پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

بررسي و بكارگيري شبكه عصبي پيچشي گاست نت

نگارنده

شفیق اشرفی

استاد راهنما: دکتر ساجدی

پایان‌نامه برای دریافت درجه کارشناسی
در رشته ریاضی محض

چکیده

پیاده سازی شبکه های پیچشی عصبی بر روی سیستم های نهان به دلیل محدودیت در فضای ذخیره سازی و همچنین محدودیت در توان محاسباتی، کاری بسیار دشوار و پیچیده است. از طرف دیگر، ویژگی های استخراج شده[1] از تصاویر در مبحث بینایی ماشین و افزونگی در تعداد آنها، نقشی بسیار اساسی در خوب و موفق ظاهر شدن مدل طراحی شده در این مبحث دارد. حال هدف این مقاله این است که ماژول و یا افزونه ای بر روی مدل های عصبی پیاده کند تا با استفاده از پردازش هایی کم هزینه(تبدیلات خطی) از نظر زمان و توان محاسباتی، افزونگی این ویژگی های استخراج شده را بر اساس تعدادی ویژگی پایه ای برای مدل فراهم کند. دقت شود که ارائه این افزونه علاوه بر کاهش حجم محاسبات و پیچیدگی مدل، باعث ارتقای دقت مدل و عملکرد آن می شود. نهایتا با بررسی و تست های انجام شده بر روی این مدل، مشخص می شود که این مدل که از نظر حجم محاسبات مشابه مدل MobileNetV3 است، عملکرد و دقت بالاتری را بر روی مجموعه داده ی ImageNet ILSVRC-2012 دارد.

[1] Feature Maps

پیشگفتار

شبکه های پیچشی عصبی عمیق، بازدهی بسیار بالا و خروجی ایده آلی در حل مسائل بینایی ماشین از جمله تشخیص تصاویر، شناسایی اشیا و تقسیم بندی معنایی تصاویر داشته و دارند. اما این بازدهی به دو مورد بسیار وابسته است. مورد اول آن که این شبکه ها به حجم بسیار بالایی از اطلاعات برای یادگیری نیازمندند. این حجم از اطلاعات با توجه به پیدایش اینترنت و متصل شدن تعداد بسیار زیادی از انسان ها به هم، ایجاد شده است و در فضای اینترنت قرار گرفته است. برای مثال، پایگاه داده ImageNet بیش از ۱۴ میلیون عکس درون خود دارد که هر کدام از آنها به صورت دستی و با کمک تعداد بسیار زیادی از افراد لیبیل گذاری شده اند.
مورد دوم آن است که این شبکه ها به تعداد بسیار بالایی از محاسبات اعشاری(FLOPs[2]) نیاز دارند تا بتوانند خروجی مطلوب را داشته باشند. برای مثال، شبکه ی عصبی ResNet-50 ۶.۲۵ میلیون پارامتر قابل یادگیری دارد و همچنین به ۱.۴ میلیارد محاسبه ی اعشاری نیاز دارد تا بتواند یک قطعه عکس 224×224 را پردازش کند. با توجه به پیشرفت تکنولوژی در بحث کارت های گرافیک و افزایش بسیار بالای قدرت پردازشی سخت افزار ها، استفاده از شبکه های عصبی علارغم سنگین بودن محاسبات، در دنیا رواج یافته است. اما با وجود همه ی پیشرفت ها، باز هم حجم محاسبات برای کامپیوتر های نهان مانند کامپیوتر های داخل ماشین های خودران و یا گوشی های هوشمند بسیار بالاست. در این نقطه است که اهمیت پروژه هایی مشابه با این پروژه مشخص می شود. چرا که ما به کارایی بالای شبکه های عصبی در حل مسائلی که به یادگیری ماشین نیازمندند، نیاز داریم و از طرفی در بعضی مواقع، حجم بالای محاسبات آنها برای ما مقدور نیست. در نتیجه به الگوریتم های کارآمدتری دراین حوزه نیازمندیم.

در طول سال ها، تحقیقات بسیاری پیرامون فشرده کردن شبکه های عصبی به منظور کاهش حجم محاسبات آنها صورت گرفته است و روش هایی مانند هرس کردن شبکه[3] ، کمی کردن بیت

[2]Floating Point Operations
[3]Network Pruning

ها[۴] و چکیده سازی دانش[۵] مورد استفاده قرار گرفته است. دقت شود که از بین روش های بالا، آن روشی بهتر است که حجم مدل را به میزان کافی کاهش دهد اما میزان دقت عملکرد مدل را تغییر چندانی ندهد. چرا که الویت اصلی، میزان عملکرد مدل است.

با وجود تلاش بسیار در ساخت مدل هایی کارآمد و کم حجم مانند MobileNet که در آن از روش پیچش عمقی[۶] و همچنین مدل ShuffleNet که در آن از محاسبات مختلط[۷] استفاده شده است تا شبکه ی عصبی کارآمدی با فیلتر پیچشی[۸] کوچکتری بسازند، لایه پیچشی[۹] 1×1 همچنان از نظر تعداد پارامتر و حجم محاسبات سنگین هستند و نیاز به فشرده سازی بیشتری است.

[4]Low-Bit Quantization
[5]Knowledge Distillation
[6]Depthwise Convolution
[7]Shuffle Operation
[8]Convolution Filter
[9]Convolution Layer

فهرست مطالب

[10] GhostNet
[11] Ghost module
[12] Ghost Bottleneck

فصل ۱

مفاهیم مقدماتی

در ادامه این بخش به تعریف بعضی مفاهیم مطرح شده در بخش پیشگفتار و تعاریف مورد نیاز برای مقاله می پردازیم.

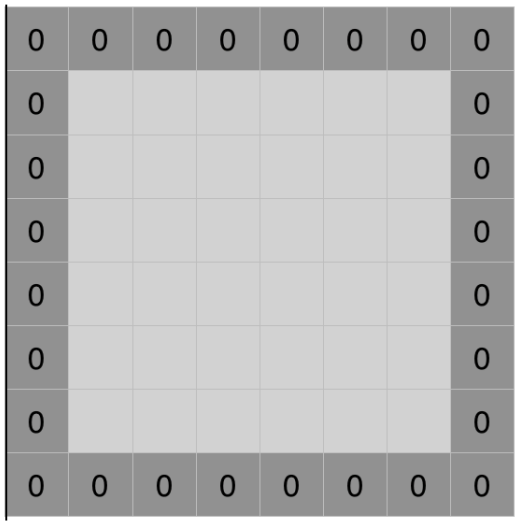
تعریف ۱.۱. *شبکه ی عصبی پیچشی*[1]

شبکه ی عصبی پیچشی یک الگوریتم با تعداد زیادی پارامتر قابل یادگیری است که به عنوان ورودی، عکسی را دریافت می کند و بعد از تعدادی محاسبات اعشاری بر داده ی ورودی، خروجی را بدست می دهد که با استفاده از آن می توان بین عکس ها تمایز قائل شد. در واقع هدف این الگوریتم دسته بندی تصاویر و فهمیدن آنها همانند انسان است و این کار را بسیار خوب و دقیق انجام می دهد. پارامتر های یادگیری مدل که در این نوع شبکه ها فیلتر ها هستند، با مقدار اولیه شروع به کار می کنند و مدل با رویت و بررسی داده های ورودی، مقدار این پارامتر ها را در راستای خروجی بهتر تغییر می دهد. به این مرحله از تغییرات پارامترها، مرحله ی یادگیری می گویند چرا که مدل را قادر به یادگیری مفاهیم داخل ورودی خود می کند. بعد از مرحله ی یادگیری، نوبت به مرحله ی تست می رسد که در آن سعی می شود با تنظیم کردن تعدادی ابرپارامتر[2] ، عملکرد مدل را بهبود بخشند.
همان طور که اشاره شد، فیلترها با مقدار اولیه ای شروع به کار می کنند و در طول فرآیند یادگیری، مدل فیلتر های خود را به گونه ای تغییر می دهد تا ویژگی هایی که به تعداد دفعات بالا در تصاویر تکرار می شوند را از تصاویر استخراج کند. این ها همان ویژگی های استخراج شده هستند.

