دانشکده مهندسی صنایع

پروژه درس هوش مصنوعی و سیستم های خبره

پردازش و طبقه بندی تصاویر(موضوع اول)

دانشجویان

علیرضا فاضلی کیا

سعید صرافزاده جهرمی

استاد درس

دکتر مهدی غضنفری

دکتر حسین حضرتی

تیر ماه ۱۴۰۱


بسم الله الرحمن الرحیم

فهرست مطالب

چکیده

هوش مصنوعی[1] در سال‌های اخیر شاهد رشد بسیار بزرگ و مهمی در پر کردن شکاف بین توانایی‌های انسان و ماشین بوده است. یکی از این جنبه‌ها، بینایی ماشین[2] است. هدف این زمینه قادر ساختن ماشین‌ها به دیدن جهان به صورتی است که انسان‌ها می‌بینند، و درک آن به گونه‌ای مشابه با انسان‌ها است. پیشرفت‌های حاصل شده در بینایی کامپیوتری با ظهور یادگیری عمیق در طول زمان ایجاد و کامل شده و در درجه اول برمبنای الگوریتم خاصی به نام شبکه عصبی پیچشی[3] بوده است.

بررسی و دسته‌بندی دستی تصاویر، کار بسیار سخت و خسته کننده‌ای است. این چالش، زمانی که حجم زیادی از تصاویر باید بررسی و دسته‌بندی شوند، عملا غیر ممکن خواهد شد. استفاده از الگوریتم های هوش مصنوعی که می‌توانند فرایند دسته‌بندی تصاویر را خودکار و از این طریق، تصاویر را در کلاس یا طبقه صحیح دسته‌بندی کنند علاوه بر کاهش زحمت انسان، سبب نیل به مقصود در سریعترین زمان ممکن می گردد. امروزه با پیشرفت این حوزه، الگوریتم های پیشرفته ای توسعه پیدا کرده اند که حتی می توانند از دقت انسان پیشی گرفته و برتری خود را مستحکم تر سازند.

این پژوهش با پیاده سازی یک معماری شبکه عصبی پیچشی بر روی مجموعه تصاویری از سگ و گربه با استفاده از کتابخانه های تنسورفلو و کراس در پایتون، به دنبال طبقه بندی داده های مسئله با حداکثر دقت ممکن است تا بتواند داده های بدون برچسب را به درستی پیش بینی نماید. جهت بهبود عملکرد مدل چندین معماری مختلف پردازش و بررسی شده و از Dropout و Augmantation نیز استفاده شد. در نهایت با استفاده از معماری شبکه VGG16 به مدلی دست یافتیم که توانست تصاویر را با دقت ۹۷ درصد طبقه بندی کند.

پس از بیان مقدمه در فصل اول، در فصل دوم تعاریف و مبانی نظری حوزه دسته بندی و پردازش تصاویر و سپس مرور ادبیات بیان شده اند. فصل سوم به مدل سازی معماری یک شبکه هوش مصنوعی کانوولوشنی در جهت طبقه بندی تصاویر سگ و گربه و سپس بهبود حداکثری آن اختصاص دارد. نتایج مدل به دست آمده در فصل چهار مورد بحث و بررسی قرار می گیرند و مدل برای پیش بینی تصاویر جدید بدون برچسب ارزیابی می شود. در نهایت در فصل پنجم به جمع بندی و نتیجه گیری از این پژوهش می پردازیم.

کلید واژه ها: پردازش تصاویر، دسته بندی، شبکه های عصبی پیچشی، یادگیری عمیق، هوش مصنوعی

---

[1] Artificial Intelligence

[2] Machine Vision

[3] Convolutional Neural Network

فصل اول

مقدمه

هوش مصنوعی[1] در سال‌های اخیر شاهد رشد بسیار بزرگ و مهمی در پر کردن شکاف بین توانایی‌های انسان و ماشین بوده است. یکی از این جنبه‌ها، بینایی ماشین[2] است. هدف این زمینه قادر ساختن ماشین‌ها به دیدن جهان به صورتی است که انسان‌ها می‌بینند، و درک آن به گونه‌ای مشابه با انسان‌ها و حتی استفاده از دانش حاصل از این بینایی برای وظایف بسیاری مانند بازشناسی تصویر و ویدئو، تحلیل تصویر، دسته‌بندی تصاویر، سیستم‌های توصیه‌گر[3] و پردازش زبان طبیعی است. پیشرفت‌های حاصل شده در بینایی کامپیوتری با ظهور یادگیری عمیق در طول زمان ایجاد و کامل شده و در درجه اول برمبنای الگوریتم خاصی به نام شبکه عصبی پیچشی[4] بوده است.

بررسی و دسته‌بندی دستی تصاویر، کار بسیار سخت و خسته کننده‌ای است. این چالش، زمانی که حجم زیادی از تصاویر (به عنوان نمونه، ۱۰ هزار یا ۱۰۰ هزار تصویر) باید بررسی و دسته‌بندی شوند، عملا غیر ممکن خواهد شد. در چنین حالتی، نقش محققان فعال در حوزه یادگیری عمیق و بینایی کامپیوتر، بیش از پیش پر رنگ‌تر خواهد شد؛ چنین افرادی قادر خواهند بود سیستم‌هایی را طراحی کنند که می‌توانند فرایند دسته‌بندی تصاویر را خودکار کنند و از این طریق، تصاویر را در کلاس یا طبقه صحیح دسته‌بندی کنند.

استفاده از الگوریتم های هوش مصنوعی علاوه بر کاهش زحمت انسان، سبب نیل به مقصود در سریعترین زمان ممکن می گردد. امروزه با پیشرفت این حوزه، الگوریتم های پیشرفته ای توسعه پیدا کرده اند که حتی می توانند از دقت انسان پیشی گرفته و برتری خود را مستحکم تر سازند. بنابراین توسعه و استفاده از این الگوریتم ها فواید بسیاری به همراه دارد.

این پژوهش با پیاده سازی یک معماری شبکه عصبی پیچشی بر روی مجموعه تصاویری از سگ و گربه با استفاده از کتابخانه های تنسورفلو و کراس در پایتون، به دنبال طبقه بندی داده های مسئله با حداکثر دقت ممکن است تا بتواند داده های بدون برچسب را به درستی پیش بینی نماید. لذا در فصل های آینده مراحل توضیح، تشکیل و ارزیابی این شبکه بیان خواهد شد.

---

[1] Artificial Intelligence

[2] Machine Vision

[3] Recommendation Systems

[4] Convolutional Neural Network

فصل دوم

مروری بر ادبیات

۲-۱مقدمه

در این فصل ابتدا به مفاهیم اولیه پردازش تصویر آشنا می شویم. سپس به مبانی نظری شبکه های عصبی پیچشی و تعاریف مربوط به آن پرداخته، و در انتهای فصل مروری گذرا بر جدیدترین مقالات پژوهشی منتشر شده در این حوزه خواهیم داشت.

۲-۲ماهیت تصویر

پیش از آنکه به پردازش تصویر بپردازیم، ابتدا باید مبانی یک تصویر را بشناسیم. یک تصویر با ابعاد (ارتفاع و عرض) و براساس تعداد پیکسل‌هایش مشخص می‌شود. یک پیکسل، نقطه‌ای در عکس است که سایه، کدری یا رنگ مشخصی دارد. به طور مثال، اگر ابعاد عکسی ۵۰۰×۴۰۰ (ارتفاع×عرض) باشد، مجموع تمام پیکسل‌های این عکس ۲۰۰ هزار پیکسل خواهد بود. پیکسل معمولاً به یکی از شیوه‌های زیر وجود دارد:

- طیف خاکستری: این مشخصه در یک پیکسل دارای عددی صحیح است بین صفر تا ۲۵۵ (که صفر بیانگر کاملاً مشکی و ۲۵۵ بیانگر کاملاً سفید است).
- قرمز سبز آبی :(RGB)هر پیکسل از سه عدد صحیح بین صفر تا ۲۵۵ تشکیل شده است. (که هرکدام از اعداد صحیح بیانگر شدت رنگ قرمز، سبز و آبی هستند)
- RGBA : نسخه‌ای توسعه یافته از RGB است که زمینه آلفا نیز به آن افزوده شده که بیانگر کدری تصویر است.

پردازش تصاویر دیجیتال نیازمند عملیاتی است که مراحل آن مشخص و ثابت باشد و بر روی هر پیکسل از عکس اعمال شود. در مرحله اول پردازش تصویر، این عملیات بصورت پیکسل به پیکسل اعمال می‌شود. زمانی که این مرحله به طور کامل انجام شد، وارد مرحله دوم شده و مقادیر خروجی این عملیات‌ها برای هر پیکسل از عکس قابل اندازه‌گیری خواهد بود.

۲-۳پردازش تصویر[۱]

به طور کلی به مجموعه فرایندهایی که بر روی تصویر انجام می‌شود تا اطلاعات و خروجی مشخصی از همان تصویر به دست آید، پردازش تصویر گفته می‌شود. پردازش تصویر به اندازه‌گیری اشیا در داخل عکس کمک می‌کند. همچنین پردازش تصویر از طریق شناخت الگوها، دسته بندی اشیا در تصویر را آسان می‌کند، موقعیت آن‌ها را تشخیص می‌دهد و درک کلی از تصویر ارائه می‌کند. برای پردازش تصویر دو روش وجود دارد:

---

[۱] Image Processing

۱. پردازش تصویر آنالوگ؛ این روش برای پردازش عکس‌های پرینت شده و چاپ شده استفاده می‌شود.
۲. پردازش تصویر دیجیتال؛ پردازش تصویر در این روش به کمک الگوریتم‌های پیچیده و دستکاری تصاویر انجام می‌شود.

پردازش تصویر داده‌های بسیار زیاد به صورت دستی، کار ساده‌ای نیست. استفاده از هوش مصنوعی و یادگیری ماشین سرعت پردازش اطلاعات را افزایش می‌دهد و خروجی با کیفیت تولید می‌کند.

نتایج به شکل پایگاه داده در درون سامانه پردازش تصویر ذخیره می شود. عملیات اصلی در پردازش تصویر شامل تبدیلات هندسی مثل تغییر اندازه، چرخش و...، اصلاح رنگ مثل تغییر روشنایی، وضوح ویا تغییر فضای رنگ، ترکیب تصاویر ، فشرده سازی تصویر، بهبود کیفیت پرونده مثل کاهش نویز و افزایش کنتراست، سنجش کیفیت تصویر و ... است.

پردازش تصاویر دارای دو شاخه عمده بهبود تصاویر و بینایی ماشین است. بهبود تصاویر شامل روش‌هایی مثل استفاده از فیلتر محوکننده و افزایش تضاد برای بهتر کردن کیفیت دیداری تصاویر و اطمینان از نمایش درست آن‌ها در محیط مقصد مانند چاپگر یا نمایشگر رایانه است. در حالی که بینایی ماشین به روش‌هایی می‌پردازد که به کمک آن‌ها می‌توان معنی و محتوای تصاویر را درک کرد تا از آن‌ها در کارهایی چون رباتیک و محور تصاویر استفاده شود.

۲-۴مراحل پردازش تصویر

برای پردازش تصویر ۸ مرحله وجود دارد که آن‌ها را قدم به قدم بیان می‌کنیم.

۱ .تهیه تصویر(Image acquisition)

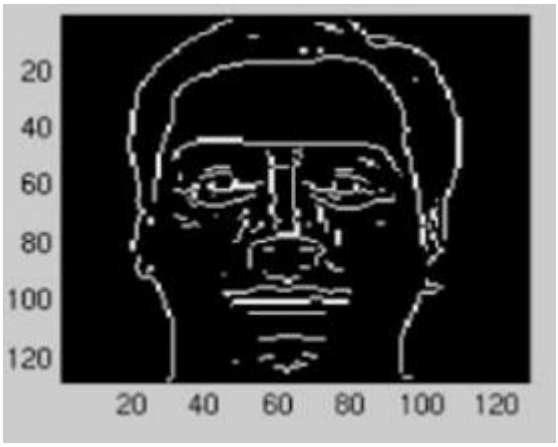
فرایند به دست آوردن تصویر به وسیله یک گیرنده (مانند دوربین) و تبدیل آن به یک وضعیت قابل کنترل است.

۲ .افزایش کیفیت عکس(Image enhencement)

پس از بدست آوردن اطلاعات ورودی تصویر، عملیات پیش پردازش انجام می شود. این مرحله شامل روش هایی برای حذف نویز و افزایش کیفیت تصویر ورودی با هدف جداسازی و تمایز نواحی دارای احتمال وجود اطلاعات حرفی و عددی است.

