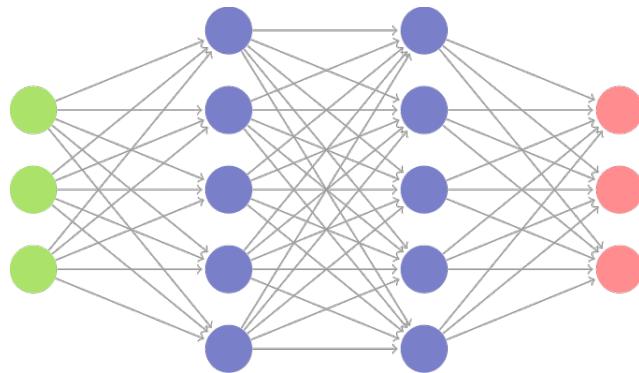




به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر



Neural Networks

Project 1

نام و نام خانوادگی	علیرضا محمدی	محمدعلی شاکردرگاه
شماره دانشجویی	۸۱۰۱۹۹۳۶۵	۸۱۰۱۹۶۴۸۷
تاریخ ارسال گزارش	۱۴۰۰/۰۲/۲۷	

فهرست مطالب

۱ ۴ بخش اول ۴ ۱.۱ قسمت اول ۴ ۱.۱.۱ مشکلات ۴ ۲.۱.۱ راه حل ها ۶ قسمت دوم ۸ قسمت سوم ۸ قسمت چهارم ۸ قسمت پنجم ۸ قسمت ششم 	۲ ۱۰ بخش دوم ۱۰ ۱.۲ مشخصات شبکه عصبی ۱۱ ۲.۲ آموزش شبکه ۱۱ ۳.۲ تعداد لایه های متفاوت ۱۲ ۱.۳.۲ دو لایه ۱۲ ۲.۳.۲ یک لایه ۱۲ ۳.۳.۲ صفر لایه ۱۳ بررسی توابع فعال سازی ۱۵ بررسی روش بهینه سازی ۱۵ کاهش حجم دیتا ۱۶ استفاده از کرنل بزرگ تر ۱۶ Dropout
۳ ۱۸ بخش سوم ۱۸ ۱.۳ Augmentation ۱۸ ۲.۳ ایجاد نمونه مصنوعی ۱۹ ۳.۳ کاهش حجم دیتا ۲۰ ۴.۳ حل مشکل عدم توازن دیتابست 	
۴ ۲۲ بخش چهارم ۲۲ ۱.۴ مدل Inception ۲۳ ۱.۱.۴ ورودی شبکه ۲۳ ۲.۱.۴ پیش پردازش های لازم ۲۳ ۳.۱.۴ خروجی شبکه ۲۳ ۲.۴ Transfer Learning ۲۴ ۳.۴ Transfer Learning ۲۴ ۱.۳.۴ تعریف مدل ۲۴ ۲.۳.۴ دیتابست جدید ۲۵ ۳.۳.۴ آموزش مدل 	

۲۶	اشیای قابل تشخیص توسط شبکه	۴.۴
۲۶	بررسی عملکرد شبکه	۵.۴
۲۶	۱.۵.۴ تصویر اول	
۲۷	۲.۵.۴ تصویر دوم	
۲۷	۳.۵.۴ تصویر سوم	

۱ بخش اول

۱.۱ قسمت اول

روش گرادیان کاهشی یکی از روش‌های بهینه‌سازی غیر خطی به منظور یافتن وزن‌های شبکه عصبی است. اما این روش می‌تواند با مشکلاتی همراه باشد.

۱.۱.۱ مشکلات

۱. اولین مشکل این روش آن است که می‌تواند به سادگی در نقطه بهینه‌ی محلی گرفتار شود. این حالت مخصوصاً در توابع پیچیده‌ای با تعداد زیادی نقطه بحرانی، شایع است. در این حالت نمی‌توانیم نقطه بهینه‌ی سراسری را برای تابع هزینه‌ای که به دنبال کمینه کردن آن هستیم برسیم.