[1] Convolutional Neural Network
[2] Hyperparameter

تعریف ۲.۱. *افزایش ابعاد تصویر*[3]

فرض کنید تصویر ورودی شبکه ی عصبی، یک تصویر سیاه سفید با ابعاد $(n \times n)$ باشد و ابعاد فیلتری که بر آن اعمال می شود، $(f \times f)$ باشد. در این صورت ابعاد تصویر بعد از اعمال فیلتر بر روی آن در این لایه ی پیچش، برابر $(n - f + 1) \times (n - f + 1)$ خواهد بود. پس در هر مرحله با انجام عملیات پیچش بر روی تصاویر، از ابعاد آن کاسته می شود. این موضوع یک کران بالایی بر روی تعداد لایه های پیچش ایجاد می کند که همین نکته در میزان دقت و عملکرد مدل تاثیر خواهد داشت. علاوه بر نکته ی مطرح شده، پیکسل های گوشه و لبه ی تصویر در فرآیند یادگیری مدل خیلی کمتر استفاده می شوند و مورد بررسی قرار می گیرند.
با توجه به نکات مطرح شده، نیاز است که روشی استفاده شود تا مشکلات مطرح شده را برطرف سازد. این روش، همان افزایش ابعاد تصویر خواهد بود. افزایش ابعاد تصویر عبارت است از اضافه کردن اعدادی یکسان به گوشه های تصویر.



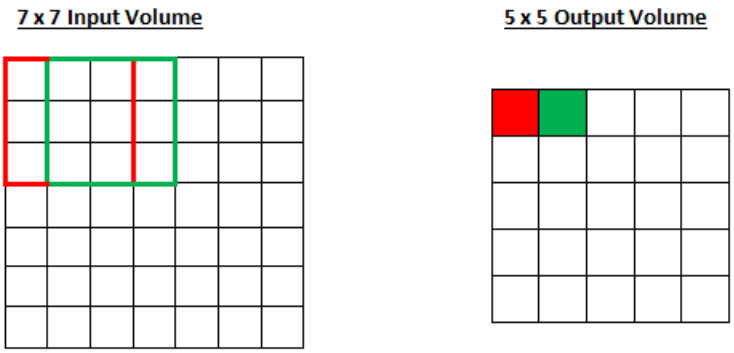
شکل ۱.۱: padding zero

در شکل بالا، عدد اضافه شده به اطراف تصویر، ۰ است. به همین دلیل به این عملیات Zero-padding می گویند.

[3]padding

تعریف ۳.۱. *طول گام*[۴]

طول گام یک پارامتر از شبکه ی عصبی است که مربوط به نحوه ی حرکت فیلتر بر روی تصاویر و ویدیوها است. برای مثال، اگر که طول گام برابر با ۱ باشد، فیلتر هر بار به اندازه یک پیکسل یا یک واحد بر روی تصویر حرکت می کند. دقت شود که طول گام عددی طبیعی است چرا که فیلتر ابعادی طبیعی دارد و بر روی یک عکس با ابعاد طبیعی اعمال می شود.



شکل ۲.۱: اعمال یک فیلتر 3×3 بر روی یک تصویر یا ویژگی های استخراج شده 7×7 با طول گام ۱ که یک خروجی با ابعاد 5×5 را ایجاد می کند.

تعریف ۴.۱. *نرمال سازی دسته ای*[۵]

نرمال سازی دسته ای یک روش الگوریتمیک است که فرآیند یادگیری شبکه های عصبی را سریع تر و دقیق تر می کند. در این روش، خروجی لایه های پیچشی با استفاده از میانگین و واریانس نرمال می شوند. این عملیات می تواند قبل و یا بعد از تاثیر تابع غیر خطی بر روی خروجی ها انجام شود.

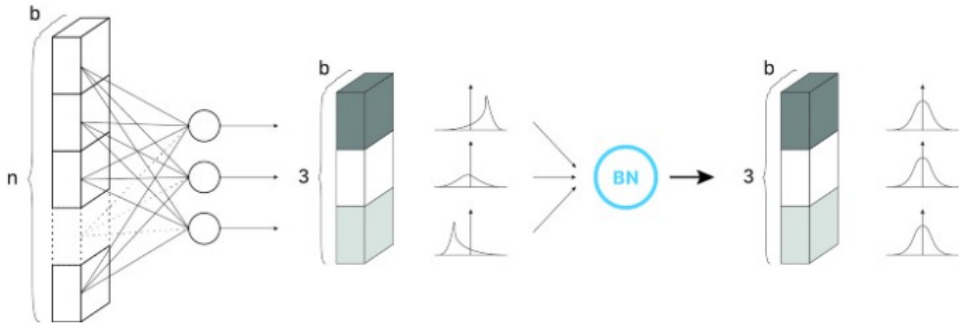
[4] stride
[5] batch normalization

$$(1)\quad \mu = \frac{1}{n}\sum_i Z^{(i)} \qquad (2)\quad \sigma = \frac{1}{n}\sum_i (Z^{(i)} - \mu)$$

$$(3)\quad Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \epsilon}} \qquad (4)\quad \check{Z} = \gamma * Z^{(i)}_{norm} + \beta$$

شکل ۳.۱: عملیات نرمال سازی بر روی خروجی لایه پیچش (Z) که در آنها μ میانگین و σ^2 واریانس است.

در معادلات بالا، ϵ برای جلوگیری از تقسیم بر صفر استفاده می شود چرا که ممکن است واریانس به صفر نزدیک باشد. در معادله چهارم نیز دو پارامتر λ و β برای تنظیم انحراف از معیار و پارامتر جانبدار[۶] خروجی لایه خواهد بود. دقت شود که این دو پارامتر توسط مدل قابل یادگیری است.



Batch Normalization first step. Example of a 3-neurons hidden layer, with a batch of size b. Each neuron follows a standard normal distribution. | Credit : author - Design : Lou HD

شکل ۴.۱: اعمال نرمال سازی دسته ای بر روی خروجی لایه های پیچش

در شکل بالا، اثر نرمال سازی دسته ای بر خروجی لایه ها به خوبی نمایش داده شده است.