۳ .ترمیم عکس(Image restoration)

هر گونه خطای احتمالی مانند تار شدگی تصویر، نویز یا عدم فوکوس دوربین حذف می‌شود تا دید بهتری برای به دست آوردن مدل احتمالی و اساس مدل ریاضی به دست آورید. فیلترسازی و آشکارسازی لبه‌ها دو مورد از رایج‌ترین روش‌های پردازش تصاویر دیجیتال هستند. فیلترسازی برای تقویت و اصلاح تصویر ورودی به کار می‌رود. با کمک فیلترهای مختلف می‌توانید ویژگی‌های خاصی از یک تصویر را تقویت کنید یا بردارید یا نویز تصویر را کاهش دهید. آشکارسازی لبه‌ها از فیلترهایی برای تقسیم‌بندی تصویر و استخراج داده استفاده می‌کند. این روش با آشکارسازی مواردی که روشنایی تصویر قطع شده است، به یافتن لبه‌های معنادار اشیا در تصاویر پردازش شده کمک می‌کند.



شکل ۲-۱ نمونه ای از آشکارسازی لبه

۴ .پردازش تصویر رنگی(Color image processing)

تصاویر رنگی و فضاهای متنوع رنگی به روش شبه رنگی (pseudocolor) یا RGB پردازش می‌شوند.

۵ .فشرده سازی تصویر(Image compression)

این کار به شما اجازه می‌دهد بسته به نیاز خود سایز و رزولوشن تصویر را تغییر دهید.

۶ .پردازش مورفولوژیکی(Morphological processing)

در این مرحله ساختار و شکل شی در تصویر تعریف می‌شود.

۷ .تشخیص تصویر(Image recognition)

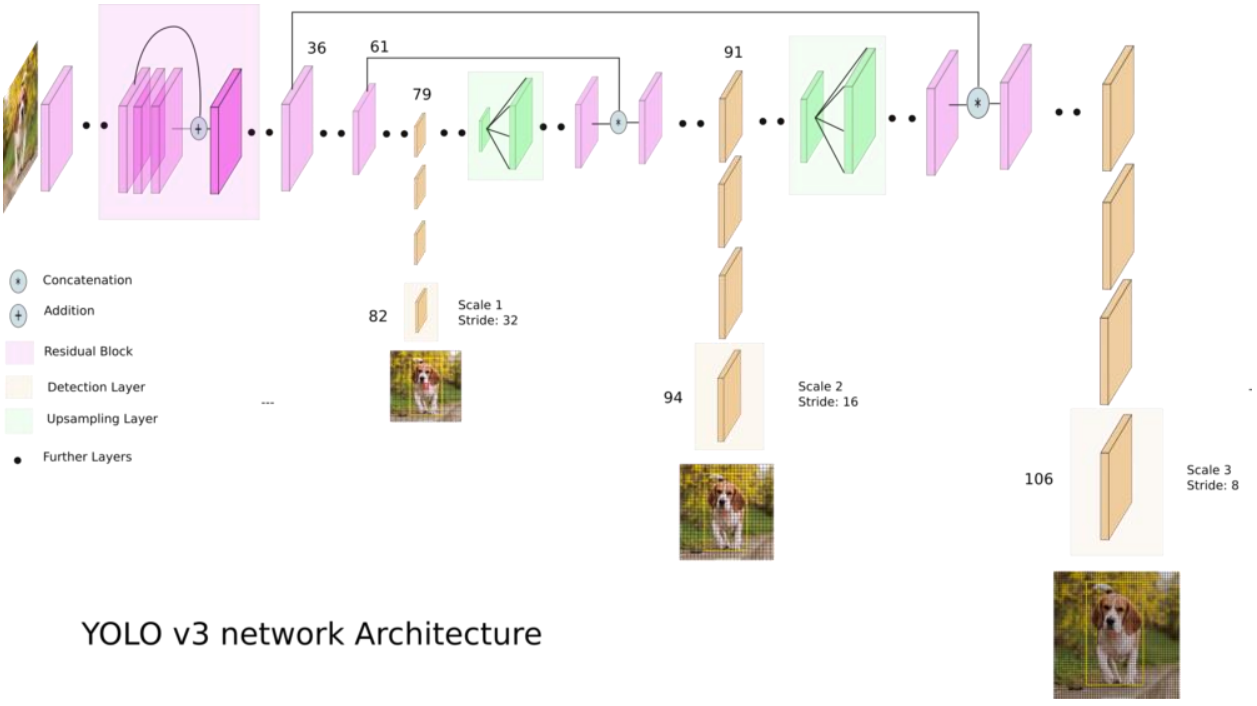
ویژگی‌های منحصربه‌فرد یک شی خاص با استفاده از تکنیک‌هایی مثل object detection تشخیص داده می‌شود.

۸. نمایش و توصیف (Representation and description)

می‌توان با استفاده از ابزارهای تصویری‌سازی خاص، این اعداد و مقادیر بدست آمده از یک سیستم هوش مصنوعی را به تصاویر قابل‌خواندن و مناسب برای تحلیل بیشتر تبدیل کرد.

۲-۵شبکه عصبی پیچشی (کانولوشنی)

شبکه عصبی پیچشی (Convolutional Neural Network | CNN / ConvNet) یک الگوریتم یادگیری عمیق است که تصویر ورودی را دریافت می‌کند و به هر یک از اشیا/جنبه‌های موجود در تصویر میزان اهمیت (وزن‌های قابل یادگیری و بایاس) تخصیص می‌دهد و قادر به متمایزسازی آن‌ها از یکدیگر است. در الگوریتم ConvNet در مقایسه با دیگر الگوریتم‌های دسته‌بندی به پیش‌پردازش کمتری نیاز است. در حالیکه فیلترهای روش‌های اولیه به صورت دستی مهندسی شده‌اند، شبکه عصبی پیچشی (ConvNets)، با آموزش دیدن به اندازه کافی، توانایی فراگیری این فیلترها و مشخصات را کسب می‌کند.

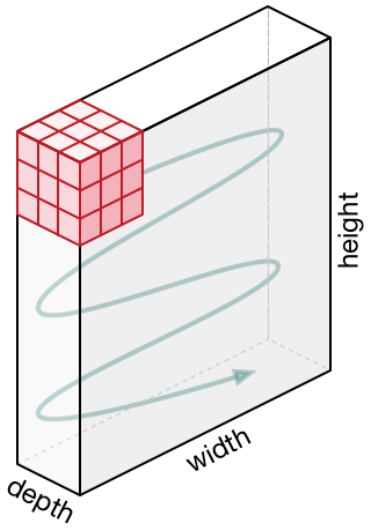


شکل ۲-۲ ساختار کلی شبکه عصبی پیچشی

معماری ConvNet مشابه با الگوی اتصال نورون‌ها در مغز انسان است و از سازمان‌دهی «قشر بصری» (Visual Cortex) در مغز الهام گرفته شده است. هر نورون به محرک‌ها تنها در منطقه محدودی از میدان بصری که تحت عنوان «میدان تاثیر» (Receptive Field)شناخته شده است پاسخ می‌دهد. یک مجموعه از این میدان‌ها برای پوشش دادن کل ناحیه بصری با یکدیگر هم‌پوشانی دارند.

ConvNet قادر است به طور موفقی وابستگی‌های زمانی و فضایی را در یک تصویر با استفاده از فیلترهای مرتبط ثبت کند و همچنین، معماری فیلترگذاری بهتری را روی مجموعه داده تصویر به دلیل کاهش تعداد پارامترهای درگیر و استفاده مجدد از وزن‌ها انجام می‌دهد. به بیان دیگر، شبکه می‌تواند برای درک تصاویر پیچیده به طور بهتری آموزش ببیند. نقش ConvNet کاهش تصاویر به شکلی است که پردازش آن‌ها آسان‌تر و بدون از دست دادن ویژگی‌هایی باشد که برای انجام یک پیش‌بینی خوب حیاتی هستند. این مساله هنگامی حائز اهمیت می‌شود که کاربر قصد طراحی معماری را دارد که صرفا در یادگیری ویژگی‌ها خوب نیست، بلکه برای مجموعه داده‌های بزرگ مقیاس‌پذیر نیز هست. هر شبکه عصبی پیچشی شامل چند لایه با وظایف مختلف است که در ادامه به آنها می پردازیم.

۲-۵-۱لایه پیچشی



شکل ۲-۳ لایه پیچشی و نحوه حرکت آن روی پیکسل های تصویر

هدف عملیات پیچش، استخراج ویژگی‌های سطح بالا مانند «لبه‌ها» از تصاویر ورودی است. ConvNets نیاز دارد که صرفا به یک «لایه پیچشی[۱]» محدود نشود. به طور معمول، اولین ConvLayer مسئول ثبت ویژگی‌های سطح پایین مانند لبه‌ها، رنگ، جهت گرادیان و دیگر موارد است. با لایه‌های افزوده، معماری با ویژگی‌های سطح بالا نیز سازگار می‌شود، و شبکه‌ای را به دست می‌دهد که دارای درک کاملی از تصاویر موجود در مجموعه داده به صورتی است که انسان‌ها تصاویر را درک می‌کنند. دو نوع نتیجه برای عملیات وجود دارد، یکی برای آنکه ابعاد ویژگی پیچانده شده در مقایسه با ورودی کاهش پیدا می‌کند، و دیگری آنکه ابعاد افزایش پیدا می‌کند یا برابر با مقدار پیشین باقی می‌ماند. این کار با اعمال Valid Padding در حالت پیشین یا Same Padding در حالت بعدی انجام می‌شود.

---

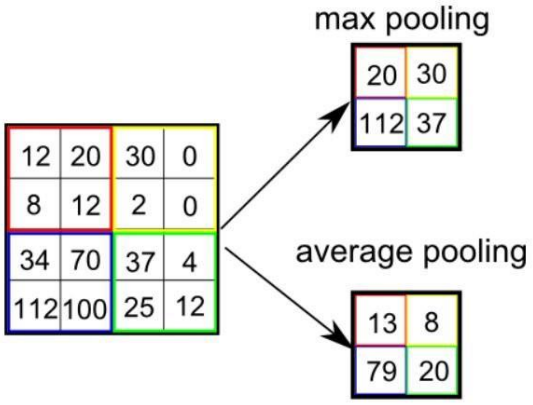
۱ Convolutional Layer

۲-۵-۲لایه ادغام (Pooling)

همچون لایه پیچشی، لایه ادغام[۱]نیز مسئول کاهش سایز فضای ویژگی پیچانده[۲]شده است. این کار با هدف کاهش قدرت محاسباتی مورد نیاز برای پردازش داده‌ها از طریق کاهش ابعاد، انجام می‌شود. علاوه بر این، برای استخراج ویژگی‌های غالبی (Dominant) مفید است که چرخش و موقعیت بدون تغییر دارند (ثابت)، بنابراین منجر به حفظ فرآیند آموزش موثر می‌شوند.

دو نوع ادغام (Pooling) وجود دارد:

Max Pooling وAverage Pooling . ادغام Max Pooling مقدار بیشینه را از قسمتی از تصویر بازمی‌گرداند که توسط کرنل پوشش داده شده است. از سوی دیگر، Average Pooling میانگین همه مقادیر را از قسمتی از تصویر باز می‌گرداند که توسط کرنل پوشش داده شده است.



شکل ۲-۴ انواع لایه ادغام

Max Pooling کار «حذف نویز» را نیز انجام می‌دهد. این ادغام، همه فعال‌سازهای نویزی را هم‌زمان رها می‌کند و همچنین، کار کاهش ابعاد را همراه با حذف نویز انجام می‌دهد. از سوی دیگر، Average Pooling کار کاهش ابعاد را به عنوان مکانیزمی برای حذف نویز اجرا می‌کند. بنابراین شاید بتوان گفت که Max Pooling خیلی بهتر از Average Pooling است.

