بسم الله الرحمن الرحیم

دانشکده ریاضی و علوم کامپیوتر

پایان‌نامه برای دریافت درجه کارشناسی ارشد در رشته علوم تصمیم و مهندسی دانش

بهبود وضوح تصویر توسط شبکه عصبی پیچشی

استادان راهنما

دکتر محمود امین طوسی و دکتر مهدی زعفرانیه

استاد مشاور

دکتر سمیه ثباتی مقدم

پژوهشگر:

محمد مهدی صاحبی

بهمن ۱۴۰۰

باسمه تعالی

فرم ارزشیابی و صورتجلسه دفاع از پایان‌نامه کارشناسی ارشد

فرم ۱۱۳-ت

جلسه دفاع از پایان‌نامه آقای /خانم **محمد مهدی صاحبی** دانشجوی رشته علوم تصمیم و مهندسی دانش به شماره دانشجویی ۹۶۲۳۱۳۷۰۲۱ با عنوان:

بهبود وضوح تصویر توسط شبکه عصبی پیچشی

در مورخه ..... در دانشکده ریاضی و علوم کامپیوتر تشکیل و توسط هیات داوران مورد ارزشیابی قرار گرفت و نمره ..... برابر درجه ..... برای آن تعیین گردید .

به‌این ترتیب از این تاریخ آقای/ خانم محمد مهدی صاحبی به عنوان کارشناس ارشد در رشته مذکور شناخته می‌شود .

موارد ارزشیابی	موارد	حداکثر نمره	نمره کسب شده
۱- کیفیت نگارش	رعایت اصول نگارش انسجام در تنظیم بخشهای مختلف ،کیفیت تصاویر ، جداول و اشکال ،تنظیم فهرست ها ، منابع و ماخذ	٤	
۲- کیفیت علمی	بررسی تاریخچه و سابقه تجربی و نظری موضوع انسجام منطقی دربخش های مختلف پایان نامه ،ابتکار و نوآوری ،اهمیت و ارزش علمی پایان نامه ،استفاده از منابع معتبر و جدید ،کیفیت تجزیه و تحلیل یافته ها و نتیجه گیری ،روشن بودن روش کار ،هدف ها و فرضیه های تحقیق ،جدید بودن روش تحقیق	۱۰	
۳-کیفیت ارایه در جلسه دفاع	تسلط بر موضوع و بیان واضح وتفهیم آن ،توانایی در پاسخگویی به سوالات مطرح شده در جلسه ، رعایت زمان ارایه , روش ارائه	٤	
٤-ارزشیابی گزارشات	گزارش های دوره ای پیشرفت کار(حداقل ٤مورد)	۱	
٥- خروجی پایان نامه	مقاله مستخرج از پایان نامه: این نمره به صورت زیر اختصاص می یابد ۱) چکیده کنفرانسی هر مورد ۰/۲۵ نمره تا سقف۰/۵ نمره ۲) مقاله کامل در مجموع مقالات همایشهای معتبر یا مقاله در مجلات علمی-ترویجی معتبر پذیرفته شده یا چاپ شده هر مورد ۰/۵ نمره تا سقف ۱نمره ۳)مقاله پذیرفته شده یا چاپ شده در مجلات علمی پژوهشی معتبر ۱ نمره ٤)مقاله ارسال شدهبه مجلات علمی پژوهشی معتبرهر مورد ۲۵/. نمره تا سقف ۰/۵ نمره ٥) دستگاه ساخته شده دارای گواهی ثبت اختراع یا به سفارش سازمان ها تا سقف ۱ نمره ٦) دستگاه ساخته شده کاربردی که به تایید رئیس دانشکده رسیده باشد تا سقف ۰/۵ نمره	۱	
جمع			

درجه معادل کسب شده: (از ۱۹تا ۲۰ عالی☐ از۱۸تا ۱۸/۹۹بسیارخوب ☐ از ۱٦تا ۱۷/۹۹خوب ☐ از ۱٤ تا ۱۵/۹۹قابل قبول☐ کمتر از ۱٤ غیر قابل قبول☐)

مشخصات هیات دوران

ردیف	نام و نام خانوادگی	سمت	مرتبه علمی	محل کار	امضـــــــــــا
۱	دکتر محمود امین طوسی	استاد راهنما	استادیار	دانشگاه حکیم سبزواری	
۲	دکتر مهدی زعفرانیه	استاد راهنما	استادیار	دانشگاه حکیم سبزواری	
۳	دکتر سمیه ثباتی مقدم	استاد مشاور	استادیار	دانشگاه حکیم سبزواری	
۴	دکتر	استاد داور	استادیار	دانشگاه حکیم سبزواری	
۵	دکتر	نماینده تحصیلات تکمیلی	استادیار	دانشگاه حکیم سبزواری	

امضا

مدیر گروه

امضا

رئیس دانشکده

سوگند نامه دانش آموختگان دانشگاه حکیم سبزواری

به نام خداوند جان و خرد کزین برتر اندیشه بر نگذرد

اینک که به خواست آفریدگار پاک، کوشش خویش و بهره گیری از دانش استادان و سرمایه‌های مادی و معنوی این مرز و بوم، توشه‌ای از دانش و خرد گردآورده‌ام، در پیشگاه خداوند بزرگ سوگند یاد می کنم که در به کارگیری دانش خویش، همواره بر راه راست ودرست گام بردارم. خداوند بزرگ، شما شاهدان، دانشجویان و دیگر حاضران را به عنوان داورانی امین گواه می گیرم که ازهمه دانش و توان خود برای گسترش مرزهای دانش بهره گیرم و از هیچ کوششی برای تبدیل جهان به جایی بهتر برای زیستن، دریغ نورزم. پیمان می‌بندم که همواره کرامت انسانی را در نظر داشته باشم و همنوعان خود را در هر زمان و مکان تا سر حد امکان یاری دهم. سوگند می خورم که در به کارگیری دانش خویش به کاری که با راه و رسم انسانی، آیین پرهیزگاری، شرافت و اصول اخلاقی برخاسته از ادیان بزرگ الهی، به ویژه دین مبین اسلام، مباینت دارد دست نیازم. همچنین در سایه اصول جهان شمول انسانی و اسلامی، پیمان می‌بندم از هیچ کوششی برای آبادانی و سرافرازی میهن و هم میهنانم فروگذاری نکنم و خداوند بزرگ را به یاری طلبم تا همواره در پیشگاه او و در برابر وجدان بیدار خویش و ملت سرافراز، بر این پیمان تا ابد استوار بمانم.

نام و نام خانوادگی: محمد مهدی صاحبی

تاریخ و امضا:

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب محمد مهدی صاحبی به شماره دانشجویی ۹۶۲۳۱۳۷۰۲۱ دانشجوی رشته علوم تصمیم و مهندسی دانش مقطع تحصیلی کارشناسی ارشد تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه برداری شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی صلاح (اعم از اداری و قضایی) به عهده ی اینجانب خواهد بود و دانشگاه هیچ گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: محمد مهدی صاحبی

تاریخ و امضا:

مجوز بهره برداری از پایان‌نامه

بهره برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

☐ بهره برداری از این پایان‌نامه برای همگان بلامانع است.

☐ بهره برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

☐ بهره برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

استادان راهنما: دکتر محمود امین طوسی

دکتر مهدی زعفرانیه

تاریخ و امضا:

تقدیم به:

همسر و فرزندم

و

پدر و مادرم

سپاس خداوندگار حکیم را که با لطف بی کران خود، آدمی را زیور عقل آراست.

در آغاز وظیفه خود می دانم از زحمات بی دریغ استاد راهنمای خود، جناب آقای دکتر محمود امین‌طوسی و جناب آقای دکتر مهدی زعفرانیه، صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی های ارزنده ایشان، این مجموعه به انجام نمی رسید.

از جناب خانم دکتر سمیه ثباتی مقدم که زحمت مطالعه و مشاوره این رساله را تقبل فرمودند و در آماده سازی این رساله، به نحو احسن اینجانب را مورد راهنمایی قرار دادند، کمال امتنان را دارم.

در پایان، بوسه می زنم بر دستان خداوندگاران مهر و مهربانی، پدر و مادر عزیزم و بعد از خدا، ستایش می‌کنم وجود مقدس شان را و تشکر می‌کنم از خانواده عزیزم به پاس عاطفه سرشار و گرمای امیدبخش وجودشان، که بهترین پشتیبان من بودند.

محمد مهدی صاحبی

بهمن ۱۴۰۰

فهرست مطالب

فهرست جداول

فهرست تصاویر

فرم چکیده ی پایان‌نامه ی دوره ی تحصیلات تکمیلی

مدیریت تحصیلات تکمیلی

دانشگاه حکیم سبزواری

نام خانوادگی دانشجو:صاحبی	نام:محمد مهدی	ش. دانشجویی:۹۶۲۳۱۳۷۰۲۱
استادان راهنما: دکتر محمود امین طوسی و دکتر مهدی زعفرانیه		
استاد مشاور: دکتر سمیه ثباتی مقدم		
دانشکده ریاضی و علوم کامپیوتر	رشته: علوم تصمیم و مهندسی دانش	
مقطع: کارشناسی ارشد	تاریخ دفاع:بهمن ۱۴۰۰	تعداد صفحات: ؟؟
عنوان پایان‌نامه: بهبود وضوح تصویر توسط شبکه عصبی پیچشی		
کلید واژه‌ها:بهبود وضوح تصویر، یادگیری عمیق، شبکه عصبی، شبکه عصبی پیچشی		

چکیده:

سیستم‌های تصویربرداری دیجیتال به دلیل راحتی کاربرد و هزینه مناسب به‌طور چشمگیری گسترش یافته‌اند، اما هنوز به دلیل پائین بودن وضوح تصویر نسبت به سیستم‌های تصویر برداری پیشین (سیستم‌های نوری)، دچار ضعف می‌باشند. تلاش‌های بسیاری جهت افزایش وضوح تصاویر دیجیتالی صورت گرفته که به دو بخش کلی نرم‌افزاری و سخت‌افزاری قابل تقسیم‌بندی می‌باشند. در بخش سخت‌افزاری با هرچه غنی‌تر نمودن تعداد پیکسل‌های موجود بر روی حسگرهای دوربین‌های دیجیتالی در واحد سطح، می‌توان درجه تفکیک تصویر را افزایش داد. بعلاوه، با هرچه کوچکتر نمودن سلول‌های حسگرهای دوربین‌های دیجیتالی، مقدار نور مؤثر دریافت شده توسط هر سلول، کاهش می‌یابد؛ روش‌های سخت‌افزاری جهت رسیدن به تصاویری با کیفیت و وضوح بالاتر، بسیار پرهزینه و عملاً تا حدی غیرممکن می‌باشد و معمولاً نمی‌توان از حد معینی، به دلیل محدودیت‌های تکنیکی موجود در تکنولوژی ساخت مدارات مجتمع، فراتر رفت.

استفاده از روش نرم‌افزاری، جهت افزایش وضوح تصاویر دیجیتالی موضوعی است که به عنوان راه حل جایگزین روش‌های سخت‌افزاری مطرح می‌گردد که از لحاظ اقتصادی مقرون به صرفه می‌باشد. هدف در چنین روش‌های نرم‌افزاری، تولید تصویر با وضوح بالاتر توسط همان دوربین‌های تصویر برداری دیجیتالی با وضوح پائین می‌باشد به طوری که تصویر نهایی از لحاظ وضوح همانند تصویر برداشت شده توسط دوربینی با وضوح بالاتر گردد که اگر در دسترس می‌بود، می‌توان برداشت نمود. پس از توسعه شبکه‌های عصبی و معرفی لایه‌های پیچشی، متدهای وضوح تصویر توانسته‌اند به دقت‌های مطلوبی دست پیدا کنند. متدهای مختلفی مبتنی بر لایه‌های پیچشی همانند SRCNN، FSRCNN، MemNet، SubPixel، VDSR معرفی شده‌اند که در این پایان‌نامه این متدها مورد استفاده قرارگرفته‌اند و تاثیر الگوریتم‌های بهینه سازی SGD، RProp، RMSProp، Lamb، AdaMax، AdaGrad، AdaDelta در میزان بهبود وضوح تصویر و سرعت بهبود وضوح تصویر مورد بررسی قرار گرفته است.

پیش‌گفتار

بهبود وضوح تصویر، که هدف آن بازیابی تصویر با وضوح بالا از تصویر با وضوح پایین است، یک مشکل کلاسیک در بینایی ماشین است. این مشکل ذاتاً بد وضع است زیرا راه حل‌های متعددی برای هر پیکسل با وضوح پایین وجود دارد تا تبدیل به تصویر وضوح‌برتر شود. به عبارت دیگر، این یک مسئله معکوس تعریف نشده است که راه حل آن منحصر به فرد ندارد. چنین مشکلی معمولاً با محدود کردن فضای راه حل توسط اطلاعات قبلی کاهش می‌یابد. روش‌های کلاسیک زیادی برای بهبود تصویر پیشنهاد شده‌اند، از جمله روش‌های مبتنی بر پیش‌بینی[۱]، روش‌های مبتنی بر لبه [۲] ، روش‌های آماری[۳]، روش‌های مبتنی بر پچ[۴] و روش‌های ماترس تنک[۵].

با توسعه سریع تکنیک‌های یادگیری عمیق در سال‌های اخیر، مدل‌های بهبود وضوح تصویر مبتنی بر یادگیری عمیق به طور گسترده‌ای مورد بررسی قرار گرفته‌اند و اغلب به عملکرد بسیار خوبی در معیارهای مختلف بهبود وضوح تصویر دست یافته‌اند. انواع روش‌های یادگیری عمیق بهبود وضوح تصویر مورد استفاده قرار گرفته‌اند، از روش مبتنی بر شبکه‌های عصبی پیچشی (CNN) تا رویکردهای اخیر با استفاده از شبکه‌های متخاصم مولد به.(GAN) طور کلی، الگوریتم‌های بهبود وضوح تصویر با استفاده از تکنیک‌های یادگیری عمیق در جنبه‌های انواع مختلف معماری شبکه، انواع مختلف توابع هزینه با هم تفاوت دارند.

در این پایان‌نامه، به بررسی متدهای وضوح تصویر یادگیری عمیق بر پایه متدهای پیچشی ارائه شده پرداخته شده است. متدهای مورد بررسی قرارگرفته شامل متدهای SRCNN، FSRCNN، MemNet، SubPixel، VDSR می‌باشد. این متدهای بر روی مجموعه داده‌هایی که از تصویر برداری هوایی از دانشگاه زوریخ تهیه شده و پاک سازی شده‌اند تست و آزمایش شده‌اند و هدف از این امر آزمایش بهبود وضوح تصاویر هوایی که توسط ماهواره‌ها و یا هواپیماهای بدون سرنشین، پهبادها انجام می‌پذیرد بوده است.

---

[۱] prediction-based methods [۲] edge-based methods [۳] statistical methods [۴] patch-based

[۵] sparse representation methods methods

فصل ۱

مرور تاریخی

در این فصل مروری بر شرح مسئله بهبود وضوح تصاویر می‌شود، و به شرح متدهای مختلف بهبود وضوح تصویر پرداخته می‌شود.

۱-۱ مقدمه

در بسیاری از کاربردهایی که از تصاویر دیجیتال استفاده می‌شود، تصاویری با وضوح بالا مورد نیاز است. وضوح تصویر بالا به معنای تراکم بالای پیکسل ها در تصاویر می‌باشد. از این رو تصاویر وضوح بالا جزئیات بیشتری از موضوع را فراهم می‌کنند. در کاربردهای مختلف که از تصاویر استفاده می‌شود همانند تصویربرداری پزشکی، نظامی، ماهواره‌ای، زیرآب، سنجش از راه دور وضوح برتر تصویر اهمیت بالایی دارند.

در گذشته تنها ابزار تصویر برداری فقط دو دستگاه آنالوگ ویدیکان و ارتوکان، در دسترس عکاسان بوده است. در دهه ۱۹۷۰ سنسورهای دستگاه‌های باز جفت شده [۲] و سیماس [۳] معرفی شدند و پس از آن این دو سنسور به طور گسترده در تصویر برداری دیجیتال استفاده می‌شود. سطوح وضوح تصویر در دسترس و قیمت‌های سنسورهای فوق برای تقاضای موجود در آینده مناسب نیستند. تقاضا برای وضوح تصویر بالاتر، انگیزه‌ای شده است برای یافتن راهکارهایی که تصاویر وضوح پایین تهیه شده توسط دستگاه‌های تصویر برداری دیجیتال فعلی را بهبود دهد.

یکی از راه‌حل‌های مستقیم برای افزایش وضوح تصویر را در فن‌آوری تولید ساخت سنسورها باید جستجو کرد. یکی از راه‌های فن‌آورانه بهبود وضوح تصویر در کارخانه‌های تولید کننده سنسور، کاهش اندازه پیکسل‌های سنسورها است، که کوچتر شدن اندازه پیکسل‌ها باعث می‌شود تعداد پیکسل‌ها در واحد سطح افزایش یابد. اما اشکال این راه حل سخت‌افزاری، کاهش نور در دسترس برای هر پیکسل می‌باشد. کاهش نور در دسترس منجر به ایجاد نویز می‌شود و به طور اثرگذاری کیفیت تصاویر را کاهش می‌دهد. متاسفانه اندازه هر پیکسل به علت دوری گزیدن از نویز تصویر برداری نمی‌تواند از یک سطح

[۲] CCD [۳] CMOS

معین کوچکتر تولید شود ($۴۰\mu m$ مربع برای سیماس‌های $۳۵m$). سطح فن‌آوری تولید سنسورها در اندازه فوق در دسترس کارخانه‌های سازنده سنسور می‌باشد و بهبودی در این زمینه دیگر نتوانسته‌اند داشته باشند.

راه حل دیگر تولید کننده‌های سخت‌افزارهای سنسور، در راستای بهبود وضوح تصاویر، افزایش اندازه سنسورها با اندازه پیکسل‌های ثابت است. این روش باعث افزایش ظرفیت خازنی تراشه می‌شود. این افزایش ظرفت خارنی سرعت انتقال شارژ را محدود می‌کند و کاهش انتقال شارژ در فرآیند تشکیل تصور تبدیل به یک مشکل بزرگ خواهد شد. به طور کل تمام راه‌حل‌های سخت افزاری برای مسئله بهبود وضوح تصویر با محدودیت‌هایی سخت افزاری و افزایش قیمت همراه خواهد بود و بهبود سخت افزاری برای بهبود وضوح تصویر بسیار گران و گاه نشدنی هستند.

راه‌حل مناسب برای این مسئله بهبود وضوح تصویر ادغام نمودن روش‌های سخت افزاری و نرم افزاری با یکدیگر است. روند جدیدی که تولید کنندگان دستگاه‌های تصویر برداری نیز به سمت آن حرکت کرده‌اند استفاده موثر از الگوریتم‌های بهبود وضوح برتر است.

۱-۲ راه حل‌های نرم افزاری بهبود وضوح تصویر

هدف بهبود وضوح تصویر، بازیابی یک تصویر با وضوح بالا از یک یا چند تصویر ورودی با وضوح پایین می‌باشد، به نحوی که اطلاعات مورد نیاز از عکس بهبود وضوح یافته قابل بازیابی شود. بهبود وضوح تصویر مهمترین کلاس از مسائل پردازش تصویر و بینایی ماشین می‌باشد[۵]. در مسئله بهبود وضوح تصویر سعی می‌شود مسئله به جای روش‌های سخت افزار به کمک الگوریتم‌های نرم افزار حل شود. بهبود وضوح تصویر کاربردهای بسیاری در حوزه‌های مختلف همانند پزشکی، امنیت، نظارت، تصویر برداری ماهواره‌ای و کاربردهایی که از تصاویر استفاده می‌شود دارد، علاوه بر اینکه بهبود وضوح تصویر باعث افزایش ادراکی از تصویر می‌شود و به بهبود عملکرد سایر کاربردهای بینایی ماشین نیز کمک می‌کند. نسبت به تعداد تصویر وارده به الگوریتم بهبود وضوح تصویر الگوریتم وضوح تصویر به دو گروه تقسیم می‌شوند:

۱-۲-۱ روش‌های کلاسیک بهبود وضوح چند تصویر

در روش‌های کلاسیک بهبود وضوح تصویر از چند تصویر و ترکیب تصویرها با یکدیگر سعی می‌شود تصویری با کیفیت بهبود داده شده تولید شود. در روش‌های کلاسیک فرض می‌شود که چند عکس با کیفیت پایین (مثلا: از یک ویدئو) می‌توانند تبدیل به یک فایل تصویر با وضوح تصویر بالا شود البته تصاویر با وضوح پایین لازم نیست که یکسان باشند، باید بین تصاویر تفاوت‌هایی وجود داشته باشد مثل انتقال خطی، چرخشی، حرکت به دور دوربین یا زاویه دید متفاوت. در تئوری، اطلاعات مربوط به اشیاء در فریم‌های متفاوت وجود دارد و تغییرات بین فریم‌ها ما را قادر می‌سازد تصاویر بهتری از اشیاء را داشته باشیم.

۱-۲-۲ بهبود وضوح تک تصویر

در روش بهبود وضوح تک تصویر[۱]، تنها از یک تصویر با وضوح پایین، تصویر با وضوح برتر ساخته شود. الگوریتم‌های بهبود وضوح تک تصویر عمدتا به سه بخش تقسیم‌بندی می‌شوند:

۱. روش‌های مبتنی بر درونیابی:

روش بهبود وضوح تصویر مبتنی بر درون‌یابی مانند درونیابی دومکعبی و باز نمونه برداری لانزوس روش‌های ساده و بسیار سریعی هستند که دارای خروجی مطلوبی به نسبت شیوه‌های جدید نمی‌باشند.

۲. روش‌های مبتنی بر بازسازی:

روش‌های بهبود وضوح تصویر مبتنی بر بازسازی اغلب دانش پیشین پیچیده‌ای را اتخاذ می‌کنند و معمولا روش‌های کندی هستند که عملکردشان بر اساس افزایش فاکتور بزرگ‌نمایی کاهش می‌یابد.

۳. روش‌های مبتنی بر یادگیری:

روش‌های مبتنی بر یادگیری که به روش‌های مبتنی بر مثال نیز شناخته می‌شوند به دلیل محاسبات سریع و عملکرد برجسته آن‌ها مورد توجه زیادی هستند. این الگوریتم‌ها معمولا از الگوریتم‌های یادگیری ماشین برای تحلیل روابط آماری میان تصویر با وضوح پایین و همتای تصویر با وضوح بالا مربوطه از مثال‌های آموزشی استفاده می‌کنند. متد میدان تصادفی مارکفی اولین بار توسط فریمن و همکاران بر روی تصاویر موجود در دنیای واقعی استفاده کردند تا تصاویری با بافت‌های تصویری بهتری را تولید نمایند. روش جداسازی همسایگی توسط چانگ و همکاران پیشنهاد شد از هندسه محلی برای پیدا کردن شباهت بین تصویر با وضوح پایین و تصویر با وضوح بالا استفاده کردند تا پچ‌های تصویر با وضوح بالا را بازیابی کنند. با الهام از تئوری بازیابی سیگنال‌های تنک محققین روش کدگذاری تنک را برای حل مسئله بهبود وضوح تصویر به کار گرفتند. جنگل تصادفی همچنین برای بهبود عملکرد وضوح تصویر مورد استفاده قرارگرفته است.

۱-۲-۳ درون‌یابی

درون‌یابی[۲] روشی است برای یافتن مقدار تابع درون یک بازه، زمانی که مقدار تابع در تعدادی از نقاط گسسته معلوم است. یافتن مقدار تابع در خارج از این بازه را برون‌یابی گویند که عموماً از روش‌های مشابهی برای هر دو استفاده می‌شود. در بسیاری از کاربردها در مهندسی و علوم پایه تعدادی نقاط معلوم در دسترس است، مانند داده‌های بدست آمده از آزمایش یا نمونه‌برداری. در چنین مواردی سعی می‌شود تابعی یافت که حتی‌المقدور به داده‌ها نزدیک‌تر باشد. یکی از روش‌های یافتن چنین تابعی درون‌یابی می‌باشد که وجه مشخصه این روش آن است که تابع یافت شده از این روش از تمامی نقاط داده شده می‌گذرد. روشهای درون یابی مجموعه ای از مدلهای مختلف ریاضی و آماری را برای پیش بینی مقادیر نامعلوم بکار

[۱] Single Image Super Resolution [۲] Interpolation

می‌گیرند. آنچه مسلم است شباهت نقاط مجهول به نزدیکترین نقاط معلوم یا اصل نزدیکترین همسایه پایه روشهای دورن یابی است . روش درون‌یابی تصویر روشی شناخته شده در بین بیشتر محققین در حوزه پردازش تصویر می‌باشد. دورن‌یابی در بسیاری از متدهای پردازش تصویر به عنوان یک مرحله ساده ممکن است مشاهده شود. در بهبود وضوح تصویر هدف متد درون‌یابی، برآورد حد واسط پیکسل(هایی) بین پیکسل‌های موجود در تصویر می‌باشد[۶].

در روش درون‌یابی تصویر، پیکسل‌های جدید بر اساس روشنایی پیکسل‌های پیرامونی بهبود وضوح داده می‌شوند. درون‌یابی روشی برای تخمین مقادیر ناشناخته (یا دیده نشده) با استفاده از داده‌های شناخته شده (دیده شده) است. درون‌یابی به شیوه نزدیک‌ترین همسایه و یا درون‌یابی خطی در دو جهت افقی و عمودی صورت می‌پذیرد. درون‌یابی تصویر قصد روشنایی (شدت رنگ) یک نقطه داده‌شده بین نقاط اولیه تصویر (شدت رنگ همسایگی) محاسبه می‌شود. روش درون‌یابی نزدیک‌ترین همسایه ساده‌ترین و سریعترین روش درون‌یابی است. در این شیوه مقدار روشنایی پیکسل نزدیک‌ترین همسایه به پیکسل فوق قرار داده خواهد شد.هرچند این شیوه بسیار سریع است اما باعث کاهش کیفیت تصویر خروجی و گاها شطرنجی شدن تصویر می‌شود این عدم کیفیت با افزایش ضریب بهبود وضوح و بزرگنمایی بیشتر خودشان را نشان می‌دهند. روش درون‌یابی خطی دوسویه برای پیدا کردن شدت روشنایی پیکسل جاداده شده از مقادیر چهار همسایگی و ضریب نسبت نزدیکی پیکسل فوق به همسایگان محاسبه می‌شود، این شیوه تصویری غیرشطرنجی و بسیار نرم‌تر از تصویر نتیجه شده در روش نزدیک‌ترین همسایگی است.

درونیابی دومکعبی یک توسعه از درون‌یابی مکعبی برای تعبیه نقاط داده‌شده در یک شبکه تصویر دو بعدی می‌باشد. شدت روشنایی‌های پیدا شده به شیوه درون‌یابی دومکعبی به نسبت دو شیوه درون‌یابی نزدیک‌ترین همسایه و درون‌یابی خطی دوسویه بسیار صاف‌تر و یکنواخت‌تر هست و بهبود وضوح تصویر بهتری را ارائه می‌دهد. در شیوه درون‌یابی دو معکبی از شانزده نقطه همسایه برای انجام درون‌یابی استفاده می‌شود.

۱-۲-۴ شبکه‌های عصبی

یادگیری عمیق یکی از شاخه‌های یادگیری ماشین است. یادگیری عمیق نتایج بسیار خوبی نسبت به سایر الگوریتم‌های یادگیری ماشین در حوزه‌های کاربری مختلف همانند بینایی‌ماشین، شناسایی صوت، پردازش زبان طبیعی کسب کرده است. دو رویکرد بر روی شبکه‌های عصبی برای بهبود وضوح تک تصویر وجود دارد:

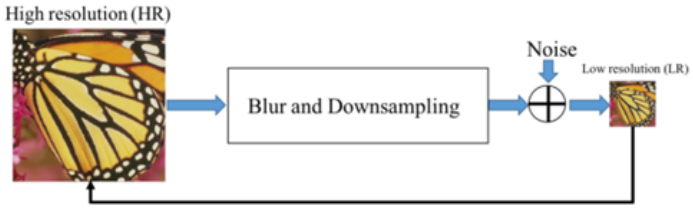
- طراحی معماری شبکه عصبی کارآمد برای بهبود وضوح تک تصویر
- بهینه سازی موثر تابع هدف وضوح تک تصویر

چهارچوب اکثر روش‌های بهبود وضوح یک تصویر[۱] مشابه به تصویر ۱-۱ می‌باشد. و عبارت ۱-۱ متناظر با تصویر ۱-۱ نوشته شده است، در این عبارت، $x \otimes k$ یک پیچش محو کننده بر روی تصویر وضوح بالای x بر اساس فیلتر k می‌باشد،

[۱] SISR - Single Image Super Resolution

و $\downarrow_s$ عملوند کاهش اندازه تصویر بر اساس ضریب s می‌باشد و n نیز یک نویز می‌باشد که به تصویر اضافه می‌شود.

$$y = (x \otimes k) \downarrow_s + n \tag{۱-۱}$$



شکل ۱-۱: فرآیند ایجاد تصویر وضوح پایین و ایجاد وضوح برتر

حل مسئله بهبود وضوح تصویر و ایجاد تصویری که نزدیک به تصویر اولیه باشد جز مسائل بد وضع [۱] است، زیرا یک تصویر با وضوح پایین به خروجی‌های متنوعی از تصویر باوضوح برتر منتهی می‌شود.