[6] bias

تعریف ۵.۱. *پیچش عمقی*

با استفاده از فیلتر ها در شبکه های عصبی پیچشی، مدل باید تعداد بسیار زیادی از پارامتر ها را تغییر دهد و یاد بگیرد. این تعداد بسیار زیاد ممکن است باعث شود مدل بیش از حد به تصاویر ورودی وابسته شود و به اصطلاح Over-Fitting رخ بدهد که تعمیم مدل به ورودی های دیگر را با مشکل مواجه می کند و در نتیجه دقت مدل را پایین می آورد. به همین دلیل گاهی لازم است از نوع دیگر پیچش مثل پیچش عمقی استفاده کنیم. این روش را با رسم یک شکل توضیح می دهیم. فرآیند پیچش معمولی همانند شکل زیر است:

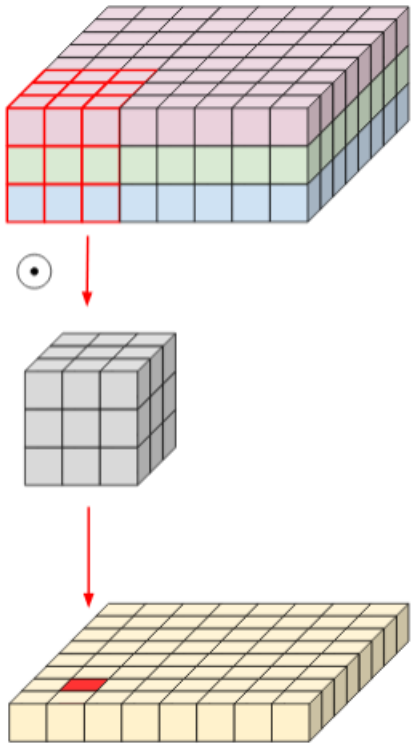


Fig 1. Normal convolution

شکل ۵.۱: عملیات معمولی پیچش

اما در این روش، ما برای هر کانال داده ی ورودی به لایه، از فیلتر متفاوتی استفاده می کنیم و

در نهایت، خروجی های هر کانال را با یکدیگر ترکیب کرده تا یک خروجی چند کانال تولید شود.
شکل زیر فرآیند را به خوبی نمایش می دهد.

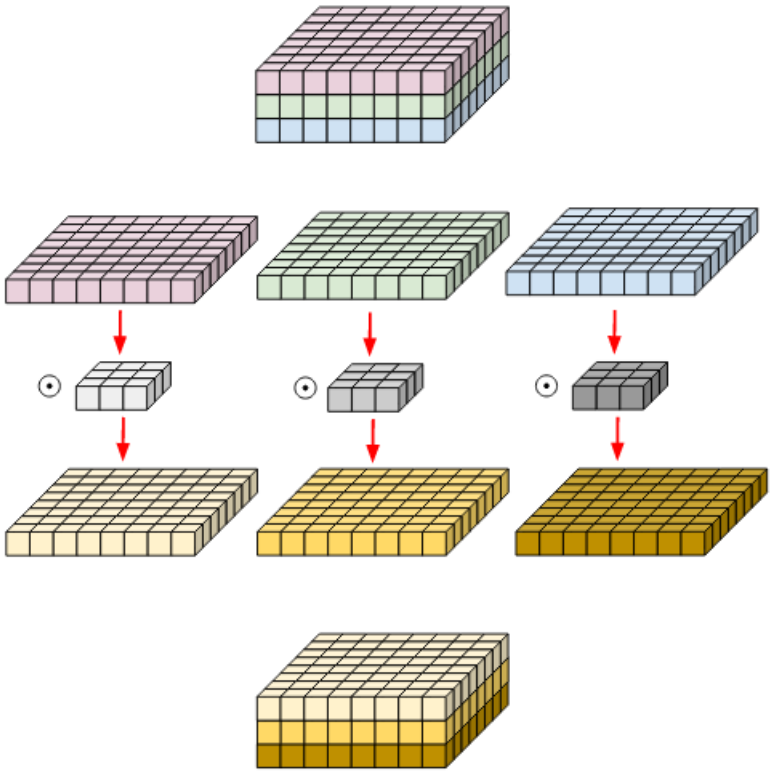


Fig 2. Depth-wise convolution. Filters and image have been broken into three different channels and then convolved separately and stacked thereafter

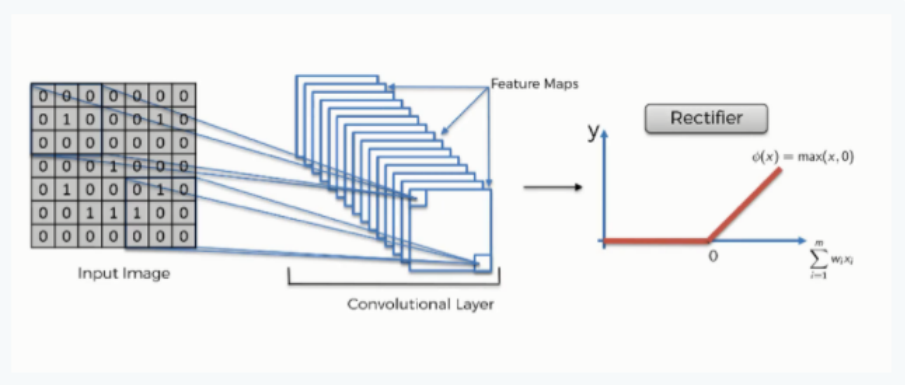
شکل ۱.۶: نمایشی از عملیات پیچش عمقی

تعریف ۱.۶. *محاسبات مختلط*

Shuffle Operation یا Channel Shuffle فرآیندی است که کمک می کند تا اطلاعات بین کانال های مختلف جابجا شود و اولین بار در مدل ShuffleNet معرفی شد.

تعریف ۱.۷. *تابع غیر خطی(Rectified Linear Unit)RELU*

این تابع بر روی خروجی لایه های پیچشی عمل می کند و هدف آن اضافه کردن خاصیت غیر خطی به مدل است تا مدل بتواند علاوه بر ویژگی های خطی، ویژگی های پیچیده تر غیر خطی را نیز یاد بگیرد.



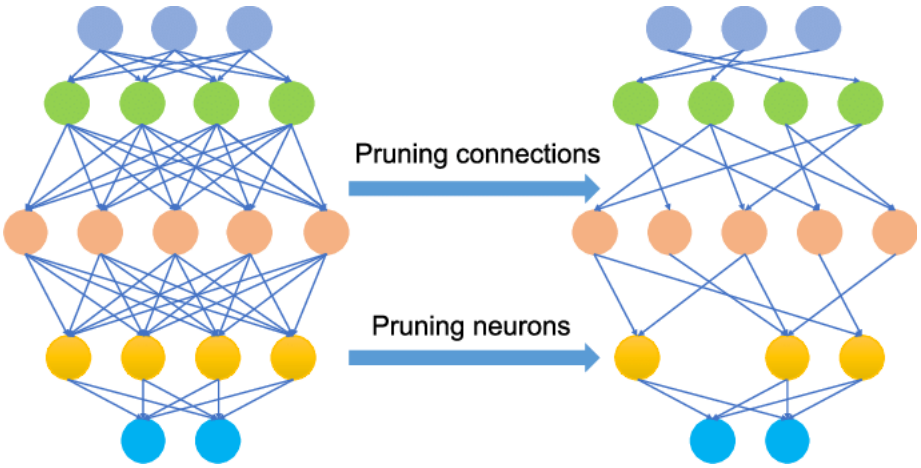
شکل ۷.۱: نمایش گراف تابع غیر خطی RELU

در چکیده ی مقاله، روش های مختلفی برای فشرده کردن و کاهش حجم مدل ارائه شد که به هر کدام به صورت مختصر می پردازیم.

تعریف ۸.۱. *هرس کردن شبکه*

در مبحث یادگیری ماشین، هرس کردن شبکه عبارت است از حذف نورون ها و یا وزن های غیر ضروی. غیر ضروری بودن به این معناست که بودن یا نبودن آنها، تاثیری در میزان عملکرد و خروجی مدل ندارد.
دو روش برای هرس کردن شبکه وجود دارد. روش اول آن است که وزن های غیر ضروری را از مدل حذف کنیم. برای این کار کافیست پارامتر های مد نظر مدل را برابر با صفر قرار داده و با این کار شبکه را پراکنده[۷] می کنیم.
این روش، تعداد پارامتر های مدل را کاهش می دهد بدون آن که تاثیری بر ساختار کلی مدل داشته باشد. روش دوم برای هرس کردن این است که نورون ها و یا گره های شبکه را به طور مستقیم حذف کنیم. این حرکت، ساختار مدل را کوچکتر می کند و هدف آن این است که دقت و عملکرد مدل اولیه را حفظ کند.