۲-۵-۳لایه کاملا متصل

افزودن یک لایه کاملا متصل[۳]معمولا راهی ارزان برای یادگیری ترکیب‌های غیر خطی سطح بالای ویژگی‌ها به صورتی است که به وسیله خروجی لایه پیچشی ارائه شد. لایه کاملا متصل در آن فضا یک تابع احتمالا

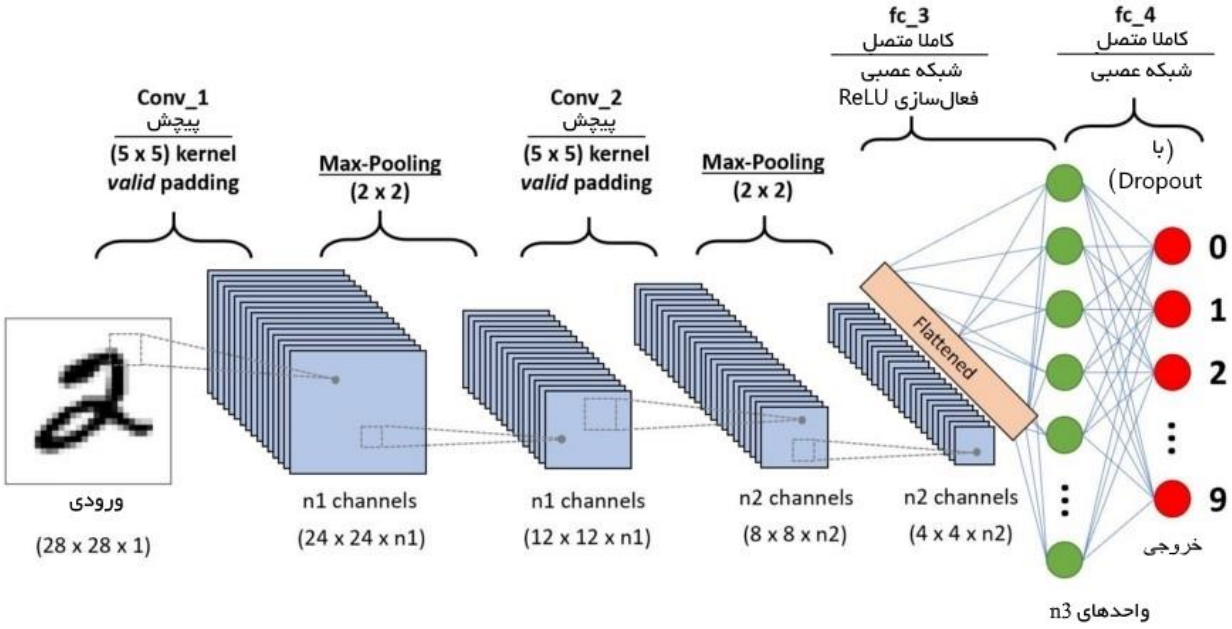
---

۱ Pooling layer

۲ Convolved

۳ Fully-Connected layer

غیر خطی را می‌آموزد. اکنون که تصویر ورودی به شکل مناسبی از پرسپترون چند لایه مبدل شد، باید تصویر را در یک بردار ستونی مسطح کرد. خروجی مسطح شده به یک شبکه عصبی پیش‌خور خورانده می‌شود و بازگشت به عقب[1] در هر تکرار از آموزش اعمال می‌شود.



شکل ۲-۵ شبکه عصبی پیچشی با دولایه آخر کاملا متصل

پس از یک مجموعه از دوره‌ها، مدل قادر به ایجاد تمایز بین ویژگی‌های غالب و برخی ویژگی‌های مشخص سطح پایین در تصویر و دسته‌بندی آن‌ها با استفاده از روش دسته‌بندی «بیشینه همواره» (Softmax) است.

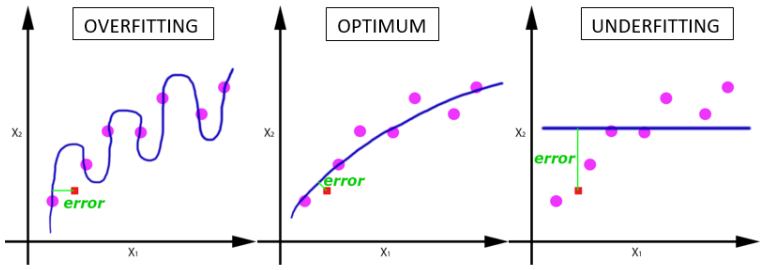
۲-۶تعاریف مهم

ما به اصطلاحاتی مانند دوره‌ها(epochs)، اندازه دسته‌ای (batch size) و تکرارها (iterations) فقط زمانی نیاز داریم که داده‌ها خیلی بزرگ باشند، که این مورد همیشه در یادگیری ماشین اتفاق می‌افتد و ما نمی‌توانیم همه داده‌ها را به یکباره به سیستم منتقل کنیم. درنتیجه، برای غلبه بر این مشکل باید داده‌ها را به اندازه‌های کوچکتر تقسیم کرده و یکی یکی به سیستم بدهیم و وزن شبکه‌های عصبی را در پایان هر مرحله به‌روز کنیم تا با داده‌های ورودی مطابقت داشته باشد.

[1] Backpropagation

Epoch ۲-۶-۱

یک دوره یا epoch زمانی است که کل مجموعه داده تنها یک بار از طریق شبکه عصبی به جلو و عقب منتقل می‌شود. می دانیم که انتقال کل مجموعه داده از طریق یک شبکه عصبی کافی نبوده و باید مجموعه داده را چندین بار به یک شبکه عصبی ارسال کنیم. اما به خاطر داشته باشید که ما از یک مجموعه داده محدود استفاده می کنیم و برای بهینه سازی یادگیری و نمودار از گرادیان کاهشی استفاده می‌کنیم که فرآیندی تکراری است. بنابراین، به روزرسانی وزن‌ها با یک بار پاس دادن یا یک دوره کافی نیست. در واقع؛ یک دوره یا epoch منجر به عدم تناسب منحنی در نمودار زیر می‌شود.



شکل ۲-۶ تاثیر تعیین دوره در یادگیری شبکه عصبی

با افزایش تعداد دوره‌ها، تعداد دفعات تغییر وزن در شبکه عصبی بیشتر می‌شود و منحنی از کم‌برازش[۱] به منحنی بهینه و بعد به منحنی بیش‌برازش[۲] می‌رود. پاسخ درستی برای تعداد مناسب دوره‌ها وجود ندارد. پاسخ برای مجموعه داده‌های مختلف متفاوت است، اما می‌توانید بگویید که تعداد دوره‌ها به تنوع داده‌های شما مربوط می‌شود... بعنوان مثال - آیا فقط گربه‌های سیاه در مجموعه داده‌های خود دارید یا مجموعه داده‌ها بسیار متنوع‌تر است؟

Batch Size ۲-۶-۲

در هر مرحله از یادگیری، شبکه تعدادی از نمونه ها را آموزش داده و پس از آن پارامترهایش را مرتبط با آنها تنظیم می کند. به این تعداد نمونه batch size می گویند. در انتخاب batch size باید دقت شود که عدد آن زیاد بزرگ یا کوچک نباشد. اگر batch size برابر کل نمونه های آموزش انتخاب شود گرادیان پایدارتر شده و شبکه به کندی همگرا می گردد. از طرف دیگر اگر batch size خیلی کوچک انتخاب شود، گرادیان ناپایدار خواهد بود که به طبع آن می بایست نرخ یادگیری را کاهش دهیم.

---

۱ Underfitting

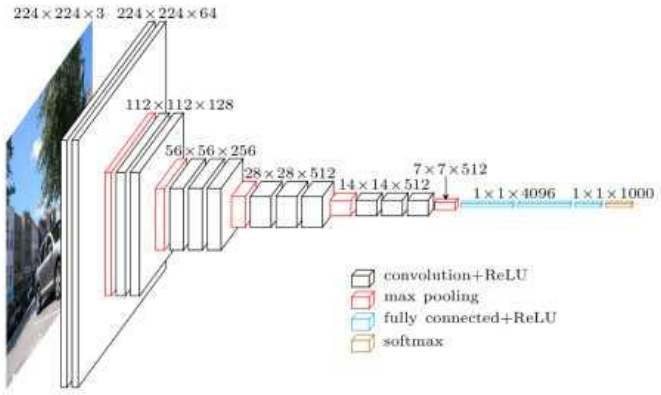
۲ Overfitting

۲-۶-۳تکرارها (iterations)

تکرارها؛ تعداد دسته‌های (batch) مورد نیاز برای تکمیل یک دوره (epoch) است. تعداد دسته‌ها (batchs) برابر است با تعداد دفعات یک دوره. فرض کنید ۲۰۰۰ نمونه آموزشی داریم که قرار است از آنها استفاده کنیم. ما می‌توانیم مجموعه داده‌های ۲۰۰۰ نمونه را به دسته‌های 500 (batchs) تایی تقسیم کنیم، سپس ۴ تکرار (iteration) طول می‌کشد تا یک دوره (epoch)کامل شود که در آن اندازه دسته ۵۰۰، تکرارها ۴ که برای ۱ دوره کامل است.

۲-۷معماری پیشرفته VGG Net

می‌توانیم معماری پیشرفته را اینگونه تعریف کنیم: یک شبکه عصبی که بر اساس بررسی‌ها، عملکرد موفق آن اثبات شده است. این معماری‌ها در دسته‌بندی «مدل‌های عمیق» قرار می‌گیرند، در نتیجه در مقابل همتای سطحی‌ترشان، بهتر عمل می‌کنند.VGG Network توسط محققان گروه Visual Graphics Group در آکسفورد معرفی شده‌است. این شبکه بیشتر بخاطر شکل هرمی مانندش شناخته می‌شود که در آن لایه‌هایی که به تصویر نزدیکتر هستند، پهن‌تر، و لایه‌های دورتر، عمیق‌تر هستند.



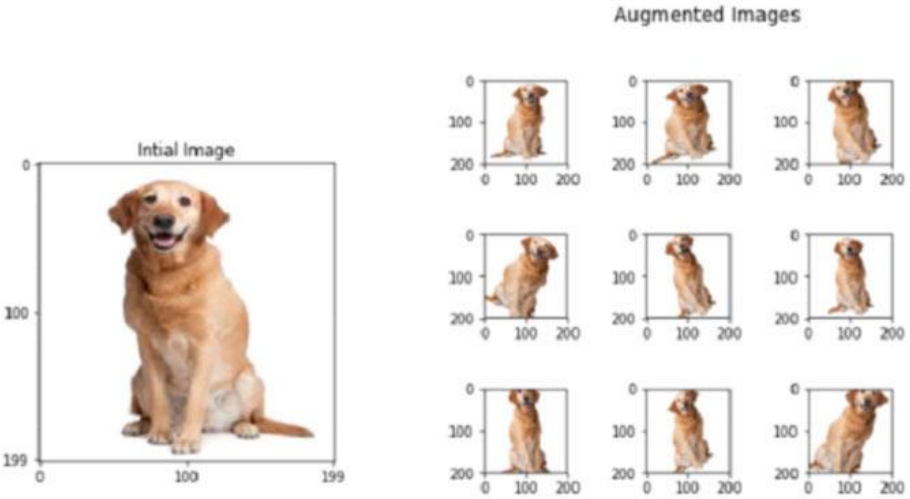
شکل ۲-۷ معماری هرمی شکل VGG Net

همانطور که در تصویر مشخص است، VGG شامل یک سری از لایه‌های محاسباتی (Convolutional) است که پشت آن‌ها لایه‌های ادغام وجود دارند که لایه‌ها را کوچکتر می‌کنند. این گروه در تحقیقات خود شبکه‌های مختلفی را مطرح کرده‌است که هرکدام آن‌ها عمق این معماری را تغییر می‌دهند. از مزایای VGG می‌توان موارد زیر را نام برد:

- یک معماری خیلی خوب برای سنجش یک وظیفه مشخص است.
- شبکه‌های از قبل تعلیم دیده VGG به طور رایگان در اینترنت قرار دارند، به همین جهت در خیلی از اپلیکیشن‌ها استفاده می‌شوند.

۲-۸داده افزایی[۱]

هر چقدر داده‌های بیشتری برای آموزش مدل طراحی شده در اختیار سیستم دسته بندی تصاویر قرار بگیرد، احتمال اینکه عملکرد و دقت سیستم در دسته بندی تصاویر افزایش یابد بیشتر می‌شود. تقویت دیتاست و افزودن داده های تصویری تکنیکی است که با استفاده از آن می توان به طور مصنوعی و بدون تصویربرداری جدید، و تنها با ایجاد نسخه های دستکاری شده از تصاویر موجود اندازه دیتاست را افزایش داد. به‌عنوان‌مثال با چرخش، انتقال و یا کشش تصویر اصلی. این امر به شبکه عصبی اجازه می‌دهد تا یاد بگیرند که چگونه یک شیء را به‌طور مستقل از زاویه چرخش و موقعیت در تصویر تشخیص دهد.