۱-۳ تعریف‌ها

۱-۳-۱ نرم

نرم تابعی می‌باشد که موکدا یک طول مثبت و اندازه بردار را در فضای برداری برگشت می‌دهد. به جز بردار صفر که طول صفر اختصاص داده می‌شود. تابع حقیقی $\|.\|$ تعریف شده بر فضای برداری X را نرم می‌نامیم اگر در خاصیت‌های زیر صدق کند:

۱. به ازای هر $x \in X$، $\|x\| \geq 0$ و $\|x\| = 0$ اگر و فقط اگر $x = 0$

۲. به ازای هر x و $\alpha \in \mathbb{R}$، $\|\alpha x\| = |\alpha|\|x\|$

۳. به ازای هر x و $y \in X$، $\|x + y\| \leq \|x\| + \|y\|$

نرم-p

فرض کنید $1 \leq p < \infty$ باشد، p-نرم به صورت زیر تعریف می‌شود:

[۱] ill-posed

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

اگر مقدار p برابر ۲ باشد، نرم حاصله، نرم اقلیدسی[۱] نامیده می‌شود، و اگر مقدار p مقدار ۱ باشد، نرم حاصله، نرم منهتن[۲] نامیده می‌شود.

۱-۳-۲ تنسور

با افزایش تقاضاهای بشر و توسعه سریع علم اطلاعات، جمع آوری، ذخیره و پردازش داده ها به مشکلی اساسی تبدیل شده است. داده‌های واقعی، مانند داده‌های تصویر، داده‌های ویدئویی، داده‌های صوتی، معمولاً عظیم، غنی و پیچیده هستند. به طور کلی، بردارها و ماتریس‌ها نمی‌توانند ویژگی‌های ساختاری داده‌های واقعی را به خوبی بیان کنند. بنابراین، داده‌های واقعی اغلب در ساختار چند بعدی به نام تنسور [۳] سازماندهی می‌شوند[۷]. اجزا تنسورها با متغیرهای پیوسته یا گسسته اندیس گذاری می‌شوند. به عنوان مثال، یک تصویر رنگی توسط یک تنسور رنک ۳ با اندیس ردیف، ستون و رنگ اندیس گذاری می‌شود، و یک ویدیو خاکستری نیز با یک تنسور رنک ۳ با دو مولفه برای تصویر خاکستری و یک مولفه زمان نمایش داده می‌شود. علاوه بر این، داده های تنسور در مسائل واقعی معمولاً دارای ابعادی بالاتری هستند. داده‌های تنسور و روش‌های داده کاوی متناظر توجه زیادی را به خود جلب کرده‌اند و به طور گسترده ای در داده کاوی، بینایی ماشین، پردازش سیگنال و غیره به کار گرفته شده‌اند.

تنسور به عنوان آرایه‌های چندبعدی

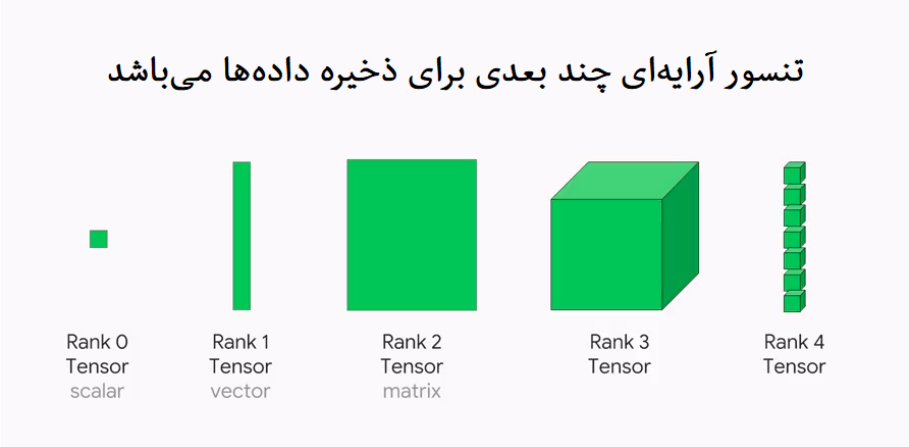
تنسور را می‌توان به صورت آرایه چندبعدی نمایش داد. درست همانگونه که یک بردار در یک فضای n-بعدی به صورت آرایه یک بعدی با n مؤلفه و نسبت به پایه دلخواهی نمایش داده می‌شود، هر تنسور را نیز می‌توان برحسب یک پایه و با کمک آرایه‌ای چندبعدی نمایش داد. به عنوان مثال، یک عملگر خطی را برحسب یک پایه و به صورت آرایه‌ای $n \times n$ نمایش داده می‌شود. درایه‌های این آرایه چندبعدی را مؤلفه‌های اسکالر تنسور نامیده یا صرفاً به آن‌ها مؤلفه‌ها می‌گویند. به کمک اندیس‌ها، موقعیت این درایه‌ها را در آرایه با کمک بالانویس‌ها و پایین نویس‌ها در کنار نماد تنسور مشخص می‌کنند. به عنوان مثال، مؤلفه‌های تنسور مرتبه ۲ را می‌توان به صورت T_{ij} که در آن i و j اندیس‌هایی هستند که مقادیرشان را از ۱ تا n انتخاب می‌شود.

---

[۱] Euclidean [۲] Manhattan [۳] Tensors

تنسورها در توسعه نرم‌افزاری

تنسورها نگهدارنده‌ای برای ذخیره داده‌ها هستند که داده‌ها با ابعاد مختلف را ذخیره می‌کنند تا در یادگیری ماشین مورد استفاده قرار بگیرند. به تنسور دارای رنک ۰ اسکالر گفته می‌شود، مجموعه‌ای از اسکالرها تنسور رنک ۱ را تشکیل می‌دهند که به آن بردار گفته می‌شود و از ترکیب بردارها تنسور رنک ۲ حاصل می‌شود، که ماتریس نامیده می‌شود و به تنسور دارای رنک ۳ مکعب گفته می‌شود و برای تنسورهایی با رنک بالاتر نام‌گذاری انتخاب نشده است و آن‌ها را با رنک‌ای که دارند بیان می‌گردند.



شکل ۱-۲: نمایش آرایه، بردار، ماتریس و تنسور

۱. اسکالر یا تنسور رنک صفر

- یک تنسور که فقط یک داده دارد، اسکالر نامیده می‌شود.
- تنسور اسکالر داری صفر محور است. $ndim == ۰$

۲. بردار یا تنسور رنک یک

- آرایه‌ای از اطلاعات یک بردار نامیده می‌شود.
- تنسور $D - ۱$ است.

۳. ماتریس یا تنسور رنک دو

- آرایه‌ای از بردارها یک ماتریس است.
- دارای دو محور سطر و ستون می‌باشد.

۴. تنسور $D - ۳$ و تنسورهای با ابعاد بزرگتر

- اگر آرایه‌ای ماتریس‌ها را در یک آرایه جدید قرار دهیم یک تنسور $D - ۳$ داریم.
- با قرار دادن یک تنسور رنک سه درون یک آرایه، تنسوری با ابعاد چهار ساخته می‌شود و همینگونه می‌توان ادامه داد تا تنسورهایی با ابعاد بالاتر ایجاد نمود.

۱-۳-۳ ضرب هادامارد

محاسبات شبکه عصبی به خصوص الگوریتم پس‌انتشار که توضیح داده خواهد شد، محاسباتی بر پایه عملیات جبر خطی می‌باشند. مواردی مانند جمع برداری، ضرب بردار در ماتریس و سایر عملیات برداری و ماتریسی، اما هاماراد با عملوند $\odot$، عملوندی است که کمتر مورد استفاده قرار می‌گیرد. دو بردار s و t دارای ابعاد مشابه هستند. آن‌گاه از $s \odot t$ برای نمایش ضرب عناصر دو بردار استفاده می‌کنیم. بنابر این اجزا $s \odot t$ به صورت $(s \odot t)_j = s_j t_j$ محاسبه می‌شوند. به عنوان مثال:

$$\begin{bmatrix} ۱ \\ ۲ \end{bmatrix} \odot \begin{bmatrix} ۳ \\ ۴ \end{bmatrix} = \begin{bmatrix} ۱ * ۳ \\ ۲ * ۴ \end{bmatrix} = \begin{bmatrix} ۳ \\ ۸ \end{bmatrix}.$$

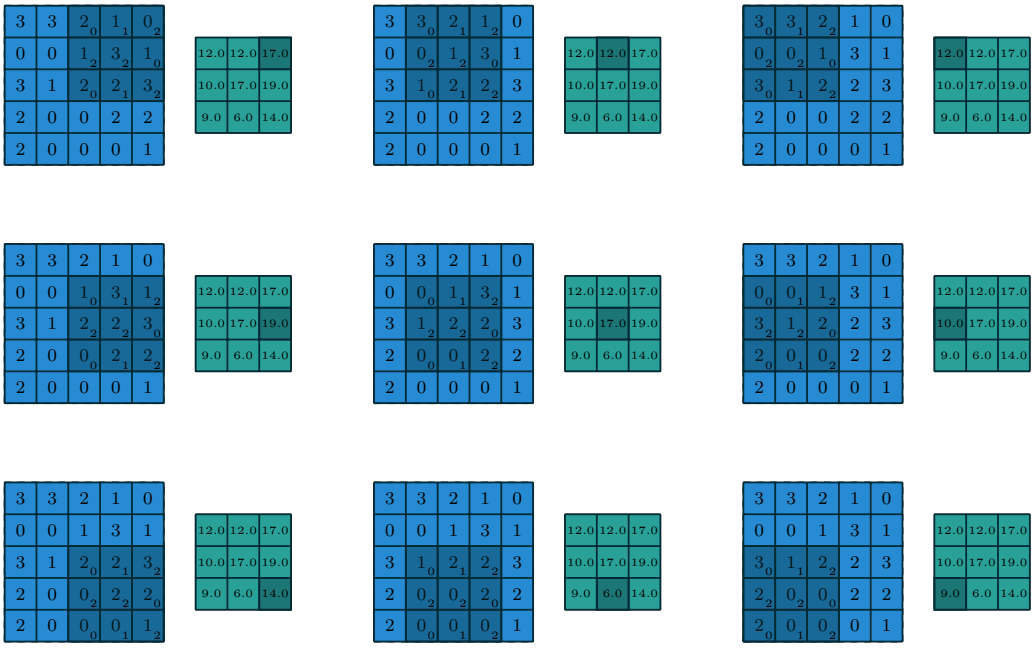
۱-۳-۴ شبکه پیچشی

شبکه‌های عصبی پیچشی یا هم‌گشتی[۱] (مخفف: CNN یا $ConvNet$) گونه‌ای از شبکه‌های عصبی عمیق هستند که معمولاً برای انجام تحلیل‌های تصویری یا گفتاری در یادگیری ماشین استفاده می‌شوند. شبکه‌های عصبی پیچشی به منظور کمینه کردن پیش‌پردازش‌ها از گونه‌ای از پرسپترون‌های چندلایه استفاده می‌کنند. به جای شبکه عصبی پیچشی گاهی از این شبکه‌ها با نام شبکه‌های عصبی تغییرناپذیر با انتقال [۲] یا تغییرناپذیر با فضا [۳] هم یاد می‌شود. این نام‌گذاری بر مبنای ساختار این شبکه است که در ادامه به آن اشاره خواهد شد. ساختار شبکه‌های پیچشی از فرایندهای زیستی قشر بینایی گربه الهام گرفته‌شده‌است. این ساختار به گونه‌ای است که تک‌نورون‌ها تنها در یک ناحیه محدود به تحریک پاسخ می‌دهند که به آن ناحیه پذیرش گفته می‌شود. نواحی پذیرش نورون‌های مختلف به صورت جزئی با هم همپوشانی دارند به گونه ای که کل میدان دید را پوشش می‌دهند. شبکه‌های عصبی پیچشی نسبت به بقیه رویکردهای دسته‌بندی تصاویر به میزان کمتری از پیش‌پردازش استفاده می‌کنند. این امر به معنی آن است که شبکه معیارهایی را فرامی‌گیرد که در رویکردهای قبلی به صورت دستی فراگرفته می‌شدند. این استقلال از دانش پیشین و دستکاری‌های انسانی در شبکه‌های عصبی پیچشی یک مزیت اساسی است. تا کنون کاربردهای مختلفی برای شبکه‌های عصبی از جمله در بینایی کامپیوتر، سیستم‌های پیشنهاددهنده و پردازش زبان طبیعی پیشنهاد شده‌اند. لایه‌های پیچشی یک عمل پیچش را روی ورودی اعمال می‌کنند، سپس نتیجه را به لایه بعدی می‌دهند. این پیچش در واقع پاسخ یک تک‌نورون را به یک تحریک دیداری شبیه‌سازی می‌کند. هر نورون پیچشی

[۱] convolutional neural network [۲] shift invariant [۳] space invariant

داده‌ها را تنها برای ناحیه پذیرش خودش پردازش می‌کند. مشبک‌کردن به شبکه‌های پیچشی این اجازه را می‌دهد که انتقال، دوران یا اعوجاج ورودی را تصحیح کنند. اگرچه شبکه‌های عصبی پیش‌خور کاملاً همبند می‌توانند برای یادگیری ویژگی‌ها و طبقه‌بندی داده به کار روند، این معماری در کاربرد برای تصاویر به کار نمی‌رود. در این حالت حتی برای یک شبکه کم‌عمق تعداد بسیار زیادی نورون لازم است. عمل پیچش یک راه‌حل برای این شرایط است که تعداد پارامترهای آزاد را به عمیق‌تر کردن شبکه کاهش می‌دهد. شکل خروجی یک لایه پیچش تحت تأثیر شکل ورودی آن و همچنین انتخاب فراسنج‌های ابعاد هسته[۱]، لایه صفر[۲] و گام‌ها[۳] قرار می‌گیرد. تصویر ۱-۳ نمونه‌ای از یک کرنل با ابعاد ۳ × ۳ می‌باشد.

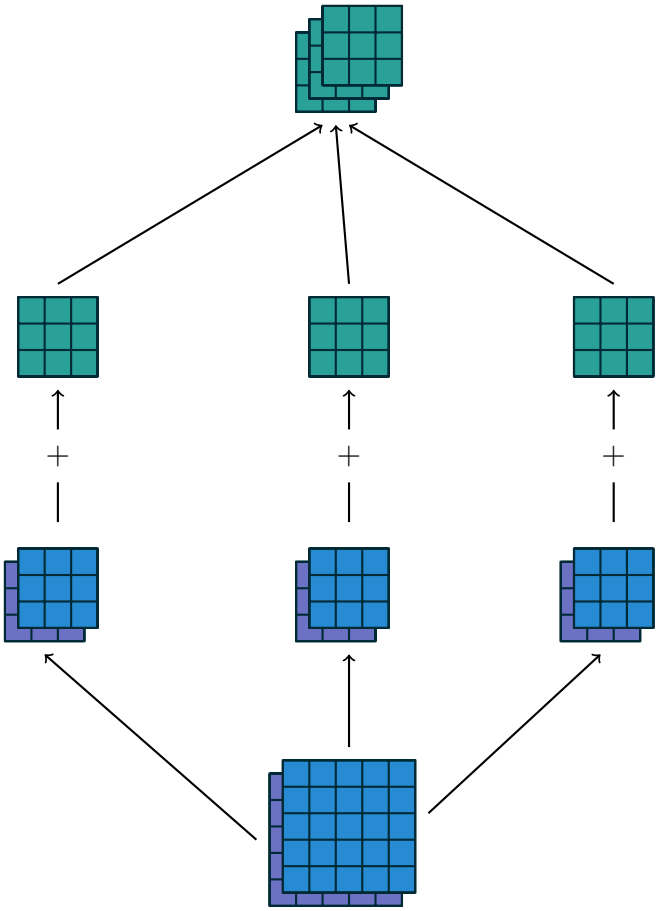
شکل ۱-۳: کرنل ۳ × ۳



شکل ۱-۴: مراحل محاسبه مقدار خروجی عملیات پیچش.

هسته (منطقه سایه دار در تصویر ۱-۴) بر روی تصویر ورودی حرکت می‌کند. در هر مکان، حاصلضرب بین هر عنصر هسته و عنصر ورودی که با آن همپوشانی دارد محاسبه می‌شود و نتایج حاصل شده به عنوان خروجی درج می‌شوند. این روش را می‌توان با استفاده از هسته‌های مختلف تکرار کرد تا تعداد دلخواه نقشه ویژگی خروجی ۱-۵ تشکیل شود. خروجی نهایی عملیات پیچش، نقشه ویژگی[۴] نامیده می‌شود. [۸]

[۱]Kernel [۲]zero padding [۳]strides [۴]feature maps



شکل ۱-۵: یک نگاشت پیچشی از دو نقشه ویژگی ورودی به سه نقشه ویژگی خروجی با استفاده از مجموعه کرنل $۳ \times ۳ \times ۲ \times ۳$. به عنوان نمونه در مسیر سمت چپ، نقشه ویژگی ورودی ۱ با کرنل $\mathbf{w}_{۱,۱}$ و نقشه ویژگی ورودی ۲ با هسته $\mathbf{w}_{۱,۲}$ در هم می‌پیچد و نتایج به صورت عنصری با هم جمع می‌شوند تا اولین نقشه ویژگی خروجی را تشکیل دهند. همین کار برای مسیرهای میانی و راست تکرار می شود تا نقشه های ویژگی دوم و سوم را تشکیل دهند و هر سه نقشه ویژگی خروجی با هم گروه بندی می‌شوند تا خروجی را تشکیل دهند.

۱-۳-۵ واهم‌گشت

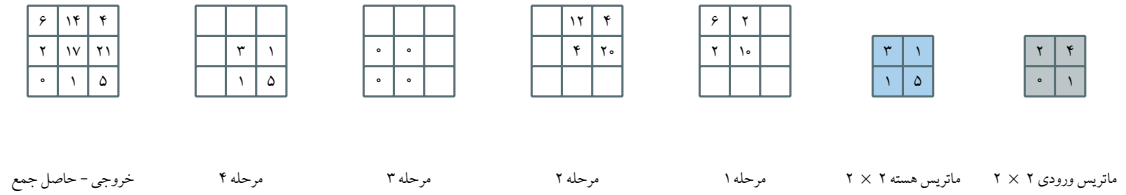
واهَم‌گَشت[۱] به طور ساده، معکوس کردن اثر ناشی از پیچش بر روی داده‌ها است. واهم‌گشت به نام ترانهاده پیچش[۲] نیز نامیده می‌شود. شبکه‌های واهم‌گشت ارتباط نزدیکی با دو روش شبکه‌های باور عمیق[۳] و رمزگذار خودکار[۴] دارد که تلاش می‌کنند ویژگی‌های سلسله مراتبی را از داده‌ها استخراج کنند.

---

[۱] Deconvolution [۲] Transposed Convolution [۳] (DBN -Deep Belief Network [۴] auto-encoders

مثال واهم‌گشت

در جدول ۱-۱ اولین ستون راست ماتریس ورودی، ماتریس ستون دوم از سمت راست، ماتریس هسته در نظر گرفته شده است، و برای سایر پیکربندی‌های واهمگشتی، گام [۱] برابر ۱ و لایه صفرگذار [۲] صفر در نظر گرفته شده است. حال در مرحل مختلف هر درایه‌های ماتریس هسته در ماتریس ورودی ضرب می‌شود و در انتها نتایج هر مرحله با هم جمع می‌شوند.



جدول ۱-۱: مراحل مثال واهم‌گشتی

۱-۴ بهبود وضوح تصویر در شبکه عصبی پیچشی

در این قسمت به معرفی روش‌های بهبود وضوح تصویر پیچشی، متد پیچشی سریع، متد پیچشی زیرپیکسل، متد MemNet و متد بسیار عمیق پرداخته می‌شود.

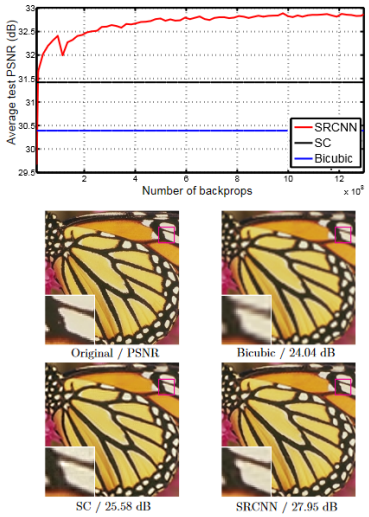
۱-۴-۱ روش بهبود وضوح تصویر شبکه عصبی پیچشی

روش بهبود وضوح تصویر شبکه عصبی پیچشی [۳] که هدفش بهبود وضوح تصویر یک تصویر با وضوح پایین می‌باشد دارای دو ویژگی جذاب در مقایسه با سایر روش‌ها است، اول اینکه ساختار این روش بسیار ساده می‌باشد و دوم اینکه دقت بالاتری را در مقایسه با روش‌های مدرن مبتنی بر مثال ارائه می‌دهد[۱].

همانگونه که در تصویر ۱-۶ نتایج مقایسه مشخص است شبکه عصبی SRCNN تنها با چند تکرار، از روش درون‌یابی دو مکعبی [۴] و روش مبتنی بر کدگذاری اسپارس [۵] بهتر عمل می‌کند.این روش یکی از سریعترین روش‌ها در مقایسه با روش‌های مبتنی بر مثال می‌باشد. آزمایشات نشان می‌دهد که کیفیت بهبود وضوح تصویر در صورت در دسترس بودن داده‌های آموزشی بیشتر و متنوع نتیجه بهبود یابد. این مدل شامل سه لایه می‌باشد، لایه استخراج ویژگی‌ها، لایه شبکه عصبی غیر خطی و لایه بازسازی تصویر بزرگتر. در این روش فیلترها در سایز ۱ × ۹ × ۹ و ۵ × ۵ هستند.

---

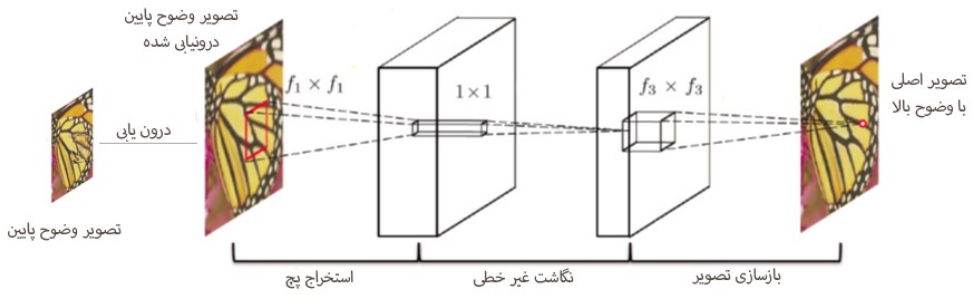
[۱] Stride [۲] Zero Padding [۳] SRCNN (Super Resolution Convolution Neural Network) [۴] Bicubic Interpolation [۵] Sparse Coding



شکل ۱-۶: نتیجه بهبود وضوح تک تصویر پیچشی[۱]

در این روش ابتدا یک مرحله پیش پردازش بر روی یک تصویر با وضوح پایین انجام می‌شود، تصویر با روش درون‌یابی دو مکعبی [۱] به اندازه تصویری که می‌خواهیم وضوح تصویر داشته باشیم تبدیل می‌شود[۹]. این تنها مرحله پیش پردازش در این روش می‌باشد. تصویر پیش پردازش شده را Y می‌نامیم. هدف ما بازیابی $F(Y)$ از روی Y می‌باشد، که تا حد ممکن شبیه تصویر وضوح بالای X خواهد بود. هرچند اندازه Y برابر اندازه تصویر X است اما برای سادگی Y را وضوح پایین می‌نامیم و عملیات پیش پردازش را در نظر نمی‌گیریم. تابع نگاشت F از نظر مفهومی از سه مرحله تشکیل شده است:

- استخراج وصله‌ها[۲]: این عملیات وصله‌ها را از تصویر وضوح پایین Y استخراج می‌کند.
- نگاشت غیر خطی[۳]: در این عملیات یک شبکه عصبی مصنوعی، نگاشت غیر خطی پچ‌های شبکه پیچشی حاصل از عملیات "استخراج وصله‌ها" را بر روی یک تصویر با وضوح اصلی X نگاشت خواهند کرد.
- بازسازی[۴]: در این عملیات بر اساس خطای خروجی برداری عملیات "نگاشات غیر خطی" و ایجاد تصویر اصلی X شبکه آموزش داده می‌شود.

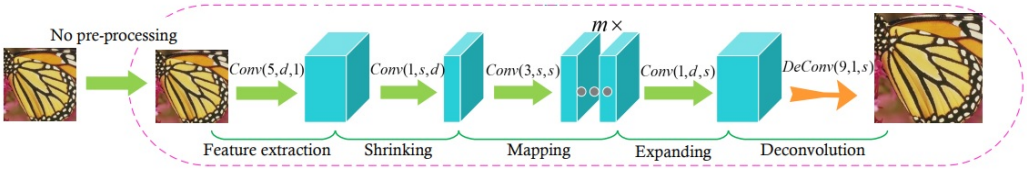


شکل ۱-۷: فرآیند بهبود وضوح تصویر شبکه پیچشی

[۱] Bicubic interpolation [۲] Patch [۳] Non Linear Mapping [۴] Reconstruction

۱-۴-۲ بهبود وضوح تصویر پیچشی سریع

روش بهبود وضوح تصویر پیچشی [۱] عملکرد برتری را از نظر سرعت و بهبود وضوح تصویر از خود به نسبت روش‌های پیشین نشان داده است. با این حال باتوجه به هزینه محاسباتی بالایی که دارد، برای کارهای بلادرنگ روش مناسبی نمی‌باشد. متد بهبود وضوح تصویر پیچشی سریع که FSRCNN نامیده می‌شود با تغییر در قسمت‌هایی از شبکه عصبی SRCNN باعث ایجاد بهبود شده است،ابتدا یک لایه واهَم‌گشت [۲] در انتهای شبکه اضافه می‌شود. با این روش نگاشت مستقیماً از تصویر با وضوح پایین بدون درون‌یابی به تصویر با وضوح بالا آموزش داده می‌شود. دومین تغییر لایه نگاشت با کاهش ابعاد ویژگی‌ها کاهش می‌یابد و سپس ابعاد به ابعاد تصویر ورودی گسترش می‌یابد و سومین تغییر کوچکتر کردن لایه‌های پیچشی و افزایش تعداد آن‌ها می‌باشد. این متد با سرعت اجرای بیشتری توانسته است بهبود وضوح تصویری همانند متد بهبود وضوح تصویر پیچشی دست پیدا کند[۲]. در تصویر ۱-۸ . متد بهبود وضوح تصویر پیچشی سریع مشاهده می‌شود، که به پنج بخش تقسیم شده است. استخراج ویژگی [۳]، کوچک کردن [۴]، نگاشت [۵]، گسترش [۶] و واهم‌گشت پنج بخش تشکیل دهنده شبکه پیچشی فوق هستند. چهار بخش ابتدایی لایه‌های پیچشی و لایه انتهایی لایه واهم‌گشت می‌باشد. برای توضیح بهتر بخش پیچشی را $Conv(f_i, n_i, c_i)$ و لایه واهم‌گشت را به صورت عبارت $DeConv(f_i, n_i, c_i)$ تعریف می‌شود. در عبارت‌های فوق f_i اندازه فیلترها، n_i تعداد فیلترها و c_i تعداد کانال‌ها می‌باشد.



شکل ۱-۸: مدل بهبود وضوح تصویر پیچشی سریع [۲]

در یک شبکه عصبی بهبود وضوح تصویر سریع ۱۰ لایه پیچشی و ۱ لایه واهم‌گشت وجود دارد (مثال: $\{\{f_i, n_i, c_i\}_{i=1}^{6}$)

استخراج ویژگی

این لایه شبکه FSRCNN مشابه با شبکه SRCNN می‌باشد که تنها تفاوتش ورودی آن می‌باشد، در متد FSRCNN استخراج ویژگی بر روی تصویر ورودی بدون انجام درونیابی انجام می‌گردد. در شبکه SRCNN بر روی تصویر درونیابی شده Y پارامتر وضوح تصویر ۱ $f_1, n_1, c-1$ وجود دارد که اندازه فیلتر لایه ابتدایی ۹ تنظیم شده است، در روش FSRCNN بر روی تصویر ورودی که تغییر اندازه پیدا نکرده است که آن Y_s نامیده می‌شود فیلتر با اندازه ۵ $f_1 = 5$ اعمال می‌شود. تعداد کانال‌ها نیز همانند SRCNN مقدار ۱ $c_1 = 1$ می‌باشد.