۲. در این روش، گرادیان تابع هزینه را برای هر یک از وزن‌های شبکه محاسبه می‌کنیم. اما به دلیل تکراری بودن یکسری از محاسبات، از روش back propagation استفاده می‌کنیم. در این روش گرادیان را از لایه‌های آخر حساب کرده و به عقب بر می‌گردیم و این گرادیان را برای محاسبه‌ی گرادیان وزن‌های لایه‌های عقب‌تر استفاده می‌کنیم. در این صورت مخصوصاً زمانی که شبکه عمیق و دارای لایه‌های زیادی باشد، ممکن است مقادیر کوچک گرادیان در لایه انتهایی، باعث صفر شدن گرادیان در لایه‌های ابتدایی و عدم آموزش وزن‌های این لایه‌ها شود. این موضوع مخصوصاً زمانی که تابع هزینه‌ی استفاده شده در شبکه دارای ناچیه اشباع زیادی باشد، بسیار شایع است.

۳. مشکل دیگر این روش هم دقیقاً برعکس مشکل قبل است. ممکن است در بعضی موارد گرادیان تابع هزینه نسبت به وزن‌های شبکه خیلی زیاد باشد و موجب ایجاد گام خیلی بزرگی در بروزرسانی وزن‌ها شده و مدل را واگرا کند.

۲.۱.۱ راه حل‌ها

حال می‌خواهیم در مورد تکنیک‌هایی در روش گرادیان کاهشی صحبت کنیم و عملکرد آن‌ها را از جهت رفع مشکلات ذکر شده بررسی کنیم.

۱. روش Momentum قاعده بروزرسانی وزن‌ها در روش گرادیان کاهشی به صورت زیر است:

$$\omega^+ = \omega^- - \alpha \nabla J$$

در این روش گرادیان تابع هزینه نسبت به وزن‌ها محاسبه می‌شود اما نکته مهم این است که گرادیان را نه به روش تحلیلی بلکه به صورت عددی و با استفاده از دیتا محاسبه می‌کنیم. در این صورت مخصوصاً زمانی که اندازه‌ی batch های استفاده شده کوچک باشند، گرادیان محاسبه شده، نویزی خواهد بود و راستای گرادیان الزاماً ما را به نقطه بهینه سراسری نمی‌رساند.

روش momentum در واقع یک moving average را معرفی می‌کند. ایده‌ی اصلی به این صورت است که وقتی یک دنباله از دیتای نویزی داشته باشیم می‌توانیم با استفاده از moving average نویز دیتا را تاحدی رفع کنیم. در واقع به نحوی فیلتر پایین‌گذاری را بر روی دیتا اعمال می‌کنیم. در صورتی که دنباله دیتای نویزی با S و مقدار moving average را با V نشان دهیم، خواهیم داشت:

$$V_t = \beta V_{t-1} + (1 - \beta) S_t$$

در این رابطه مقدار β عددی بین صفر و یک است که میزان حذف نویز را مشخص می‌کند. در صورتی که مقدار این پارامتر کم باشد، دیتای جدید اهمیت بیشتری نسبت به میانگین کنونی دارد و از این رو نتیجه همچنان نویزی و نزدیک به دیتای اصلی خواهد بود. اما اگر مقدار این پارامتر را نزدیک به یک انتخاب کنیم، نتیجه یک تابع smooth است که تغییرات دیتا را به کندی دنبال می‌کند.