شکل ۲-۸ چگونگی داده افزایی

کتابخانه یادگیری عمیق Keras قابلیت هایی را برای ساده کردن فرآیند آموزش شبکه ها با داده های Image data augmentation فراهم کرده است که در کلاس ImageDataGenerator از این کتابخانه در دسترس است.

۲-۹سوالات مهم در طراحی شبکه عصبی پیچشی

یکی از مراحل مهم در فرایند پیاده‌سازی مدل یادگیری عمیق، تعریف معماری مدل یادگیری عمیق جهت دسته بندی تصاویر است. در این مرحله، جزئیات طراحی مدل یادگیری عمیق از طریق پاسخ به سؤال‌های زیر مشخص می‌شود:

---

[۱] data augmentation

- در مدل یادگیری عمیق، به چند «لایه پیچشی» نیاز است تا عملکرد بهینه سیستم دسته بندی تصاویر تضمین شود.
- برای هر کدام از لایه‌ها، باید از کدام یک از «توابع فعال‌سازی» (Activation Function) استفاده کرد.
- هر کدام از لایه‌های نهان مدل یادگیری عمیق، باید متشکل از چند نورون یا نود نهان باشد.

این سؤالات و دیگر سؤالات مشابه، «ابَرپارامترهای[1]» مدل یادگیری عمیق را مشخص می‌کند. مشخص کردن ابَرپارامترهای بهینه برای سیستم دسته بندی تصاویر، نقش مهمی در افزایش کیفیت پیش‌بینی‌های انجام شده روی تصاویر مجموعه داده تست خواهد داشت. دو راه عمده برای مشخص کردن مقادیر مناسب برای ابَرپارامترهای مدل یادگیری عمیق عبارتند از:

- مطالعه مقالات تحقیقاتی و پژوهشی مرتبط و استنتاج مقادیر مناسب برای ابَرپارامترهای مدل یادگیری عمیق.
- آزمودن سیستم بر اساس مقادیر مختلف ابَرپارامترهای مدل و مشخص کردن ابَرپارامترهایی که بهترین عملکرد را برای سیستم به ارمغان می‌آورند (این روش، بار محاسباتی و زمان پردازشی زیادی را به سیتم تحمیل می‌کند)

۲-۱۰کتابخانه های مورد استفاده برای پردازش تصاویر در پایتون

الف)تنسورفلو (TensorFlow)، یک کتابخانه رایگان و متن‌باز برای برنامه‌نویسی جریان داده Dataflow Programmingو برنامه‌نویسی متمایزگر Differentiable Programming، جهت انجام طیف وسیعی از وظایف است. تنسورفلو، کتابخانه‌ای برای ریاضیات نمادین محسوب می‌شود و کاربردهای گوناگونی در یادگیری ماشین دارد که از آن جمله می‌توان به پیاده‌سازی شبکه‌های عصبی اشاره کرد.

ب) کراس (Keras) یک کتابخانه رایگان منبع باز قدرتمند و با کاربرد آسان برای توسعه و ارزیابی مدل های یادگیری عمیق است.keras دو کتابخانه یادگیری ماشین عددی Theano و TensorFlow را پوشش می دهد و به شما امکان می دهد فقط در چند خط کد، مدل های شبکه عصبی را تعریف و آموزش دهید.

۲-۱۱مرور ادبیات

در سال ۲۰۱۸ Sharma و همکاران در پژوهشی به دنبال بهترین الگوریتم شبکه عصبی پیچشی ممکن برای دسته بندی تصاویر، اعلام کردند که این ناتخاب وابستگی بالایی به اشیای درون مجموعه داده دارد.

[1] Hyperparameter

He و Zhang در سال ۲۰۱۹ توانستند یا مجموعه ترفندهایی دقت Resnet50 را در طبقه بندی عکس از ۷۵٪ به ۷۹,۲۹٪ برسانند.

Jajodia و Garg در سال ۲۰۱۹ مسئله طبقه بندی تصاویر سگ و گربه را بوسیله یک شبکه عصبی پیچشی عمیق برای ۱۳۰۰۰ داده تصویر آزمایش کرده و دقت ۹۰,۱٪ را گزارش کردند.

فصل سوم

مدلسازی

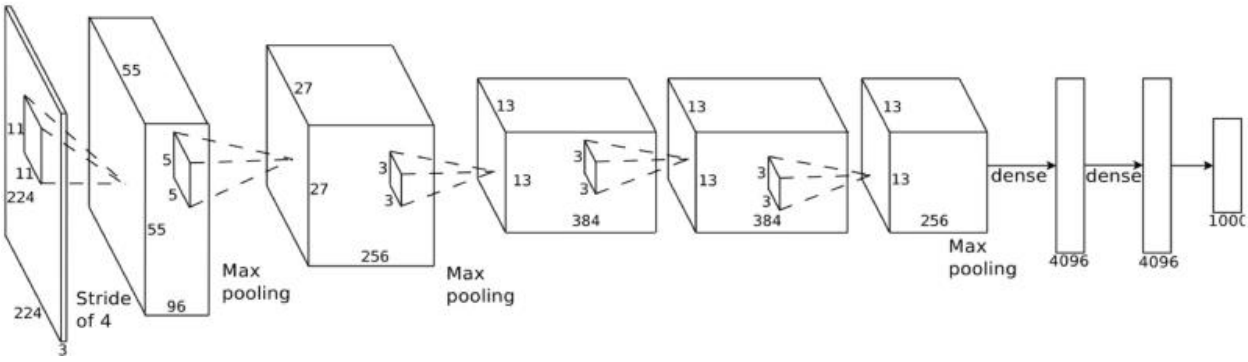
۳-۱بیان مسئله

در این مساله لازم است یک سیستم دارای هوش مصنوعی پیاده سازی شود که بتواند تمایز بین تصاویر گربه و سگ را تشخیص داده و برچسب مناسب را به تصویر جدید تخصیص دهد. علاوه بر موجودیت مورد نظر (سگ و گربه) موجودیت های دیگری نیز در تصاویر حضور دارند. این مساله با توجه به مشخص بودن برچسب داده های آموزش، یک مساله دسته بندی خواهد بود.

۳-۲ایده حل مسئله

فرآیند پردازش تصویر برای حل این مساله شامل یک مجموعه از گام‌ها از جمله پیچش، max-pooling، و در نهایت شبکه کاملا متصل است. هنگام حل مسائل جهان واقعی، این مراحل را می‌توان به تعداد دفعات مورد نظر ترکیب و تجمیع کرد. می‌توان دو، سه یا حتی ده لایه پیچشی داشت. همچنین، هر بار که نیاز باشد می‌توان از max pooling برای کاهش سایز داده‌ها استفاده کرد.

ایده اصلی آغاز کردن کار با تصاویر بزرگ و پایین آوردن مداوم بزرگی آن به صورت گام به گام، تا زمان کشف نتیجه است. هر چه تعداد گام‌های پیچشی موجود بیشتر شود، شبکه قادر خواهد بود بیاموزد که ویژگی‌های پیچیده‌تری را شناسایی کند. برای مثال، اولین گام پیچشی می‌آموزد که لبه‌های تیز را شناسایی کند، گام پیچشی دوم نوک‌های تیز را با استفاده از دانش مرزهای تیز شناسایی می‌کند و سومین گام شناسایی کل پرنده با استفاده از دانش شناسایی نوک‌های تیز است. در تصویر زیر، طرح واقع‌گرایانه‌تری از شبکه پیچشی عمیق ارائه شده است.



شکل ۳-۱ شبکه عصبی پیچشی برای دسته بندی تصاویر

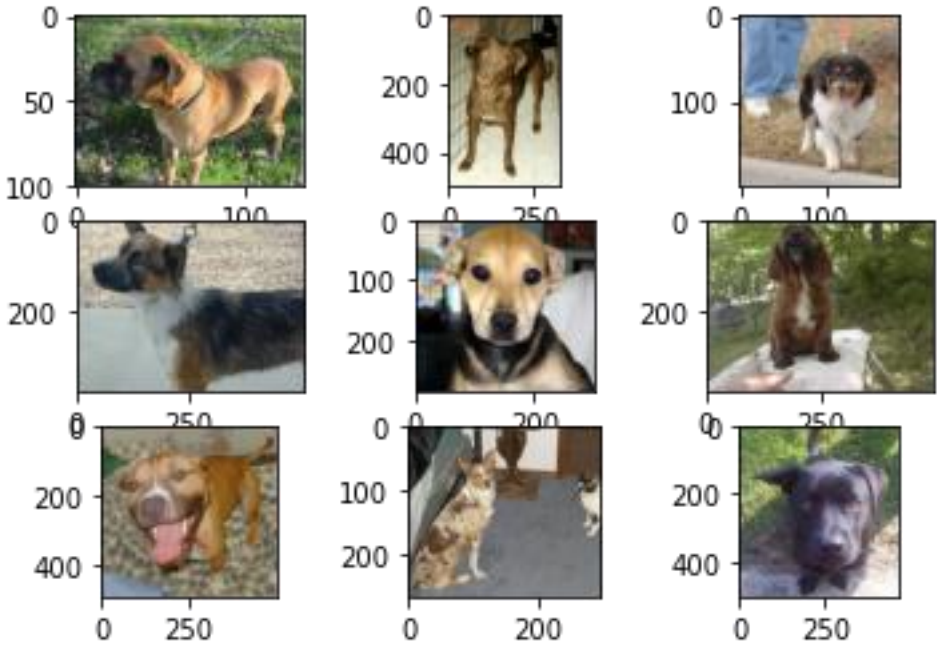
مجموعه داده سگ‌ها در مقابل گربه‌ها یک مجموعه داده استاندارد بینایی رایانه‌ای است که شامل طبقه‌بندی عکس‌ها به عنوان حاوی سگ یا گربه است. یک شبکه عصبی کانولوشن عمیق برای طبقه بندی عکس های سگ ها و گربه ها ایجاد شده است. این موضوع شامل چگونگی ایجاد یک مهار تست قوی برای تخمین

عملکرد مدل، چگونگی بررسی بهبودهای مدل، و نحوه ذخیره مدل و بارگذاری بعدی آن برای پیش‌بینی داده‌های جدید است.

۳-۳بارگذاری داده ها

با نگاهی به چند عکس تصادفی در فهرست، می بینید که عکس ها رنگی هستند و اشکال و اندازه های متفاوتی دارند. به عنوان مثال، نه عکس اول سگ ها را در یک شکل بارگیری و در زیر ترسیم کرده ایم. مثال کامل در زیر آمده است:

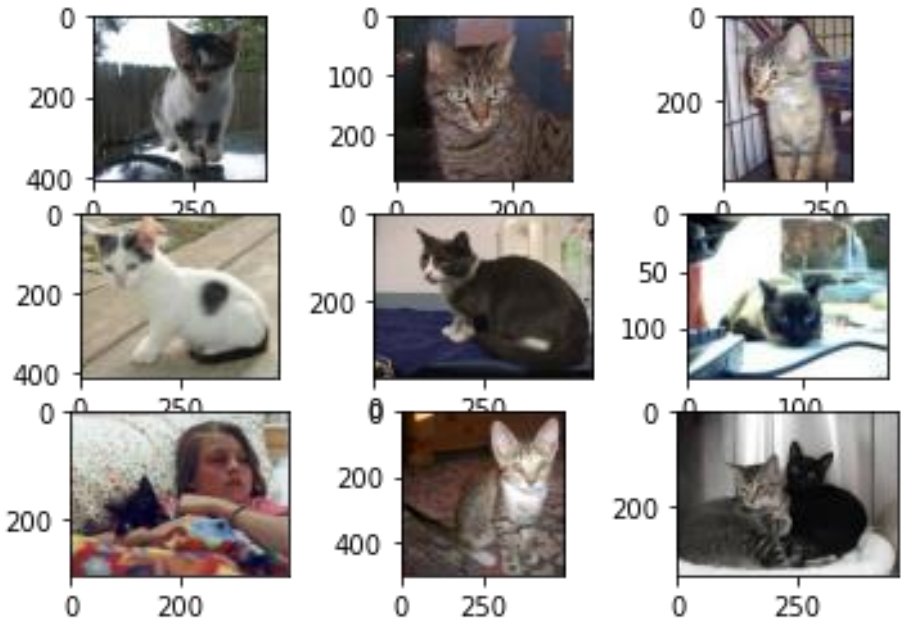
```
# plot dog photos from the dogs vs cats dataset
from matplotlib import pyplot
from matplotlib.image import imread
# define location of dataset
folder = "D:/main_dataset_dogs_vs_cats/"
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # define filename
    filename = folder + 'dog.' + str(i) + '.jpg'
    # load image pixels
    image = imread(filename)
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```



شکل ۳-۲ Plot of the First Nine Photos of Dogs in the Dogs vs Cats Dataset

با اجرای مثال، شکلی ایجاد می شود که ۹ عکس اول سگ ها را در مجموعه داده نشان می دهد. می بینیم که برخی از عکس ها فرمت افقی، برخی فرمت عمودی و برخی مربع هستند. با اجرای کد برای تصاویر گربه باز هم می بینیم که عکس ها همه اندازه های متفاوتی دارند. همچنین می توانیم عکسی را ببینیم که در آن گربه به سختی قابل مشاهده است (گوشه پایین سمت چپ) و عکسی که دارای دو گربه است (گوشه پایین سمت راست). این نشان می دهد که هر طبقه بندی متناسب با این مشکل باید قوی باشد.