[7] Sparse



شکل ۸.۱: دو روش هرس کردن شبکه

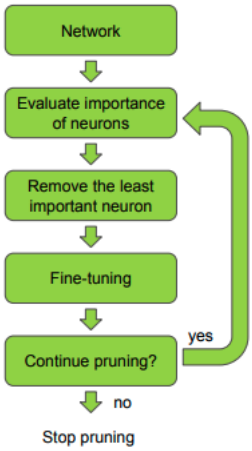
از بین دو روش مطرح شده در بالا، روش اول محبوب تر است چرا که هم پیاده کردن آن از نظر فنی راحت تر است و هم تاثیر نامطلوب کمتری بر دقت مدل می گذارد. اما از طرف دیگر، این روش نیازمند محاسبات پراکنده[۸] است تا نتیجه ی مطلوبی داشته باشد و به همین دلیل ممکن است به سخت افزار های خاصی نیاز باشد.
پیاده سازی روش دوم به ما اجازه ی انجام محاسبات فشرده[۹] را می دهد که از محاسبات پراکنده، بهینه تر است. چرا که این گونه محاسبات بر روی سخت افزار های امروزی راحت تر پیاده می شود. تنها مانع این روش، این است که با حذف نود یا گره ها از شبکه، ساختار کلی شبکه به هم می ریزد و این نکته، تاثیر نامطلوبی بر دقت مدل می گذارد.

مهم ترین نکته در مسئله هرس کردن، این است که مشخص کنیم که باید چه چیزی را در مدل هرس کنیم. اگر بخواهیم وزن ها و یا نود ها را از شبکه حذف کنیم، باید کم ارزش ترین و کم مصرف ترین آنها را حذف کنیم تا کمترین تاثیر بر عملکرد مدل وارد شود. روش های مکاشفه ای مختلفی برای تشخیص این نود ها وجود دارد. یکی از آسان ترین آنها، هرس کردن بر اساس اندازه ی وزن است. اندازه ی وزن را می توانیم با روش های مختلف ریاضی از جمله نرم اقلیدسی[۱۰] و یا نرم L1 تعیین کنیم.
این روش بدین گونه عمل می کند که وزن هایی که اندازه آنها بسیار نزدیک به صفر است را پیدا می کنیم و آنها را برابر صفر قرار می دهیم. برای این منظور، می توانیم یک عددی را به عنوان معیار قرار دهیم و تمامی وزن های مدل با اندازه کمتر از آن عدد را برابر صفر قرار دهیم.

[8] Sparse Computation
[9] Dense Computation
[10] L2 norm

روش دیگر آن است که نورون هایی را با وجود ورودی بسیار بالا، خروجی آنها باز هم نزدیک به صفر است را می توانیم به عنوان کاندید هرس کردن انتخاب کنیم. چرا که گویا آنها در حل مسئله تاثیر به سزایی ندارند.

حال که مشخص کردیم چه قسمت هایی از مدل باید هرس شوند، وقت آن است که بفهمیم کی باید عملیات هرس کردن را انجام دهیم. اگر به دنبال هرس کردن پارامترهای مدل رفتیم، بهترین زمان برای انجام هرس کردن، بعد از مرحله ی یادگیری مدل است. اما باید دقت شود که این کار به دقت مدل لطمه وارد می کند. در نتیجه مجبور هستیم بعد از هرس کردن، مجدداً کار تنظیم مدل[11] را انجام دهیم که با انجام مجدد فرآیند یادگیری، این کار صورت می گیرد.



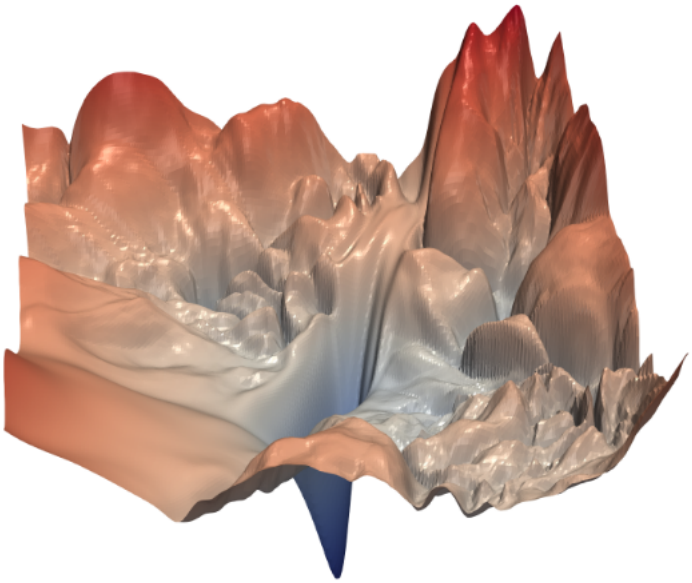
شکل ۹.۱: فرآیند تدریجی و دوره ای هرس کردن شبکه

تعریف ۹.۱. *کمی کردن بیت ها*

همان طور که در ضمن موارد بالا مطرح شد، تمامی محاسبات درون شبکه های عصبی به صورت اعشاری است (FLOPs Or Floating Point Operations) . ایده ی این روش آن است که مقادیر پارامتر ها و همچنین خروجی تمامی لایه های شبکه را از حالت اعشاری به حالت عددی صحیح تبدیل کنیم. حداکثر فشرده سازی که می توانیم انجام دهیم این است که اعداد مدل را به صورتی در بیاوریم تا در حداکثر یک بیت فضا قرار بگیرند.
اما نکته ای که در این روش باید بسیار به آن توجه کرد این است که با این کار، ممکن است دقت مدل به شدت دسخوش تغییر شود. عکس نمایش داده شده در زیر، از مقاله ای است که در آن تابع

[11] Fine-Tuning

هدف شبکه ی عصبی ResNet-56 بدون لایه پرش اتصالات[12] رسم شده است. با توجه به این شکل متوجه می شویم که ممکن است با تغییری بسیار کوچک در محاسبات یک شبکه ی عصبی، تغییرات بسیار بزرگی در خروجی آن ایجاد کنیم که نتیجه ی آن چیزی جز ار بین رفتن دقت مدل نیست.



شکل ۱۰.۱: فضای پارامتر های مدل ResNet-56 که مدل در آن به جستجوی نقطه ی بهینه می پردازد.

تعریف ۱۰.۱. *چکیده سازی دانش*

ایده ی اصلی این روش، ساختن یک مدل کوچکتر از نظر تعداد پارامتر درست مشابه مدل اصلی است. در واقع مدل کوچک طوری ساخته می شود تا رفتار دقیق مدل اصلی در تک تک لایه ها را یاد بگیرد. در این روش باید به این نکته دقت کنیم که میزان دقت و عملکرد مدل اصلی همواره یک کران بالا برای دقت مدل کوچکتر است. از آن جا که مدل کوچکتر یک کپی با حجم کمتری از مدل اصلی است، میزان دقت آن در بهترین حالت برابر با مدل اولیه خواهد بود.