---

[۱] SRCNN [۲] deconvolution [۳] feature extraction [۴] shrinking [۵] mapping [۶] expanding

کوچک کردن

در شبکه عصبی SRCNN مرحله استخراج ویژگی تصویر وضوح پایین را به تصویر وضوح بالاتر تبدیل می‌کند، به علت زیاد بودن ویژگی‌ها d در تصویر ورودی محاسبات مرحله نگاشت [۱] بسیار زیاد است. این پدیده در بسیاری از وظایف پردازش تصویر نیز مشاهده می‌شود. به این سبب در متد FSRCNN لایه کوچک کردن پس از لایه استخراج ویژگی اضافه شده است که ابعاد تصویر ورودی d را کاهش می‌دهد. در این لایه اندازه فیلتر ۱ در نظر گرفته می‌شود ($f_۲ = ۱$) که باعث می‌شود فیلتر شبیه به یک ترکیب خطی از ویژگی‌های تصویر ورودی پس از استخراج ویژگی‌ها عمل کند. با تعیین تعداد فیلتر $n_۲ = s < d$ ابعاد تصویر وضوح پایین ورودی از ابعاد d به ابعاد s کاهش می‌یابد. و دومین لایه به صورت $Conv(۱, s, d)$ تعریف می‌شود. عملیات پیچشی فوق به میزان قابل توجهی میزان پارامترها و زمان پردازش را کاهش می‌دهد.

نگاشت غیر خطی

مرحله نگاشت غیر خطی مهم‌ترین مرحله در بهبود وضوح تصویر شبکه عصبی FSRCNN می‌باشد. مهمترین عامل تاثیر گذار در این لایه تعداد فیلترها و تعداد لایه‌های نگاشت میباشد. در این لایه $f_۳ = ۳$ مقدار دهی میشود، و تعداد لایه نگاشت که با m نمایش داده می‌شود که مقدارش می‌تواند باعث افزایش دقت و افزایش پیچیدگی شبکه عصبی FSRCNN شود تعیین می‌گردد و تعداد فیلتر های نیز به اندازه ابعاد s که در مرحله کوچک کردن انتخاب شده بود تعیین می‌گردد. بر اساس پارامترهای تعریف شده رابطه نگاشت غیر خطی به صورت $m \times Conv(۳, s, s)$ نوشته می‌شود.

گسترش

گسترش [۲] این لایه معکوس لایه کوچک کردن عمل می‌کند. در لایه کوچک کردن تعداد ویژگی‌های تصویر ورودی وضوح پایین کاهش می‌داد، اگر تصویر وضوح بالا پس از نگاشت غیر خطی ساخته شود نتیجه تصویر خروجی بسیار خوب نخواهد شد. بنابراین لایه گسترش افزوده می‌شود تا ویژگی‌های تصویر را که از مرحله نگاشت غیرخطی وارد شده است را به اندازه واقعی تصویر ورودی تبدیل نماید. مشخصات لایه کوچک کننده $Conv(۱, s, d)$ بود که در این مرحله گسترش معکوس آن $Conv(۱, d, s)$ می‌شود.

واهم‌گشت

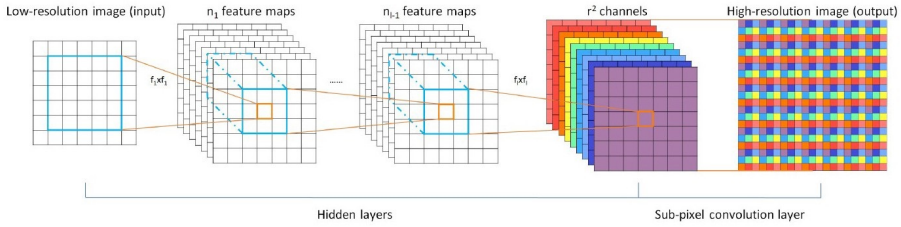
واهم‌گشت لایه انتهایی شبکه بهبود وضوح تصویر در روش FSRCNN است که ویژگی‌های قبلی را با مجموعه‌ای از فیلترهای واهم‌گشت نمونه‌برداری و جمع‌آوری می‌کند. واهم‌گشت را می توان به عنوان یک عمل معکوس پیچش در نظر

[۱] Mapping [۲] Expnding

گرفت. در عملود پیچش فیلتر با تصویر با گام k پیچش می‌شود و خروجی $1/k$ برابر ورودی است و در واهم‌گشت با گام k خروجی k برابر ورودی است. از این ویژگی واهم‌گشت برای تعیین میزان k استفاده می‌شود و $k = n$ تعیین می‌شود که به آن فاکتور بزرگ کننده تصویر اتلاق می‌شود که خروجی این لایه همان تصویر بهبود وضوح یافته می‌باشد.

۱-۴-۳ روش بهبود وضوح تصویر زیر پیکسل

رویکردهای مبتنی بر شبکه عصبی کانولوشن مانند SRCNN، FSRCNN و VDSR دارای اشکالاتی هستند. رویکردهای بهبود وضوح تصویر CNN نیاز به استفاده از روش های درون‌یابی همانند درون‌یابی دو مکعبی برای نمونه برداری مجدد از تصویر y را دارند. به منظور حل مشکلات، در متد بهبود وضوح تصویر زیر پیکسل که به صورت خلاصه SubPixel نامیده می‌شود، افزودن یک لایه پیچشی زیرپیکسلی[۱] به شبکه CNN پیشنهاد شده است. در متد SubPixel اندازه تصویر در انتهای شبکه افزایش می‌یابد. به این معنی که تصویر وضوح پایین با اندازه کوچکتر مستقیماً به شبکه داده می‌شود. بنابراین نیازی به استفاده از روش درونیابی نیست. با توجه به کاهش اندازه تصویر ورودی، می توان از اندازه فیلتر کوچکتر برای استخراج ویژگی‌ها استفاده کرد، که باعث می‌شود پیچیدگی محاسباتی و هزینه حافظه کاهش یابد، به طوری که می توان کارایی را تا حد زیادی افزایش داد.



شکل ۱-۹: مدل بهبود وضوح تصویر شبکه‌های بسیار عمیق SubPixel [۳]

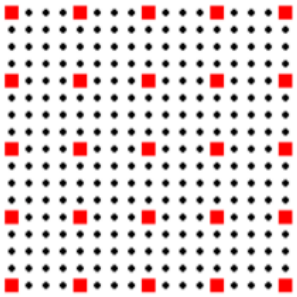
ساختار شبکه SubPixel را می توان در تصویر ۱-۹ مشاهده کرد. فرض کنید، L لایه برای شبکه وجود دارد، از لایه اول تا $L - 1$، لایه‌های پیچشی هستند که نقشه ویژگی تصویر وضوح پایین ورودی را به دست می‌آورند. و آخرین لایه شبکه SubPixel لایه پیچشی زیرپیکسلی برای بازیابی اندازه تصویر خروجی بر اساس فاکتور بزرگ کننده مشخص شده است.

پیچش زیر پیکسل

یکی از مهمترین مفاهیمی که توسط توسعه دهندگان متد SubPixel ارائه شده است، پیچش زیرپیکسلی است که به آن تغییر پیکسل [۲] نیز گفته می‌شود. در سیستم تصویربرداری دوربین‌ها، قبل از ذخیره شدن تصویر توسط دوربین، پردازش گسسته سازی[۳] بر روی تصویر اعمال می‌شود. با توجه به محدودیت سنسورهای نوری، وضوح تصاویر، به وضوح پیکسل‌های

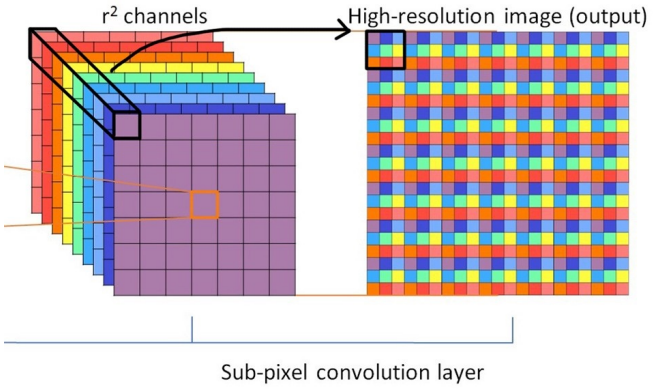
[۱] SubPixel [۲] pixel shuffle [۳] discretized processing

دوربین محدود می‌شوند، به عبارت دیگر، هر پیکسل از تصاویر نشان دهنده ناحیه کوچکی از رنگ در دنیای واقعی هستند. در تصاویر دیجیتالی که مشاهده می‌کنیم، پیکسل‌ها به هم متصل مشاهده می‌شوند، در حالی که در دنیای میکروسکوپی و دنیای واقعی تعداد پیکسل‌های ریز بین دو پیکسل فیزیکی وجود دارد. که به آن پیکسل‌های کوچک، پیکسل‌های فرعی گفته می‌شود.



شکل ۱-۱۰: تصویر SubPixel

همانطور که در تصویر ۱-۱۰ مشخص است، هر ناحیه مربعی با چهار مربع قرمز بزرگتر احاطه شده است، اگر هر کدام از این مربع‌های بزرگتر را پیکسل‌های دوربین در نظر بگیریم، نقاط سیاه کوچکتر زیر پیکسل‌ها هستند. دقت زیر پیکسل‌ها را می توان با روش‌های درون‌یابی بین پیکسل‌های مجاور تنظیم کرد. حال در روش SubPixel همانطور که در تصویر ۱-۱۱ نشان داده شده است، ترکیب هر پیکسل از نقشه‌های ویژگی چند کاناله لایه انتهایی، در یک منطقه مربع $r \times r$ در تصویر خروجی ایجاد می‌گردد. بنابراین، هر پیکسل نقشه‌های ویژگی، معادل پیکسل فرعی در تصویر خروجی تولید شده است.



شکل ۱-۱۱: لایه انتهایی شبکه SubPixel

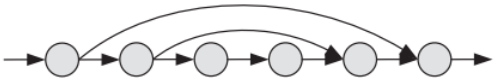
پیچش زیرپیکسل شامل دو فرآیند است: یک عملیات پیچش و به دنبال آن چینش مجدد پیکسل‌ها. کانال خروجی آخرین لایه باید دارای ابعاد $C \times r \times r$ باشد تا تعداد کل پیکسل‌ها با تصویر هدف که به دست می‌آید مطابقت داشته باشد.

۱-۴-۴ روش بهبود وضوح تصویر MemNet

در این روش یک شبکه حافظه پایدار بسیار عمیق (MemNet) توسط نانجینگ و همکاران [۱۰] معرفی شده است. ساختار شبکه‌های قبلی وضوح تصویر همانند تصاویر ۱-۱۲ و ۱-۱۳ هستند که یا به صورت شبکه‌های خطی و یا به صورت شبکه‌های باقی‌مانده طراحی شده‌اند. شبکه بهبود وضوح MemNet حاصل ترکیب روش‌های خطی و باقی‌مانده می‌باشد که از سه بخش تشکیل شده است:

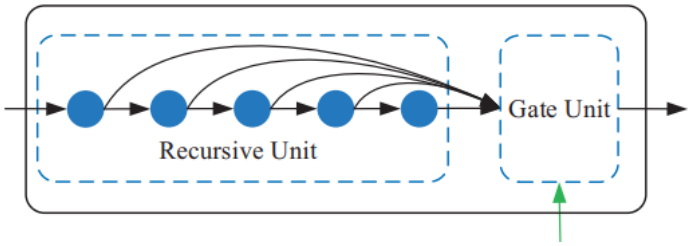
- **لایه پیچش:** لایه پیچش برای استخراج ویژگی‌ها از تصویر ورودی است.
- **بلوک‌های حافظه:** در این لایه M بلوک حافظه روی هم چیده می‌شوند.
- **بازسازی:** یک لایه پیچشی برای بازسازی تصویر نهایی است.

شکل ۱-۱۲: فرآیند شبکه عصبی بهبود وضوح تصویر



شکل ۱-۱۳: فرآیند شبکه عصبی بهبود وضوح تصویر باقی‌مانده

تصویر ۱-۱۴ نمایش دهنده بلوک حافظه شامل یک واحد بازگشتی و یک واحد دروازه است. که واحد بازگشتی[۱] یک شبکه عصبی باقی‌مانده است که خروجی‌اش ورودی واحد دروازه[۲] است.



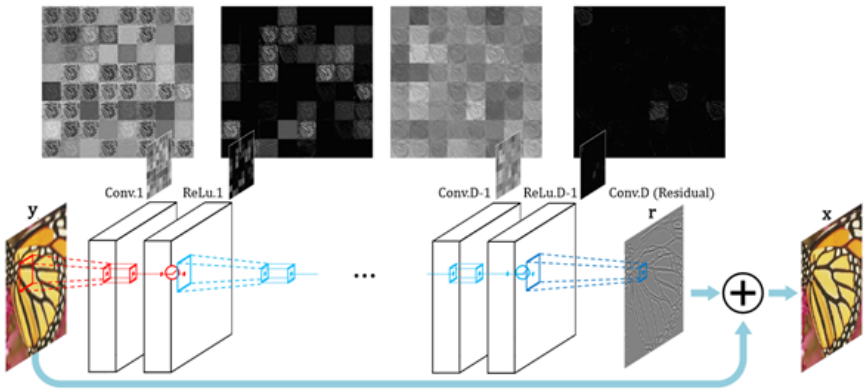
شکل ۱-۱۴: فرآیند شبکه عصبی بهبود وضوح تصویر MemNet

۱-۴-۵ روش بهبود وضوح تصویر بسیار عمیق

در طراحی متد بهبود وضوح‌تصویر شبکه عصبی بسیار عمیق[۳] که به صورت خلاصه VSRD نامیده میشود، پایه طراحی‌اش افزایش تعداد لایه‌های شبکه عصبی و شبکه‌های باقی‌مانده[۴] است [۴]. در مقایسه با شبکه عصبی SRCNN که دارای ۳

[۱] Recursive Unit [۲] Gate Unit [۳] Very Deep Super Resolution [۴] Residual

لایه می‌باشد در طراحی این شبکه از شبکه باقی‌مانده استفاده شده است، این شبکه بهبود وضوح با کمک طراحی باقی‌مانده توانسته است شبکه بهبود وضوح تصویر خود را تا ۲۰ لایه گسترش دهد. دونگ و همکارانش [۱] ارائه دهنده روش SRCNN سعی کرده‌اند که تعداد لایه‌های شبکه عصبی را افزایش دهند و پس از یک هفته آموزش شبکه عصبی با افزایش تعداد لایه‌های شبکی عصبی پیچشی SRCNN نتوانستند به نتایج بهبود یافته‌ای دست پیدا کنند.[۱] در شبکه بهبود وضوح تصویر VDSR از روش ارائه شده توسط سیمونیان و زیسرمن [۱۱] استفاده شده است. پیکربندی پیشنهادی شبکه عصبی پیشنهادی سیمونیان در تصویر ۱-۱۵ مشاهده می‌شود. در این شبکه از تعداد d لایه استفاده شده است، که به جز لایه اول و لایه آخر شبکه همگی دارای پیکربندی تعداد ۶۴ فیلتر به انداز $۳ \times ۳ \times ۶۴$ می‌باشند. ورودی شبکه همانند شبکه SRCNN یک تصویر درون‌یابی شده به اندازه تصویر نهایی می‌باشد.



شکل ۱-۱۵: مدل بهبود وضوح تصویر شبکه‌های بسیار عمیق VDSR [۴]

یکی از مشکلات استفاده از یک شبکه بسیار عمیق برای پیش بینی خروجی های متراکم این است که هر بار که عملیات پیچش اعمال می‌شود، اندازه نقشه ویژگی کاهش می‌یابد. برای رفع این مشکل، قبل از عملیات پیچش [۱] مقدار Padding صفر در هر مرحله به تصاویر افزوده می‌شود تا خروجی تصویری که فیلتر بر روی آن اعمال شده است از ابتدا تا انتها شبکه به اندازه تصویر خروجی X باشند. اعمال Padding صفر در هر مرحله جهت حفظ اندازه تصویر نیز یکی از تفاوت‌های این مدل با سایر مدل‌ها است.

[۱] Convolution

فصل ۲

شبکه عصبی عمیق

۲-۱ شبکه عصبی چیست

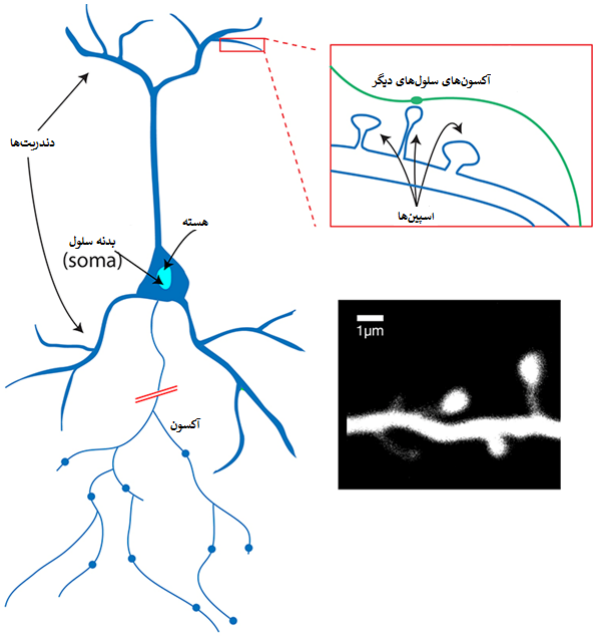
در طبیعت نورون‌ها[۲] (که به آن‌ها سلول‌های عصبی [۳] نیز گفته می‌شود) واحدهای سازنده مغز و سیستم عصبی موجودات زنده هستند، سلول‌هایی که وظیفه دریافت ورودی حسی از دنیای خارج، ارسال دستورات حرکتی به ماهیچه‌های موجودات زنده و تغییر و انتقال سیگنالهای الکتریکی در هر قسمت از کالبد موجودات را بر عهده دارند. تعاملات آن‌ها مشخص می‌کند که ما به عنوان یک انسان چه کسی هستیم. با این وجود، تقریباً ۱۰۰ میلیارد سلول عصبی انسان با سایر انواع سلول‌ها، که به طور کلی به عنوان گلیا [۴] طبقه‌بندی می‌شوند، ارتباط نزدیکی برقرار می‌کنند (ممکن است این تعداد در واقع از تعداد سلول های عصبی بیشتر باشد، هرچند واقعاً تعدادشان مشخص نیست).

یک مثال مفید برای فهمیدن نورون این است که یک نورون را یک درخت تصور کنیم. یک نورون دارای سه قسمت اصلی است: دندریت‌ها [۵]، آکسون [۶] و بدنه سلولی [۷]، همانگونه که در تصویر ۲-۱ مشاهده می‌شود، به ترتیب اجزاء تشکیل دهنده می‌توانند به عنوان شاخه، ریشه و تنه یک درخت نشان داده شوند. دندریت (شاخه درخت) جایی است که نورون ورودی از سلول های دیگر را دریافت می کند. دندریت‌ها همانطور که شاخه‌های درختان به سمت نوک خود حرکت می کنند منشعب می شوند و حتی ساختارهای برگ مانند روی آن‌ها وجود دارد که به آنها اسپین [۸] گفته می‌شود.

آکسون (ریشه‌های درخت) ساختار خروجی نورون‌ها هستند. هنگامی که یک نورون می‌خواهد با نورون دیگری صحبت کند، یک پیام الکتریکی به نام پتانسیل عملی [۹] در کل آکسون ارسال می کند. سوما (تنه درخت) جایی است که هسته سلول قرار دارد، DNA نرون در آنجا قرار دارد و پروتئین‌هایی برای انتقال در سراسر آکسون و دندریت‌ها ساخته می‌شوند.

---

[۲] Neuron [۳] Network Neural [۴] glia [۵] dendrites [۶] axon [۷] Body Cell [۸] Spines [۹] potential action



شکل ۲-۱: تصویر مدل سلول عصبی، ساختار درخت مانند یک نورون را نمایش می‌دهد. اسپین‌های دندریت ساختارهای کوچکی هستند که ورودی‌ها را از آکسون‌های دیگر نورون‌ها دریافت می‌کنند. تصویر سمت راست پایین: قسمتی از دندریت که ستون فقرات از آن منشعب می شود، مانند برگ‌های شاخه درخت است. به اندازه بسیار کوچک (تقریبا ۰.۰۰۱ میلی‌متر) توجه کنید.

۲-۲ شبکه عصبی سیستم بینایی

504192

شکل ۲-۲: تصویر دست خط اعداد

سیستم بینایی انسان یکی از شگفتی های جهان است. دنباله ارقام دست نویس تصویر ۲-۲ را در نظر بگیرید، اکثر انسان‌ها بدون هیچ زحمتی و با کمترین خطای ممکن اعداد تصویر ۲-۲ را ۵۰۴۱۹۲ تشخیص می‌دهند. این سهولت تشخص انسان‌ها فریبنده است. در مغز ما انسان‌ها هر نیمکره مغز دارای یک قشر بینایی اولیه است که با نام V1 شناخته می‌شود و شامل ۱۴۰ میلیون نرون عصبی است و ده ها میلیارد ارتباط بین این نرون‌ها وجود دارد. سیستم بینایی انسان فقط شامل کورتکس V1 نمی‌باشد، بلکه سیستم بینایی انسان مجموعه‌ای کامل از قشرهای بصری - V2، V3، V4 و V5 - را نیز شامل می‌شود که پردازش‌های تصویری پیچیده را برای انسان‌ها انجام می‌دهند. انسان‌ها یک ابر رایانه در سر دارند که طی صدها میلیون سال توسط تکامل ایجاد شده است و برای درک جهان و بقا انسان سازگار شده است. تشخیص ارقام دست نویس آسان نیست. انسان‌ها در درک آنچه چشم‌هایشان به آن‌ها نشان می‌دهد به طرز شگفت انگیز و حیرت آوری خوب عمل می‌کنند. تقریباً تمام این کارها ناخودآگاه انجام می‌شود و انسان‌ها معمولاً درک نمی‌کنند که سیستم بصری مغز چه مشکل بزرگ و سختی را در هر لحظه در حال حل کردن است[۱۲].

سختی تشخصی الگوهای بصری زمانی آشکار می‌شود که قصد شود برنامه‌ای نوشته شود که ارقامی مانند تصویر۲-۲

را تشخیص دهد. به طور مثال درباره چگونگی تشخص عدد ۹ در تصویر ۲-۲، ما یک داره تو خالی در بالا و یک خط صاف و عمودی در سمت راست دایره فوق داریم. برای بیان الگوریتم مسئله، و توسعه برنامه‌ای که عدد ۹ را تشخیص بدهد مسیر ساده‌ای پیش رو نخواهد بود و در بیان الگوریتم به یقین به تعداد زیادی از استثنائات و مشکلات روبرو خواهیم شد.

شبکه‌های عصبی با این مسئله‌ها به گونه‌ای دیگر رفتار می‌کند. برای آموزش شبکه‌های عصبی تعداد زیادی نمونه رقم‌های دست‌نویس تهیه می‌شود، داده‌های گردآوری شده داده‌های آموزشی نامیده می‌شوند. سپس سیستمی طراحی می‌شود که از داده‌های آموزشی، تشخیص اعداد را بیاموزد. به عبارت دیگر، شبکه عصبی از داده‌های آموزشی برای استنباط خودکار قوانین تشخیص ارقام دست نویس استفاده می‌کند. علاوه بر این، با افزایش تعداد نمونه‌های آموزشی، شبکه عصبی می‌تواند درباره تشخیص دست خط بیشتر بیاموزد و دقت خود را بهبود بخشد. در تصویر ۲-۳ فقط ۱۰۰ داده آموزشی نشان داده شده است، شاید بتوانیم با استفاده از هزاران یا حتی میلیون‌ها یا میلیاردها مثال آموزشی، یک تشخیص دهنده دست خط بهتر ساخته شود.

شکل ۲-۳: تعدادی از نمونه‌های آموزشی دست خط ارقام

۲-۳ تاریخچه یادگیری عمیق

یادگیری عمیق، دارای تاریخی طولانی است، دیدگاه‌هایی با نام‌گذاری‌های متفاوتی در طول تاریخ برای تکامل‌اش رشد داده شده‌اند که در طی زمان از بین رفته‌اند. یادگیری عمیق از زمانی که داده‌های آموزشی، سخت افزارها و نرم‌افزارهای در دسترس رشد پیدا کرد بسیار کاربردی شده است و توانسته است حل مسائل پیچیده را در حوزه‌های کاربردی مختلف را بهبود ببخشد. تاریخ پیدایش یادگیری عمیق به دهه ۱۹۴۰ باز می‌گردد و ظاهرا جدید به نظر می‌رسد زیرا طی سالیان قبل به علت محدودیت‌های موجود استفاده از آن غیر معمول بوده است، از طرفی به نام‌های متفاوتی نام گذاری شده بوده است و اخیرا "یادگیری عمیق" نامیده می‌شود. بعضی از اولین الگوریتم‌های یادگیری شبکه‌های عصبی که امروزه نیز معتبر هستند مدلی

از سیستم یادگیری بیولوژیکی هستند، همانند آنچه در سیستم یادگیری مغز موجودات زنده رخ می‌دهد، به همین دلیل یکی از نام‌هایی که یادگیری عمیق به آن شناخته می‌شود شبکه‌های عصبی است. دیدگاه مربوطه در مورد مدل‌های یادگیری عمیق این است که آن‌ها سیستم‌هایی هستند که از مغز بیولوژیکی الهام‌گرفته شده (چه مغز انسان و چه مغز یک حیوان دیگر). در حالی که انواع شبکه‌های عصبی مورد استفاده برای یادگیری ماشین، گاهی اوقات برای درک عملکرد مغز مورد استفاده قرار می‌گیرند[۱۳].

بر اساس واژه یابی انجام شده در کتابخانه آنلاین گوگل می‌توان سه موج از تحقیقات انجام شده روی موضوعات شبکه عصبی را مشاهده کرد. در ابتدای موج تحقیقات واژه سایبرنتیک [۱] مورد استفاده قرار می‌گرفته است و پژوهش‌های این حوزه از واژه سایبرنتیک استفاده می‌کرده‌اند، موج دوم با عبارت اتصالات [۲] و موج سوم که مربوط به زمان حال است شبکه عصبی [۳] نامیده می‌شود. اولین موج سایبرنتیک از دهه ۱۹۴۰ تا دهه ۱۹۶۰ با توسعه تئوری‌های یادگیری بیولوژیکی و پیاده‌سازی اولین مدل‌های یادگیری بیولوژیکی همانند پرسپترون [۱۴] می‌توانند یک تک نرون آموزش ببینند. دومین موج با رویکرد اتصالات در دوره ۱۹۹۵-۱۹۸۰ با الگوریتم پس انتشار خطا [۱۵] برای آموزش شبکه عصبی با یک یا دو لایه مخفی شروع شده است. موج فعلی و سومین موج، یادگیری عمیق حدود سال ۲۰۰۶ [۱۶] شروع شده است. اولین پیشگامان یادگیری عمیق مدل‌های خطی ساده بودند که از دیدگاه عصب‌شناختی الهام گرفته شده بودند. این مدل‌ها طراحی شده بودند تا یک مجموعه n ورودی از مقادی $x_1, \ldots, x_n$ را از ورودی گرفته و به یک خروجی y مرتبط کند. این مدل‌ها وزن‌های $w_1, ..., w_n$ را یاد می‌گیرند و خروجی $f(x, w) = w_1 x_1 + ... + w_n x_n$ را محاسبه می‌کنند.

نرون‌های پیت و مک‌کلاچ اولین مدل تابعی مغز بودند. این مدل خطی می‌توانست دو دسته بندی از ورودی مختلف را با چک کردن مثبت یا منفی بودن تابع $f(x, w)$ را شناسایی کند. البته برای اینکه مدل به تعریف مطلوب از دسته بندی‌ها برسد، وزن‌ها باید درست تنظیم شوند. این وزن‌ها توسط کاربر انسانی تعیین می‌شود. در دهه ۱۹۵۰ پرسپترون [۱۴] اولین مدلی بود که میتوانست مدل‌ها را یاد بگیرد و داده‌های ورودی را دسته‌بندی نماید. عنصر خطی انطباقی که در همان دهه ۱۹۵۰ معرفی شد با سادگی با پیش‌بینی مقدار $f(x)$ را برگشت می‌دهد [۱۷] و می‌تواند یادبگیرد اعداد را از روی داده‌ها پیش‌بینی نماید. این چند الگوریتم ساده تا حد زیادی بر چشم انداز مدرن یادگیری ماشین تاثیر گذار هستند.