```
# plot cat photos from the dogs vs cats dataset
from matplotlib import pyplot
from matplotlib.image import imread
# define location of dataset
folder = "D:/main_dataset_dogs_vs_cats/"
# plot first few images
for i in range(9):
	# define subplot
	pyplot.subplot(330 + 1 + i)
	# define filename
	filename = folder + 'cat.' + str(i) + '.jpg'
	# load image pixels
	image = imread(filename)
	# plot raw pixel data
	pyplot.imshow(image)
# show the figure
pyplot.show()
```



شکل ۳-۳ Plot of the First Nine Photos of Dogs in the Dogs vs Cats Dataset

۳-۴پیش پردازش داده ها

الف) تعیین اندازه استاندارد تصاویر

عکس‌ها باید قبل از مدل‌سازی تغییر شکل داده شوند تا همه تصاویر یک شکل باشند. این اغلب یک تصویر مربع کوچک است. راه‌های زیادی برای رسیدن به این هدف وجود دارد، اگرچه رایج‌ترین آنها یک عملیات ساده تغییر اندازه است که نسبت تصویر هر تصویر را کشیده و تغییر شکل می‌دهد و آن را به شکل جدید مجبور می‌کند.

ما می‌توانیم همه عکس‌ها را بارگذاری کنیم و به توزیع عرض و ارتفاع عکس نگاه کنیم، سپس یک اندازه عکس جدید طراحی کنیم که به بهترین وجه آنچه را که احتمالاً در عمل می‌بینیم، منعکس کند. ورودی‌های کوچکتر به معنای مدلی است که سریع‌تر آموزش داده می‌شود، و معمولاً این نگرانی بر انتخاب اندازه تصویر غالب است. در این صورت، این رویکرد را دنبال می کنیم و اندازه ثابت ۲۰۰×۲۰۰ پیکسل را انتخاب می کنیم.

ب) تبدیل داده ها به آرایه Numpy

در این مرحله همه تصاویر را بارگذاری می کنیم، آنها را تغییر شکل میدهیم و آنها را به عنوان یک آرایه NumPy ذخیره میکنیم. می‌توانیم کد سفارشی بنویسیم تا تصاویر را در حافظه بارگذاری کنیم و اندازه آنها را به عنوان بخشی از فرآیند بارگذاری تغییر دهیم، سپس آنها را برای مدل‌سازی ذخیره کنیم.

```
# define location of dataset
folder = "D:/main_dataset_dogs_vs_cats/"
photos, labels = list(), list()
# enumerate files in the directory
for file in listdir(folder):
    # determine class
    output = 0.0
    if file.startswith('dog'):
        output = 1.0
    # load image with load_img()
    photo = load_img(folder + file, target_size=(200, 200))
    #   convert to numpy array ,Using the img_to_array () function,
    #   we convert the image from PIL format to Numpy format in the
    #   format (width x height x channels)
    photo = img_to_array(photo)
    # store
    photos.append(photo)
    labels.append(output)
# convert to a numpy arrays
photos = asarray(photos)
labels = asarray(labels)
print(photos.shape, labels.shape)
# save the reshaped photos
save('dogs_vs_cats_photos.npy', photos)
save('dogs_vs_cats_labels.npy', labels)
```

```
(8007, 200, 200, 3) (8007,)
```

شکل ۳-۴ بارگذاری، تغییر سایز و ذخیره سازی تصاویر

اجرای مثال ممکن است حدود یک دقیقه طول بکشد تا همه تصاویر در حافظه بارگیری شوند و شکل داده های بارگذاری شده را چاپ می کند تا تأیید شود که به درستی بارگذاری شده است. در پایان اجرا، دو فایل با نام‌های «dogs_vs_cats_photos.npy» و «dogs_vs_cats_labels.npy» ایجاد می‌شوند که حاوی تمام تصاویر تغییر اندازه‌یافته و برچسب‌های کلاس مرتبط با آن‌ها هستند. داده های آماده شده را می توان مستقیماً بارگذاری کرد. مثلا:

```
# load and confirm the shape
from numpy import load
photos = load('dogs_vs_cats_photos.npy')
labels = load('dogs_vs_cats_labels.npy')
print(photos.shape, labels.shape)
```

```
(8007, 200, 200, 3) (8007,)
```

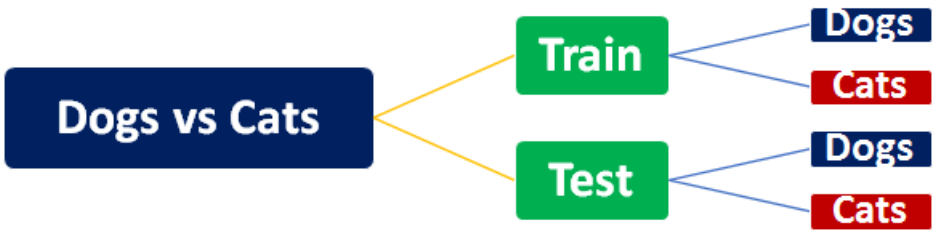
شکل ۳-۵ بارگذاری مستقیم داده ها

ج) قرار دادن تصاویر در دسته بندی های آموزش و تست

این مرحله ترجیح می دهیم داده‌ها به پوشه های train/ و test/ جداگانه تقسیم شوند و در داخل هر پوشه یک زیر شاخه برای هر کلاس وجود داشته باشد، به عنوان مثال در هر کدام از پوشه های مذکور یک زیرپوشه برای سگ و یک زیر پوشه برای گربه داشته باشیم.(شکل ۳-۶)

ما می توانیم یک اسکریپت برای ایجاد یک کپی از مجموعه داده با این ساختار ترجیحی بنویسیم. به طور تصادفی ۲۵٪ از تصاویر را برای استفاده در مجموعه داده های آزمایشی انتخاب کرده ایم.

ابتدا باید ساختار پوشه ها را به صورت زیر ایجاد کنیم:



شکل ۳-۶ ساختار پوشه ها

ما می‌توانیم با استفاده از تابع makedirs() پوشه هایی را در پایتون ایجاد کنیم و از یک حلقه برای ایجاد زیر شاخه‌های dog/ و cat/ برای پوشه های train/ و test/ استفاده کنیم.

```
# create directories using the makedirs() function and use a loop
dataset_home = "D:/dataset_dogs_vs_cats/"
subdirs = ['train/', 'test/']
for subdir in subdirs:
    # create label subdirectories
    labeldirs = ['dogs/', 'cats/']
    for labldir in labeldirs:
        newdir = dataset_home + subdir + labldir
        os.makedirs(newdir, exist_ok=True)
```

شکل ۳-۷ کد ایجاد پوشه ها و زیرشاخه های آنها

د) سازمان دهی داده در یک ساختار مفید

در مرحله بعد، می‌توانیم تمام فایل‌های تصویری موجود در مجموعه داده را شمارش کنیم و آنها را بر اساس نام فایل‌شان در زیر شاخه /dogs/ or cats کپی کنیم. علاوه بر این، می‌توانیم به‌طور تصادفی تصمیم بگیریم ۲۵ درصد از تصاویر را در مجموعه داده آزمایشی نگه داریم. این کار به طور مداوم با ثابت کردن seed برای مولد اعداد شبه تصادفی انجام می شود تا هر بار که کد اجرا می شود همان تقسیم داده ها را بدست آوریم.

```
# seed random number generator
seed(1)
# define ratio of pictures to use for validation
val_ratio = 0.25
# copy training dataset images into subdirectories
src_directory = "D:/main_dataset_dogs_vs_cats/"
for file in listdir(src_directory):
    src = src_directory + '/' + file
    dst_dir = "train/"
    if random() < val_ratio:
        dst_dir = "test/"
    if file.startswith('cat'):
        dst = dataset_home + dst_dir + 'cats/'  + file
        copyfile(src, dst)
    elif file.startswith('dog'):
        dst = dataset_home + dst_dir + 'dogs/'  + file
        copyfile(src, dst)
```

شکل ۳-۸ seed random number generator

۳-۵ایجاد معماری شبکه عصبی پیچشی اولیه

در این بخش، می‌توانیم یک مدل شبکه عصبی کانولوشنال پایه برای مجموعه داده سگ در مقابل گربه ایجاد کنیم. یک مدل پایه حداقل عملکرد مدل را ایجاد می کند که همه مدل های دیگر ما را می توان با آن مقایسه کرد، و همچنین یک معماری مدل که می توانیم به عنوان مبنای مطالعه و بهبود استفاده کنیم.

یک نقطه شروع خوب، اصول کلی معماری مدل های VGG است. اینها نقطه شروع خوبی هستند زیرا در رقابت ILSVRC 2014 به عملکرد برتر دست یافتند و به دلیل اینکه ساختار مدولار معماری قابل درک و پیاده سازی آسان است. این معماری شامل انباشته شدن لایه‌های کانولوشن با فیلترهای کوچک ۳×۳ و به دنبال آن یک لایه ترکیبی حداکثر است. این لایه‌ها با هم یک بلوک را تشکیل می‌دهند و این بلوک‌ها می‌توانند در جایی که تعداد فیلترها در هر بلوک با عمق شبکه افزایش می‌یابد مانند ۳۲، ۶۴، ۱۲۸، ۲۵۶ برای چهار بلوک اول مدل، تکرار شوند. Padding روی لایه های کانولوشن استفاده می شود تا اطمینان حاصل شود که شکل ارتفاع و عرض نقشه های ویژگی خروجی با ورودی ها مطابقت دارد. ما می‌توانیم این معماری را در مسئله سگ در مقابل گربه بررسی کنیم و مدلی را با این معماری با بلوک‌های ۱، ۲ و ۳ مقایسه کنیم.

هر لایه از تابع فعال‌سازی ReLU [۱] و مقدار اولیه وزن He استفاده می‌کند که عموماً بهترین روش‌ها هستند. به عنوان مثال، یک معماری 3-block VGG-style که در آن هر بلوک دارای یک لایه کانولوشنال و تلفیقی است می‌توان در Keras به صورت زیر تعریف کرد:

```
# block 1
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
model.add(MaxPooling2D((2, 2)))
# block 2
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
# block 3
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
```

شکل ۳-۹ معماری 3-block VGG-style

ما می‌توانیم یک تابع به نام define_model() ایجاد کنیم که یک مدل را تعریف کرده و آن را برای قرار دادن در مجموعه داده آماده می‌کند. سپس این تابع را می توان برای تعریف مدل های پایه مختلف سفارشی کرد، به عنوان مثال. نسخه های مدل با بلوک های سبک ۱، ۲ یا ۳ VGG. مدل با نزول گرادیان تصادفی مناسب خواهد بود و ما با نرخ یادگیری محافظه کارانه ۰,۰۰۱ و حرکت ۰,۹ شروع می کنیم. مشکل یک کار طبقه‌بندی باینری است که نیاز به پیش‌بینی یک مقدار ۰ یا ۱ دارد. یک لایه خروجی با ۱ گره و یک فعال‌سازی سیگموئید استفاده می‌شود و مدل با استفاده از تابع تلفات متقابل آنتروپی باینری بهینه می‌شود.

[۱] ReLU activation function

در مرحله بعد، باید داده ها را آماده کنیم. این امر ابتدا شامل تعریف نمونه ای از ImageDataGenerator است که مقادیر پیکسل را در محدوده ۰-۱ مقیاس می کند.