[12] Skip-Connections

فصل ۲

شبکه ی عصبی پیچشی مشابه

۱.۲ افزونه مشابه

همان طور که قبلاً اشاره شده بود، وجود افزونگی در ویژگی های استخراج شده از اطلاعات ورودی مدل، معیاری است که با توجه به آن می توان متوجه شد که مدل، موضوع و زمینه[1] ی اطلاعات ورودی را فرا گرفته است. نتیجه ی این موضوع، بالا بودن دقت مدل است.

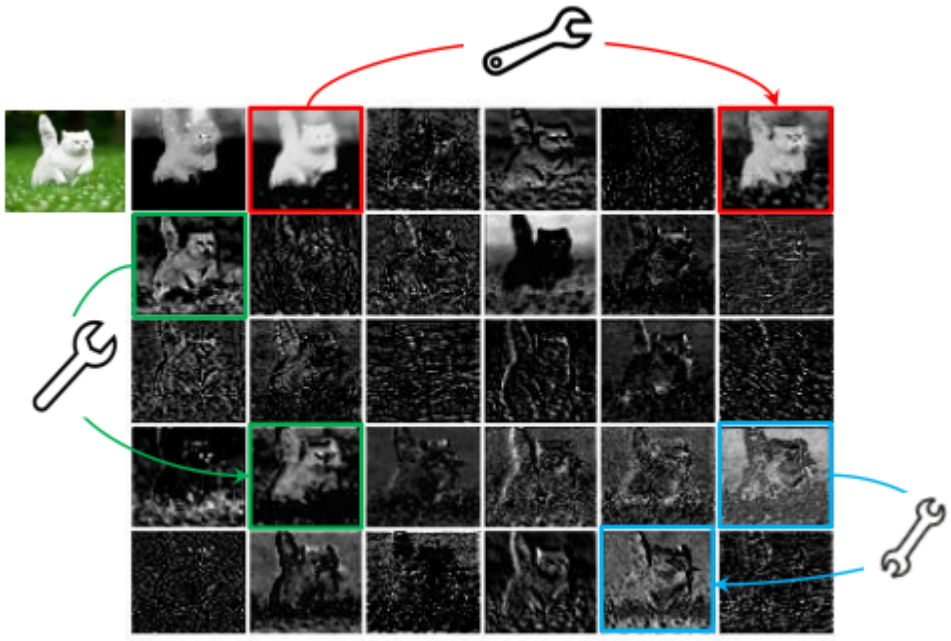
حال، هدف در این بخش این است که افزونگی ویژگی های استخراج شده از اطلاعات ورودی را با کمک افزونه ای بسازیم اما با تعداد کمتری پارامتر. استفاده از این افزونه در شبکه ی عصبی باعث می شود که شبکه ی عصبی به دو قسمت تقسیم شود. قسمت اول مشابه شبکه ی عصبی نرمال است با این تفاوت که پارامتر های آن بسیار کاهش یافته و توان محاسباتی آن به مقدار بسیار زیادی کم شده است. با کمک همین مدل محدود شده، تعدادی ویژگی از داده های ورودی مدل استخراج می کنیم. سپس بخش دوم شبکه ی عصبی افزونه ای است که در بالا به آن اشاره شد و آن افزونه، افزونگی در ویژگی های استخراج شده توسط قسمت اول مدل را با استفاده از تعدادی تبدیل خطی که از نظر محاسباتی بسیار کارآمد و کم هزینه است، ایجاد می کند. کنار هم قرار گرفتن این دو قسمت نهایتا مدل GhostNet را می سازد که از نظر عملکرد از مدل های هم رده ی خود مانند MobileNetV3 بسیار بهتر خواهد بود. خروج ها و اطلاعات در مورد عملکرد مدل در بخش آخر ارائه خواهد شد.

۱.۱.۲ استفاده از افزونه مشابه برای ایجاد ویژگی های بیشتر

علاوه بر تمامی مواردی که در قسمت های قبل به آن اشاره کردیم، دو مدل MobileNet و ShuffleNet هر دو به ترتیب دو روش جدید پیچش عمقی و محاسبات مختلط را برای ساختن

[1] Context

مدل های پیچشی کارآمد با استفاده از فیلتر هایی با ابعاد کمتر معرفی کردند اما همچنان فیلتر های با ابعاد 1×1 ، حجم بالایی از محاسبات را تحمیل و مقدار زیادی از فضای حافظه را اشغال می کنند. حال برای ایجاد افزونگی ها همانند شکل زیر به سراغ تعریف افزونه جدید خود می رویم.



شکل ۱.۲: نمایش ویژگی های استخراج شده اولین لایه ی باقی مانده ای مدل ResNet-50 که در آن سه جفت ویژگی های استخراج شده مشابه با سه رنگ مختلف مشخص شده اند.

فرض کنید تصویر ورودی $X \in \mathbb{R}^{c \times h \times w}$ باشد که c تعداد کانال های تصویر ورودی و h, w به ترتیب طول و عرض تصویر هستند. حال عمل پیچشی دلخواه بر روی تصویر ورودی به صورت زیر تعریف می شود:

$$Y = X * f + b \qquad (۱.۲)$$

* عملیات پیچش است، b نماینده پارامتر جانبدار است و $Y \in \mathbb{R}^{h' \times w' \times n}$ خروجی لایه پیچش است و $f \in \mathbb{R}^{c \times k \times k \times n}$ فیلتر اعمال شده در لایه پیچش است. h' و w' به ترتیب طول و عرض تصویر خروجی از لایه ی پیچش است و $k \times k$ ابعاد فیلتر اعمال شده بر تصویر در این لایه است. با توجه به ابعاد ذکر شده، میزان محاسبات انجام شده برابر است با $n \times h' \times w' \times c \times k \times k$ که

مشخص است که از مرتبه ی میلیون و یا میلیارد است چرا که معمولاً تعداد فیلترها(n) و تعداد کانال ها(c) خیلی زیاد است.(۲۵۶ یا ۵۱۲) حال با توجه به معادلات بالا، بهینه سازی تعداد پارامترها وابسته به ابعاد عکس ورودی و همچنین تعداد ویژگی های استخراج شده لایه خروجی است. حال با توجه به شکل ۱.۲ خروجی لایه های پیچش معمولاً از افزونگی بالایی در ویژگی های استخراج شده برخوردارند و بسیار ی از آنها شبیه به یکدیگر هستند. در نتیجه، تولید این افزونگی ها با استفاده از حجم بالای محاسبات معقول به نظر نمی رسد.
پس به سراغ روش جدید خود می رویم. فرض کنید بخشی از خروجی لایه ی پیچشی را با استفاده از تبدیلات کم هزینه از روی تعدادی ویژگی های استخراج شده اولیه بسازیم. این ویژگی های استخراج شده اولیه، معمولاً اندازه کمتری دارند و با استفاده از فیلتر های پیچشی معمولی ساخته شده اند. در واقع، m ویژگی های استخراج شده $Y' \in \mathbb{R}^{h' \times w' \times m}$ با استفاده از پیچش اولیه به صورت

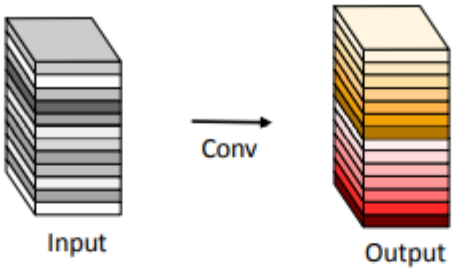
$$Y' = X * f' \qquad (۲.۲)$$

ساخته می شوند که در آن، $f' \in \mathbb{R}^{c \times k \times k \times m}$ فیلتر استفاده شده است و $m \leq n$ و پارامتر جانبدار برای سادگی از فرمول ۲.۲ حذف شده است. ابرپارامتر نظیر اندازه فیلتر، طول گام و میزان افزایش ابعاد تصویر مشابه معادله ۱.۲ خواهد بود تا ابعاد خروجی لایه همان w' و h' حفظ شود. حال برای آن که تعداد n ویژگی های استخراج شده را تولید کنیم، پیشنهاد می شود که از تعدادی عملگر کم هزینه خطی استفاده کنیم تا بتوانیم s ویژگی های مشابه[۲] را از روی خروجی لایه یعنی Y' ایجاد کنیم.