---

[۱] Cybernetic [۲] Coonection [۳] Neural Network

۲-۴ شبکه عصبی مصنوعی

(ANNs)[1] یا به زبان ساده‌تر شبکه‌های عصبی، روش‌های محاسباتی جدید برای یادگیری ماشین، نمایش دانش و اعمال دانش به دست آمده در پیش‌بینی پاسخ‌های خروجی سامانه‌های پیچیده می‌باشند. در حقیقت، هدف از ایجاد یک شبکه عصبی نرم‌افزاری، بیش از آن که شبیه‌سازی مغز انسان باشد، ایجاد مکانیسمی برای حل مسائل مهندسی با الهام از الگوی رفتاری شبکه‌های بیولوژیک است. تاکنون مدل‌های مختلف با ساختار و الگوریتم‌های متنوعی از شبکه‌های عصبی ارائه شده است و هر چند این مدل‌ها با یکدیگر تفاوت دارند، اما تمام این مدل‌ها یک هدف مشترک را دنبال می‌کنند. یک شبکه عصبی از اجزای ساده (نورون) و زنجیره‌ای تشکیل می‌شود و دارای ویژگی‌های زیر است

- مقاوم بودن
- قابلیت تقریب سراسری
- پردازش موازی
- قابلیت یادگیری و تطبیق پذیری
- قابلیت تعمیم پذیری

قابلیت یادگیری و تطبیق پذیری قابلیت یادگیری یعنی توانایی تنظیم پارامترهای شبکه عصبی. برای این منظور نمونه‌های اولیه را به شبکه اعمال می‌کنند، شبکه پارامترها را براساس این نمونه‌ها تنظیم می‌کند. اگر نمونه‌های جدید به این شبکه که به این طریق آموزش دیده اعمال شوند، خروجی مناسب را با درصد خطای کوچک می‌توان به دست آورد. به این ترتیب شبکه‌های عصبی می‌توانند با تغییر شرایط به صورت هوشمندانه خود را تطبیق یا اصلاح نمایند.

قابلیت تعمیم پذیری: پس از آن که نمونه‌های اولیه به شبکه آموزش داده شد، شبکه می‌تواند در مقابل ورودی‌های آموزش داده نشده (ورودی‌های جدید) قرار گیرد و یک خروجی مناسب تولید نماید. این خروجی براساس مکانیسم تعمیم، که چیزی جز فرآیند درون‌یابی نیست به دست می‌آید.

پردازش موازی: هنگامی که شبکه عصبی در قالب سخت افزار پیاده‌سازی می‌شود نورون‌هایی که در یک لایه قرار می‌گیرند می‌توانند به طور همزمان به ورودی‌های آن لایه پاسخ دهند. این ویژگی باعث افزایش سرعت پردازش می‌شود. در واقع در چنین سیستمی وظیفه کلی پردازش، بین پردازنده‌های کوچکتر مستقل از یکدیگر توزیع می‌گردد.

مقاوم بودن: در یک شبکه عصبی هر نورون به طور مستقل عمل می‌کند و رفتار کلی شبکه برآیند رفتارهای محلی نورون‌های متعدد است. این ویژگی باعث می‌شود تا خطاهای محلی از چشم خروجی نهایی به دور بماند. به عبارت دیگر نورون‌ها در یک روند همکاری خطاهای محلی یکدیگر را تصحیح می‌کنند. این خصوصیت باعث افزایش مقاوم بودن در سیستم می‌گردد.

---

[1] Artificial Neural Networks (ANNs)

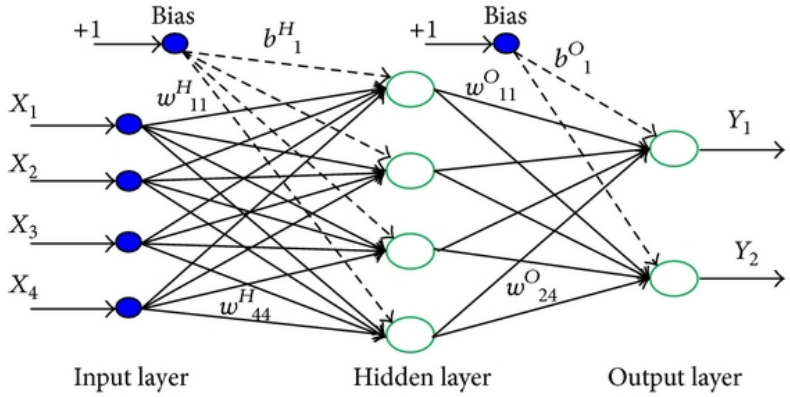
قابلیت تقریب سراسری[1]: شبکه‌های عصبی با یک یا چندلایه پنهان به شرط آن که لایه‌ها تعداد کافی نورون‌های پنهان داشته باشند، می‌توانند هر تابع غیرخطی قطعه به قطعه پیوسته [2] را تخمین بزنند.

اجزای شبکه عصبی : ورودی و خروجی، نورون، وزن و توابع فعالیت[3] (توابع انتقال) هستند.

۲-۴-۱ سبک‌های معماری شبکه عصبی

طرح اتصالات بین نورون‌ها در یک شبکه عصبی به سبک معماری شبکه عصبی معروف است. از حیث سبک معماری، انواع مختلفی از شبکه‌های عصبی وجود دارند که در یک طبقه‌بندی کلی به مدل‌های ایستا و پویا تقسیم می‌شوند. در مدل‌های ایستا مسیر پردازش اطلاعات از لایه ورودی به لایه خروجی است، بدون این که بازگشتی در سیستم ارتباطی لایه‌ها وجود داشته باشد؛ در حالی که در مدل‌های پویا مسیرهای بازگشتی از لایه خروجی یا لایه‌های میانی به لایه ورودی نیز وجود دارد. شبکه‌های ایستا را شبکه‌های «پیشخور» و شبکه‌های پویا را شبکه‌های «برگشتی» یا «پسخور» نیز می‌گویند. بنابراین تفاوت شبکه‌های پسخور با شبکه‌های پیشخور در این است که در شبکه‌های پسخور حداقل یک سیگنال برگشتی از یک نورون به همان نورون یا نورون‌های همان لایه یا لایه قبل وجود دارد

شبکه‌های عصبی پیشخور در طول چند دهه گذشته بسیار مورد مطالعه قرار گرفته‌اند. شبکه‌های عصبی پیشخور دارای یک یا چندلایه پنهان‌اند. شبکه‌های عصبی پیشخور تک لایه پنهان یکی از مدل‌های شبکه عصبی بسیار محبوب دارای ساختاری ساده شامل یک لایه ورودی، یک لایه پنهان و یک لایه خروجی هستند. در شکل ۲-۵ یک شبکه عصبی پیشخور تک لایه پنهان دیده می‌شود.



شکل ۲-۴: شمای کلی یک شبکه عصبی پیشخور تک لایه پنهان

در یک شبکه عصبی چندلایه پنهان، هر لایه ماتریس وزن ورودی (**W**) ، بردار بایاس (**b**) خود را دارد. لایه‌های مختلف می‌توانند تعداد نورون‌های متفاوتی داشته باشند. نورون‌های لایه ورودی صرفاً وظیفه توزیع مقادیر ورودی به لایه بعدی را

[2] nonlinear piecewise continuous function [3] activation functions

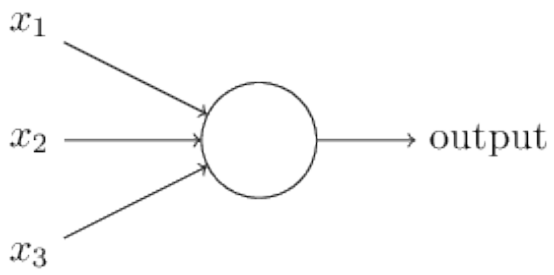
برعهده دارند و بنابراین هیچ محاسبه‌ای را انجام نمی‌دهند. لایه‌ای که خروجی آن خروجی شبکه است، لایه خروجی نامیده می‌شود. لایه‌های دیگر لایه‌های پنهان نام دارند. شبکه‌های چندلایه قدرتمندتر از شبکه‌های تک لایه هستند. برای نمونه، یک شبکه با دو لایه پنهان که شامل لایه اول سیگموئید و لایه دوم خطی می‌باشد، می‌تواند به منظور تقریب اکثر توابع اختیاری آموزش داده شود. بیشتر شبکه‌های کاربردی تنها دو یا سه لایه دارند.

۲-۴-۲ پرسپترون

در ابتدا قبل از توضیح درباره شبکه‌های عصبی عمیق، با اولین گونه ارائه شده از نرون‌های مصنوعی که پرسپترون [۱] نامیده می‌شود آشنا می‌شویم. امروزه گونه‌های دیگری از نرون‌ها مصنوعی استفاده می‌شود، نرون عصبی که امروزه در شبکه‌های عصبی کاربرد دارد نرون‌های سیگمئودی [۲] است، برای درک ساخته شدن نرون‌های سیگمئود باید از نحوه کارکرد پرسپترون‌ها آگاهی داشت.

چگونه یک پرسپترون کار می‌کند

یک پرسپترون چندین ورودی دودویی [۳] $x_1, x_2, \ldots$ و یک خروجی دودویی دارد.



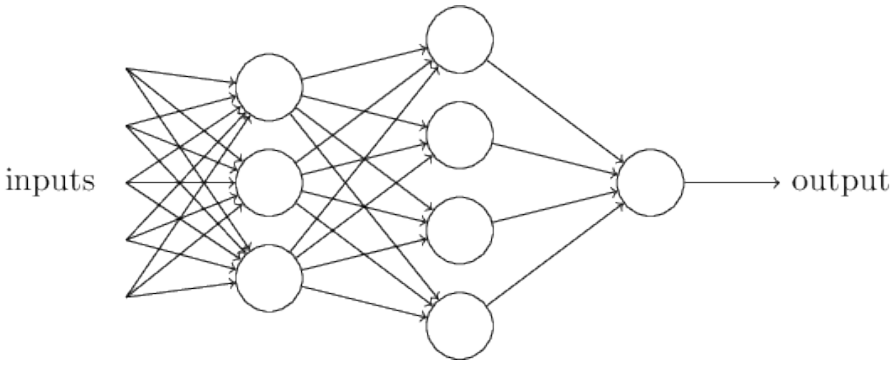
شکل ۲-۵: نمای مدل اصلی نورون

در مثال تصویر ۲-۵ پرسپترون با سه ورودی x_1, x_2, x_3 دودویی ترسیم شده است. به طور کلی این تعداد می‌تواند کمتر و یا بیشتر باشد. روزنبلات یک قانون ساده برای نحوه محاسبه خروجی پیشنهاد کرده است. او وزن‌های $w_1, w_2, \ldots$ که اعداد حقیقی هستند را معرفی می‌کند. اعداد حقیقی که بیان کننده اهمیت ورودی‌های مربوط به خروجی می‌باشد. خروجی‌های نرون‌ها ۰ و یا ۱ می‌باشد که خروجی جمع حاصل‌ضرب وزن‌ها در ورودی‌ها ($\sum_j w_j x_j$) در مقایسه جبری با یک مقدار آستانه [۴] می‌باشد. مقدار آستانه عددی جبری جزء پارامترهای نرون است. فرم جبری توضیح فوق به فرم ۲-۱ می‌باشد.

---
[۱] Perceptron [۲] Sigmoid Neuron [۳] binary [۴] threshold

$$\text{خروجی پرسپترون} = \begin{cases} ۰ & \text{اگر } \sum_j w_j x_j \leq \text{آستانه} \\ ۱ & \text{اگر } \sum_j w_j x_j > \text{آستانه} \end{cases} \tag{۱-۲}$$

این تمام چیزی است که درباره نحوه کارکردن پرسپترون وجود دارد. بدیهی است که پرسپترون یک مدل کامل از سیستم تصمیم‌گیری مغز موجودات زنده نمی‌باشد اما منطقی به نظر می‌رسد که یک شبکه پیچیده از پرسپترون بتواند تصمیمات، با خطاهای کمتری را بگیرد.



شکل ۲-۶: نمای مدل چند لایه پرسپترون

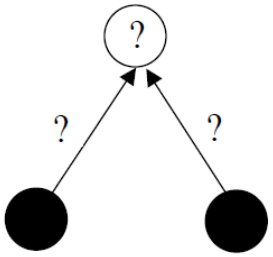
در این شبکه (تصویر ۲-۶)، اولین ستون پرسپترون‌ها - که ما آن را لایه اول پرسپترون‌ها می‌نامیم - با سنجش شواهد ورودی، سه تصمیم می‌گیرد. در مورد پرسپترون‌های لایه دوم هر یک از این پرسپترون‌ها نیز با سنجش نتایج حاصل از لایه اول، تصمیم‌گیری می‌کنند. به این ترتیب پرسپترون در لایه دوم می‌تواند در سطحی پیچیده‌تر و انتزاعی‌تری از پرسپترون در لایه اول تصمیم‌گیری کند. و حتی تصمیمات پیچیده‌تری می تواند توسط پرسپترون در لایه سوم اتخاذ شود. به این ترتیب، یک شبکه چند لایه از پرسپترون می‌تواند در تصمیم‌گیری پیچیده مشارکت داشته باشد. زمانی که پرسپترون‌ها تعریف شدند، بیان شد که پرسپترون فقط یک خروجی دارد. در شبکه فوق پرسپترون‌ها به نظر می‌رسد که دارای چندین خروجی هستند. در حقیقت، آن‌ها هنوز تک خروجی هستند. پیکان‌های خروجی چندگانه فقط یک راه برای نشان دادن این است که خروجی از پرسپترون به عنوان ورودی چندین پرسپترون دیگر استفاده می شود. این کار سخت‌تر از رسم یک خط خروجی واحد است که سپس تقسیم می‌شود. جهت ساده سازی عبارت $\sum_j w_j x_j > \text{threshold}$، اول عبارت $\sum_j w_j x_j$ را می‌توان به فرم حاصل‌ضرب نقطه‌ای $w \cdot x \equiv \sum_j w_j x_j$ بیان کرد که در اینجا x و w به ترتیب بردارهای ورودی‌ها و وزن‌ها هستند. و دومین تغییر جابه‌جا کردن مقدار آستانه و بردنش به طرف دیگر نامساوی می‌باشد. و نام آن را به نام بایاس نرون نیز تغییر می‌دهیم $b \equiv -\text{threshold}$ با انجام این دو تغییر، قوانین پرسپترون به عبارت ۲-۲ بازنویسی می‌شوند.

$$\text{output} = \begin{cases} ۰ & \text{if } w \cdot x + b \leq ۰ \\ ۱ & \text{if } w \cdot x + b > ۰ \end{cases} \qquad (۲-۲)$$

در نظر بگیرید که پرسپترون به دنبال خطی است که طبقه‌ها را تفکیک کند. پرسپترون به راحتی می‌تواند طبقاتی را که در دوسوی خط قرار دارند تفکیک کند، مسائل فراوانی هستند که جدایی دسته‌ها بسیار پیچیده‌تر می‌باشند، همانند مسئله یای حذفی[۱](XOR) ، همان طور که در شکل ۲-۷ دیده می‌شود پرسپترون باید درنهایت یاد بگیرد که اگر ورودی اول ($in_۱$) فعال و ورودی دوم ($in_۲$) خاموش و یا به عکس $in_۲$ فعال و $in_۱$ خاموش بود جواب یک بدهد و اگر هر دو فعال یا خاموش بودند جواب صفر بدهد.

XOR

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0



شکل ۲-۷: جدول تابع یای حذفی

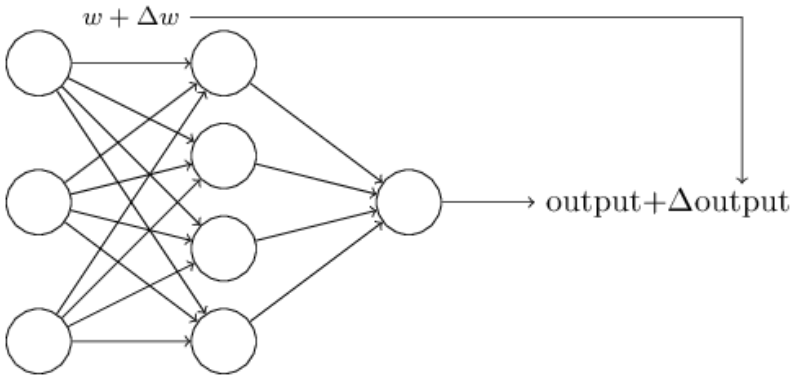
پرسپترون تک لایه‌ای با وجود سادگی مدل، هیچ مسئله جدایی ناپذیر خطی را نمی‌تواند حل کند. رفع این نیاز برای نخستین بار در سال ۱۹۸۶ توجه محافل علمی را به خود جلب کرد، زمانی که رومل هارت[۲] و مک کللند[۳] صورت جدیدی از مدل پرسپترون را به نام پرسپترون چندلایه‌ای[۴] که به صورت لایه‌ای منظم شده بود و سه لایه (ورودی، پنهان و خروجی) داشت را معرفی کردند. هر نورون در لایه پنهان و لایه خروجی مانند یک پرسپترون عمل می‌کند. لازم است که از تابع آستانه‌ای غیرخطی استفاده شود زیرا اگر همه لایه‌های پرسپترون از توابع خطی استفاده کنند توان آن‌ها از یک مدل پرسپترون تک لایه‌ای فراتر نمی‌رود. دلیل آن این است که هر لایه پرسپترون صرفاً عملیاتی خطی بر ورودی‌های خود انجام می‌دهد و عملیات خطی نهایتاً می‌توانند با هم به صورت یک عملیات واحد ترکیب شوند.

۲-۴-۳ نورون‌های سیگموئید

فرض کنید شبکه‌ای از پرسپترون‌ها داریم که می‌خواهیم از آن برای حل مسائل استفاده نماییم. به عنوان مثال، ورودی‌های شبکه ممکن است داده‌های پیکسل خام از یک تصویر اسکن شده اعداد دست نویس باشد. ما می‌خواهیم که شبکه وزن‌ها

[۱]exclusive OR (XOR) [۲]Rommel Hart [۳]McClelland [۴]Multi Layer Perceptron

و بایاس‌ها را طوری بیاموزد که خروجی شبکه عصبی، که ورودی‌اش تصاویر اعداد هستند را به درستی طبقه‌بندی نماید. برای مشاهده نحوه عملکرد یادگیری شبکه عصبی، فرض شود در وزن (یا بایاس) یک تغییر کوچک ایجاد شود. آن چه ما می‌خواهیم این است که این تغییر وزن کوچک فقط یک تغییر کوچک در خروجی از شبکه ایجاد کند.

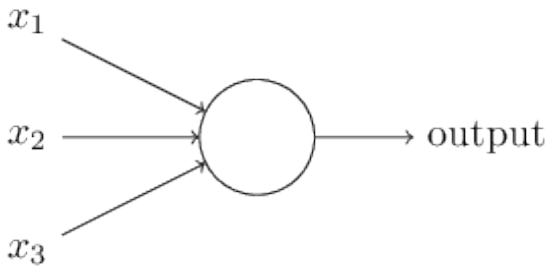


شکل ۲-۸: نمایش تغییر کوچک در وزن یا بایاس هر لایه، باعث یک تغییر کوچک در خروجی می‌شود

همانند تصویر ۲-۸ یک تغییر کوچک در وزن (یا بایاس) فقط باعث تغییر کوچکی در خروجی می شود. برای مثال، فرض شود که شبکه به اشتباه تصویری را به عنوان عدد "۸" طبقه بندی می‌کند در حالی که باید عدد "۹" را تشخیص میداده است. می‌شود تغییرات کوچک در وزن‌ها و بایاس‌ها را پیدا نمود تا شبکه کمی به طبقه بندی تصویر به عنوان "۹" نزدیک شود. وزن‌ها و بایاس‌ها در یک حلقه تکراری تغییر داده می‌شوند تا خروجی با کمترین خطا ساخته شود.

مشکل این است که وقتی شبکه حاوی پرسپترون باشد، بهبود مطلوبی در دقت خروجی شبکه رخ نمی‌دهد. در حقیقت، یک تغییر کوچک در وزن یا بایاس هر پرسپترون در شبکه می‌تواند گاهی اوقات باعث شود خروجی آن پرسپترون به طور کامل از ۰ به ۱ تغییر کند. این تغییر خروجی‌ها باعث تغییر بسیار پیچیده شبکه می‌شود. بنابراین در حالی که ممکن است "۹" به درستی طبقه‌بندی شود، اما خروجی شبکه در سایر تصاویر به احتمال زیاد به طریقی که کنترل آن سخت است کاملاً تغییر کرده است. این اتفاق باعث می‌شود که برای تغییر تدریجی وزن‌ها و بایاس‌ها به گونه‌ای که شبکه به خروجی مورد نظر نزدیک شود، با مشکل مواجه شویم.

با معرفی نوع جدیدی از نورون مصنوعی به نام نورون سیگموئید این مشکل حل می‌شود. نورون‌های سیگموئید شبیه پرسپترون ها هستند، اما به گونه‌ای اصلاح شده‌اند که تغییرات کوچک در وزن و بایاس آن‌ها فقط باعث تغییر کوچکی در خروجی آن‌ها می‌شود. این واقعیت مهمی است که به شبکه‌ای از نورون‌های سیگموئید اجازه یادگیری می‌دهد. نورون‌های سیگموئید همانند پرسپترون‌ها تصویر ۲-۹ تصویر می‌شوند.



شکل ۲-۹: نورون سیگموئید

شبیه به مدل شبکه پرسپترون، نورون‌های سیگموئیدی هم ورودی‌های $x_1, x_2, \ldots$ را دارند، اما در پرسپترون ورودی‌ها دودویی ۰ و ۱ بودند، در اینجا ورودی هر عددی بین ۰ و ۱ می‌تواند اختیار شود. به عنوان مثال عدد ۰٫۶۳۸... یک ورودی صحیح برای نورون سیگموئید می‌باشد. همچنین نورون‌های سیگموئیدی مشابه پرسپترون به ازای هر ورودی پارامتر وزن $w_1, w_2, \ldots$ و بایاس b وجود دارد. و اما خروجی در پرسپترون دودویی ۰ و یا ۱ بود و در نورون سیگموئید خروجی $\sigma(w \cdot x + b)$ می‌باشد که σ تابع سیگما نامیده می‌شود. و به صورت رابطه ۲-۳ تعریف می‌شود.

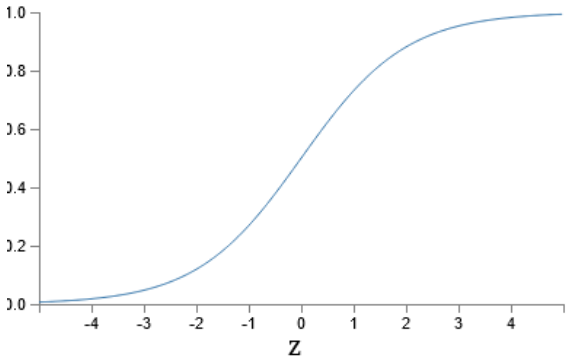
$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}. \tag{۲-۳}$$

در نگاه اول، نورون‌های سیگموئید بسیار متفاوت از پرسپترون‌ها به نظر می‌رسند. اگر قبلاً با آن آشنا نباشید، ممکن است شکل جبری عملکرد سیگموئید مبهم به نظر برسد. در واقع، شباهت‌های زیادی بین پرسپترون‌ها و نورون‌های سیگموئید وجود دارد. برای فهمیدن شباهت مدل پرسپترون و نورون سیگموئید، فرض می‌شود $z \equiv w \cdot x + b$ یک عدد مثبت بزرگ است، آنگاه $e^{-z} \approx 0$ و همچنین $\sigma(z) \approx 1$ می‌شود. به بیان دیگر زمانی که $z = w \cdot x + b$ عددی بزرگ و مثبت است، خروجی نورون سیگموئید به سمت ۱ میل می‌کند، درست همانند یک پرسپترون. از طرف دیگر فرض کنید که $z = w \cdot x + b$ خیلی منفی باشد آنگاه $e^{-z} \to \infty$ و $\sigma(z) \approx 0$، همچنین وقتی $z = w \cdot x + b$ خیلی منفی است رفتار نورون سیگموئید نزدیک به پرسپترون است.

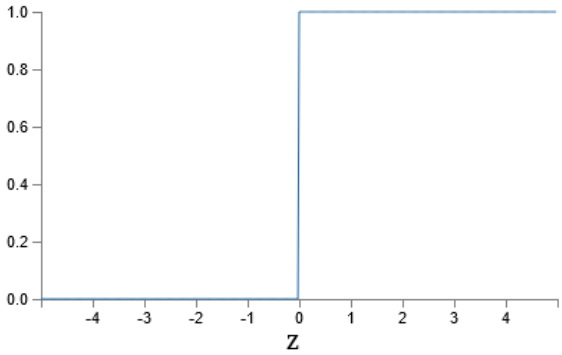
شکل جبری سیگموئید

در تصویر ۲-۱۰ یک تابع سیگموئید ترسیم شده است، این شکل یک نسخه هموار شده از تابع پله‌ای [1] تصویر ۲-۱۱ می‌باشد. اگر به جای تابع سیگموئید یک تابع پله‌ای قرار بگیرد، نورون سیگموئید یک پرسپترون خواهد شد، زیرا خروجی وابسته به مثبت و یا منفی بودن $w \cdot x + b$، مقدار ۰ یا ۱ خواهد شد. در واقع این انحناهای تابع σ مهم هستند، از این منظر که تغییر کوچک Δw_j در وزن‌ها و Δb در بایاس باعث ایجاد تغییر کوچکی در Δoutput خروجی نورون می‌شود. و رابطه ۲-۴ تقریب خوبی برای محاسبه Δoutput می‌باشد.

[1] Step function



شکل ۲-۱۰: تصویر ترسیم تابع σ



شکل ۲-۱۱: تصویر تابع پله‌ای

$$\Delta \text{output} \approx \sum_j \frac{\partial \,\text{output}}{\partial w_j} \Delta w_j + \frac{\partial \,\text{output}}{\partial b} \Delta b, \tag{۴-۲}$$

۲-۴-۴ یادگیری شبکه عصبی با گرادیان کاهشی

آنچه ما نیاز داریم الگوریتمی است که که مقدار مناسب وزن‌ها و بایاس‌ها را پیدا کند تا خروجی شبکه تقریبی از $y(x)$ برای تمامی ورودی‌های آموزشی x به شبکه باشد. برای تعیین این هدف، تابعی را به عنوان تابع هزینه تعریف [۱] [۲] می‌نماییم:

$$C(w,b) \equiv \frac{۱}{۲n} \sum_x \|y(x) - a_x\|^۲. \tag{۵-۲}$$

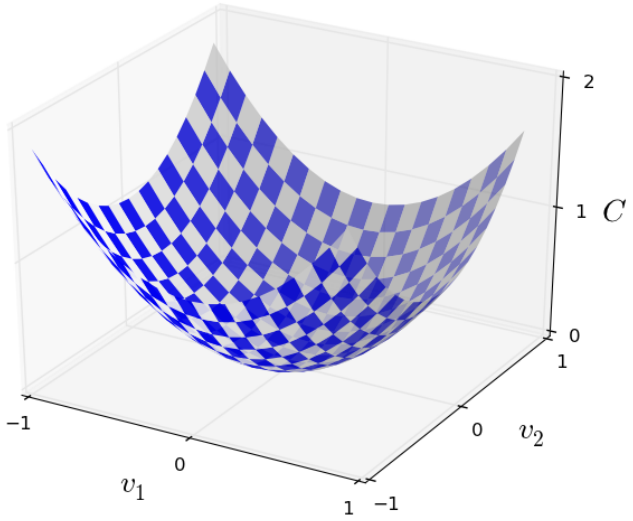
---

[۱] Cost Function

[۲] گاهی تابع هزینه، تابع خطا و یا تابع هدف نیز نامیده می‌شود

در رابطه ۲-۵، w به مجموعه تمام وزن‌های شبکه اشاره می‌کند و b همه بایاس‌ها و n تعداد تمام نمونه‌های آموزشی، a بردار خروجی از شبکه زمانی که x ورودی آن است. عبارت $||v||$ اشاره به نرم و یا اندازه بردار v دارد. ما C را تابع هزینه درجه دوم می‌نامیم. این تابع میانگین مربعات خطا [۱] نامیده می‌شود. با بررسی فرم تابع هزینه درجه دوم، مشاهده می‌شود که $C(w,b)$ مقداری همیشه مثبت و غیر منفی دارد. علاوه بر این زمانی که تقریبا $y(x)$ برای تمام نمونه‌های آموزشی x برابر با خروجی شبکه شود، مقدار $C(w,b)$ کوچک می‌شود برای مثال $C(w,b) \approx ۰$. بنابراین اگر الگوریتم آموزشی بتواند وزن و بایاس را پیدا کند به طوری که $C(w,b) \approx ۰$ کارش را انجام داده است. بنابراین هدف الگوریتم آموزشی به حداقل رساندن تابع هزینه $C(w,b)$، تابعی از w و b است. به عبارت دیگر، هدف الگوریتم آموزشی پیدا کردن مجموعه‌ای از وزن‌ها و بایاس‌ها است که هزینه را تا آن‌جا که ممکن است کوچک کند، این کار را با استفاده از الگوریتمی به نام گرادیان کاهشی [۲] انجام می‌شود.