اما این روش چگونه در گرادیان کاهشی استفاده می‌شود و چرا نتیجه مطلوبی دارد؟

همانطور که گفته شد گرادیان مخصوصاً در روش‌های mini batch نویزی است و ممکن است وزن‌های مدل را به نقطه اشتباہی سوق دهد. از این رو می‌توانیم ایده moving average را با روش گرادیان کاهشی ترکیب کنیم. بنابراین قاعده‌ی بروزرسانی زیر پیشنهاد شده است:

$$\begin{aligned} V_t &= \beta V_{t-1} + (1 - \beta) \nabla J(\omega) \\ \omega^+ &= \omega^- - \alpha V_t \end{aligned}$$

در این معادله به جای استفاده مستقیم از مقدار گرادیان محاسبه شده که می‌تواند نویزی باشد، از مقدار moving average استفاده می‌شود که با انتخاب پارامتر β می‌توانیم آن را smooth تر کنیم.

در واقع همانطور که گفته شد، محاسبه گرادیان در حالت عادی نمی‌تواند الزاماً ما را به در جهت بهینه سوق دهد زیرا دارای نویز است. اما استفاده از روش momentum تخمین بهتری را از گرادیان به ما می‌دهد که با استفاده از آن می‌توانیم با اطمینان بیشتری در جهت بهینه حرکت کنیم. همین موضوع باعث می‌شود تا الگوریتم گرادیان کاهشی با momentum بتواند همگرایی سریع‌تری داشته باشد.

۲. روش Adam این روش یکی از شاخه‌ها و توسعه‌های روش گرادیان کاهشی است. در این روش از دو مزیت روش‌های RMSProp و AdaGrad استفاده می‌شود. این روش از میانگین ممکن دوم گرادیان استفاده می‌کند و روش بروزرسانی آن به صورت زیر است:

$$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\hat{\nu}_t} + \epsilon} \hat{m}_t$$

که در این معادله داریم:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{\nu}_t = \frac{\nu_t}{1 - \beta_2^t}$$

این دو پارامتر بایاس و نرخ یادگیری به صورت زیر بروزرسانی می‌شوند:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla J \\ \nu_t &= \beta_2 \nu_{t-1} + (1 - \beta_2) [\nabla J]^2 \end{aligned}$$

این روش در مسائل بهینه‌سازی غیرمحدب مزیت‌های زیادی دارد، از جمله اینکه پیاده‌سازی آن آسان است و به لحاظ محاسباتی بهینه است. از نظر حافظه بهینه است و برای مسائل با ابعاد بالا قابل استفاده است. همچنین در مسائل نویزی نیز عملکرد خوبی دارد.

۳. روش AdaDelta این روش همانند RMSProp یک بهینه‌سازی برای Adagrad است که بیشتر روی نرخ یادگیری و تغییرات آن تمرکز دارد. معادله بروزرسانی در این روش به صورت زیر است:

$$\omega_{t+1} = \omega_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{\nu_t} + \epsilon} \nabla J$$

در این معادله داریم:

$$\begin{aligned} D_t &= \beta D_{t-1} + (1 - \beta)[\Delta\omega_t]^2 \\ \nu_t &= \beta \nu_{t-1} + (1 - \beta)[\nabla J]^2 \end{aligned}$$

$$\Delta\omega_t = \omega_t - \omega_{t-1}$$

مشاهده می‌شود که به جای ذخیره‌سازی توان دوم گرادیان که غیر بهینه است، مجموع گرادیان‌های به صورت بازگشتی ذخیره می‌شود و از تفاضل وزن‌ها برای بروزرسانی وزن جدید استفاده می‌شود. بزرگ‌ترین مزیت این روش این است که نیازی به تنظیم یک مقدار اولیه برای نرخ یادگیری نیست.