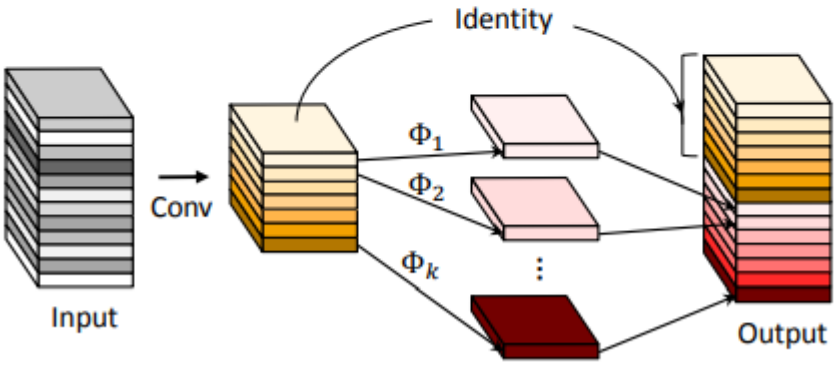
$$y_{ij} = \Phi_{i,j}(y'_i), \forall i = 1, ..., m, j = 1, ..., s \qquad (۳.۲)$$

در معادله بالا، y'_i i امین ویژگی های استخراج شده اولیه[۳] در Y' است و $\Phi_{i,j}$ j امین تبدیل خطی برای ساختن j امین ویژگی های مشابه y_{ij} است. نکته ی قابل توجه این است که y' ممکن است بیش از یک ویژگی های مشابه داشته باشد($\{y_{ij}\}_{j=1}^{s}$) . دقت شود که آخرین تبدیل خطی $\Phi_{i,s}$ تبدیل همانی است. در شکل زیر، افزونه توصیف شده رسم شده است.

[2]ghost feature
[3]intrinsic feature map



(a) The convolutional layer.



(b) The Ghost module.

شکل ۲.۲: مقایسه یک لایه از شبکه ی عصبی معمولی با افزونه جدید معرفی شده

با کمک این افزونه جدید، می توانیم $n = m \times s$ ویژگی های استخراج شده ایجاد کنیم و دقت شود که عملگر ها بر تمامی کانال های داده ها اثر می کنند و این عملیات، به مراتب از عملیات پیچش معمولی کم هزینه تر خواهد بود.

۲.۱.۲ تحلیل پیچیدگی

حال، از آن جایی که ما می توانیم از ۳.۲ استفاده کنیم تا ویژگی های استخراج شده را با حجم کمتری از محاسبات استفاده کنیم، به راحتی می توانیم این افزونه را با یک مدل پیچشی ترکیب کرده و سرعت انجام محاسبات را افزایش دهیم. حال با توجه به ساختار افزونه، یک تبدیل همانی به همراه $n \times (s-1) = \frac{n}{s} \times (s-1)$ تبدیل خطی و فیلتر هایی با ابعاد $d \times d$ داریم. به صورت ایده آل، $n \times (s-1)$ تبدیل خطی می توانند ابعاد و متغیر های گوناگونی داشته باشند اما این تفاوت ها ممکن است هنگام پیاده سازی مشکلاتی ایجاد کنند. به همین دلیل، پیشنهاد می شود

که ابعاد تبدیلات خطی ثابت فرض شوند (5×5 و یا 3×3) . نهایتا نسبت افزایش سرعتی که این افزونه نسبت به مدل های معمولی از نظر تئوری ایجاد می کند، به صورت زیر خواهد بود:

$$\begin{aligned} r_s &= \frac{n \times h' \times w' \times c \times k \times k}{\frac{n}{s} \times h' \times w' \times c \times k \times k + (s-1) \times \frac{n}{s} \times h' \times w \times d \times d} \\ &= \frac{c \times k \times k}{\frac{1}{s} \times c \times k \times k + \frac{s-1}{s} \times d \times d} \\ &\approx \frac{s \times c}{s + c - 1} \\ &\approx s \end{aligned} \tag{۴.۲}$$

در معادله بالا، ابعاد $d \times d$ با $k \times k$ یکی است و $s \ll c$ به صورت مشابه، نسبت فشرده شدن مدل جدید نیز قابل محاسبه است:

$$\begin{aligned} r_c &= \frac{n \times c \times k \times k}{\frac{n}{s} \times c \times k \times k + (s-1) \times \frac{n}{s} \times d \times d} \\ &\approx \frac{s \times c}{s + c - 1} \approx s \end{aligned} \tag{۵.۲}$$

۲.۲ ساخت مدل پیچشی کارآمد

۱.۲.۲ لایه مشابه

برای آن که بتوانیم از مزایای افزونه جدید استفاده کنیم، باید بتوانیم آن را داخل شبکه ی عصبی استفاده کنیم. در ادامه نحوه ی اضافه کردن افزونه به شبکه ی عصبی را توضیح می دهیم.

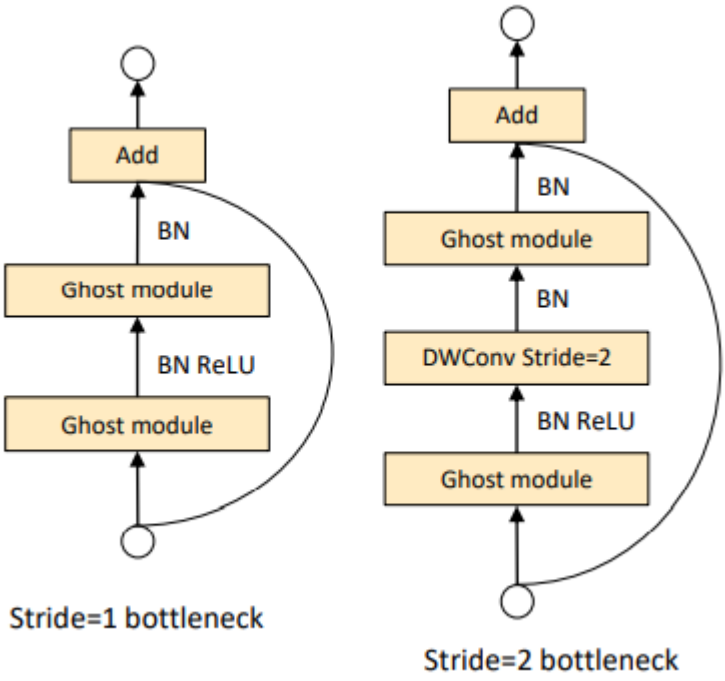
لایه ی افزونه را همانند شکل ۳.۲ مشابه با لایه ی باقی مانده در مدل ResNet تعریف می کنیم که در آن چند میان بر نیز وجود دارد. این لایه به طور کلی از دو افزونه تشکیل شده است. اولین افزونه به صورت یک بسط دهنده استفاده می شود تا تعداد کانال ها در داده ی ورودی لایه را افزایش دهد. افزونه دوم، تعداد کانال های داده ها را کاهش می دهد تا تعداد آنها با تعداد کانال های خروجی میان بر برابر باشد. میان بر نیز ورودی و خروجی این دو افزونه را به یکدیگر متصل می کند. بعد از هر لایه، نرمال سازی دسته ای و تابع غیر خطی RELU بر روی خروجی ها اعمال می شود اما تابع RELU بر روی خروجی های دومین افزونه طبق پیشنهاد مدل MobileNetV2 اعمال نمی شود. همان طور که در شکل نشان داده شده، اگر طول گام برابر با ۲ باشد، یک عملیات

میانی پیچش عمقی با طول گام ۲ صورت می گیرد. نکته ی نهایی آن که برای کارایی بیشتر، در اولین افزونه، عملیات پیچش به صورت نقطه به نقطه[۴] خواهد بود.

