حیاتی در شبکه عصبی امروزی دارد. در ادامه خلاصه ای از موارد بیان شده در این نوشته می آید.

در بخش اول ابتدا قسمت های مختلف شبکه های عصبی پیچشی مانند لایه های کاملا متصل، تابع فعال سازی، لایه های کانولوشن، کرنل یا فیلتر و لایه های شراکت بیان شده و هر یک از این مفاهیم تشریح شده و نقش آنها در شبکه های عصبی پیچشی بیان گردید. سپس پس از بیان مفاهیم اولیه، روش کاری شبکه های عصبی پیچشی برای طبقه بندی تصاویر بیان شد. در این بخش نشان داده شد که چگونه لایه های اولیه ویژگی های سطح پایین را استخراج می کنند و این ویژگی های سطح پایین در لایه های بالاتر با یکدیگر ترکیب شده و ویژگی های سطح بالاتر را مشخص می کنند. پس از آن معماری کامل شبکه عصبی پیچشی و روش عملکرد هر لایه کانولوشن بیان شده اند. و در ادامه روش عملکرد و تاثیر لایه های شراکت تشریح شده اند.

در بخش دوم سه معماری مشهور شبکه های عصبی یعنی LeNet-5، AlexNet و VGG-16/19 بررسی شده اند. معماری LeNet از پنج لایه تشکیل شده است. این لایه ها عبارت اند از دو لایه کانولوشن، دو لایه شراکت حداکثر، و یک لایه مسطح سازی. و در انتهای این شبکه با لایه های کاملا متصل کلاس بندی انجام می شود. در معماری AlexNet هشت لایه داریم و از عمل حذف تصادفی و تابع فعال سازی ReLU جهت بهبود عملکرد شبکه استفاده می کنیم. همچنین در این معماری قابلیت موازی سازی کار ها وجود دارد که باعث بهبود عملکرد آن می شود. بهبود دیگر شبکه AlexNet به نسبت LeNet استفاده از تابع Softmax در آخرین لایه کاملا متصل برای کلاس بندی است که باعث می شود دقت کلاس بندی بهتر شود. بهبود عمکرد AlexNet به نسبت LeNet کاملا قابل توجه و حدود ده درصد می باشد. پس از این دو معماری، معماری مشهور VGG را داریم که دارای دو مدل با عمق ۱۶ و ۱۹ می باشد. این شبکه اولین شبکه ای می باشد که درصد خطای آن کمتر از ده درصد است. در این شبکه تمامی فیلتر ها دارای سایز ۳ در ۳ و اندازه طول گام نیز برابر با ۱ می باشد. همچنین تمامی لایه های شراکت از نوع حداکثر شراکت بوده و با طول گام دو انجام می شوند. به صورت کلی این معماری بسیار مرتب و با قاعده های مشخص می باشد. و دقیقا پس از هر دو لایه کانولوشن، یک لایه اشتراک داریم. در انتهای شبکه نیز لایه های کاملا متصل به همراه تابع فعال سازی Softmax می باشد. همچنین در این بخش علت بهتر بودن استفاده از فیلتر های با سایز ۳ در ۳ تشریح شده است.

در بخش سوم به نسخه های اول تا سوم معماری GoogLeNet/Inception پرداخته می شود. این معماری اولین معماری است که از بلاک های مشخص کوچک تر تشکیل شده است. در هر بلاک این شبکه کانولوشن های با سایز مختلف استفاده شده و نتایج این کانولوشن ها همگی به یکدیگر الحاق شده و به عنوان خروجی بلاک در نظر گرفته می شود. همچنین کانولوشن های با سایز ۱ در ۱ برای اولین بار در این شبکه مورد استفاده قرار گرفند. از کاربرد های این کانولوشن می توان به تغییر ابعاد اشاره کرد. در این انتهای این شبکه نیز بر خلاف شبکه های قبلی از شراکت میانگین گیری استفاده می شود. درصد خطای این شبکه بسیار کم و در حد سه درصد می باشد. در نسخه دوم این شبکه از نرمال سازی وزن ها برای بهبود عملکرد شبکه استفاده شده است. همچین در این نسخه به جای فیلتر با سایز ۵ در ۵ از دو فیلتر با سایز ۳ در ۳ استفاده شده است که همان عملکرد را با کارایی بهتر دارد. در نسخه سوم نیز به جای استفاده از کانولوشن های با ابعاد n در n از دو کانولوشن با ابعاد n در 1 و 1 در n استفاده شده است. که باعث می شود همان عملکرد را با پیچیدگی کمتر داشته باشیم.

در بخش چهارم به شبکه ResNet و مشکلاتی که در اثر افزایش عمق شبکه به وجود می آید پرداخته شده است. ابتدا به بیان مشکل ناپدید شدن گرادیان پرداخته شده که برای سال ها مشکل اصلی برای افزایش عمق شبکه های عصبی بوده است. اما این مشکل با روش هایی مانند نرمال سازی دسته ای تا حد خوبی حل می شود. سپس به مشکل تنزل پرداخته شده است که پس از حل شدن مشکل قبلی به وجود آمد و این برای حل این مشکل شبکه ResNet بوجود آمده است. شبکه ResNet نیز مانند شبکه Inception از بلاک هایی تشکیل شده است. این بلاک ها همگی دارای این ویژگی هستند که مقدار ورودی بلاک را با مقدار خروجی جمع کرده و به تابع فعال سازی می دهند. در نهایت نتیجه این کار خروجی بلاک را مشخص می کند. در حقیقت در این شبکه به یادگیری باقیمانده به جای یادگیری تابع اصلی پرداخته می شود. این کار باعث بهبود بسیار زیاد شبکه می شود و در این حالت می توانیم شبکه های با ۱۰۰تا ۱۰۰۰ لایه داشته باشیم.

همان طور که مشخص است شبکه های عصبی پیچشی به سرعت رو به بهبود هستند و در معماری انتهایی یعنی inception و ResNet دقت شان فراتر از دقت انسان ها رفته است و هر دو درصد خطایی در حد ۳ درصد دارند که رقم بسیار بالاییست. انتظار می رود در سال های آینده با بهبود هایی که در این شبکه ها انجام شود و شبکه های جدید بوجود بیایند، شبکه هایی با دقت نزدیک ۱۰۰ درصد داشته باشیم که قطعا چنین دقتی در کلاس بندی عکس ها می تواند باعث پیشرفت چشمگیری در تمامی زمینه های مربوط به کلاس بندی عکس و تشخیص الگو بشود.

منابع و مراجع

[1] Wang, Wei, Yujing Yang, Xin Wang, Weizheng Wang, and Ji Li. "Development of convolutional neural network and its application in image classification: a survey." *Optical Engineering* 58, no. 4 (2019): 040901.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012): 1097-1105..

[3] Aggarwal, Charu C. "Neural networks and deep learning." Springer 10 (2018): 978-3

[4] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[5] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[6] https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[7] https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11