```
# create data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)
```

شکل ۳-۱۰ تعریف ImageDataGenerator

در مرحله بعد، تکرار کننده ها باید برای مجموعه داده های train و آزمایش آماده شوند.

می‌توانیم از تابع flow_from_directory() در مولد داده استفاده کنیم و برای هر یک از دایرکتوری‌های train/ and test/ یک تکرار بسازیم. باید مشخص کنیم که مشکل از طریق آرگومان "class_mode" یک مشکل طبقه بندی باینری است و از طریق آرگومان "target_size" تصاویر را با اندازه ۲۰۰×۲۰۰ پیکسل بارگذاری کنیم. ما اندازه دسته را در ۶۴ تعیین می کنیم.

```
# prepare iterators
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
    class_mode='binary', batch_size=64, target_size=(200, 200))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
    class_mode='binary', batch_size=64, target_size=(200, 200))
```

شکل ۳-۱۱ تعیین تکرار و تعداد دسته

سپس می‌توانیم مدل را با استفاده از تکرارکننده train (train_it) برازش دهیم و از تکرارکننده test (test_it) به عنوان مجموعه داده اعتبارسنجی در طول آموزش استفاده کنیم. تعداد مراحل train و تکرار کننده های test باید مشخص شود. این تعداد دسته هایی است که یک دوره را شامل می شود. این را می توان از طریق طول هر تکرار کننده مشخص کرد و تعداد کل تصاویر در فهرست راهنمای train و test تقسیم بر اندازه دسته (۶۴) خواهد بود. این مدل برای ۲۰ دوره مناسب است، تعداد کمی برای بررسی اینکه آیا مدل می تواند مشکل را یاد بگیرد.

```
# fit model
history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
    validation_data=test_it, validation_steps=len(test_it), epochs=20, verbose=0)
```

شکل ۳-۱۲ فیت کردن مدل

پس از تناسب، مدل نهایی را می توان مستقیماً روی مجموعه داده آزمایشی ارزیابی کرد و دقت طبقه بندی را گزارش کرد.

```
# evaluate model
_, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
print('> %.3f' % (acc * 100.0))
```

شکل ۳-۱۳ ارزیابی مدل

در نهایت، می‌توانیم نموداری از تاریخچه جمع‌آوری‌شده در طول آموزش مدل را در فهرست «history» ایجاد کنیم. تاریخچه شامل دقت و loss مدل در مجموعه آزمایشی و آموزشی در پایان هر دوره است. نمودارهای خطی این معیارها در دوره‌های آموزشی، منحنی‌های یادگیری را ارائه می‌کنند که می‌توانیم از آنها برای دریافت ایده‌ای در مورد اینکه آیا مدل overfitting, underfitting است یا good fit دارد، استفاده کنیم.

تابع summarize_diagnostics() زیر دایرکتوری history را می گیرد و یک شکل واحد با نمودار خطی loss و دیگری برای دقت ایجاد می کند. سپس شکل در فایلی با نام فایل بر اساس نام اسکریپت ذخیره می شود. اگر بخواهیم بسیاری از تغییرات مدل را در فایل های مختلف ارزیابی کنیم و برای هر کدام به طور خودکار نمودارهای خطی ایجاد کنیم، این کار مفید است. ما می‌توانیم همه این‌ها را به یک simple test harness برای testing a model configuration متصل کنیم.

```
# plot diagnostic learning curves
def summarize_diagnostics(history):
    # plot loss
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='blue', label='train')
    pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
    # save plot to file
    filename = sys.argv[0].split('/')[-1]
    pyplot.savefig(filename + '_plot.png')
    pyplot.close()
```

شکل ۳-۱۴ plot diagnostic learning curves

۳-۶ایجاد معماری شبکه عمیق VGGNet

مدل VGG تک بلوکی دارای یک لایه کانولوشنیک با ۳۲ فیلتر و به دنبال آن یک لایه ادغام حداکثر است.

تابع define_model() برای این مدل در بخش قبل تعریف شد اما برای کامل بودن دوباره در زیر ارائه شده است.

```
# define cnn model
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3
	model.add(MaxPooling2D((2, 2)))
	model.add(Flatten())
	model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dense(1, activation='sigmoid'))
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
	return model
```

شکل ۳-۱۵ One Block VGG cnn model

مدل VGG دو بلوکی مدل یک بلوکی را گسترش می دهد و بلوک دوم را با ۶۴ فیلتر اضافه می کند.

```
# define cnn model
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3
	model.add(MaxPooling2D((2, 2)))
	model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
	model.add(MaxPooling2D((2, 2)))
	model.add(Flatten())
	model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dense(1, activation='sigmoid'))
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
	return model
```

شکل ۳-۱۵ Two Block VGG cnn model

مدل VGG سه بلوکی مدل دو بلوکی را گسترش می‌دهد و بلوک سوم را با ۱۲۸ فیلتر اضافه می‌کند.

```
# define cnn model
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3
	model.add(MaxPooling2D((2, 2)))
	model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
	model.add(MaxPooling2D((2, 2)))
	model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
	model.add(MaxPooling2D((2, 2)))
	model.add(Flatten())
	model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dense(1, activation='sigmoid'))
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
	return model
```

شکل ۳-۱۵ 3 Block VGG cnn model

۳-۷توسعه مدل بهبود یافته به وسیله Augmentation و Dropout

در بخش قبل، یک مدل پایه با استفاده از بلوک‌های سبک VGG ایجاد کردیم و روند بهبود عملکرد با افزایش ظرفیت مدل را کشف کردیم. در این بخش، ما با مدل پایه با سه بلوک VGG (یعنی VGG 3) شروع می کنیم و برخی از پیشرفت های ساده را در مدل بررسی می کنیم.

ما می‌توانیم دو رویکرد را برای جلوگیری از overfitting بررسی کنیم: Dropout Regularization و

Data Augmentation. انتظار می رود هر دوی این رویکردها سرعت بهبود را در طول آموزش کاهش دهند و امیدواریم با overfitting مقابله کنند. به این ترتیب، ما تعداد دوره های آموزشی را از ۲۰ به ۵۰ افزایش خواهیم داد تا فضای بیشتری برای اصلاح به مدل بدهیم.

Dropout regularization یک روش محاسباتی ارزان برای منظم کردن یک شبکه عصبی عمیق است. Dropout با حذف احتمالی، یا حذف کردن[۱]ورودی‌های یک لایه، که ممکن است متغیرهای ورودی در نمونه داده یا فعال‌سازی‌های لایه قبلی باشد، کار می‌کند. این اثر شبیه سازی تعداد زیادی از شبکه ها با ساختارهای شبکه بسیار متفاوت است و به نوبه خود، گره ها را در شبکه به طور کلی برای ورودی ها قوی تر[۲]می کند. به طور معمول، پس از هر بلوک VGG می‌توان مقدار کمی حذف را اعمال کرد، با حذف بیشتر روی لایه‌های کاملا متصل نزدیک لایه خروجی مدل اعمال می‌شود.

در زیر تابع define_model() برای یک نسخه به روز شده از مدل پایه با اضافه کردن Dropout است. در این حالت، پس از هر بلوک VGG، افت ۲۰ درصدی اعمال می‌شود، با نرخ افت بزرگ‌تر ۵۰ درصدی پس از لایه کاملا متصل در بخش طبقه‌بندی‌کننده مدل اعمال می‌شود.

```
# define cnn model
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3
	model.add(MaxPooling2D((2, 2)))
	model.add(Dropout(0.2))
	model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
	model.add(MaxPooling2D((2, 2)))
	model.add(Dropout(0.2))
	model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
	model.add(MaxPooling2D((2, 2)))
	model.add(Dropout(0.2))
	model.add(Flatten())
	model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dropout(0.5))
	model.add(Dense(1, activation='sigmoid'))
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
	return model
```

شکل ۳-۱۶ Develop Model with Dropout regularization

[۱] dropping out

[۲] robust

در این مورد، مشاهده می کنیم با اضافه کردن dropout دقت ما به حدود ۷۳ درصد میرسد .

```
# entry point, run the test harness
run_test_harness()
```

```
Found 5979 images belonging to 2 classes.
Found 2028 images belonging to 2 classes.
> 72.929
```

شکل ۳-۱۷ دقت جدید پس از افزودنDropout به مدل

تغییرات کوچک در عکس های ورودی سگ ها و گربه ها مانند جابجایی های کوچک و چرخش های افقی نمونه هایی از Data Augmentation هستند. این افزایش‌ها را می‌توان به‌عنوان آرگومان‌هایی برای ImageDataGenerator مورد استفاده برای مجموعه داده آموزشی مشخص کرد. تقویت‌ها نباید برای مجموعه داده test استفاده شوند، زیرا می‌خواهیم عملکرد مدل را روی عکس‌های اصلاح نشده ارزیابی کنیم. این مستلزم آن است که یک نمونه ImageDataGenerator جداگانه برای مجموعه داده های train و test داشته باشیم، سپس تکرار کننده ها برای train و مجموعه های آزمایشی ایجاد شده از تولید کننده های داده مربوطه. مثلا:

```
# create data generators
train_datagen = ImageDataGenerator(rescale=1.0/255.0,
	width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
# prepare iterators
train_it = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
	class_mode='binary', batch_size=64, target_size=(200, 200))
test_it = test_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
	class_mode='binary', batch_size=64, target_size=(200, 200))
```

شکل ۳-۱۸ Develop Model with Image Data Augmentation

در این حالت، عکس‌های موجود در مجموعه داده آموزشی با جابجایی‌های افقی و عمودی تصادفی کوچک (۱۰٪) و چرخش‌های افقی تصادفی که یک تصویر آینه‌ای از یک عکس ایجاد می‌کنند، افزوده می‌شوند. در این مورد، می‌توانیم شاهد افزایش عملکرد در حدود ۸ درصد از حدود ۷۳ درصد برای مدل پایه به حدود ۸۱ درصد برای مدل پایه با افزایش داده‌های ساده باشیم.

```
# entry point, run the test harness
run_test_harness()
```

```
Found 5979 images belonging to 2 classes.
Found 2028 images belonging to 2 classes.
> 80.917
```

شکل ۳-۱۹ دقت جدید پس از افزودن Augmentation به مدل

۳-۸ یادگیری انتقالی[۱]

یادگیری انتقالی شامل استفاده از تمام یا بخش‌هایی از یک مدل آموزش‌دیده در یک کار مرتبط است.Keras طیف وسیعی از مدل های از پیش آموزش دیده را ارائه می دهد که می توانند به طور کامل یا جزئی از طریق Keras Applications API بارگیری و استفاده شوند.

یک مدل مفید برای یادگیری انتقال یکی از مدل های VGG است، مانند VGG-16 با ۱۶ لایه که در زمان توسعه آن، نتایج برتر را در چالش طبقه بندی عکس ImageNet به دست آورد. این مدل از دو بخش اصلی تشکیل شده است، بخش استخراج کننده ویژگی مدل که از بلوک های VGG تشکیل شده است، و بخش طبقه بندی کننده مدل که از لایه های کاملاً متصل و لایه خروجی تشکیل شده است. می‌توانیم از بخش استخراج ویژگی مدل استفاده کنیم و یک بخش طبقه‌بندی‌کننده جدید از مدل را اضافه کنیم که برای مجموعه داده سگ‌ها و گربه‌ها طراحی شده است. به طور خاص، ما می‌توانیم وزن تمام لایه‌های کانولوشن را در طول تمرین ثابت نگه داریم و فقط لایه‌های کاملاً متصل جدیدی را آموزش دهیم که یاد بگیرند ویژگی‌های استخراج‌شده از مدل را تفسیر کنند و یک طبقه‌بندی باینری ایجاد کنند.

این را می‌توان با بارگذاری مدل VGG-16، حذف لایه‌های کاملاً متصل از انتهای خروجی مدل، سپس اضافه کردن لایه‌های کاملاً متصل جدید برای تفسیر خروجی مدل و انجام یک پیش‌بینی به دست آورد. بخش طبقه‌بندی‌کننده مدل را می‌توان با تنظیم آرگومان "include_top" روی "False" به طور خودکار حذف کرد، که همچنین مستلزم آن است که شکل ورودی نیز برای مدل مشخص شود، در این مورد (۲۲۴، ۲۲۴، ۳). این به این معنی است که مدل بارگذاری شده به آخرین لایه ادغام حداکثر می رسد، پس از آن می توانیم به صورت دستی یک لایه Flatten و لایه های طبقه بندی جدید اضافه کنیم.

تابع define_model() زیر، این مطلب را پیاده سازی می کند و یک مدل جدید آماده برای آموزش برمی گرداند.

[۱] Explore Transfer Learning

```python
# define cnn model
def define_model():
    # load model
    model = VGG16(include_top=False, input_shape=(224, 224, 3))
    # mark loaded layers as not trainable
    for layer in model.layers:
        layer.trainable = False
    # add new classifier layers
    flat1 = Flatten()(model.layers[-1].output)
    class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
    output = Dense(1, activation='sigmoid')(class1)
    # define new model
    model = Model(inputs=model.inputs, outputs=output)
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

شکل ۰–۲۰ Develop Model with Transfer Learning

در این مورد به آموزش زیادی نیاز نخواهد بود، زیرا تنها لایه جدید کاملا متصل و خروجی دارای وزنه های قابل آموزش هستند. به این ترتیب، تعداد دوره‌های آموزشی را ۱۰ می‌کنیم. مدل VGG16 بر روی یک مجموعه داده چالشی ImageNet خاص آموزش داده شد. به این ترتیب، به گونه ای پیکربندی شده است که تصاویر ورودی مورد انتظار دارای شکل ۲۲۴×۲۲۴ پیکسل باشند. هنگام بارگذاری عکس‌ها از مجموعه داده سگ‌ها و گربه‌ها، از این به عنوان اندازه هدف استفاده می‌کنیم.