به نسبت مسئله‌ای و ساختار شبکه عصبی که طراحی می‌شود اندازه v، ($v = v_۱, v_۲, \ldots$) تعیین می‌شود. به عنوان مثال برای پیدا کردن حداقل مقدار تابع $C(v)$ اگر تابع C تابعی ساخته شده از دو متغییر $v_۱$ و $v_۲$ باشد، تابع هزینه نموداری شبیه به تصویر ۲-۱۲ خواهد داشت.



شکل ۲-۱۲: نمونه تابع C دارای دو متغییر $v_۱$ و $v_۲$

هدف الگوریتم بهینه‌سازی گرادیان کاهشی پیدا کردن برداری است که تابع در آن مکان در نقطه بهینه سراسری‌اش قراردارد. در تصویر ۲-۱۲ می‌توانیم به صورت بصری، نقطه تقریبی، حداقلی تابع را مشخص نمود. اما در مسائل واقعی توابع بسیار پیچیده‌تر و دارای ابعاد بالاتری هستند که بصری سازی آن‌ها فعلا امکان پذیر نمی‌باشد.

یکی از راه‌های حل این مشکل استفاده از روش‌های بهینه سازی ریاضی و تلاش برای یافتن حداقل مقدار تابع به صورت

[۱] MSE, Mean Squared Error [۲] Gradient Descent

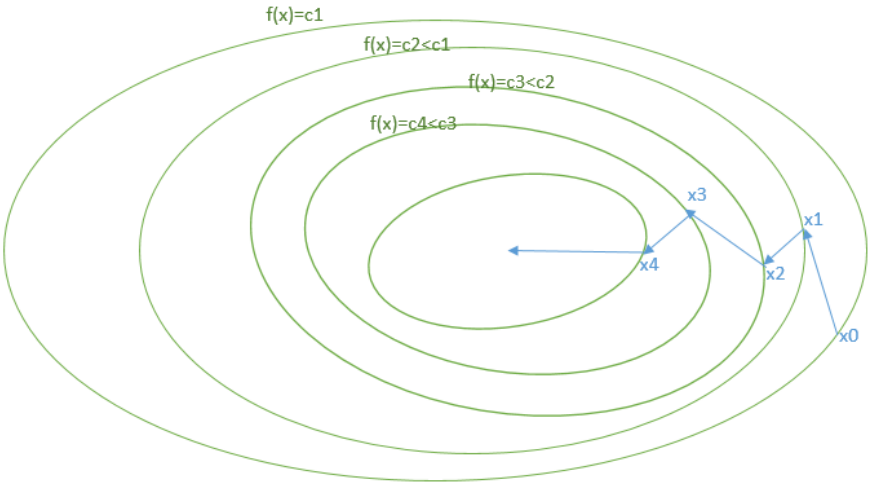
تحلیلی است. ما می‌توانیم مشتق را محاسبه کرده و سپس از آنها برای یافتن مکان‌هایی که C یک اکسترموم است استفاده کنیم. این روش زمانی کار می‌کند که C تابعی از یک یا چند متغیر باشد. اما وقتی متغیرهای بیشتری وجود داشته باشد، استفاده از این روش به یک مسئله سخت تبدیل می‌شود. در مسائل شبکه‌های عصبی اغلب متغیرهای بیشتری وجود دارد، شبکه‌های عصبی وجود دارد که دارای توابع هزینه‌ای هستند که از میلیاردها وزن و بایاس تشکل شده‌اند. عملا استفاده از مشتق برای به حداقل رساندن آن‌ها پاسخ نمی‌دهد. بنابراین روش فوق استفاده از مشتق، پاسخ‌گوی نیاز شبکه‌های عصبی نمی‌باشد.

روش تکرار نزولی

مسئله حداقلی‌سازی تابع پیوسته و بدون محدودیت مشتق‌پذیر $f : \mathbb{R}^n \to \mathbb{R}$ را در نظر بگیرید. بیشتر الگوریتم‌ها برای حل این مسائل بر روی یک ایده مهم بنا شده‌اند، که "نزول تکراری[1]" نامیده می‌شود و به صورت زیر کار می‌کند:

ما از یک نقطه بردار تصادفی x^0 شروع می‌کنیم و به صورت مداوم بردارهای $x^1, x^2, \ldots$ را تولید می‌کنیم که مقدار تابع f در هر تکرار کاهش می‌یابد.

$$f(x^{k+1}) < f(x^k), \text{for all} K = 0, 1, \ldots$$



شکل ۲-۱۳: روش نزول تکراری برای پیدا کردن مینیمم تابع f. هر بردار هزینه کمتری از سازنده‌اش دارد. [۱۸]

۲-۴-۵ روش‌های گرادیان

بردار $x \in \mathbb{R}^n$ با $\nabla f(x) \neq 0$ را در نظر بگیریم از بسط اولیه تیلور حول بردار x خواهیم داشت:

[1] iterative descent

$$f(x_\alpha) = f(x) + \bigtriangledown f(x)'(x_\alpha - x) + o(\|x_\alpha - x\|)$$
$$= f(x) - \alpha\| \bigtriangledown f(x)\|^2 + o(\alpha\|x_\alpha - x\|)$$

همچنین رابطه فوق را به صورت رابطه زیر می‌توانیم بنویسیم:

$$f(x_\alpha) = f(x) - \alpha\| \bigtriangledown f(x)\|^2 + o(\alpha)$$

که ترم $\alpha\| \bigtriangledown f(x)\|^2$ برای α نزدیک صفر بر $o(\alpha)$ غلبه می‌کند. برای α به اندازه کافی مثبت و کوچک، مقدار $f(x_\alpha)$ از مقدار $f(x)$ کوچکتر خواهد بود. بسیاری از متدهای گرادیان به فرم زیر بیان می‌شوند:

$$x^{k+1} = x^k - \alpha^k D^k \bigtriangledown f(x^k) \qquad (۶-۲)$$

که D^k یک ماتریس سیمتریک معین مثبت [1] است. در عبارت ۲-۶ اگر $D^k \bigtriangledown f(x^k)$ را با d^k جایگزین شود، به طوری که $d^k = -D^k \bigtriangledown f(x^k)$، رابطه ۲-۷ حاصل می‌شود.

$$x^{k+1} = x^k - \alpha^k d^k \qquad (۷-۲)$$

بر اساس d^kهای مختلف و همچنین نحوه انتخاب اندازه گام حرکت α روش‌های گرادیان مختلفی ساخته می‌شوند. و علاوه بر این دو گروه تغییرات، الگوریتم‌های گرادیان را بر اساس حجم داده‌های آموزشی که به آن داده می‌شود می‌توان سه گونه مختلف دسته بندی نمود.

۲-۴-۵-۱ گرادیان کاهشی دسته‌ای

الگوریتم گرادیان کاهشی دسته‌ای [2]، گرادیان تابع هزینه را بر اساس پارامتر θ برای تمام داده‌های موجود در دیتاست محاسبه می‌کند.

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta) \qquad (۸-۲)$$

[1] positive definite symetrix [2] Batch gradient descent

در این روش نیاز است که گرادیان را برای تمام داده‌های آموزشی محاسبه نماییم، روش گرادیان کاهشی دسته‌ای می‌تواند بسیار کند باشد و برای مجموعه داده‌ای که درون حافظه نتواند بارگذاری شود روشی اجرا ناپذیر است. در برنامه‌نویسی شبه‌کد گرادیان کاهشی دسته‌ای به صورت کد زیر خواهد بود.

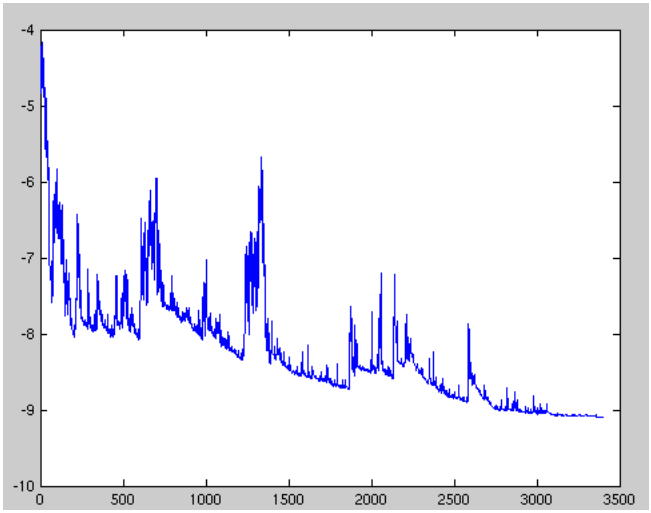
```
for i in range(nb_epochs):                                          ۱
   params_grad = evaluate_gradient(loss_function, data, params)     ۲
   params = params - learning_rate * params_grad                    ۳
```

۲-۴-۵-۲ گرادیان کاهشی تصادفی

گرادیان کاهشی تصادفی [۱] در هر مرحله از بروزرسانی پارامترهای شبکه از نمونه $x^{(i)}$ با مقدار تابع هدف $y^{(i)}$ استفاده می‌کند

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; x^{(i)}; y^{(i)}) \qquad (۹-۲)$$

در الگوریتم گرادیان کاهشی دسته‌ای، محاسبات زیادی برای مجموعه داده‌های بزرگ انجام می‌شود، قبل از بروزرسانی وزن‌ها و بایاس‌های شبکه، گرادیان برای تمام داده‌های آموزشی محاسبه می‌شود. روش گرادیان کاهشی تصادفی به نسبت گرادیان کاهشی دسته‌ای بسیار سریع‌تر می‌باشد. در گرادیان کاهشی تصادفی در هر تکرار به صورت تصادفی داده‌های درهم ریزی می‌شوند و هر داده به صورت یک به یک گرادیان‌هایشان محاسبه شده و هر داده به صورت مستقل بر روی محاسبه وزن شبکه اثر می‌گذارد، به همین علت در جریان بروزرسانی پارامترهای وزن و بایاس، پرش‌هایی را در زمان بهینه‌سازی تابع هزینه همانند تصویر ۲-۱۴ وجود دارد.



شکل ۲-۱۴: پرش در الگوریتم گرادیان کاهشی تصادفی

زمانی که گرادیان کاهشی تصادفی در حال همگرا شدن به نقطه بهینه تابع هزینه و پیدا کردن وزن‌های مربوطه است، نوسان‌های ذکر شده ممکن است نقاط بهینه محلی بهتری را پیدا نماید، و از طرفی این پرش‌ها همگرایی به یک نقطه بهینه

[۱] Stochastic gradient descent, SGD

دقیق را پیچیده‌تر می‌کنند. اگر به صورت کاهشی، نرخ یادگیری را کاهش دهیم، الگوریتم گرادیان کاهشی تصادفی رفتار همگرایی مشابه به گرادیان کاهشی دسته‌ای از خودش به نمایش می‌گذارد. شبه‌کد الگوریتم فوق به صورت ذیل می‌باشد:

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

۲-۴-۵-۳ گرادیان کاهشی دسته‌ای کوچک

گرادیان کاهشی دسته‌ای کوچک[۱] الگوریتمی است که در هر تکرار بر اساس فقط n نمونه از داده‌های آموزشی میزان گرادیان را در هر تکرار محاسبه می‌نماید.

$$\theta = \theta - \eta \cdot \nabla_\theta C(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{۲-۱۰}$$

برای تعداد داده‌های آموزشی در هر دسته (n) مقداری بین ۱۶ و ۱۰۲۴ انتخاب می‌شود، این اعداد برای برنامه‌های مختلف متفاوت هستند، و برای هر مسئله به عنوان یک فراسنج باید مقدار مناسب را برایش جستجو کرد. در آموزش شبکه‌های عصبی به صورت پیش فرض از این الگوریتم استفاده می‌شود. شبه‌کد الگوریتم فوق به صورت ذیل می‌باشد:

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

۲-۵ پس‌انتشار خطا

پس از انتشار مجله PDP [۲] در سال ۱۹۸۶، یادگیری به روش پس‌انتشار [۳] یکی از روش‌های مشهور آموزش شبکه‌های عصبی مصنوعی شده است. دلیل محبوبیت روش پس‌انتشار سادگی و قدرت نسبی الگوریتم است. از این روش بر خلاف قوانین یادگیری پرسپترون و قوانین یادگیری ویدرو-هاف [۴] ، می‌توان برای آموزش شبکه‌های عصبی غیرخطی چند لایه استفاده کرد. از آن‌جایی که شبکه‌های عصبی چند لایه برای برنامه‌های کاربردی در دنیای واقعی مورد نیاز بودند استفاده از این الگوریتم

[۱] Mini-batch gradient descent [۲] Parallel distributed processing: Exploration in microstructure of cognition. Two volume by Rumelhart, McClelland [۳] Backpropgation [۴] Widrow-Hoff

اهمیت بالایی پیدا کرد. تقریبا به اندازه قدرت الگوریتم پس‌انتشار در توضیح محبوبیت، سادگی آن است. ایده اصلی قدیمی و ساده است. الگوریتم به قدری ساده است که می‌توان آن را چند خط کد پیاده‌سازی کرد[۱۹].

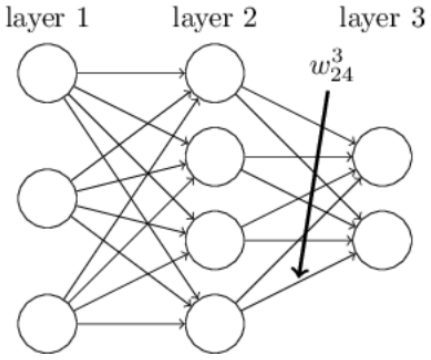
نام الگوریتم پس‌انتشار در واقع از واژه‌ای است که رزنبلات در سال ۱۹۶۲ برای تلاش در راستای تعمیم الگوریتم یادگیری پرسپترون چند لایه به کار برده است. تلاش‌های زیادی برای تعمیم روش یادگیری پرسپترون به چندین لایه در طول دهه ۱۹۶۰ و دهه ۱۹۷۰ انجام شد، اما هیچ کدام از آن‌ها به موفق نبودند. به نظر می‌رسد سه ارائه مستقل از نسخه مدرن الگوریتم پس‌انتشار مجدد وجود دارد:

پاول وربوس [۱] ایده اصلی را در سال ۱۹۷۴ در پایان نامه مقطع دکتری خود به نام فراتر از رگرسیون [۲] توسعه داد، و دیوید پارکر [۳] و دیوید روملهارت [۴] این ایده را در بهار سال ۱۹۸۲ توسعه دادند. با این حال، تا انتشار مقاله توسط روملهارت، هینتون و ویلیامز در سال ۱۹۸۶ در توضیح این ایده و نشان دادن تعدادی برنامه شبکه عصبی و هوش مصنوعی ارتباط پس‌انتشار را با شبکه عصبی به تصویر کشیدند، و نشان دادند که روش پس‌انتشار بسیار سریع‌تر از روش‌های قبلی یادگیری عمل می‌کند و امکان حل مشکلاتی که قبلا حل نشده بودند را ممکن می‌سازد، و پس از آن الگوریتم پس‌انتشار توسط تعداد زیادی از محققان مورد استفاده قرار گرفته است. و امروزه الگوریتم پس‌انتشار کار اصلی یادگیری در شبکه‌های عصبی را انجام می‌دهد[۲۰].

۲-۵-۱ پس‌انتشار

در قلب الگوریتم پس‌انتشار، مشتق‌های جزئی $\partial C/\partial w$ از تابع هزینه C نسبت به تمام وزن‌های w و بایاس‌ها b در شبکه عصبی قرار دارد. مشتق جزئی میزان سرعت تغییر تابع هزینه C است زمانی که وزن‌ها w و بایاس‌ها b تغییر می‌کند.

برای توصیف وزن شبکه عصبی از w^l_{jk} برای اشاره به وزن‌ها استفاده می‌کنیم که نشان دهنده ارتباط از نرون kام لایه $l-1$ به نرون jام لایه l است. در تصویر ۲-۱۵ وزن ارتباط از نرون چهارم لایه دوم به نرون دوم لایه سوم مشاهده می‌شود.

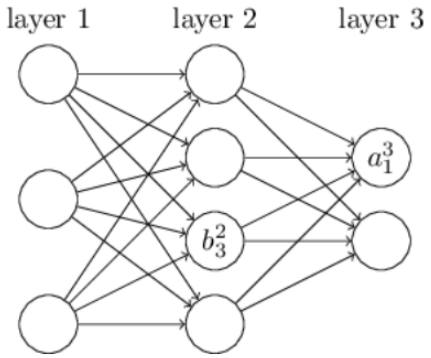


شکل ۲-۱۵: شماره گذاری وزن در شبکه عصبی

برای راحتی درک اندیس‌ها، توجه کنید که اندیس j به نرون ورودی و اندیس k به نرون خروجی اشاره دارد. از اندیس

[۱] Paul Werbos [۲] Beyond Regression [۳] David Parker [۴] David Rumelhart

گذاری مشابهی برای بایاس‌ها و توابع فعال‌ساز [۱] استفاده می‌نماییم. عبارت b_j^l به بایاس نرون jام لایه l اشاره می‌کند. و a_j^l اشاره به تابع فعال‌ساز نرون jام لایه l دارد. تصویر ۲-۱۶ مثالی از این اندیس گذاری‌ها می‌باشد.



شکل ۲-۱۶: شماره گذاری نودها در شبکه عصبی

با این اندیس گذاری‌ها همانگونه که در رابطه ۲-۱۱ مشاهده می‌شود تابع فعال‌ساز a_j^l از نرون jام لایه l وابسته به تابع فعال‌ساز در لایه $l-1$ می‌باشد.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right), \tag{۲-۱۱}$$

برای نوشتن رابطه ۲-۱۱ در فرم ماتریسی، ماتریس وزن w^l برای هر لایه l تعریف می‌کنیم که هر خانه ماتریس فوق در سطر jام و ستون kام مقدار $w_j^l k$ قرار دارد.و به صورت مشابه برای هر لایه l بردار بایاس‌ها b^l را تعریف می‌نماییم که اجزا بردار b^l مقادیر بایاس هر نرون b_j^l می‌باشد. در نهایت بردار تابع فعال‌ساز a^l را که اجزا بردار را a_j^l تشکیل می‌دهند را تعریف می‌نماییم. اکنون رابطه ۲-۱۱ را با توجه به نمادهای تعریف شده به فرم ماتریسی رابطه ۲-۱۲ باز نویسی می‌نماییم.

$$a^l = \sigma(w^l a^{l-1} + b^l). \tag{۲-۱۲}$$

رابطه ۲-۱۲ به ما طرز فکر بهتری درباره نحوه ارتباط تابع فعال‌ساز هر لایه با لایه‌های پیشین ارائه می‌دهد. همانگونه که مشاهده می‌شودما فقط ماتریس وزن‌ها را در بردار فعال‌ساز لایه پیشین ضرب و مقدار بردار بایاس را با حاصل فوق جمع می‌نماییم و در انتها تابع σ را بر روی خروجی اعمال می‌کنیم تا مقدار تابع فعال ساز لایه lام محاسبه شود. زمانی که از رابطه ۲-۱۲ برای محاسبه a^l استفاده می‌کنیم، مقدار $z^l \equiv w^l a^{l-1} + b^l$ را محاسبه می‌کنیم. ما z^l را وزن ورودی نرون‌های لایه l می‌نامیم. رابطه ۲-۱۲ را می‌توان به صورت ترم $a^l = \sigma(z^l)$ بنویسیم.

هدف الگوریتم پس‌انتشار محاسبه مشتق جزئی $\partial C/\partial w$ و $\partial C/\partial b$ تابع هزینه C نسبت به تمام وزن‌ها w یا بایاس‌ها b در شبکه می‌باشد. برای آنکه الگوریتم کار کند نیاز به دو فرضیه درباره فرم توابع هزینه داریم. برای ادامه مطلب از تابع

[۱] Activation function

هزینه درجه دوم [1] رابطه ۲-۱۳ استفاده می‌نماییم.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2, \tag{۱۳-۲}$$

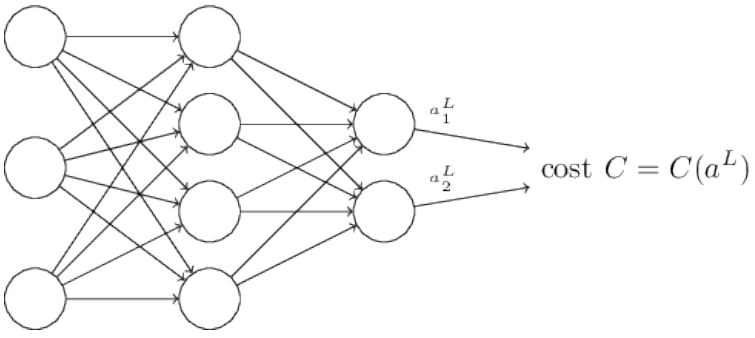
در این رابطه n تعداد مثال‌های آموزشی می‌باشد؛ جمع برای تمام نمونه‌های آموزشی x می‌باشد؛ $y = y(x)$ خروجی مورد نظر است؛ L اشاره به تعداد لایه‌های شبکه دارد؛ $a^L = a^L(x)$ [2] یک بردار فعال‌ساز خروجی شبکه زمانی که x ورودی شبکه بوده، می‌باشد.

فرضیه اول

اولین فرضی که نیاز داریم این است که تابع هزینه C را بتوانیم به فرم $C = \frac{1}{n} \sum_x C_x$ میانگین برای هر تابع هزینه C_x برای هر ورودی مجزا ورودی x بنویسیم. در تابع هزینه درجه دو هزینه برای یک نمونه آموزشی $C_x = \frac{1}{2}\|y - a^L\|^2$ می‌باشد. این فرض برای تمام توابع هزینه در شبکه‌های عصبی باید صادق باشد. دلیلی که به این فرض را نیاز داریم این است که الگوریتم پس‌انتشار در واقع مشتقات جزئی را بر هر یک از نمونه‌های آموزشی $\partial C_x/\partial w$ و $\partial C_x/\partial b$ محاسبه می‌کند. و ما مقدار $\partial C/\partial w$ و $\partial C/\partial b$ را با میانگیری روی داده‌های آموزشی محاسبه می‌نماییم. در تمام محاسبات ما در حال محاسبه C_x هستیم اما برای راحتی از نوشتن زیرنویس x اجتناب می‌نماییم.

فرضیه دوم

فرض دوم ما در مورد تابع هزینه این است، که بتوان تابع هزینه را به عنوان تابعی از خروجی‌های شبکه عصبی نوشت:



شکل ۲-۱۷: مدل لایه آخر شبکه عصبی

برای مثال تابع هزینه درجه دوم این فرضیه را برآورده می‌سازد. زیرا تابع هزینه درجه دو برای یک نمونه آموزشی به فرم زیر نوشته می‌شود:

---

[1] quadratic cost function

[2] در بعضی منابع $\hat{y}$ نام گذاری می‌شود

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2, \tag{۱۴-۲}$$

و بنابر این تابعی از فعال‌سازهای خروجی است. البته خروجی فوق وابسته به متغییر y نیز می‌باشد.

پس‌انتشار درباره فهمیدن چگونگی تاثیر تغییر وزن‌ها و بایاس‌های شبکه در تغییر مقدار تابع هزینه است. درنهایت، این به معنی محاسبه مشتقات جزئی $\partial C/\partial w^l_{jk}$ و $\partial C/\partial b^l_j$ می‌باشد. برای محاسبه آن‌ها ابتدا مقدار δ^l_j تعریف می‌شود. که به آن خطای نورون jام در لایه l نامیده می‌شود. پس‌انتشار یک رویه و روش برای محاسبه خطای δ^l_j ارائه می‌دهد. و سپس رابطه δ^l_j با $\partial C/\partial w^l_{jk}$ و $\partial C/\partial b^l_j$ را نمایش می‌دهیم. پس‌انتشار برپایه چهار معادله پایه‌ای می‌باشد. این معادلات چهارگانه راهی برای محاسبه خطای δ^l و گرادیان تابع هزینه فراهم می‌کند.

معادله خطا در لایه خروجی، δ^l

اجزا تشکیل دهنده δ^l، که یک رابطه طبیعی می‌باشد.

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j). \tag{BP۱}$$

در معادله BP۱ اولین ترم از راست $\partial C/\partial a^L_j$، اندازه گیری سرعت تغییر تابع هزینه به عنوان تابعی از خروجی تابع فعال‌ساز jام است. اگر C خیلی زیاد وابسته به خروجی نورون jام نباشد آنگاه δ^L_j کوچک خواهد بود. دومین ترم در راست $\sigma'(z^L_j)$ سرعت تغییر تابع σ در z^L_j را اندازه می‌گیرد. توجه کنید که همه چیز در رابطه BP۱ به راحتی قابل محاسبه است. به طور خاص ما مقدار z^L_j را در محاسبات رو به جلو شبکه انجام می‌دهیم و برای محاسبه $\sigma'(z^L_j)$ سربار محاسباتی کمی را خواهیم داشت. و فرم دقیق $\partial C/\partial a^L_j$ وابسته به فرم تابع هزینه می‌باشد. با این حال اگر تابع هزینه مشخص باشد، در محاسبات $\partial C/\partial a^L_j$ مشکل کمی وجود خواهد داشت. برای مثال ما اگر تابع هزینه درجه دوم $C = \frac{1}{2}\sum_j (y_j - a^L_j)^2$ استفاده کنیم آنگاه $\partial C/\partial a^L_j = (a^L_j - y_j)$ را خواهیم داشت که به سادگی قابل محاسبه می‌باشد.

معادله BP۱ یک عبارت جز به جز برای محاسبه δ^L است. به راحتی می‌توان فرم ماتریسی رابطه را بازنویسی نمود.

$$\delta^L = \nabla_a C \odot \sigma'(z^L). \tag{BP۱a}$$

در معادله BP۱a، اجزا بردار $\nabla_a C$ مقادیر مشتق‌های جزئی $\partial C/\partial a^L_j$ می‌باشد. می‌توانید تصور کنید که $\nabla_a C$ بیان کننده میزان تغییر C نسبت تابع فعال‌ساز خروجی می‌باشد. در مثال تابع هزینه درجه دوم ما $\nabla_a C = (a^L - y)$ را داریم، و فرم

ماتریسی فرمول برای تابع هزینه درجه دوم به فرم رابطه ۲-۱۵ می‌شود.

$$\delta^L = (a^L - y) \odot \sigma'(z^L). \tag{۱۵-۲}$$

هم اکنون رابطه ۲-۱۵ به فرم برداری مناسبی برای پیاده‌سازی توسط زبان‌های برنامه نویسی و کتابخانه‌هایی همانند Numpy, PyTorch, Tensorflow رسیده است.

معادله خطا δ^l به لحاظ نقش خطای لایه بعدی δ^{l+1}

به طور خاص:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{BP۲}$$

در اینجا $(w^{l+1})^T$ ترنهاده وزن ماتریس w^{l+1} برای لایه $(l+1)^{\text{th}}$ می‌باشد. مقدار خطای لایه بعدی δ^{l+1} را هم داریم، به علت اینکه از انتها به ابتدای [1] شبکه در حال انجام محاسبات هستیم. و عبارت محاسبه شده ضرب هادامارد $\odot\sigma'(z^l)$ می‌شود. با فرمول BP۱ و BP۲ میزان خطای δ^l هر لایه از شبکه را محاسبه نماییم. ما از فرمول BP۱ شروع می‌نماییم و مقدار خطای δ^L را محاسبه می‌نماییم، سپس فرمول $BP۲$ برای محاسبه خطای δ^{L-1} استفاده می‌نماییم، مجدد از فرمول $BP۲$ برای محاسبه خطای δ^{L-2} استفاده می‌نماییم، و همینطور تا لایه اول شبکه عصبی برای محاسبه خطا پیش می‌رویم.

به طور خاص:

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j. \tag{BP۳}$$

که خطای δ^l_j دقیقا برابر با نرخ تغییرات $\partial C/\partial b^l_j$ است. در معادلات BP۱ و BP۲ مقدار δ^l_j را محاسبه نموده‌ایم و نحوه محاسبه‌اش را بیان شده است. و می‌توانیم فرمول BP۳ را به فرم خلاصه زیر بازنویسی نماییم:

$$\frac{\partial C}{\partial b} = \delta, \tag{۱۶-۲}$$

از رابطه فوق اینگونه استنباط می‌شود که بایاس b همای خطای محاسبه شده δ می‌باشد.