۲.۲.۲ شبکه ی عصبی پیچشی مشابه

حال همه چیز را برای ساختن یک مدل عصبی کارآمد در دست داریم. برای ساختن شبکه ی عصبی پیچشی مشابه ، از ساختار کلی مدل MobileNetV3 استفاده می کنیم و تمامی لایه های آن را با لایه های مشابه خود که در قسمت قبل معرفی کردیم، جایگذاری می کنیم. اولین لایه ی مدل ما، یک لایه ی پیچشی معمولی با ۱۶ فیلتر است و سپس سری از لایه های مشابه را داریم که در آنها، به آرامی تعداد کانال ها افزایش پیدا می کند. این لایه های مشابه با توجه به تعداد لایه های ورودی خود به تعدادی طبقه دسته بندی می شوند. تمامی لایه های مشابه با طول گام ۱ هستند به غیر از آخرین آنها در هر طبقه که مقدار طول گام آن ۲ خواهد بود. در آخر، یک لایه ی میانگین[۵] سراسری[۶] و بعد ازآن یک لایه پیچشی وجود خواهد داشت تا در نهایت ویژگی های استخراج شده به یک بردار ۱۲۸۰ تایی برای دسته بندی نهایی تبدیل شود.
ساختار ارائه شده، یک مدل اولیه خواهد بود که می توان آن را با استفاده از تنظیم کردن ابرپارامتر سریع تر و کارآمد تر کرد.

[4] pointwise
[5] average pooling
[6] global



شکل ۳.۲: لایه مشابه. شکل سمت چپ لایه مشابه با طول گام ۱ و شکل سمت راست، لایه افزونه با طول گام ۲

فصل ۳

بررسی عملکرد مدل

در این بخش آزمایش های انجام شده بر روی مدل برای بررسی دقت و عملکرد مدل معرفی می شوند. مدل بر روی مجموعه داده[1] ی CIFAR-10 و ImageNet ILSVRC 2012 آزمایش شده است. مجموعه داده ها CIFAR-10 مجموعه ای از ۶۰۰۰۰ عکس رنگی 32×32 است که ۵۰۰۰۰ تای آنها برای مرحله یادگیری و ۱۰۰۰۰ تای آن برای مرحله ی آزمایش مدل است. ImageNet ۲.۱ میلیون عکس برای فاز یادگیری و ۵۰ هزار عکس برای فاز تست که در مجموع شامل ۱۰۰۰ کلاس مختلف است را در خود دارد. در ادامه خروجی ها را بر روی هر کدام از مجموعه داده گزارش می کنیم.

قبل از پرداختن به خروجی مدل بر روی مجموعه داده ، به بررسی ویژگی های استخراج شده مشابه در شکل ۱.۲ می پردازیم. سعی می کنیم خطای باز تولید ویژگی های استخراج شده مشابه را بدست آوریم. ویژگی های استخراج شده با استفاده از مدل ResNet-50 بدست آمده اند. ویژگی های استخراج شده در سمت چپ و راست شکل ۱.۲ را در نظر می گیریم و سعی می کنیم با استفاده از پیچش عمقی تبدیل بین آنها را یاد بگیریم. d اندازه فیلتر استفاده شده است و میزان خطاها برای جفت ها در جدول زیر نمایش داده شده است.

$MSE(10^{-3})$	red pair	green pair	blue pair
d = 1	4.0	25.0	12.1
d = 3	3.3	24.3	11.2
d = 5	3.3	24.1	11.1
d = 7	3.2	11.2	11.0

جدول ۱.۳: میزان خطای محاسبات به ازای فیلتر های متفاوت

[1] Dataset

همان طور که مشاهده می شود، میزان خطاها بسیار کم و ناچیز هستند که این موضوع میزان همبستگی بالای میان ویژگی های استخراج شده بدست آمده در شبکه ی عصبی و ویژگی های ساخته شده از روی ویژگی استخراج شده اولیه را نشان می دهد.

CIFAR-۱۰ ۱.۰.۳

همان طور که اشاره شد، حال نوبت بررسی خروجی های مدل است. افزونه ی معرفی شده بر روی دو مدل VGG-16 و ResNet-56 پیاده شد و از آن جایی که VGG-16 مخصوص مجموعه داده ها ImageNet طراحی شده است، از نوع دیگر آن استفاده شد. تمامی لایه های پیچش دو مدل با افزونه جایگذاری شدند و دو مدل جدید را Ghost-VGG-16 و Ghost-ResNet-56 نامیدیم. همان طور که قبلا اشاره شد، مدل جدید دو ابرپارامتر d برای اندازه فیلتر و s که همان تعداد ویژگی های استخراج شده است که از روی intrinsic feature map می سازیم. حال به ازای مقادیر مختلفی از این ابرپارامتر ها، دو جدول زیر، عملکرد مدل را نمایش می دهند.

d	VGG-16	1	3	5	7
Weights(M)	15.0	7.6	7.7	7.7	7.7
FLOPs(M)	313	157	158	160	163
Acc.(%)	93.6	93.5	93.7	93.4	93.1

جدول ۲.۳: عملکرد مدل مشابه بر روی مجموعه داده ی CIFAR-10 به ازای مقادیر مختلف d

s	VGG-16	2	3	4	5
Weights(M)	15.0	7.7	5.2	4.0	3.3
FLOPs(M)	313	158	107	80	65
Acc.(%)	93.6	9.7	93.4	93.0	92.9

جدول ۳.۳: عملکرد مدل مشابه بر روی مجموعه داده ی CIFAR-10 به ازای مقادیر مختلف s

ImageNet ۲.۰.۳

در این قسمت خروجی های مدل معرفی شده در مقایسه با گونه های مختلف مدل ResNet بر روی مجموعه داده ی ImageNet در جدول های زیر نمایش داده شده است.