این مدل همچنین انتظار دارد که تصاویر در مرکز قرار گیرند. یعنی مقادیر میانگین پیکسل از هر کانال (قرمز، سبز و آبی) که در مجموعه داده آموزشی ImageNet محاسبه شده است، از ورودی کم شود. Keras تابعی را برای انجام این آماده سازی برای عکس های جداگانه از طریق تابع preprocess_input() فراهم می کند. با این وجود، می‌توانیم با تنظیم آرگومان «featurewise_center» روی «True» و تعیین دستی مقادیر میانگین پیکسل‌ها برای استفاده در مرکز به‌عنوان مقادیر میانگین از مجموعه داده آموزشی ImageNet، به همان اثر ImageDataGenerator برسیم: [۱۲۳,۶۸، ۱۱۶,۷۷۹، ۱۰۳,۹۳۹] .

```python
# entry point, run the test harness
run_test_harness()
```

```
Epoch 5/10
94/94 [==============================] - 864s 9s/step - loss: 9.7627e-04 - accuracy: 1.0000 - val_loss: 0.0866 - val_accur
acy: 0.9763
Epoch 6/10
94/94 [==============================] - 829s 9s/step - loss: 6.7199e-04 - accuracy: 1.0000 - val_loss: 0.0883 - val_accur
acy: 0.9763
Epoch 7/10
94/94 [==============================] - 1221s 13s/step - loss: 5.0325e-04 - accuracy: 1.0000 - val_loss: 0.0914 - val_acc
uracy: 0.9763
Epoch 8/10
94/94 [==============================] - 1174s 13s/step - loss: 4.0299e-04 - accuracy: 1.0000 - val_loss: 0.0925 - val_acc
uracy: 0.9763
Epoch 9/10
94/94 [==============================] - 1172s 13s/step - loss: 3.3311e-04 - accuracy: 1.0000 - val_loss: 0.0930 - val_acc
uracy: 0.9758
Epoch 10/10
94/94 [==============================] - 1174s 13s/step - loss: 2.8260e-04 - accuracy: 1.0000 - val_loss: 0.0942 - val_acc
uracy: 0.9758
> 97.584
```

شکل ۰–۲۱ run the test harness with Transfer Learning

در این مورد، می‌توانیم ببینیم که مدل به نتایج بسیار چشمگیری با دقت طبقه‌بندی در حدود ۹۷ درصد در مجموعه داده‌های آزمون نگهدارنده دست یافته است.

۳-۹نهایی کردن مدل و ذخیره آن

روند بهبود مدل ممکن است تا زمانی ادامه یابد که ما ایده ها و زمان و منابع لازم برای آزمایش آنها را داشته باشیم. یک مدل نهایی معمولاً بر روی همه داده‌های موجود، مانند ترکیبی از مجموعه‌های داده‌های train و test ، مناسب است. در این آموزش، تناسب مدل نهایی را فقط در مجموعه داده train نشان می‌دهیم زیرا فقط برچسب‌هایی برای مجموعه داده آموزشی داریم.

اولین گام این است که مجموعه داده train را آماده کنیم تا بتوان آن را توسط کلاس ImageDataGenerator از طریق تابع flow_from_directory بارگذاری کرد. به طور خاص، ما باید یک دایرکتوری جدید با تمام تصاویر train سازماندهی شده در دایرکتوری های سگ/و گربه/بدون هیچ گونه جداسازی در دایرکتوری های train /یا test/ ایجاد کنیم. این را می توان با به روز رسانی اسکریپتی که در ابتدای آموزش ایجاد کردیم به دست آورد. در این مورد، ما یک پوشه finalize_dogs_vs_cats/ جدید با زیرپوشه‌های cats and /dogs/ برای کل مجموعه داده train ایجاد می‌کنیم. ساختار به صورت زیر خواهد بود:

```
1 finalize_dogs_vs_cats
2 ├── cats
3 └── dogs
```

شکل ۳-۲۰ ساختار مجموعه داده نهایی

اسکریپت به روز شده برای کامل بودن در ادامه فهرست شده است.

```
# organize dataset into a useful structure
from os import makedirs
from os import listdir
from shutil import copyfile
# create directories
dataset_home = "D:/finalize_dogs_vs_cats/"
# create label subdirectories
labeldirs = ['dogs/', 'cats/']
for labldir in labeldirs:
    newdir = dataset_home + labldir
    makedirs(newdir, exist_ok=True)
# copy training dataset images into subdirectories
src_directory = "D:/dataset_dogs_vs_cats - train"
for file in listdir(src_directory):
    src = src_directory + '/' + file
    if file.startswith('cat'):
        dst = dataset_home + 'cats/'  + file
        copyfile(src, dst)
    elif file.startswith('dog'):
        dst = dataset_home + 'dogs/'  + file
        copyfile(src, dst)
```

شکل ۳-۲۱ سازمان دهی ساختار مجموعه داده

ما اکنون آماده هستیم تا یک مدل نهایی را در کل مجموعه داده آموزشی قرار دهیم.flow_from_directory() باید به روز شود تا همه تصاویر از پوشه finalize_dogs_vs_cats/ جدید بارگذاری شوند.

```
# prepare iterator
train_it = datagen.flow_from_directory('finalize_dogs_vs_cats/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
```

شکل ۳-۲۲ آماده سازی تعداد تکرار برای مدل نهایی

علاوه بر این، فراخوانی fit_generator() دیگر نیازی به تعیین مجموعه داده اعتبارسنجی[1] ندارد.

```
# fit model
model.fit_generator(train_it, steps_per_epoch=len(train_it), epochs=10, verbose=0)
```

شکل ۳-۲۳ فیت کردن نهایی

پس از fit، می‌توانیم مدل نهایی را در فایل H5 با فراخوانی تابع save() روی مدل ذخیره کرده و نام فایل انتخابی را ارسال کنیم.

[1] validation dataset

```
# save model
model.save('final_model.h5')
```

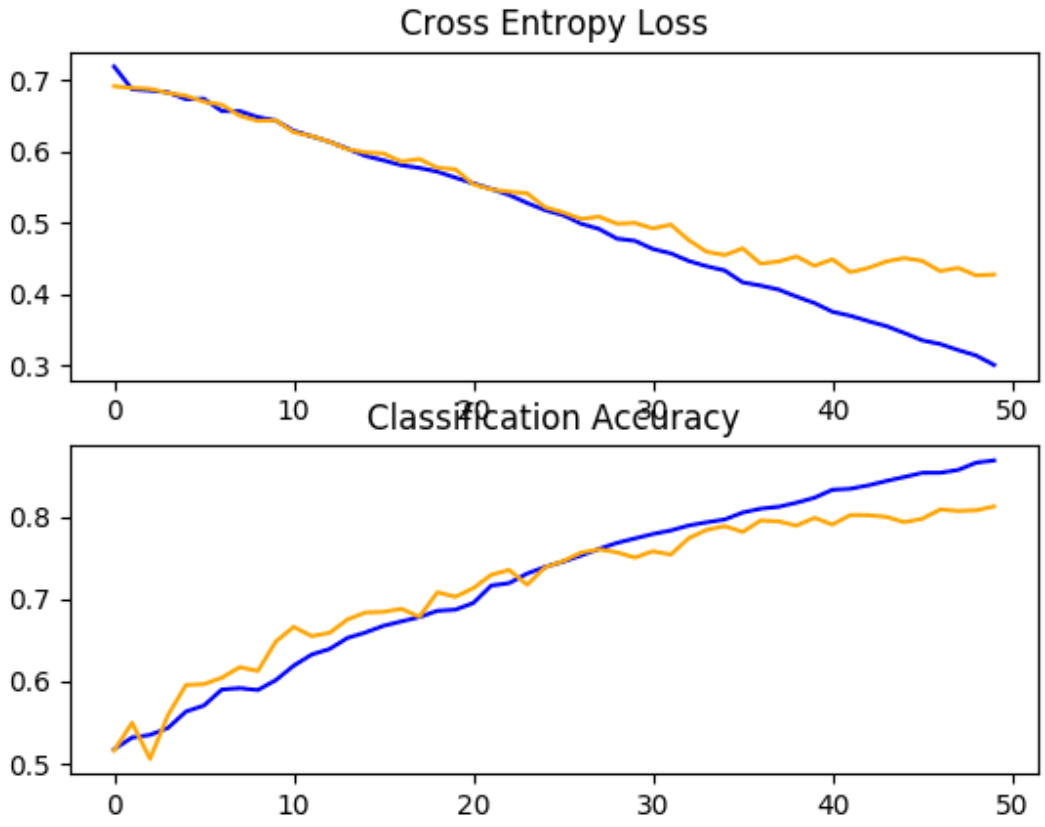
شکل ۳-۲۴ ذخیره سازی مدل نهایی

فصل چهارم

تحلیل و ارزیابی

۴-۱تاثیر Dropout بر مدل

با مرور منحنی های یادگیری، می بینیم که Dropout بر میزان بهبود مدل در هر دو مجموعه train وtest تأثیر داشته است. اضافه برازش کاهش یافته یا به تعویق افتاده است، اگرچه ممکن است عملکرد در پایان اجرا شروع به توقف کند. نتایج نشان می دهد که دوره های آموزشی بیشتر ممکن است منجر به بهبود بیشتر مدل شود. همچنین ممکن است جالب باشد که در کنار افزایش دوره‌های آموزشی[۱] میزان dropout rate کمی بالاتر بعد از بلوک‌های VGG را بررسی کنیم.