[1] Backward

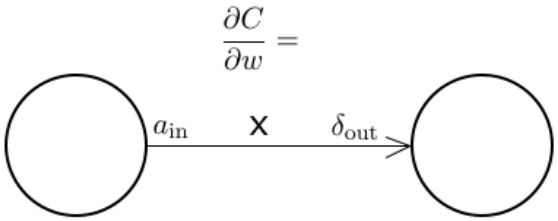
معادله نرخ تغییر تابع هزینه نسبت به تمام وزن‌های شبکه

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j. \tag{BP۴}$$

معادله BP۴ روش محاسبه مشتق جزئی $\partial C/\partial w^l_{jk}$ با ترم‌های محاسبه شده δ^l و a^{l-1} را که روش محاسبه آن‌ها را می‌دانیم به ما می‌گوید. فرمول می‌تواند بدون اندیس‌های سنگین بازنویسی شود:

$$\frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}, \tag{۲-۱۷}$$

ار فرمول فوق اینگونه استباط می‌شود که a_{in} یک فعال‌ساز از نرون ورودی به وزن w است و δ_{out} خطای نورون خروجی از وزن w است. برای راحتی درک این موضوع به وزن w توجه کنید که دو نورون به آن وزن متصل شده‌اند، مشابه تصویر ۲-۱۸



شکل ۲-۱۸: اتصال دو نورون به یک وزن

۲-۵-۲ الگوریتم پس‌انتشار

معادلات پس‌انتشار راه حلی برای محاسبه گرادیان تابع هزینه فراهم ساخته است. حال آن را به صورت الگوریتم بازنویسی می‌کنیم:

۱. ورودی x: محاسبه تابع فعال‌ساز a^1 برای لایه ورودی.

۲. محاسبه $a^l = \sigma(z^l)$ برای $l = ۲, ۳, \ldots, L$.

۳. محاسبه بردار خطای خروجی $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

۴. پس‌انتشار خطا: محاسبه $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ برای $l = L-1, L-2, \ldots, 2$.

۵. محاسبه گرادیان تابع هزینه $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ و $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

۲-۶ الگوریتم‌های گرادیان کاهشی

همانطور که پیشتر گفته شد، بر اساس روش محاسبه جهت کاهش و روش محاسبه ضریب نرخ یادگیری روش‌های مختلفی ارائه شده است. در این بخش چند الگوریتم‌های بهینه‌سازی ،Momentum شتاب گرادیان نسترو، گرادیان تطبیقی، برآورد لحظه‌ای تطبیقی، AdaDelta، RProp و RMSProp و LAMB مورد بررسی قرار می‌گیرد.

۲-۶-۱ Momentum

روش گرادیان کاهشی تصادفی در حرکت درون دره‌ها مشکل دارد، یعنی در یکی از ابعاد مسئله که سطح دارای شیب بیشتری است گرادیان کاهشی به آن سمت خم می‌شود [۲۱]. این اتفاق به صورت مداوم در اطراف نقاط بهینه محلی رخ می‌دهد. روش مومنتوم سرعت پیدا کردن مینیمم تابع هزینه را سرعت می‌بخشد و میزان نوسانات را کاهش می‌دهد. در این الگوریتم جهت و درصدی از میزان اندازه بردار محاسبه شده مرحله پیشین، با محاسبات جهت بردار جدید جمع می‌شود. تا از میزان خطای ممکن مقداری بکاهد.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta C(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{۲-۱۸}$$

همانگونه که در عبارت ۲-۱۸، مشاهده می‌شود، با افزودن مقدار اعشاری و بین ۱ و ۰ متغییر γ از بردار بروزرسانی مرحله پیشین به بردار بروزرسانی فعلی، جهت بردار بهینه‌سازی محاسبه می‌شود. معمولا مقدار γ به صورت پیش‌فرض ۰/۹ در نظر گرفته می‌شود و یا عددی مشابه بین ۰ و ۱ برایش در نظر گرفته می‌شود.

۲-۶-۲ شتاب گرادیان نسترو

الگوریتم شتاب گرادیان نسترو (NAG) [۲۲] مشابه الگوریتم مومنتوم ۲-۶-۱، از بردار γv_{t-1} در جهت کاهش سرعت سقوط در دره‌های یک یا چند همسایگی استفاده می‌کند، و با تغییر پارامتر θ به $\theta - \gamma v_{t-1}$ که یک تخمین از مقدار بعدی پارامتر θ می‌باشد تغییر می‌دهد.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta C(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t \tag{۱۹-۲}$$

در الگوریتم ۲-۱۹ نیز مشابه الگوریتم مومنتوم، مقدار γ عدی حدود ۰/۹ تعیین می‌شود و یا عددی بین ۰ و ۱ برایش در نظر گرفته می‌شود.

۲-۶-۳ گرادیان تطبیقی

گرادیان تطبیقی [۱] الگوریتم بهینه‌سازی می‌باشد که سعی می‌کند ضریب نرخ یادگیری را در هر مرحله منطبق بر پارامترهای شبکه محاسبه و بروزرسانی‌های کوچکتری را انجام دهد. به عبارتی، یعنی نرخ یادگیری پایین‌تر برای پارامترهای مرتبط با ویژگی های مکرر، و به‌روزرسانی بزرگتر یعنی نرخ یادگیری بالاتر، برای پارامترهای مرتبط با ویژگی‌های نادر. به همین دلیل برای داده‌های اسپارس بسیار مناسب است. دیان و همکارانش [۲۳] دریافتند که آداگراد قدرت SGD را تا حد زیادی افزایش داده است و از این روش برای آموزش شبکه‌های عصبی در مقیاس بزرگ در گوگل استفاده می‌کنند. پنینگتون و همکارانش [۲۴] از آداگراد برای اموزش تعبیه‌سازی کلمات GloVe [۲] استفاده کرده‌اند.

در چند روش پیشین، ما برای همه پارامترهای θ به طور همزمان یک بروز رسانی را انجام دادیم زیرا هر پارامتر θ_i از یک نرخ یادگیری η مشابه یکدیگر استفاده می‌کردند. از آنجا که آداگراد از نرخ یادگیری متفاوتی برای هر پارامتر θ_i در هر مرحله t استفاده می‌کند، ابتدا بروز رسانی پارامتر آداگراد را انجام می‌دهیم، به طور اختصار، از g_t برای نشان دادن گرادیان در مرحله زمانی t استفاده می‌کنیم. $g_{t,i}$ مشتق جزئی از تابع هدف نسبت به پارامتر θ_i در مرحله زمانی t می‌باشد.

$$g_{t,i} = \nabla_\theta C(\theta_{t,i})$$
$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{۲۰-۲}$$

آداگراد در بروز رسانی خود، ضریب نرخ یادگیری η را در هر مرحله t برای هر پارامتر θ_i بر اساس گرادیان‌های قبلی که برای θ_i محاسبه شده است، تغییر می‌دهد:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{۲۱-۲}$$

[۱] Adaptive gradient [۲] Word embedding

$G_t \in \mathbb{R}^{d\times d}$ یک ماتریس مربعی است که عناصر قطر اصلی i, i مجموع مربعات گرادیان‌ها هستند. یکی از مزایای اصلی آداگراد این است که نیازی به تنظیم دستی نرخ یادگیری ندارد. اکثر پیاده‌سازی‌ها از مقدار پیش فرض ۰٫۰۱ استفاده می‌کنند و این مقدار را تغییر نیز نمی‌دهند. ضعف اصلی آداگراد انباشتن مربع گرادیان‌ها در مخرج است: از آنجا که هر عبارت اضافه شده مثبت است، مجموع انباشته شده در طول یادگیری همچنان افزایش می‌یابد. این به نوبه خود باعث می شود که میزان یادگیری کاهش یابد و در نهایت بی نهایت کوچک شود، و بهینه‌سازی تابع هزینه با سرعت کمتری پیش می‌رود.

۲-۶-۴ برآورد لحظه تطبیقی

متد برآورد لحظه تطبیقی [۱] که به صورت خلاصه Adam نامیده می‌شود متدی دیگر برای محاسبه تطبیقی نرخ یادگیری برای هر پارامترها می‌باشد. در این متد مشابه متدهای AdaGrad و RMSProp از میانگین مجذور مربعات گرادیان استفاده می‌کند. همچنین این متد بهینه‌سازی میانگین گرادیان‌های گذشته را همانند متد Momentum در محاسبات خود لحاظ می‌کند. دو بخش عنوان شده توسط عبارت ۲-۲۲ تعریف می‌شوند.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{۲-۲۲}$$

متغییرهای m_t و v_t به ترتیب تخمین لحظه اول میانگین و لحظه دوم واریانس بدون مرکز [۲] است. در مرحله شروع مقدار دو متغییر فوق با بردارد ۰ مقدار دهی می‌شوند و این باعث ایجاد بایاس در محاسبات فوق می‌شود از این رو برای مقابله با بایاس ایجاد شده عبارت‌های ۲-۲۳ محاسبه می‌شوند.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{۲-۲۳}$$

و از مقادیر $\hat{m}_t$ و $\hat{v}_t$ برای بروزرسانی پارامترها استفاده می‌شود.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{۲-۲۴}$$

مقدار پیش فرض برای β_1 مقدار ۰٫۹ است و برای β_2 مقدار ۰٫۹۹۹ است. برای ϵ هم پیش فرض مقدار 10^{-8} در نظر گرفته می‌شود.

[۱] Adaptive Moment Estimation [۲] uncentered variance

AdaDelta ۵-۶-۲

این روش بهینه سازی گسترش یافته الگوریتم گرادیان تطبیقی می‌باشد، در الگوریتم گرادیان تطبیقی میزان یادگیری به صورت یکنواخت در حال کاهش است و مربع گرادیان‌های گذشته را به صورت انباشه درون ماتریس G قرار می‌دهد. در الگوریتم AdaDelta پنجره‌ای [۱] از گرادیان‌های گذشته را در روابط خود لحاظ می‌کند. در این روش به جای ذخیره ناکارآمد w مجذور گرادیان‌های قبلی، مجموع گرادیان‌ها به صورت بازگشتی به عنوان میانگین در حال کاهش همه گرادیان‌های مجذور گذشته تعریف می شود.

$$E[g^۲]_t = \gamma E[g^۲]_{t-۱} + (۱ - \gamma) g_t^۲ \tag{۲۵-۲}$$

که در اینجا اصطلاح γ ضریبی با کارکردی مشابه در روش Momentum دارد و به صورت پیش فرض مقدار ۰/۹ برایش در نظر گرفته می‌شود. با تغییر ماتریس G در عبارت ۲-۲۱ با مقدار $E[g^۲]_t$ عبارت ۲-۲۶ حاصل می‌شود.

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^۲]_t + \epsilon}} g_t \tag{۲۶-۲}$$

و از آنجایی که مقدار ریشه کسر عبارت ۲-۲۷ فقط ریشه میانگین مربعات خطای گرادیان می‌باشد می‌توانیم به صورت عبارت ۲-۲۶ بازنویسی‌اش نماییم.

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \tag{۲۷-۲}$$

ارائه دهندگان این متد به علت تفاوت داشتن واحد بروزرسانی این متد را به صورت زیر بازنویسی می‌کنند و دیگر نیازی به تعیین پارامتر η در محاسبات نمی‌باشد.

$$\begin{aligned} E[\Delta\theta^۲]_t &= \gamma E[\Delta\theta^۲]_{t-۱} + (۱ - \gamma)\Delta\theta_t^۲ \\ RMS[\Delta\theta]_t &= \sqrt{E[\Delta\theta^۲]_t + \epsilon} \end{aligned} \tag{۲۸-۲}$$

[۱] Window

RProp ۶-۶-۲

در بهینه سازی به کمک گرادیان کاهشی مشکلی وجود دارد و آن تفاوت اندازه‌های گرادیان‌ها می‌باشد، برخی از گرادیان‌ها ممکن است اندازه‌ای بسیار کوچک و بعضی دیگر ممکن است اندازه‌ای بسیار بزرگی داشته باشند، این اختلاف گرادیان‌ها در پیدا کردن نقطه بهینه برای مسئله مشکل ایجاد می‌کند. الگوریتم بهینه‌سازی پس‌انتشار ارتجاعی[۱][۲۵] که به صورت مخفف RProp نامیده می‌شود سعی می‌کند با کمک علامت گرادیان‌ها این مشکل را حل کنند، در این روش اندازه نرخ یادگیری به صورت پویا محاسبه می‌شود.

به طور مشخص، RProp از یک اندازه گام متفاوت برای هر بعد استفاده می‌کند. مقدار $\eta_i^{(t)}$ نرخ یادگیری برای وزن i در تکرار tام در نظر گرفته می‌شود. مقدار تکرار اول و دوم، $\eta_i^{(۰)}$ و $\eta_i^{(۱)}$، یک ابرپارمتر هستند. برای پیاده سازی این طرح به روز رسانی، از عبارت ۲-۲۹ استفاده می شود. در این رابطه $\alpha > ۱ > \beta$ هستند. و بسته به اینکه سرعت باید افزایش یا کاهش یابد، اندازه نرخ یادگیری مقیاس می‌شود.

$$\eta_i^{(t)} = \begin{cases} \min(\eta_i^{(t-۱)} * \alpha, \eta_{\max}) & \text{if} \frac{\partial C^{(t)}}{\partial w_i^{(t)}} * \frac{\partial C^{(t-۱)}}{\partial w_i^{(t-۱)}} > ۰ \\ \max(\eta_i^{(t-۱)} * \beta, \eta_{\min}) & \text{if} \frac{\partial C^{(t)}}{\partial w_i^{(t)}} * \frac{\partial C^{(t-۱)}}{\partial w_i^{(t-۱)}} < ۰ \\ \eta_i^{(t-۱)} & \text{otherwise} \end{cases} \quad (۲۹-۲)$$

و با این متد تضمین می‌شود که تمام گرادیان‌هایی که برای بروزرسانی وزن‌های شبکه عصبی استفاده می‌شود داری اندازه‌ای یکسان هستند. این الگوریتم کمک می‌کند نقطه بهینه در نقاط زینی[۲] و فلات‌ها[۳] گیر نکنند زیرا گرادیان‌های کوچک بر روی وزن‌ها تاثیری بزرگتر از اندازه واقعی‌شان خواهند گذاشت.

RMSProp ۷-۶-۲

RMSprop یک روش نرخ‌یادگیری تطبیقی است که در هیچ مجله‌ای منتشر نشده است و توسط دکتر جفری هینتون [۴] در سخنرانی ۶e در کلاس‌های ایشان در Coursera ارائه شده است. RMSprop و Adadelta هر دو به طور مستقل همزمان توسعه یافته‌اند که ناشی از نیاز به حل مشکل کاهش شدید نرخ یادگیری متد گرادیان تطبیقی است. RMSprop در واقع مشابه اولین بردار به روز رسانی Adadelta است که در بالا استخراج کردیم:

---

[۱] Resilient Propagation [۲] saddle points [۳] plateaus [۴] Jeffrey Hinton

$$E[g^۲]_t = \gamma E[g^۲]_{t-۱} + (۱-\gamma) g_t^۲$$
$$\theta_{t+۱} = \theta_t - \frac{\eta}{\sqrt{E[g^۲]_t + \epsilon}} g_t \qquad (۳۰-۲)$$

مشابه به AdaDelta این متد هم نرخ یادگیری η را بر مجذوز میانگین مربعات نرخ خطا تقسیم می‌کند. پیشنهاد دکتر هینتون برای مقدار γ عدد ۰/۹ می‌باشد و مقدار بهینه را برای نرخ یادگیری η را مقدار ۰/۰۰۱ عنوان می‌کنند.

۲-۶-۸ بهینه سازی دسته‌ای بزرگ

آموزش شبکه های عصبی عمیق که دارای پارامترهای زیادی هستند، در زمان آموزش مجموعه داده‌های عظیم از نظر محاسباتی بسیار چالش برانگیز هستند. یک راه متداول برای سرعت بخشیدن به آموزش شبکه های پیچشی بزرگ، اضافه کردن واحدهای محاسباتی GPU موازی می‌باشد. در این روش داده‌ها به دسته‌های کوچک تقسیم می‌شوند و بین واحدهای محاسباتی توزیع می‌شوند. با افزایش تعداد گره‌ها، اندازه دسته افزایش می‌یابد. اما آموزش با اندازه دسته بزرگ اغلب منجر به دقت کمتر مدل می‌شود[۲۶]. با ارائه متد بهینه سازی دسته‌ای بزرگ که به صورت خلاصه LAMB نامیده می‌شود، می‌توان اندازه دسته‌ها را به میزان محدودیت حافظه افزایش داد. در روش بهینه سازی LAMB بر اساس گزارش ارائه شده در [۲۶] با افزایش اندازه دسته‌ها تا مقدار ۳۲۸۶۸ بدون کاهش کارایی می‌تواند شبکه BERT[۱] را از ۳ روز به ۷۶ دقیقه کاهش دهد.

الگوریتم بهینه‌سازی LAMB برای هر دسته آموزشی x و در مرحله آموزش i، مقدار گرادیان $g_l^j(i)$ را بر روی وزن‌های $w_l^j(i)$ محاسبه می‌کند، و در مرحله بعد نرمال شده نرم ۲ گرادیان‌ها (عبارت ۲-۳۱) را محاسبه می‌نماید.

$$\hat{g}_l^j(i) = g_l^j(i) / \left\| g(i) \right\|_۲ \qquad (۳۱-۲)$$

سپس مقدار متغییرهای سرعت[۲] (رابطه ۲-۳۳) و مومنتوم (رابطه ۲-۳۲) محاسبه میشود.

$$m_l^j(i) = \beta_۱ m_l^j(i-۱) + (۱-\beta_۱)\hat{g}_l^j(i) \qquad (۳۲-۲)$$

$$v_l^j(i) = \beta_۲ v_l^j(i-۱) + (۱-\beta_۲)(\hat{g}_l^j(i))^۲ \qquad (۳۳-۲)$$

---

[۱] BERT مدل شبکه عصبی در حوزه پردازش زبان طبیعی می‌باشد که توسط گوگل توسعه یافته است.

[۲] Velocity

و همانطور که در الگوریتم بهینه سازی Adam (۲-۶-۴) بیان شد، در مرحله شروع مقدار دو متغییر $m_۰$ و $v_۰$ با بردارد ۰ مقدار دهی می‌شوند و این مقدار دهی اولیه ۰ باعث ایجاد بایاس در محاسبات می‌شود از این رو برای مقابله با بایاس ایجاد شده عبارت‌های ۲-۳۴ محاسبه می‌شوند و مقادیر $\hat{m}_l^j(i)$ و $\hat{v}_l^j(i)$ حاصل می‌شوند.

$$\hat{m}_l^j(i) = \frac{m_l^j(i)}{۱ - \beta_l^j}$$
$$\hat{v}_l^j(i) = \frac{v_l^j(i)}{۱ - \beta_۲^j} \qquad (۲-۳۴)$$

در مرحله بعد پارامتر بروزرسانی[۱]، از رابطه $u_l^j(i) = \frac{\hat{m_l^j(i)}}{\sqrt{\hat{v}_l^j(i)+\epsilon}} + \gamma w_l^j(i)$ که در γ محاسبه می‌شود که [۲] پارامتر کاهش وزن است. و در آخرین مرحله برای هر لایه l بر اساس نرم ۲ نرخ u_l و نرم ۲ وزن لایه w_l مقدار r_l بر اساس رابطه ۲-۳۵ محاسبه می‌گردد.

$$r_l(i) = \frac{\|w_l(i)\|_۲}{\|u_l(i)\|_۲} \qquad (۲-۳۵)$$

[۱] update [۲] wight decay

فصل ۳

بهبود وضوح تصویر

در این بخش دقت و سرعت ۵ روش بهبود وضوح تصوبر در دسته بندی روش‌های مبتنی بر نمونه شامل متدهای: SRCNN، FSRCNN، MemNet، SubPixel، VDSR که توسط الگوریتم‌های بهینه‌سازی برپایه گرادیان شامل: AdaDelta، AdaGrad، Adam، AdaMax، Lamb، RMSProp، RProp، SGD مورد بررسی قرار گرفته است. تمام متدهای بهبود وضوح تصویر ارائه شده در این آزمایش برای افزایش ابعاد تصویر ارائه شده بودند تا تصاویر با ابعد دارای طول و عرض ورودی را به تصویر با ابعاد ۲ برابر، ۳ برابر و . . . تبدیل کنند. اما با تغییر در لایه‌ها، شبکه‌های عصبی به گونه‌ای تغییر داده شدند که تمرکز شبکه‌های عصبی به جای آنکه بر بزرگتر کردن اندازه تصویرها باشد بر روی وضوح تصویر بدون تغییر اندازه تصویر قرار بگیرد. فصل فوق و مطالب پیش رو نتایج پیاده سازی فوق می‌باشد.

۳-۱ داده‌ها

داده‌های آموزشی بر اساس نقشه‌برداری هوایی به دقت ۴ سانتیمتر از دانشگاه زوریخ دانلود شده‌اند (نمونه تصویر ۳-۱) و طی فرآیندی که جلوتر توضیح داده می‌شود، مراحل تهیه و تولید دیتاست بیان می‌شود. تصاویر نقشه اصلی به تصاویر کاشی [۲] به ابعاد مربعی به طول و ارتفاع ۱۲۸ پیکسل تبدیل شده‌اند، هر کدام از این کاشی‌ها به عنوان تصویر اصلی[۳] یا همان y در شبکه عصبی در نظر میگیریم. و از این تصاویر کاشی، برای تولید تصاویر آموزشی x استفاده می‌کنیم، برای تولید x تصاویر کاشی اصلی y را یک چهارم کوچکتر می‌کنیم و مجدد تصاویر را با توابع کتابخانه cv2 به اندازه قبل خود بر میگردانیم. برنامه نوشته شده تبدیل عکس به کاشی در پیوست آ در دسترس می‌باشد.

---

[۲] Tile [۳] Original

شکل ۳-۱: تصویر هوایی تهیه شده از دانشگاه زوریخ

۳-۲ بررسی خروجی روش‌ها

در این بخش خروجی کد توسعه داده شده متدهای بهبود وضوح تصویر در قالب جداول، گراف‌ها و تصاویر مورد بررسی قرار می‌گیرد. برنامه توسعه داده شده بر روی ماشین‌های مجازی سیستم آنلاین کگل [۱] اجرا شده است. ماشین مجازی مورد استفاده کگل دارای ۱۳GB رم، کارت گرافیک Tesla P100-PCIE دارای ۱۶GB رم و پردازه دو هسته‌ای Intel(R) Xeon(R) CPU @ 2.00GHz بوده است، زمان اجرا کد بر روی ماشین فوق حدود ۶ ساعت بوده است.

۳-۲-۱ جداول نتایج

در جدول ۳-۲ بهترین (کمترین) خطا بر اساس سطر جدول یعنی الگوریتم‌های بهینه‌سازی، علامت گذاری شده‌اند. همانگونه که مشاهده می‌شود الگوریتم بهینه‌سازی MemNet و VDSR در تمام الگوریتم‌های بهینه‌سازی توانسته‌اند بهترین نقطه بهینه به نسبت سایر الگوریتم‌ها برسند. و در جدول ۳-۳ علامت گذاری بهترین‌ها بر اساس ستون جدول گزارشات انجام شده است و مشاهده می‌شود که الگوریتم‌های Lamb و RProp بهترین عملکرد را از خود نشان داده‌اند. با بررسی جداول ۳-۴ و ۳-۵ که بر روی داده‌های تست انجام شده است نیز نتایج بالا مجدد تکرار شده است یعنی بهترین نتایج برای روش‌های بهینه‌سازی MemNet و VDSR و بهترین الگوریتم بهینه سازی برای Lamb و RProp بدست آمده است.

و از منظر زمان اجرای الگوریتم‌های بهینه‌سازی و متدهای بهبود وضوح تصویر، از داده‌های جدول ۳-۶ استنباط می‌شود که متد SRCNN سریعترین متد در بین سایر متدها بوده است. و پس از آن در جایگاه دوم متد FSRCNN قرار دارد.

[۱] Kaggle

Report MSE on train data

method	AdaDelta	AdaGrad	AdaMax	Adam	Lamb	RMSProp	RProp	SGD
FSRCNN	0.004217	0.005548	0.004729	0.004010	0.003178	0.208352	0.001491	0.004512
MemNet	0.002135	0.001514	0.001599	0.001626	0.001436	0.001643	0.001554	0.001885
SRCNN	0.004264	0.003175	0.002828	0.002689	0.001895	0.005902	0.001459	0.003826
SubPixel	0.002257	0.002675	0.002658	0.003092	0.002090	0.005478	0.001410	0.002264
VDSR	0.001647	0.001643	0.001643	0.001643	0.001321	0.001687	0.001344	0.001647

شکل ۳-۲: در این جدول گزارش MSE تمام روش‌های اجرا شده بر روی داده‌های آموزشی وجود دارد که بهترین روش‌های SR با رنگ تیره‌تر مشخص شده‌اند

Report MSE on train data

method	AdaDelta	AdaGrad	AdaMax	Adam	Lamb	RMSProp	RProp	SGD
FSRCNN	0.004217	0.005548	0.004729	0.004010	0.003178	0.208352	0.001491	0.004512
MemNet	0.002135	0.001514	0.001599	0.001626	0.001436	0.001643	0.001554	0.001885
SRCNN	0.004264	0.003175	0.002828	0.002689	0.001895	0.005902	0.001459	0.003826
SubPixel	0.002257	0.002675	0.002658	0.003092	0.002090	0.005478	0.001410	0.002264
VDSR	0.001647	0.001643	0.001643	0.001643	0.001321	0.001687	0.001344	0.001647

شکل ۳-۳: در این جدول گزارش MSE تمام روش‌های اجرا شده بر روی داده‌های آموزشی وجود دارد که بهترین روش‌های بهینه‌سازی با رنگ تیره‌تر مشخص شده‌اند

Report MSE on test data

method	AdaDelta	AdaGrad	AdaMax	Adam	Lamb	RMSProp	RProp	SGD
FSRCNN	0.004370	0.005689	0.004877	0.004130	0.001765	0.151378	0.001640	0.004683
MemNet	0.001878	0.001680	0.001764	0.001793	0.001591	0.001822	0.001741	0.001847
SRCNN	0.004376	0.003303	0.002969	0.002852	0.001839	0.004896	0.001607	0.003963
SubPixel	0.002418	0.002827	0.002796	0.003264	0.002030	0.005288	0.001560	0.002427
VDSR	0.001822	0.001822	0.001822	0.001822	0.001465	0.001822	0.001489	0.001822

شکل ۳-۴: در این جدول گزارش MSE تمام روش‌های اجرا شده بر روی داده‌های آزمایشی وجود دارد که بهترین روش‌های SR با رنگ تیره‌تر مشخص شده‌اند

Report MSE on test data

method	AdaDelta	AdaGrad	AdaMax	Adam	Lamb	RMSProp	RProp	SGD
FSRCNN	0.004370	0.005689	0.004877	0.004130	0.001765	0.151378	0.001640	0.004683
MemNet	0.001878	0.001680	0.001764	0.001793	0.001591	0.001822	0.001741	0.001847
SRCNN	0.004376	0.003303	0.002969	0.002852	0.001839	0.004896	0.001607	0.003963
SubPixel	0.002418	0.002827	0.002796	0.003264	0.002030	0.005288	0.001560	0.002427
VDSR	0.001822	0.001822	0.001822	0.001822	0.001465	0.001822	0.001489	0.001822