Model	ResNet-50	Thinet-ResNet-50	NISP-ResNet-50-B	Versatile-ResNet-50
Weights(M)	25.6	16.9	14.4	11.0
FLOPs(B)	4.1	2.6	2.3	3.0
Top-1 Ac.(%)	75.3	72.1	-	74.5
Top-5 Acc.(%)	92.2	90.3	90.8	91.8

جدول ۴.۳: مقایسه مدل های مختلف بر روی مجموعه داده ی ImageNet (جدول اول)

Model	Ghost-ResNet-50(s = 2)	Shift-ResNet-50	Taylor-FO-BN-ResNet-50
Weights(M)	13.0	6.0	7.9
FLOPs(B)	2.2	-	1.3
Top-1 Ac.(%)	**75.0**	70.6	71.7
Top-5 Acc.(%)	**92.3**	90.1	-

جدول ۵.۳: مقایسه مدل های مختلف بر روی مجموعه داده ی ImageNet (جدول دوم)

Model	MetaPruning-ResNet-50	Ghost-ResNet-50(s = 4)
Weights(M)	-	6.5
FLOPs(B)	1.0	1.2
Top-1 Ac.(%)	73.4	**74.1**
Top-5 Acc.(%)	-	**91.9**

جدول ۶.۳: مقایسه مدل های مختلف بر روی مجموعه داده ی ImageNet (جدول سوم)

فصل ۴

نتیجه گیری و جمع بندی

در نهایت، با توجه به خروجی های بدست آمده از مدل بر روی مجموعه داده ی مختلف و مقایسه ی آن با مدل های مشابه و هم رده از نظر تعداد پارامتر ها و همچنین حجم محاسبات، مشاهده می کنیم که ساختار جدید ارائه شده موثر واقع شده است. از طرف دیگر با توجه به نحوه ی ساخت این مدل که به عنوان افزونه می توان آن را با اندکی تغییرات به هر شبکه ی عصبی متصل کرد، می توانیم از عملکرد خوب مدل پایه و همچنین فشرده سازی که این افزونه برای ما به ارمغان می آورد، هم از حجم محاسبات بکاهیم و هم مدلی با دقت و عملکردی مطلوب تولید کنیم. این خاصیت افزونه بودن این ساختار، دست محققان در این زمینه را باز می گذارد تا با بررسی روش های مختلف استفاده از این مدل، به مدل هایی جدید تر، سریع تر و کم حجم تر برسند و پیاده سازی الگوریتم شبکه ی عصبی را بر روی دستگاه های نهان مانند کامپیوتر های داخل ماشین های خودران و سایر لوازم الکترونیکی راحت تر سازند.

کتاب‌نامه

[1] Blalock, Davis, et al. “What is the state of neural network pruning?.” arXiv preprint arXiv:2003.03033 (2020).

[2] Han, Song, et al. “Learning both weights and connections for efficient neural network.” Advances in neural information processing systems. 2015.

[3] Molchanov, Pavlo, et al. “Pruning convolutional neural networks for resource efficient inference.” arXiv preprint arXiv:1611.06440 (2016).

[4] Babaeizadeh, Mohammad, Paris Smaragdis, and Roy H. Campbell. “Noiseout: A simple way to prune neural networks.” arXiv preprint arXiv:1611.06211 (2016).

[5] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, ´ Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol ´ Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In ICLR, 2019.

[7] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian. Data-free learning of student networks. In ICCV, 2019.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In ICLR, 2016.

[9] Wuyang Chen, Xinyu Gong, Xianming Liu, Qian Zhang, Yuan Li, and Zhangyang Wang. Fasterseg: Searching for faster real-time semantic segmentation. In ICLR, 2020.

[10] Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In CVPR, 2019.

[11] Franc¸ois Chollet. Xception: Deep learning with depthwise separable convolutions. In CVPR, pages 1251–1258, 2017.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In CVPR, pages 248–255. Ieee, 2009.

[13] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In NeurIPS, pages 1269–1277, 2014.

[14] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In ICCV, 2019.

[15] Shupeng Gui, Haotao N Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. In NeurIPS, 2019.

[16] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In NeurIPS, 2018.

[17] Kai Han, Jianyuan Guo, Chao Zhang, and Mingjian Zhu. Attribute-aware attention model for fine-grained representation learning. In ACM MM, 2018.

[18] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In ICLR, 2016.

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In NeurIPS, pages 1135–1143, 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pages 770–778, 2016.

[21] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In ECCV, 2018.

[22] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In ICCV, 2017.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In ICCV, 2019.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[26] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, 2018.

[27] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In ECCV, pages 304–320, 2018.

[28] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran ElYaniv, and Yoshua Bengio. Binarized neural networks. In NeurIPS, pages 4107–4115, 2016.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

[30] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In CVPR, pages 2704–2713, 2018.

[31] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In BMVC, 2014.

[32] Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In NeurIPS, 2018.

[33] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NeurIPS, pages 1097–1105, 2012.

[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In ICLR, 2017.

[36] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, ´ Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In CVPR, 2017. 9

[37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In ´ ICCV, 2017.

[38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C Lawrence ´ Zitnick. Microsoft coco: Common objects in context. In ECCV. Springer, 2014.

[39] Chuanjian Liu, Yunhe Wang, Kai Han, Chunjing Xu, and Chang Xu. Learning instance-wise sparsity for accelerating deep models. In IJCAI, 2019.

[40] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In ICCV, 2019.

[41] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In ICLR, 2019.

[42] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In ECCV, 2018.

[43] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In ICCV, pages 5058–5066, 2017.

[44] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In ECCV, 2018.

[45] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In CVPR, 2019.

[46] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In ECCV, pages 525–542. Springer, 2016.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NeurIPS, 2015.

[48] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In CVPR, pages 4510–4520, 2018.

[49] Mingzhu Shen, Kai Han, Chunjing Xu, and Yunhe Wang. Searching for accurate binary neural architectures. In ICCV Workshops, 2019.

[50] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[51] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In CVPR, pages 2820–2828, 2019.

[52] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80NeurIPS, 2019.

[53] Yunhe Wang, Chang Xu, Chunjing XU, Chao Xu, and Dacheng Tao. Learning versatile filters for efficient convolutional neural networks. In NeurIPS, 2018.

[54] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. Cnnpack: packing convolutional neural networks in the frequency domain. In NeurIPS, pages 253–261, 2016.

[55] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In NeurIPS, pages 2074–2082, 2016.

[56] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In CVPR, pages 10734–10742, 2019.

[57] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In CVPR, 2018.

[58] Yixing Xu, Yunhe Wang, Hanting Chen, Kai Han, XU Chunjing, Dacheng Tao, and Chang Xu. Positive-unlabeled compression on the cloud. In NeurIPS, 2019.

[59] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. arXiv preprint arXiv:1909.04977, 2019.

[60] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In ICML, 2019.

[61] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In SIGKDD, 2017.

[62] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In ICLR, 2019.

[63] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In CVPR, 2018.

[64] Sergey Zagoruyko. 92.45 on cifar-10 in torch, 2015. `http://torch.ch/blog/2015/07/30/cifar.html`.

[65] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. CVPR, 2018.

[66] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In CVPR, pages 8697–8710, 2018

Abstract

Deploying convolutional neural networks (CNNs) on embedded devices is difficult due to the limited memory and computation resources. The redundancy in feature maps is an important characteristic of those successful CNNs, but has rarely been investigated in neural architecture design. This paper proposes a novel Ghost module to generate more feature maps from cheap operations. Based on a set of intrinsic feature maps, we apply a series of linear transformations with cheap cost to generate many ghost feature maps that could fully reveal information underlying intrinsic features. The proposed Ghost module can be taken as a plug-and-play component to upgrade existing convolutional neural networks. Ghost bottlenecks are designed to stack Ghost modules, and then the lightweight GhostNet can be easily established. Experiments conducted on benchmarks demonstrate that the proposed Ghost module is an impressive alternative of convolution layers in baseline models, and our GhostNet can achieve higher recognition performance (e.g. 75.7% top-1 accuracy) than MobileNetV3 with similar computational cost on the ImageNet ILSVRC2012 classification dataset.

College of Science
School of Mathematics, Statistics, and Computer Science

GhostNet: More Features from Cheap Operations

Shafigh Ashrafi

Supervisor: Dr. Sajedi

A thesis submitted in partial fulfillment of the requirements for
the degree of B.Sc. in Pure Mathematics