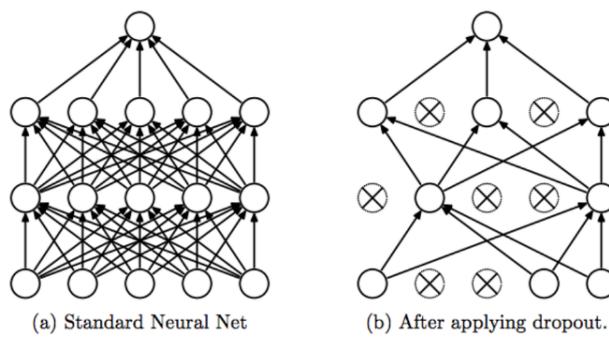
۲.۱ قسمت دوم

در یک مدل یادگیری ماشین، مثلاً شبکه عصبی، زمانی که پیچیدگی شبکه یا مدل از مدل اصلی دیتا بیشتر باشد، یا حجم دیتا به نسبت تعداد پارامترهای مدل خیلی کم باشد، ممکن است شبکه چسبیدگی زیادی به دیتا پیدا کند و از این رو کوچک‌ترین نویز و دیتاها پردازش را هم حفظ کند. از این رو مدل عمومیت خود را برای پیش‌بینی و مواجهه با دیتاها جدید و دیده نشده از دست می‌دهد که به این پدیده فرابرازش می‌گویند. بیش برآذش پدیده‌ی نامطلوبی در آمار است که در آن درجه آزادی مدل بیشتر از درجه آزادی واقعی انتخاب شده، این اتفاق می‌تواند بر اثر تکرار بی روابط بررسیدن به دقت بیشتر نیز باشد که در نتیجه اگرچه مدل روی داده استفاده شده برای یادگیری بسیار خوب نتیجه می‌دهد، اما بر روی داده جدید دارای خطای زیاد است. انتخاب درجه آزادی مناسب به کمک وارسی اعتبار (Cross-validation) و تنظیم‌کردن (Regularization) از راههای مقابله با این پدیده است.

روش‌های متعددی برای رفع این مشکل ارائه شده‌اند که در ادامه در مورد سه روش مهم صحبت می‌کنیم:

۱. روش اول dropout نام دارد. این روش به این صورت است که در روند یادگیری شبکه عصبی، درصد مشخصی از نورون‌های یک لایه را (مثلاً ۳۰ درصد) به صورت تصادفی (با احتمال p) حذف کرده و اتصالات مربوط به آن‌ها را نیز قطع می‌کنیم. بدین صورت در روند back propagation نیز این نورون‌ها بروزرسانی نخواهند شد.

ایدهی کلی این روش آن است که در روند آموزش شبکه عصبی، به نحوی یک وابستگی مشترک بین نورون‌ها شکل می‌گیرد و از این رو توانایی یک تک نورون برای یادگیری محدود می‌شود. این روش باعث می‌شود تا با حذف تصادفی تعدادی از نورون‌ها در هر epoch وابستگی بین نورون‌ها کاهش پیدا کند و نورون‌هایی که به هر دلیلی تاکنون فعال نشده‌اند و نقش کمی در یادگیری داشته‌اند، در یادگیری نقش بیشتری داشته باشند. چون ممکن است در روند یادگیری تنها تعدادی از نورون‌ها فعال باشند و خروجی باقی نورون‌ها مقدار خیلی کمی باشد، از این رو با حذف نورون‌هایی که تاکنون نقش مهمی در یادگیری داشته‌اند باعث می‌شویم تا سایر نورون‌ها نیز آموزش بیینند.



در واقع این روش با ایجاد یک معماری متفاوت از نورون‌ها در هر epoch، شبکه عصبی را قادر می‌کند تا ویژگی‌های robust تری را از دنیا استخراج کند و یاد بگیرد.

۲. در حالت کلی ما به دنبال یافتن وزن‌هایی هستیم که تابع هزینه‌ی تعریف شده برای شبکه عصبی را کمینه کند.
اما این روش می‌تواند منجر به بزرگ شدن وزن‌های شبکه شود. برای همین منظور ترم جدیدی را به تابع هزینه اضافه می‌کنیم:

$$\ell = \text{loss} + \lambda \omega^T \omega$$

افزودن این ترم که در واقع نرم دوم بردار وزن‌های شبکه است، موجب می‌شود که افزایش وزن‌های شبکه جریمه شود. زیرا در حالت کلی با کمینه کردن تابع هزینه جدید، به بردار وزن‌هایی مرسیم که درایه‌های آن بیش از اندازه بزرگ نمی‌شوند.