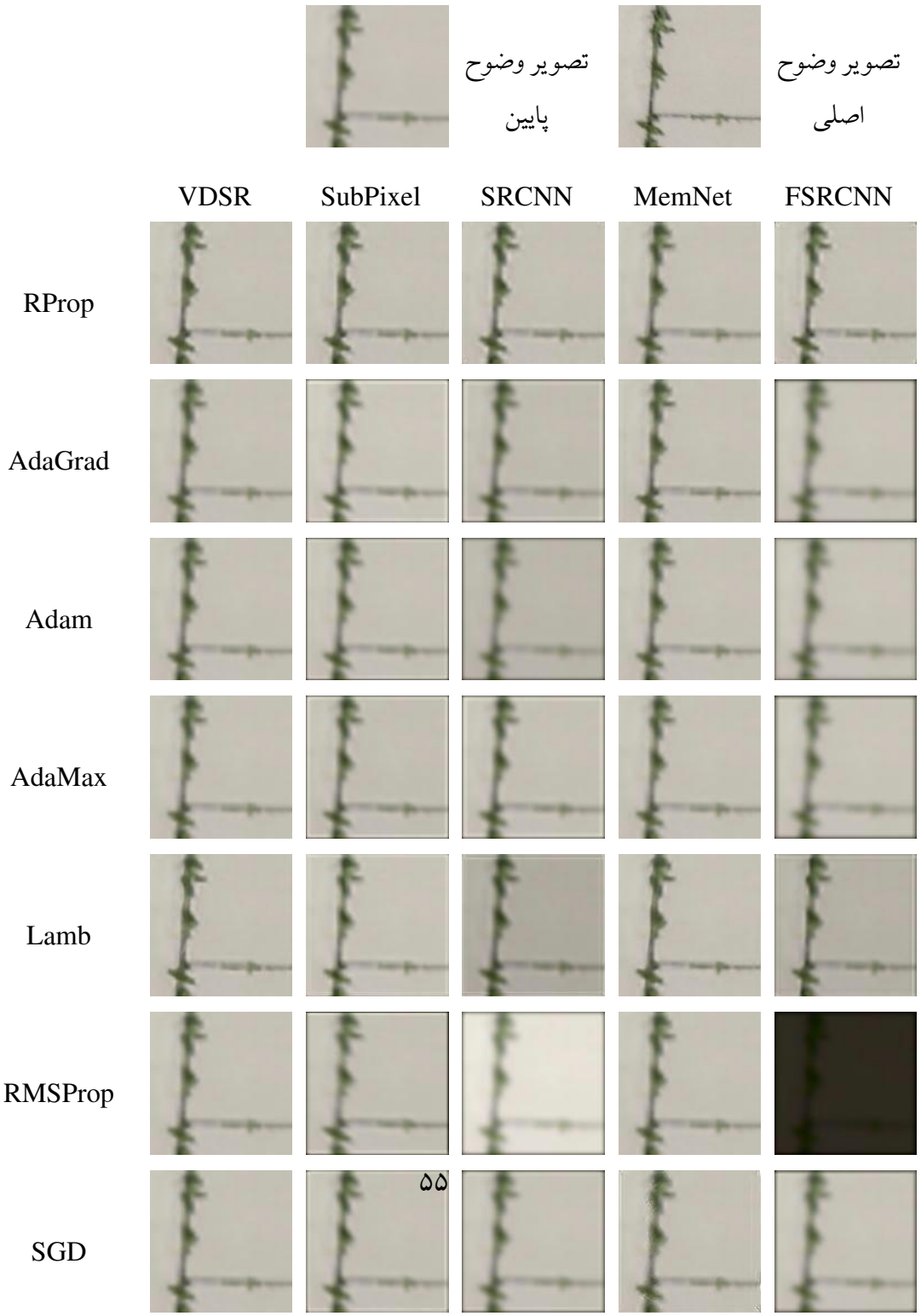
شکل ۳-۵: در این جدول گزارش MSE تمام روش‌های اجرا شده بر روی داده‌های آزمایشی وجود دارد که بهترین روش‌های بهینه‌سازی با رنگ تیره‌تر مشخص شده‌اند

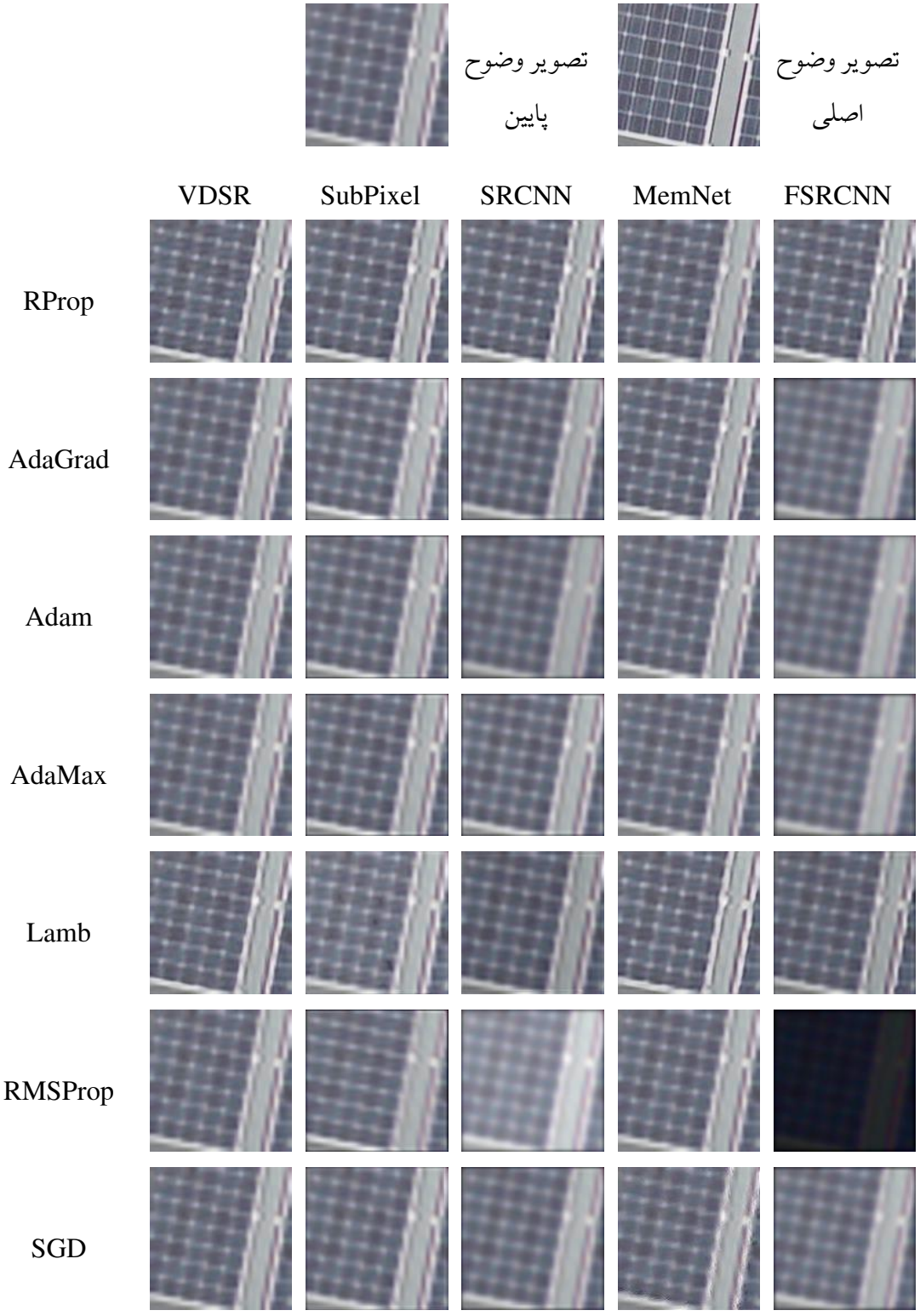
Report time on train data

method	AdaDelta	AdaGrad	AdaMax	Adam	Lamb	RMSProp	RProp	SGD
FSRCNN	4.233476	4.270683	4.376673	4.308409	4.447297	4.269916	4.430624	4.195116
MemNet	5.501371	5.376960	5.426692	5.528989	5.762273	5.547517	5.866873	5.540056
SRCNN	3.539109	3.507294	3.732320	3.528203	3.576241	3.649217	3.900883	3.516654
SubPixel	4.340369	4.404957	4.389576	4.216705	4.506655	4.403957	4.391614	4.223523
VDSR	5.707761	5.735224	5.785811	5.808139	5.545127	5.693678	5.797860	5.714559

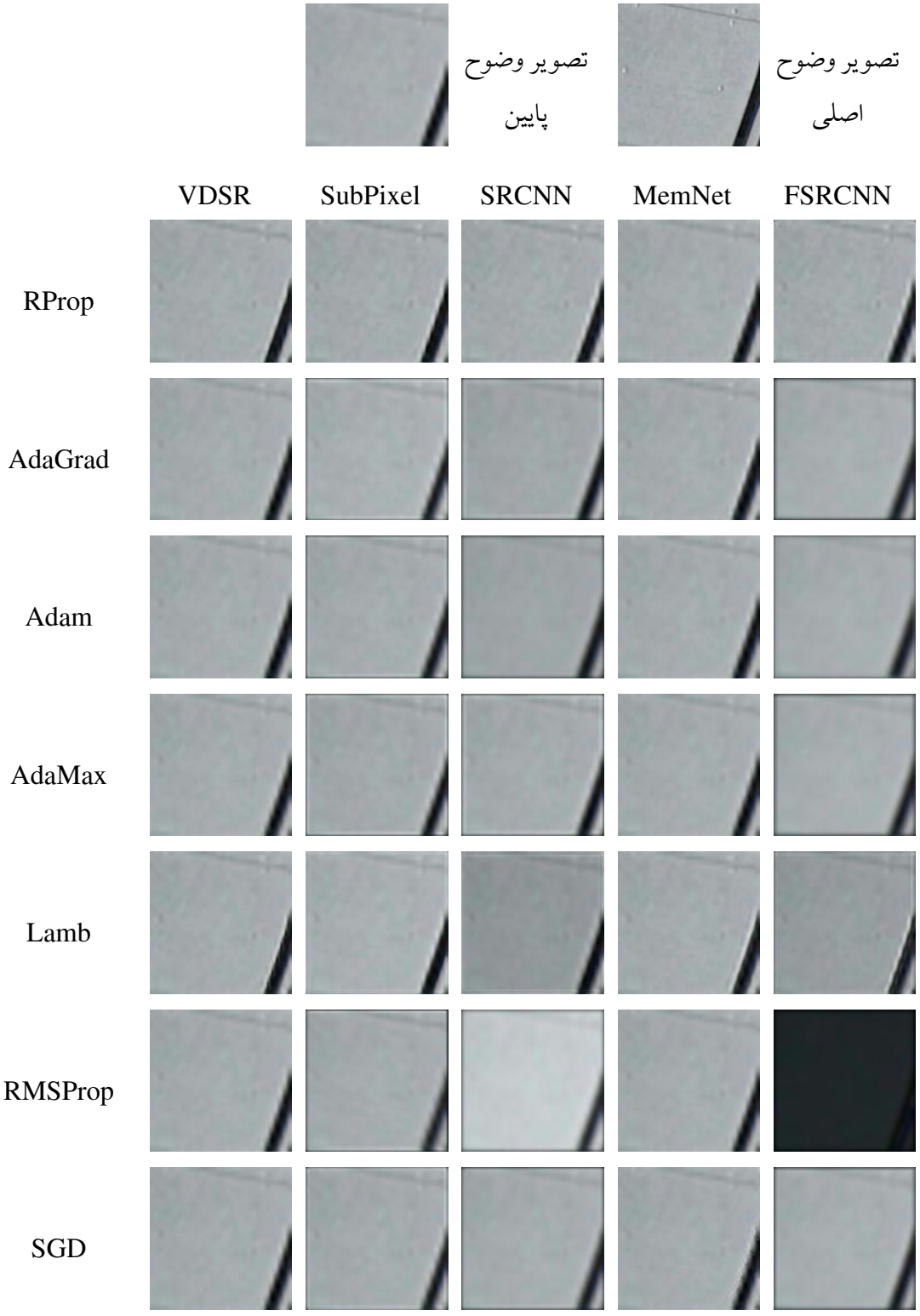
شکل ۳-۶: جدول زمان آموزش الگوریتم‌های در مرحله آموزش شبکه عصبی

۳-۲-۲ نمونه

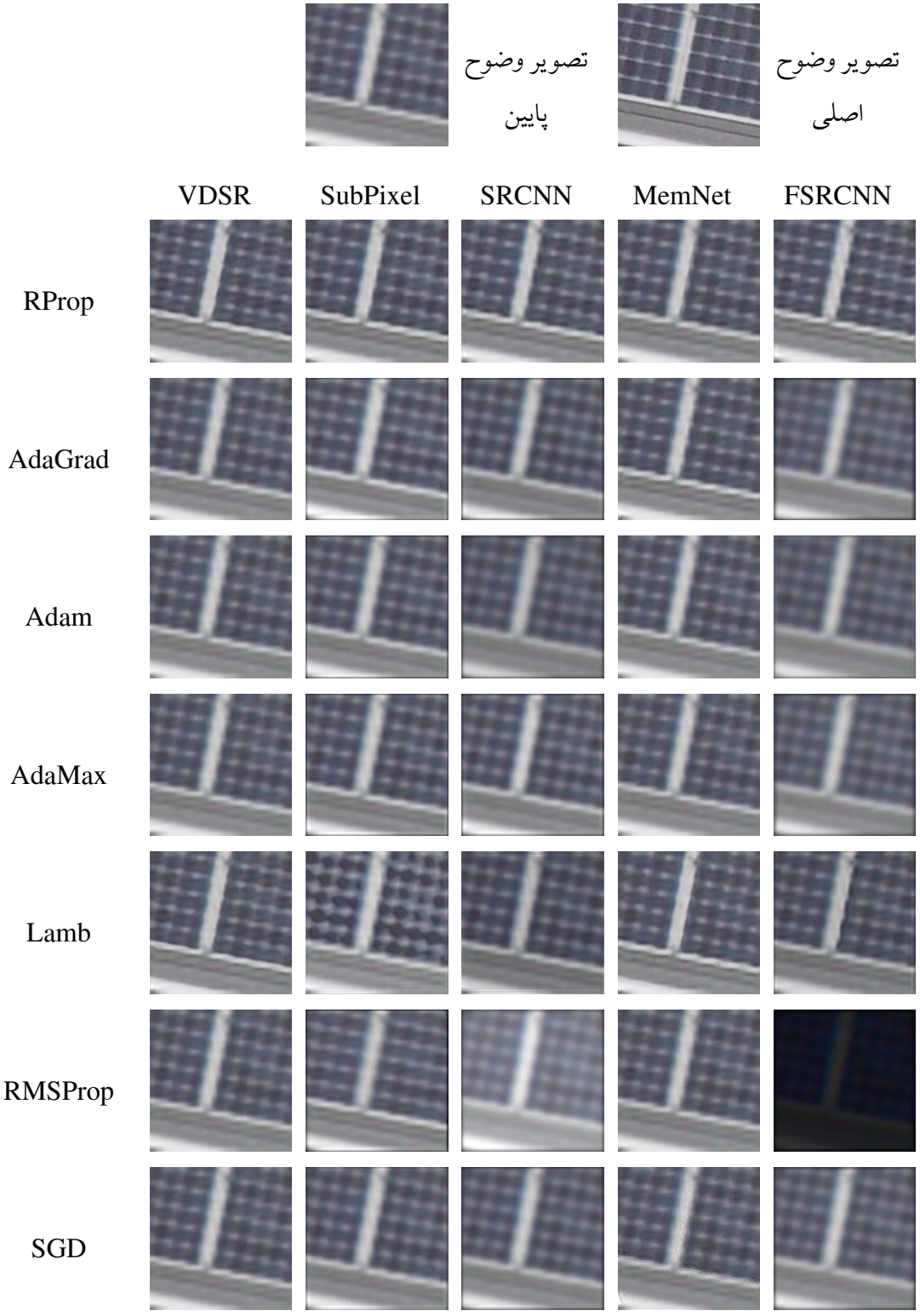




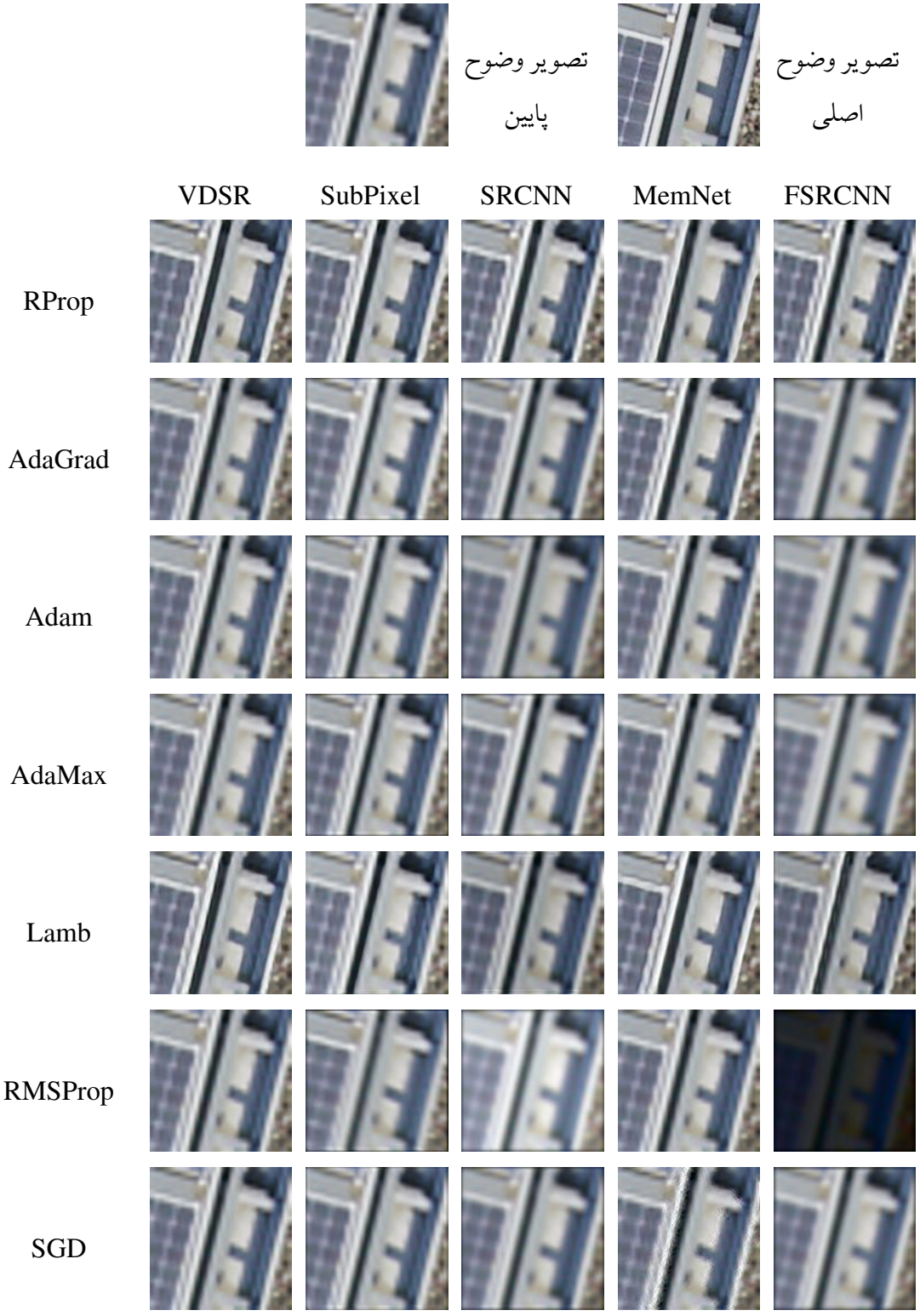
جدول ۳-۲: تصویر نمونه ۲



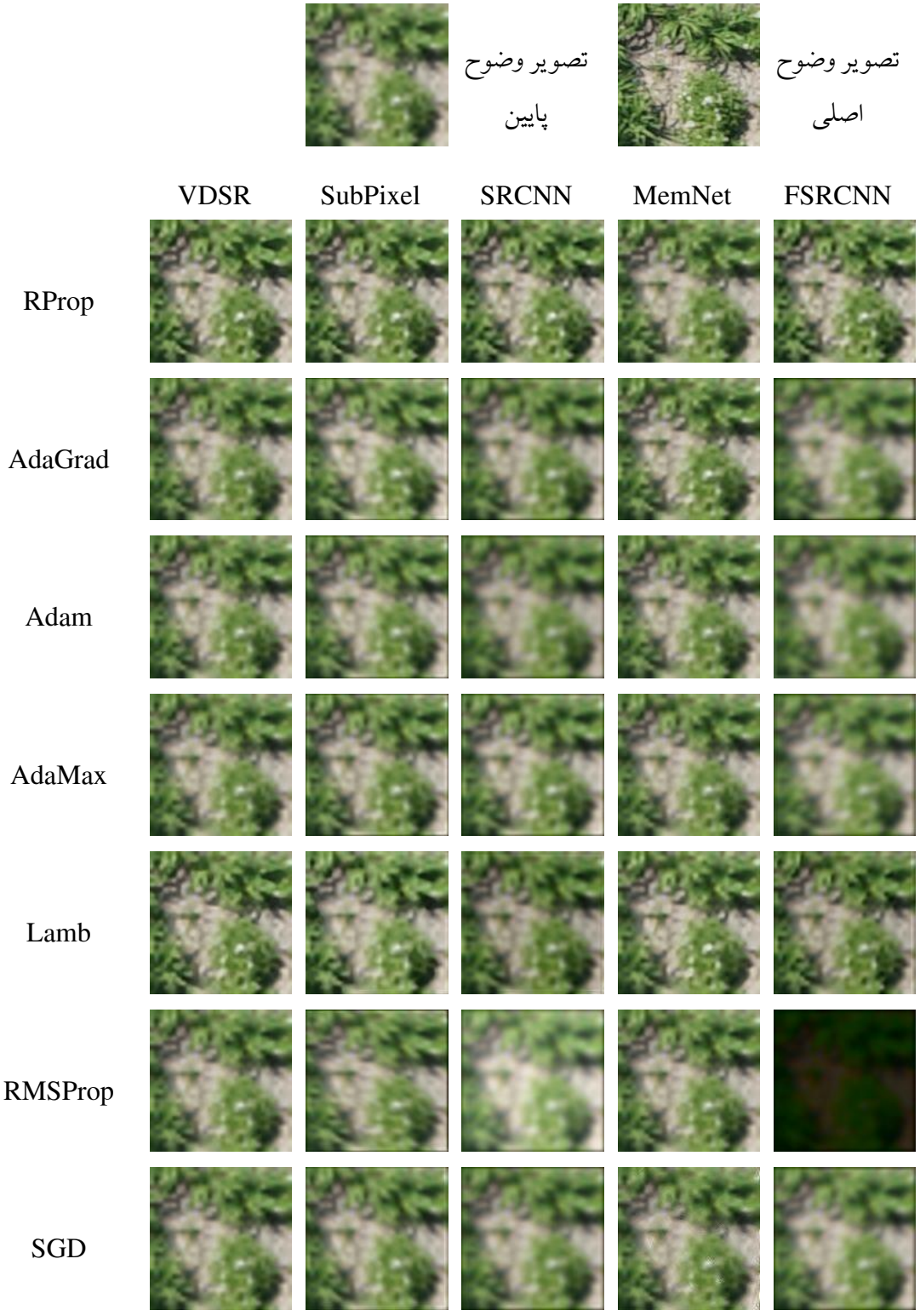
جدول ۳-۳: تصویر نمونه ۳



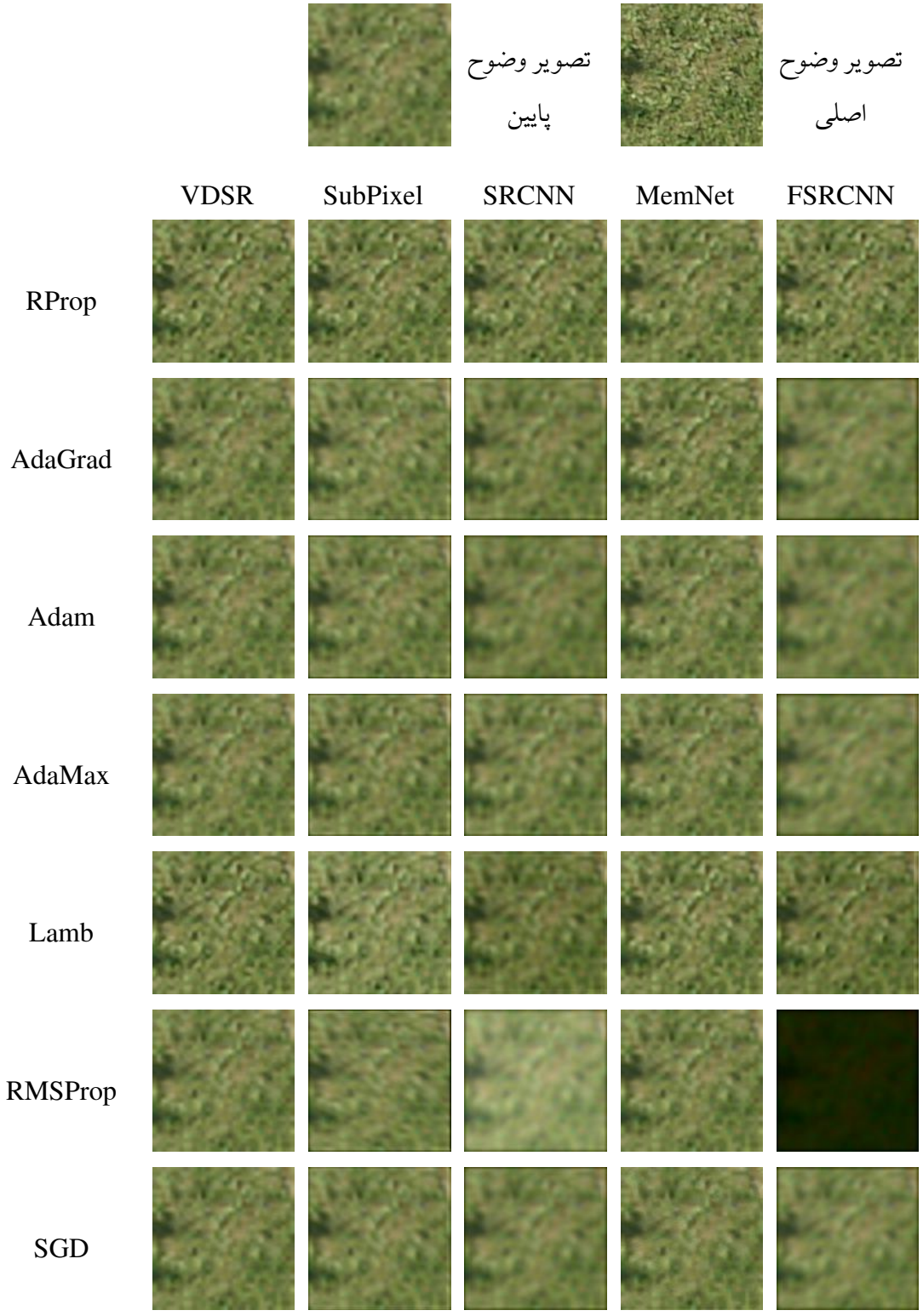
جدول ۳-۴: تصویر نمونه ۴



جدول ۳-۵: تصویر نمونه ۵



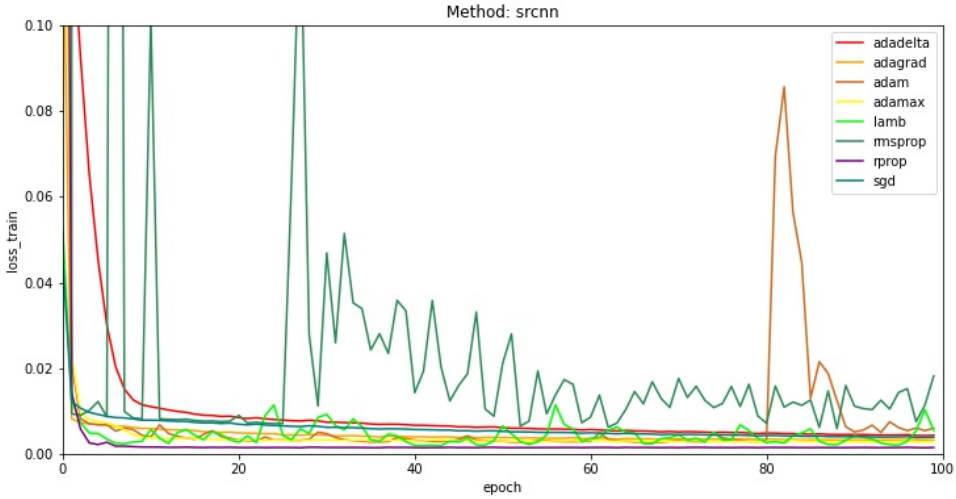
جدول ۳-۶: تصویر نمونه ۶



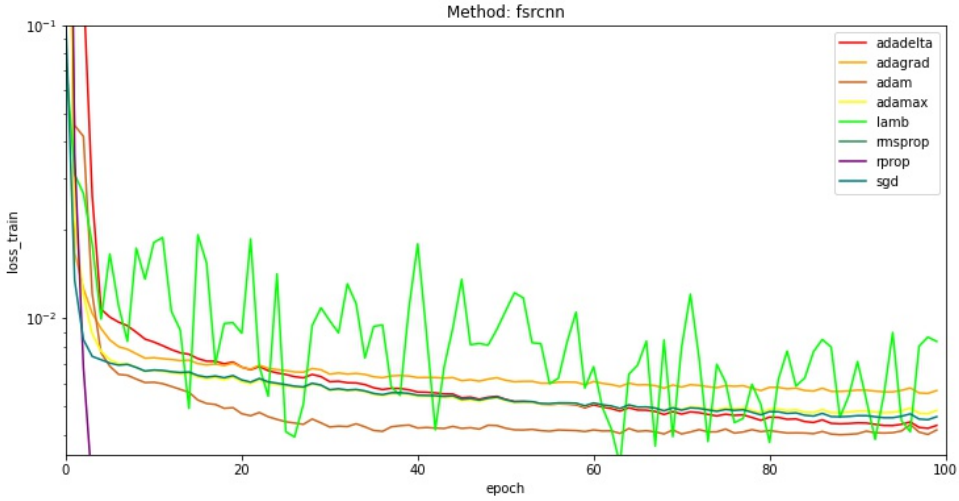
جدول ۳-۷: تصویر نمونه ۷

۳-۲-۳ نمودار نتایج

در این قسمت نمودارهایی به تفکیک برای روش‌های بهبود وضوح تصویر SRCNN، FSRCNN، MemNet، SubPixel و VDSR برای ۱۰۰ ایپاک آموزش برای تمام الگوریتم‌های بهینه‌سازی استفاده شده ترسیم شده‌اند. هر نمودار نمایش دهنده میزان خطای میانگین مربعات خطا در هر دوره آموزش می‌باشد.در تصویر ۳-۷ مربوط به شبکه عصبی SRCNN مشاهده می‌شود بهترین عملکرد را الگوریتم بهینه‌سازی RProp از خود نشان داده‌است، در کمترین ایپاک توانسته است به نقطه بهینه همگرا شود. و الگوریتم بهینه‌سازی SGD رفتاری تصادفی دارای تکانش‌های شدید در همگرا شدن به نقطه بهینه از خود نشان داده است.