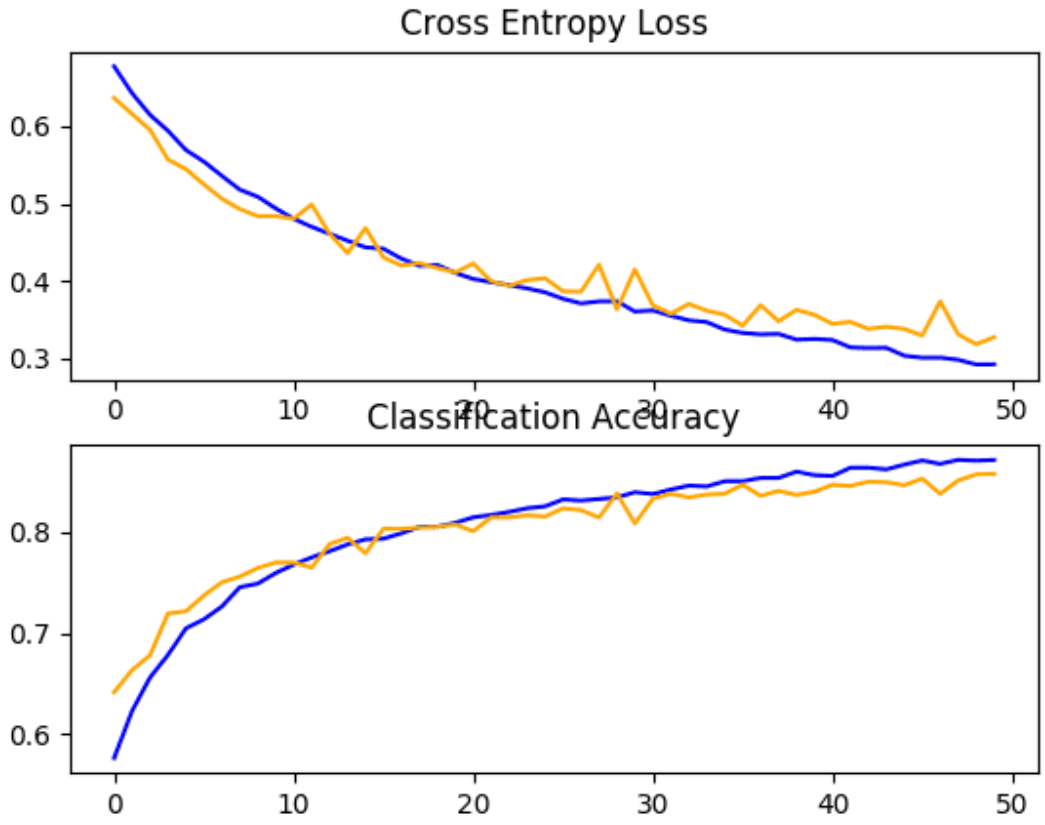
شکل ۴-۱ Line Plots of Loss and Accuracy Learning Curves for the Baseline Model With Dropout on the Dogs and Cats Dataset

[۱] training epochs

۴-۲تاثیر Augmentation بر مدل

با مرور منحنی یادگیری، می‌توانیم ببینیم که به نظر می‌رسد این مدل قادر به یادگیری بیشتر است، هم‌زمان با از دست دادن مجموعه داده‌های train و test حتی در پایان اجرا همچنان کاهش می‌یابد. تکرار آزمایش با ۱۰۰ دوره یا بیشتر به احتمال زیاد منجر به عملکرد بهتر مدل می شود.

ممکن است جالب باشد که افزایش‌های دیگری را کشف کنید که ممکن است یادگیری ویژگی‌های غیرمتغیر در موقعیت آنها در ورودی را بیشتر تشویق کنند، مانند چرخش‌های جزئی و بزرگ‌نمایی.

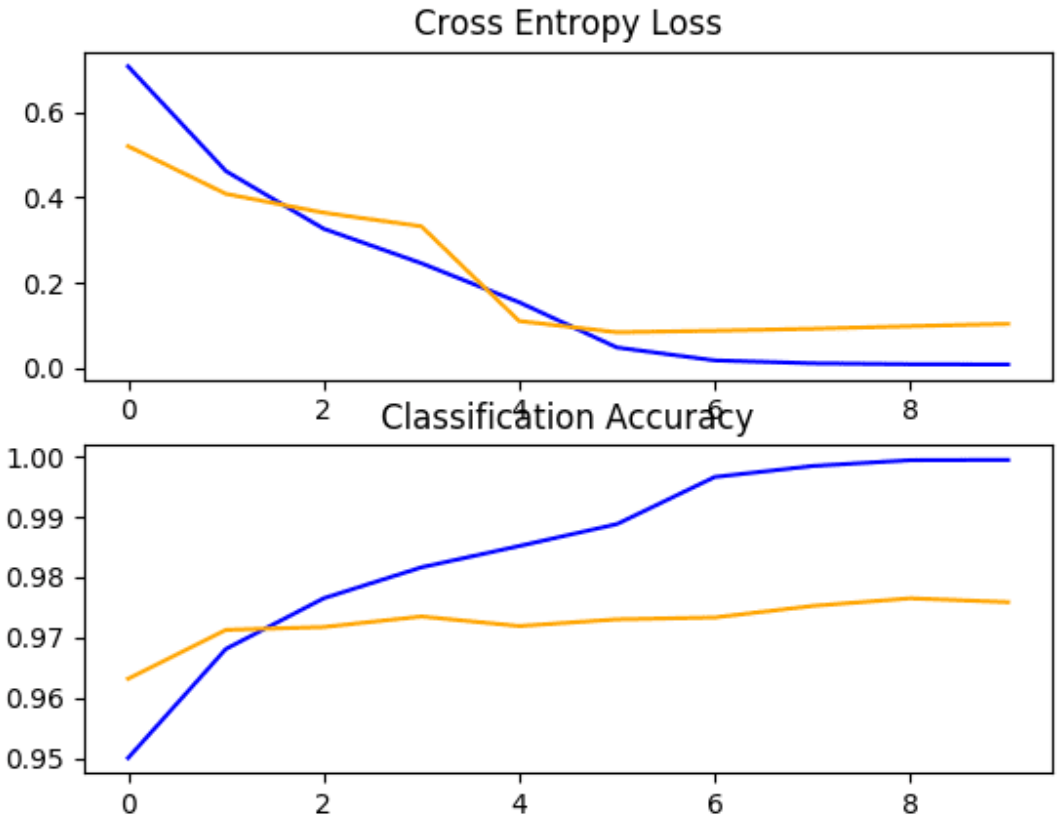


شکل ۴-۲ Line Plots of Loss and Accuracy Learning Curves for the Baseline Model With Data Augmentation on the Dogs and Cats Dataset

۴-۳تاثیر استفاده از یادگیری انتقالی با VGG16

با مرور منحنی های یادگیری، می بینیم که مدل به سرعت با مجموعه داده مطابقت دارد. این بیش از حد برازش قوی را نشان نمی دهد، اگرچه نتایج نشان می دهد که شاید ظرفیت اضافی در طبقه بندی کننده و/یا استفاده از منظم سازی ممکن است مفید باشد.

پیشرفت‌های زیادی می‌توان در این رویکرد ایجاد کرد، از جمله افزودن dropout regularization به بخش طبقه‌بندی‌کننده مدل و شاید حتی تنظیم دقیق وزن برخی یا همه لایه‌ها در بخش آشکارساز ویژگی مدل.



شکل ۴-۳ Line Plots of Loss and Accuracy Learning Curves for the VGG16 Transfer Learning Model on the Dogs and Cats Dataset

۴-۴پیش بینی با استفاده از مدل نهایی و گزارش نتایج

ما می توانیم از مدل ذخیره شده خود برای پیش بینی تصاویر جدید استفاده کنیم. این مدل فرض می‌کند که تصاویر جدید رنگی هستند و به گونه‌ای تقسیم‌بندی شده‌اند که یک تصویر شامل حداقل یک سگ یا گربه باشد.

در زیر یک تصویر استخراج شده از مجموعه داده آزمایشی برای مسابقه سگ و گربه است. هیچ برچسبی ندارد، اما به وضوح می توان گفت که عکس یک سگ است. آن را با نام فایل "sample_image.jpg" ذخیره می کنیم.

شکل ۴-۴ Dog (sample_image.jpg)

وانمود می کنیم که این یک تصویر کاملاً جدید و دیده نشده است که به روش لازم تهیه شده است، و می بینیم که چگونه می توانیم از مدل ذخیره شده خود برای پیش بینی عدد صحیحی که تصویر نشان می دهد استفاده کنیم. برای این مثال، ما کلاس "۱" را برای "سگ" انتظار داریم.

توجه: زیر شاخه های تصاویر، یکی برای هر کلاس، توسط تابع flow_from_directory() به ترتیب حروف الفبا بارگذاری می شوند و برای هر کلاس یک عدد صحیح اختصاص می دهند. زیر شاخه "cat" قبل از "dog" قرار می گیرد، بنابراین به برچسب های کلاس اعداد صحیح اختصاص داده می شود: cat=0، dog=1. این را می توان از طریق آرگومان "classes" در فراخوانی flow_from_directory() هنگام آموزش مدل تغییر داد.

ابتدا می‌توانیم تصویر را بارگذاری کنیم و اندازه آن را ۲۲۴×۲۲۴ پیکسل کنیم. سپس می توان اندازه تصویر بارگذاری شده را تغییر داد تا یک نمونه در یک مجموعه داده داشته باشد. مقادیر پیکسل نیز باید در مرکز

قرار گیرند تا با روشی که داده ها در طول آموزش مدل تهیه شده اند مطابقت داشته باشند. تابع load_image() این را پیاده سازی می کند و تصویر بارگذاری شده را آماده طبقه بندی می کند.

```
# make a prediction for a new image.
import keras
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.models import load_model

# load and prepare the image
def load_image(filename):
    # load the image
    img = load_img(filename, target_size=(224, 224))
    # convert to array
    img = img_to_array(img)
    # reshape into a single sample with 3 channels
    img = img.reshape(1, 224, 224, 3)
    # center pixel data
    img = img.astype('float32')
    img = img - [123.68, 116.779, 103.939]
    return img

# load an image and predict the class
def run_example():
    # load the image
    img = load_image('sample_image.jpg')
    # load model
    model = load_model('final_model_VGG16.h5')
    # predict the class
    result = model.predict(img)
    print(result[0])

# entry point, run the example
run_example()
```

```
1/1 [==============================] - 1s 772ms/step
[1.]
```

make a prediction for a new image شکل ۴-۵

فصل پنجم:

نتیجه گیری

رشد سریع هوش مصنوعی در سال‌های اخیر و کاربرد های فراوان آن در محول کردن وظایف انسانی سبب شد تا شاهد بکارگیری این فناوری در دسته بندی تصاویر نیز باشیم. با ظهور یادگیری عمیق پیشرفت‌های حاصل شده در طول زمان ایجاد و کامل شده و امروزه در الگوریتم شبکه های عصبی پیچشی به نهایت خود رسیده است.

در این پژوهش ما به دنبال دسته بندی تصاویری از دو موجودیت سگ و گربه با استفاده از یادگیری ماشین تحت نظارت بودیم. در این راستا یک معماری شبکه عصبی پیچشی طراحی و با استفاده از زبان برنامه نویسی پایتون کدزنی شد. ابزار کار شامل کتابخانه های تنسورفلو و کراس در پایتون بود و اعضای گروه بنا به شرایط از دو محیط Jupiter و Google Colab کار کدزنی را انجام دادند.

با پیاده سازی مدل بر روی مجموعه تصاویر ابتدا دقت ۷۱٪ حاصل شد. جهت بهبود عملکرد مدل چندین معماری مختلف پردازش و بررسی شدند تا در نهایت با استفاده از معماری شبکه VGG16 به مدلی دست یافتیم که توانست تصاویر را با دقت ۹۷ درصد طبقه بندی کند. همچنین برای جلوگیری از Overfitting در مدل از Dropout و Augmantation در مجموعه داده اولیه نیز استفاده شد.

پس از انجام پیش بینی با داده های بدون برچسب و با بررسی نتایج به دست آمده از پژوهش می توان اظهار داشت که مدل ارائه شده در این پژوهش از دقت مناسبی برخوردار بوده و می تواند در پیش بینی تصاویر مربوطه با موفقیت عمل نماید. همچنین بررسی نمودار های ارزیابی مدل بیانگر آن است که استفاده از Dropout و Augmantation به دقت مدل و جلوگیری از Overfit شدن آن کمک کرده است. بنابراین می توان نتیجه گیری کرد مدل حاضر توانسته است مارا در رسیدن به هدف یاری کند.

مراجع

مراجع

[1] Jajodia, T., & Garg, P. (2019). Image classification–cat and dog images. Image, 6(23), 570-572.

[2] Ramprasath, M., Anand, M. V., & Hariharan, S. (2018). Image classification using convolutional neural networks. International Journal of Pure and Applied Mathematics, 119(17), 1307-1319.

[3] Wang, J., & Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. Convolutional Neural Networks Vis. Recognit, 11, 1-8.

[4] Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. Procedia computer science, 132, 377-384.

[5] He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2019). Bag of tricks for image classification with convolutional neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 558-567).

[6] Mikołajczyk, A., & Grochowski, M. (2018, May). Data augmentation for improving deep learning in image classification problem. In 2018 international interdisciplinary PhD workshop (IIPhDW) (pp. 117-122). IEEE.

[7] Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization, 2007.

[8] Machine Learning Attacks Against the Asirra CAPTCHA, 2007.

[9] OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, 2013.

[10] https://blog.faradars.org/convolutional-neural-networks/

[11] https://blog.faradars.org/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%DB%8C%D8%A7%D8%AF%DA%AF%DB%8C%D8%B1%DB%8C-%D9%85%D8%A7%D8%B4%DB%8C%D9%86-%D8%A8%D8%AE%D8%B4-%D8%B3%D9%88%D9%85/