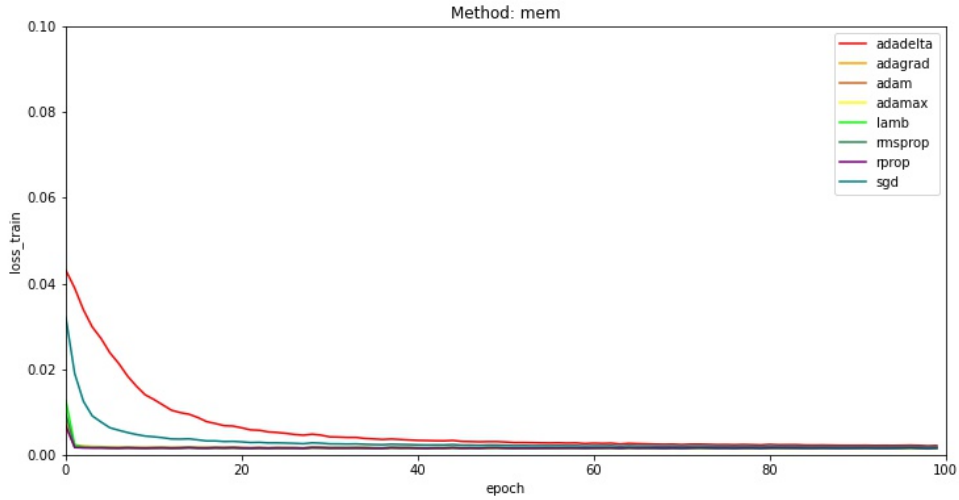
شکل ۳-۷: مقایسه الگوریتم‌های مختلف بهینه‌سازی در همگرا شدن به نقطه بهینه و گزارش خطای MSE در متد SRCNN در ۱۰۰ مرحله آموزش داده‌های آموزشی



شکل ۳-۸: نمودار MSE آموزش الگوریتم‌های مختلف بهینه‌سازی متد FSRCNN

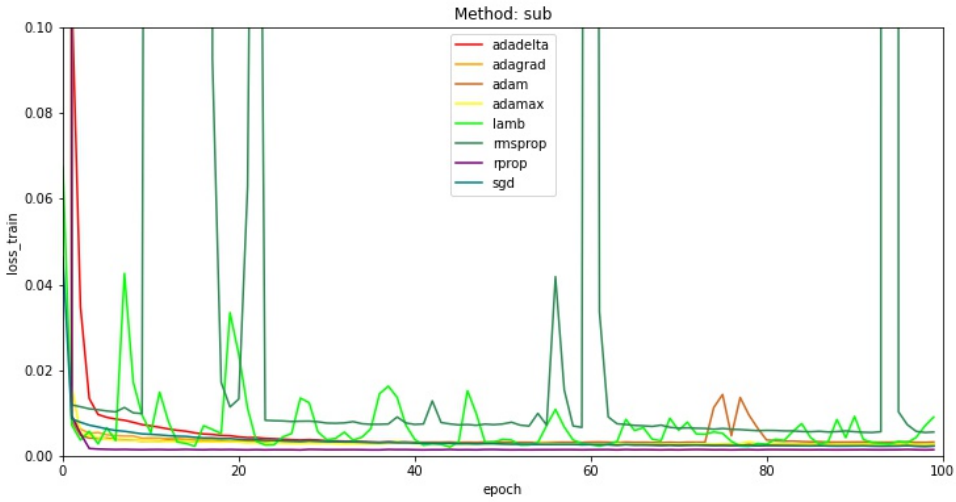
در تصویر ۳-۸ نتایج تمام الگوریتم‌های بهینه‌سازی متد FSRCNN در ۱۰۰ ایپاک ترسیم شده است. مشاهده می‌شود که الگوریتم بهینه‌سازی RProp سریعترین همگرایی را داشته است، و پس از آن الگوریتم Adam توانسته است در ایپاک کمتری به نقطه بهینه برسد. و از طرفی مشاهده می‌شود که الگوریتم بهینه‌سازی Lamb رفتاری تصادفی دارد.

در تصویر ۳-۹ مربوط به آموزش شبکه‌عصبی MemNet به غیر از الگوریتم‌های بهینه‌سازی AdaDelta و SGD سایر الگوریتم‌ها همگرایی سریعی به نقطه بهینه داشته‌اند مخصوصا RProp که سریعتر از الگوریتم‌های دیگر به نقطه بهینه همگرا شده است.



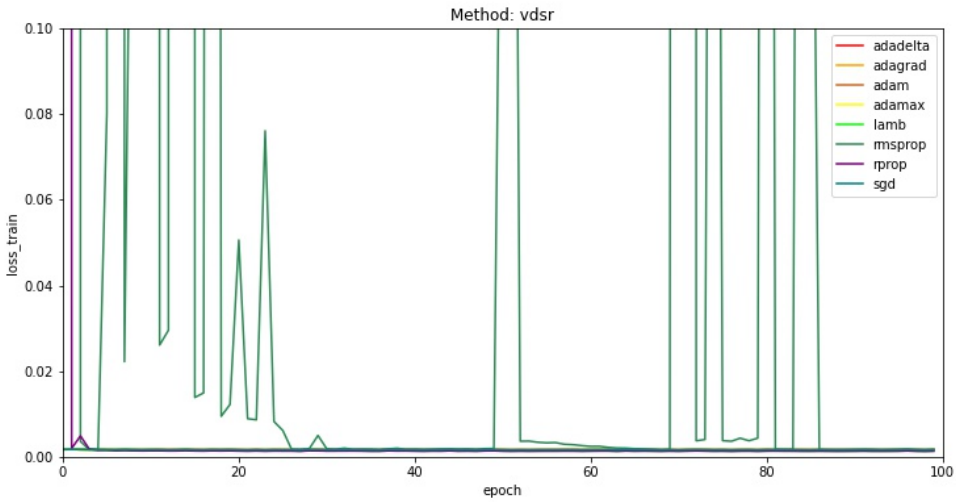
شکل ۳-۹: مقایسه الگوریتم‌های مختلف بهینه‌سازی در همگرا شدن به نقطه بهینه و گزارش خطای MSE در متد MemNet در ۱۰۰ مرحله آموزش داده‌های آموزشی

در تصویر ۳-۱۰ متد SubPixel مشابه متد SRCNN الگوریتم بهینه‌سازی RProp سریعترین همگرایی را به نقطه بهینه داشته است.

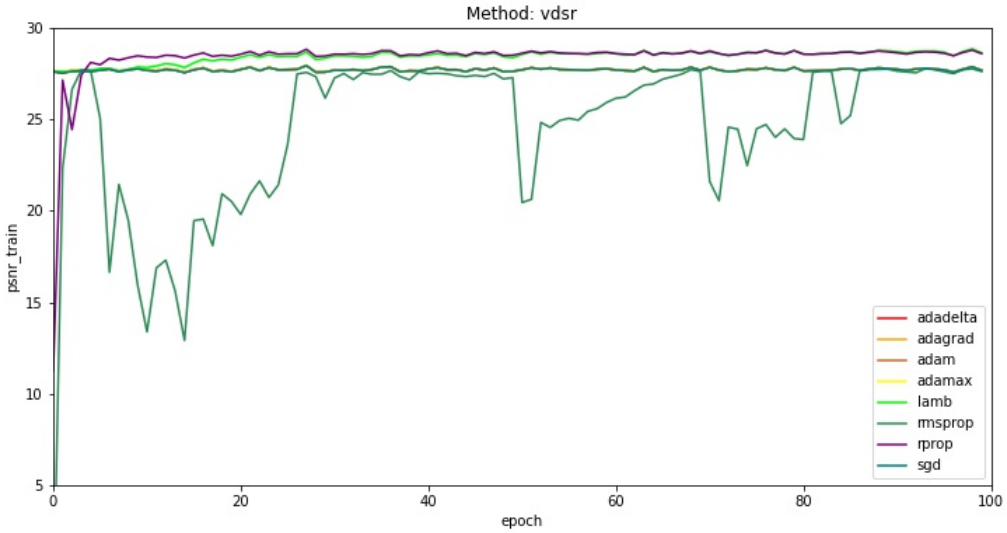


شکل ۳-۱۰: مقایسه الگوریتم‌های مختلف بهینه‌سازی در همگرا شدن به نقطه بهینه و گزارش خطای MSE در متد SubPixel در ۱۰۰ مرحله آموزش داده‌های آموزشی

در تصویر ۳-۱۱ مربوط به متد VDSR نمودار خطای MSE الگوریتم‌های مختلف بهینه‌سازی گزارش شده است، از روی نمودار MSE فوق نمی‌توان برداشت مناسبی داشت، با ترسیم نمودار PSNR تصویر ۳-۱۲ برای ۱۰۰ اپیاک آموزش مشاده می‌شود که الگوریتم بهینه سازی RProp توانسته است بهترین زمان و پایداری را برای رسیدن به نقطه بهینه داشته باشد.



شکل ۳-۱۱: مقایسه الگوریتم‌های مختلف بهینه‌سازی در همگرا شدن به نقطه بهینه و گزارش خطای MSE در متد vdsr در ۱۰۰ مرحله آموزش داده‌های آموزشی



شکل ۳-۱۲: مقایسه الگوریتم‌های مختلف بهینه‌سازی در همگرا شدن به نقطه بهینه و گزارش خطای PSNR در متد VDSR در ۱۰۰ مرحله آموزش داده‌های آموزشی

۳-۳ نتیجه گیری

بر اساس گزارشات ارائه شده در ۳-۲-۱ سریعترین الگوریتم‌های بهینه‌سازی در بهبود وضوح تصویر با تغییراتی که در شبکه‌های عصبی داده شده بود، الگوریتم RProp و Lamb می‌باشند و بهترین متد بهبود وضوح تصویر متدهای MemNet و VDSR می‌باشد.

فهرست منابع

[1] Dong, Chao, Loy, Chen Change, He, Kaiming, and Tang, Xiaoou. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[2] Dong, Chao, Loy, Chen Change, and Tang, Xiaoou. Accelerating the super-resolution convolutional neural network. in *European conference on computer vision*, pp. 391–407. Springer, 2016.

[3] Shi, Wenzhe, Caballero, Jose, Huszár, Ferenc, Totz, Johannes, Aitken, Andrew P., Bishop, Rob, Rueckert, Daniel, and Wang, Zehan. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[4] Kim, Jiwon, Lee, Jung Kwon, and Lee, Kyoung Mu. Accurate image super-resolution using very deep convolutional networks. in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016.

[5] Wang, Zhihao, Chen, Jian, and Hoi, Steven C. H. Deep learning for image super-resolution: A survey, 2020.

[6] Prasad, Navneel, Singh, Rajeshni, and Lal, Sunil Pranit. Comparison of back propagation and resilient propagation algorithm for spam classification. in *2013 Fifth international conference on computational intelligence, modelling and simulation*, pp. 29–34. IEEE, 2013.

[7] Tucker, Ledyard R. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, Sep 1966.

[8] Dumoulin, Vincent and Visin, Francesco. A guide to convolution arithmetic for deep learning, 2018.

[9] Umehara, Kensuke, Ota, Junko, and Ishida, Takayuki. Application of super-resolution convolutional neural network for enhancing image resolution in chest ct. *Journal of Digital Imaging*, 31:441–450, 08 2018.

[10] Tai, Ying, Yang, Jian, Liu, Xiaoming, and Xu, Chunyan. Memnet: A persistent memory network for image restoration. in *Proceedings of International Conference on Computer Vision*, 2017.

[11] Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[12] Nielsen, Michael A. Neural networks and deep learning, 2018.

[13] Hinton, Geoffrey E and Shallice, Tim. Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological review*, 98(1):74, 1991.

[14] Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[15] Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[16] Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[17] Widrow, Bernard and Hoff, Marcian E. Adaptive switching circuits. tech. rep., Stanford Univ Ca Stanford Electronics Labs, 1960.

[18] Bertsekas, D.P. *Nonlinear Programming*. Athena Scientific, 1999.

[19] Chauvin, Yves and Rumelhart, David E. *Backpropagation: theory, architectures, and applications*. Psychology press, 2013.

[20] Nielsen, Michael A. *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA, 2015.

[21] Qian, Ning. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[22] Bengio, Yoshua, Boulanger-Lewandowski, Nicolas, and Pascanu, Razvan. Advances in optimizing recurrent networks, 2012.

[23] Dean, Jeffrey, Corrado, Greg, Monga, Rajat, Chen, Kai, Devin, Matthieu, Mao, Mark, Ranzato, Marc'aurelio, Senior, Andrew, Tucker, Paul, Yang, Ke, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25:1223–1231, 2012.

[24] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. GloVe: Global vectors for word representation. in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[25] Prasad, Navneel, Singh, Rajeshni, and Lal, Sunil Pranit. Comparison of back propagation and resilient propagation algorithm for spam classification. in *2013 Fifth international conference on computational intelligence, modelling and simulation*, pp. 29–34. IEEE, 2013.

[26] You, Yang, Li, Jing, Reddi, Sashank, Hseu, Jonathan, Kumar, Sanjiv, Bhojanapalli, Srinadh, Song, Xiaodan, Demmel, James, Keutzer, Kurt, and Hsieh, Cho-Jui. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

پیوست آ

نمونه کدهای پایان‌نامه

آ-۱ منبع سورس کدهای بهبود وضوح تصویر

منبع برنامه توسعه داده شده برای بهبود وضوح تصویر در آدرس گیت‌هاب [۲]، Sahebi@ [۳] قرار گرفته است. تمام اجراهایی که برروی کگلKaggle و گوگل کولبGoogle Colab از همین منبع استفاده شده است.

آ-۲ کد بروز رسانی دسته‌ای

در فصل ۲ بخش ۲-۵ درباره الگوریتم پس‌انتشار صحبت شد، در زیر پیاده‌سازی‌های الگوریتم پس انتشار خطا، برای یک شبکه مبتنی بر بهینه‌سازی دسته حداقلی [۴] آورده شده است.

```
class Network(object):
        ...
    def update_mini_batch(self, mini_batch, eta):
        """Update the network's weights and biases by applying
        gradient descent using backpropagation to a single mini batch.
        The "mini_batch" is a list of tuples "(x, y)", and "eta"
        is the learning rate."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b
                )]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w
                )]
        self.weights = [w-(eta/len(mini_batch))*nw
```

```
                        for w, nw in zip(self.weights, nabla_w)
                        ]
        self.biases = [b-(eta/len(mini_batch))*nb
                       for b, nb in zip(self.biases, nabla_b)]
```

بیشتر کارهای کد آ-۴ در خط ۱۱ و محاسبه دو متغییر delta_nabla_b و delta_nabla_w انجام می‌شود، که از تابع backprop که خروجی‌اش مشتق‌های جزئی $\partial C_x/\partial b^l_j$ و $\partial C_x/\partial w^l_{jk}$ استفاده می‌کند. تابع backprop پیاده سازی دقیق الگوریتم‌های ۲-۵-۲ است.

آ-۳ کد پس‌انتشار

```
class Network(object):
    ...
    def backprop(self, x, y):
        """Return a tuple "(nabla_b, nabla_w)" representing the
        gradient for the cost function C_x.  "nabla_b" and
        "nabla_w" are layer-by-layer lists of numpy arrays, similar
        to "self.biases" and "self.weights"."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        # feedforward
        activation = x
        activations = [x] # list to store all the activations, layer by
            layer
        zs = [] # list to store all the z vectors, layer by layer
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)
        # backward pass
        delta = self.cost_derivative(activations[-1], y) * \
            sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())
        # Note that the variable l in the loop below is used a little
        # differently to the notation in Chapter 2 of the book.  Here,
        # l = 1 means the last layer of neurons, l = 2 is the
        # second-last layer, and so on.  It's a renumbering of the
        # scheme in the book, used here to take advantage of the fact
        # that Python can use negative indices in lists.
        for l in xrange(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
        return (nabla_b, nabla_w)
```

```
    ...                                                                    ۳۷
    def cost_derivative(self, output_activations, y):                      ۳۸
        """Return the vector of partial derivatives \partial C_x /         ۳۹
           \partial a for the output activations."""                       ۴۰
        return (output_activations-y)                                      ۴۱
                                                                           ۴۲
def sigmoid(z):                                                            ۴۳
    """The sigmoid function."""                                            ۴۴
    return 1.0/(1.0+np.exp(-z))                                            ۴۵
                                                                           ۴۶
def sigmoid_prime(z):                                                      ۴۷
    """Derivative of the sigmoid function."""                              ۴۸
    return sigmoid(z)*(1-sigmoid(z))                                       ۴۹
```

آ-۴ کد ایجاد کاشی

کد زیر تبدیل کننده تصاویر نقشه‌های یک تکه به کاشی‌هایی برای آموزش شبکه عصبی می‌باشد. [۱]

```
from tqdm import tqdm
from PIL import Image
import numpy as np
import torch
import math
import glob
import cv2
import os

def calculate_weights_indices(in_length, out_length, scale, kernel,
    kernel_width, antialiasing):
    if (scale < 1) and (antialiasing):
        # Use a modified kernel to simultaneously interpolate and
            antialias- larger kernel width
        kernel_width = kernel_width / scale

    # Output-space coordinates
    x = torch.linspace(1, out_length, out_length)

    # Input-space coordinates. Calculate the inverse mapping such that
        0.5
    # in output space maps to 0.5 in input space, and 0.5+scale in
        output
    # space maps to 1.5 in input space.
    u = x / scale + 0.5 * (1 - 1 / scale)

    # What is the left-most pixel that can be involved in the
        computation?
    left = torch.floor(u - kernel_width / 2)

    # What is the maximum number of pixels that can be involved in the
    # computation?  Note: it's OK to use an extra pixel here; if the
    # corresponding weights are all zero, it will be eliminated at the
        end
    # of this function.
    P = math.ceil(kernel_width) + 2

    # The indices of the input pixels involved in computing the k-th
        output
    # pixel are in row k of the indices matrix.
    indices = left.view(out_length, 1).expand(out_length, P) + torch.
        linspace(0, P - 1, P).view(
        1, P).expand(out_length, P)

    # The weights used to compute the k-th output pixel are in row k of
        the
    # weights matrix.
    distance_to_center = u.view(out_length, 1).expand(out_length, P) -
        indices

```

```
    # apply cubic kernel
    if (scale < 1) and (antialiasing):
        weights = scale * cubic(distance_to_center * scale)
    else:
        weights = cubic(distance_to_center)
    # Normalize the weights matrix so that each row sums to 1.
    weights_sum = torch.sum(weights, 1).view(out_length, 1)
    weights = weights / weights_sum.expand(out_length, P)

    # If a column in weights is all zero, get rid of it. only consider
        the first and last column.
    weights_zero_tmp = torch.sum((weights == 0), 0)
    if not math.isclose(weights_zero_tmp[0], 0, rel_tol=1e-6):
        indices = indices.narrow(1, 1, P - 2)
        weights = weights.narrow(1, 1, P - 2)
    if not math.isclose(weights_zero_tmp[-1], 0, rel_tol=1e-6):
        indices = indices.narrow(1, 0, P - 2)
        weights = weights.narrow(1, 0, P - 2)
    weights = weights.contiguous()
    indices = indices.contiguous()
    sym_len_s = -indices.min() + 1
    sym_len_e = indices.max() - in_length
    indices = indices + sym_len_s - 1
    return weights, indices, int(sym_len_s), int(sym_len_e)

# matlab 'imresize' function, now only support 'bicubic'
def cubic(x):
    absx = torch.abs(x)
    absx2 = absx**2
    absx3 = absx**3
    return (1.5 * absx3 - 2.5 * absx2 + 1) * (
        (absx <= 1).type_as(absx)) + (-0.5 * absx3 + 2.5 * absx2 - 4 *
            absx + 2) * ((
             (absx > 1) * (absx <= 2)).type_as(absx))

def imresize_np(img, scale, antialiasing=True):
    # Now the scale should be the same for H and W
    # input: img: Numpy, HWC BGR [0,1]
    # output: HWC BGR [0,1] w/o round
    img = torch.from_numpy(img)

    in_H, in_W, in_C = img.size()
    _, out_H, out_W = in_C, math.ceil(in_H * scale), math.ceil(in_W *
        scale)
    kernel_width = 4
    kernel = 'cubic'

    # Return the desired dimension order for performing the resize.
        The
    # strategy is to perform the resize first along the dimension with
        the
    # smallest scale factor.
    # Now we do not support this.

    # get weights and indices
    weights_H, indices_H, sym_len_Hs, sym_len_He =
```

```
    calculate_weights_indices(
     in_H, out_H, scale, kernel, kernel_width, antialiasing)
weights_W, indices_W, sym_len_Ws, sym_len_We =
    calculate_weights_indices(
     in_W, out_W, scale, kernel, kernel_width, antialiasing)
# process H dimension
# symmetric copying
img_aug = torch.FloatTensor(in_H + sym_len_Hs + sym_len_He, in_W,
    in_C)
img_aug.narrow(0, sym_len_Hs, in_H).copy_(img)

sym_patch = img[:sym_len_Hs, :, :]
inv_idx = torch.arange(sym_patch.size(0) - 1, -1, -1).long()
sym_patch_inv = sym_patch.index_select(0, inv_idx)
img_aug.narrow(0, 0, sym_len_Hs).copy_(sym_patch_inv)

sym_patch = img[-sym_len_He:, :, :]
inv_idx = torch.arange(sym_patch.size(0) - 1, -1, -1).long()
sym_patch_inv = sym_patch.index_select(0, inv_idx)
img_aug.narrow(0, sym_len_Hs + in_H, sym_len_He).copy_(
    sym_patch_inv)

out_1 = torch.FloatTensor(out_H, in_W, in_C)
kernel_width = weights_H.size(1)
for i in range(out_H):
    idx = int(indices_H[i][0])
    out_1[i, :, 0] = img_aug[idx:idx + kernel_width, :, 0].
        transpose(0, 1).mv(weights_H[i])
    out_1[i, :, 1] = img_aug[idx:idx + kernel_width, :, 1].
        transpose(0, 1).mv(weights_H[i])
    out_1[i, :, 2] = img_aug[idx:idx + kernel_width, :, 2].
        transpose(0, 1).mv(weights_H[i])

# process W dimension
# symmetric copying
out_1_aug = torch.FloatTensor(out_H, in_W + sym_len_Ws + sym_len_We
    , in_C)
out_1_aug.narrow(1, sym_len_Ws, in_W).copy_(out_1)

sym_patch = out_1[:, :sym_len_Ws, :]
inv_idx = torch.arange(sym_patch.size(1) - 1, -1, -1).long()
sym_patch_inv = sym_patch.index_select(1, inv_idx)
out_1_aug.narrow(1, 0, sym_len_Ws).copy_(sym_patch_inv)

sym_patch = out_1[:, -sym_len_We:, :]
inv_idx = torch.arange(sym_patch.size(1) - 1, -1, -1).long()
sym_patch_inv = sym_patch.index_select(1, inv_idx)
out_1_aug.narrow(1, sym_len_Ws + in_W, sym_len_We).copy_(
    sym_patch_inv)

out_2 = torch.FloatTensor(out_H, out_W, in_C)
kernel_width = weights_W.size(1)
for i in range(out_W):
    idx = int(indices_W[i][0])
    out_2[:, i, 0] = out_1_aug[:, idx:idx + kernel_width, 0].mv(
        weights_W[i])
```

```
        out_2[:, i, 1] = out_1_aug[:, idx:idx + kernel_width, 1].mv(
            weights_W[i])
        out_2[:, i, 2] = out_1_aug[:, idx:idx + kernel_width, 2].mv(
            weights_W[i])

    return out_2.numpy()

def crop(infile,height,width):
    im = Image.open(infile)
    imgwidth, imgheight = im.size
    for i in range(imgheight//height):
        for j in range(imgwidth//width):
            box = (j*width, i*height, (j+1)*width, (i+1)*height)
            yield im.crop(box)

if __name__=='__main__':
    height    = 512
    width     = 512
    start_num = 0
    k = 0
    for ext in ('jpg',):
        file_list = glob.glob(f'.\images\src\*.{ext}')
        for infile in file_list:
            src_path = os.path.dirname(infile)
            if not os.path.isdir(infile):
                for piece in tqdm(crop(infile,height,width)):
                    k += 1
                    img  = Image.new('RGB', (height,width), 255)
                    img.paste(piece)
                    dest = os.path.join('./images/tile/',"tile-%s.png"
                        % k)
                    img.save(dest)

                    # Make Samples
                    image    = cv2.imread(dest)

                    up_scale = 2
                    widthx  = int(np.floor(image.shape[1] / up_scale))
                    heightx = int(np.floor(image.shape[0] / up_scale))

                    # High Resolution Sample
                    image_HR = image[0:up_scale * height, 0:up_scale *
                        width]
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.2.orginal.{ext}'), image_HR)
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.4.orginal.{ext}'), image_HR)
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.8.orginal.{ext}'), image_HR)

                    image_LR2   = imresize_np(image_HR,  1 / 2, True)
                    image_lr2bc = imresize_np(image_LR2, 2, True)
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.2.lr.{ext}'), image_lr2bc)

                    image_LR4   = imresize_np(image_LR2,  1 / 2, True)
```

```
                    image_lr4bc = imresize_np(image_LR4, 4, True)
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.4.lr.{ext}'), image_lr2bc)

                    image_LR8   = imresize_np(image_LR4,  1 / 2, True)
                    image_lr8bc = imresize_np(image_LR8, 8, True)
                    cv2.imwrite(os.path.join("./images/dataset", f'{k
                        }.8.lr.{ext}'), image_lr8bc)
```

Hakim Sabzevari University

An Outline of MSc. Thesis

Surname:Sahebi	Name:Mohammad Mahdi	Student No.:9623137021
Supervisors: Dr. Mahmood Amintoosi and Dr. Mehdi Zaferanieh		
Advisor: Dr. Somaye Sebati Moghadam		
Faculty of Mathematics and Computer Science		
Program: Decision Science and Knowledge engineering Field:		
Title of thesis: Super resolution with convolutional neural network		

Keywords:Super-Resolution, Deep Learning, Conolution Neural Network, Conolutional Single Image Super Resolution

Abstract: Digital imaging systems have expanded dramatically due to ease of use and affordability, but are still weak due to their lower image resolution than previous imaging systems (optical systems). Many attempts have been made to increase the resolution of digital images, which can be divided into two general parts, software and hardware. In the hardware section, the degree of image resolution can be increased by enriching the number of pixels available on digital camera sensors per unit area.

In addition, as the cells of the digital camera sensors become smaller, the amount of effective light received by each cell decreases; Hardware methods to achieve higher quality images and resolution are very expensive and practically impossible to some extent, and usually can not be exceeded due to technical limitations in integrated circuit technology. The use of software method to increase the sharpness of digital images is an issue that is considered as an alternative to hardware methods that are economically viable. The goal of such software methods is to produce a higher resolution image with the same low-resolution digital cameras, so that the final image is as sharp as the image captured by a higher-resolution camera that can be captured if available. After the development of neural networks and the introduction of torsional layers, image resolution methods have been able to achieve the desired accuracy. Various methods based on torsional layers such as SRCNN, FSRCNN, MemNet, SubPixel, VDSR have been introduced, which are used in this dissertation. Optimization SGD, RProp, RMSProp, Lamb, AdaMax, AdaGrad, AdaDelta The extent of image resolution improvement and the speed of image resolution improvement have been examined.

Hakim Sabzevari University
Faculty of Mathematics and Computer Science

A Thesis Submitted in Partial Fulfilment of the Requirement for the Degree of Master of Science in Decision Science and Knowledge engineering

Super resolution with convolutional neural network

Supervisors:
Dr. Mahmood Amintoosi and Dr. Mehdi Zaferanieh

Advisor:
Dr. Somaye Sebati Moghadam

By:
Mohammad Mahdi Sahebi

February 2022