اما بزرگ بودن وزن ها چه مشکلی می تواند داشته باشد؟ مساله این است که اگر یک سری از وزن های شبکه زیادی بزرگ شوند، مدل پیچیده تر می شود، زیرا بعضی وزن ها خیلی بزرگ و بعضی خیلی کوچک هستند، از این رو مدل نسبت به ورودی خاصی حساسیت بیشتری پیدا کرده است یا به عبارتی مدل واریانس زیادی دارد. در حالی که اگر دامنه مقادیر وزن های شبکه کوچک تر باشد، توانایی محدود تری برای مدل کردن الگوهای پیچیده مثل نویز خواهد داشت. در واقع شبکه با وزن های بزرگ پیچیده تر می شود، در حالی که طبق اصل Occam's razor ما به دنبال شبکه های ساده تری هستیم تا به generalization بیشتری برسیم.

نکته دیگر این است که بردار ویژگی‌های شبکه شامل تعدادی ورودی است که هر یک از آن‌ها همبستگی متفاوتی با خروجی دارند و بعضی از آن‌ها حتی غیرمربوط هستند. بعضی موقع نمی‌توانیم به سادگی این بردار ویژگی را اصلاح کنیم و ویژگی‌های ناهمبسته را حذف کنیم. اما ایجاد یک جریمه برای اندازه بردار وزن‌های شبکه و کاهش اندازه وزن‌ها باعث می‌شود تا وزن‌های مربوط به ویژگی‌هایی با همبستگی پایین یا ناهمبسته، نزدیک به صفر شود و از این رو به مدل ساده‌تری برسیم که نتیجتاً باعث کاهش فرایبارازش مدل می‌شود.

۳. روش سوم برای جلوگیری از فرایرداش مدل استفاده از Early Stopping است. در روند یادگیری وزن‌های شبکه عصبی به تدریج مقدار تابع هزینه برای دیتای آموزشی کاهش پیدا می‌کند. اما مقدار این تابع را برای دیتای تست نیز باید بررسی کنیم، زیرا در صورتی که شبکه دچار فرایرداش شود، این مقدار برای دیتای آموزشی همچنان کاهش می‌یابد اما چون مدل به تدریج عمومیت و توانایی پیش‌بینی برای دیتای جدید را از دست می‌دهد، مقدار تابع هزینه برای دیتای تست زیاد خواهد شد.

بنابراین با بررسی مقادیر تابع هزینه برای دیتای آموزشی و تست می‌توانیم دریابیم که مدل چه زمانی دچار فراگیرازش می‌شود و هر زمان که این مقدار برای دیتای تست، ثابت ماند یا افزایش یافت، فرآیند آموزش را متوقف کنیم.

۳.۱ قسمت سوم

مساله اصلی این است که با استفاده از دو لایه ما می‌توانیم هر تابعی را مدل کنیم، از این رو می‌توانیم هر مپینگ را از هر دیتای آموزشی به فضای خروجی آن، پیدا کنیم. اما نکته اینجاست که در اکثر مواقع دیتا دارای عدم قطعیت‌هایی است و ممکن است شامل نویز باشد. از این رو اگر شبکه تنها تابع مپینگ بین ورودی و خروجی را یاد بگیرد، نمی‌تواند generalization مناسبی برای پیش‌بینی دیتاها دیده نشده داشته باشد. از این رو در بسیاری از موارد، تعداد لایه‌های شبکه را افزایش می‌دهیم تا شبکه را در مقابل این نایقینی مقاوم کنیم و بتوانیم تابع بهتر و با generalization بالاتری را برای مپینگ ورودی خروجی، پیدا کنیم.

۴.۱ قسمت چهارم

یکی از دلایلی که در انتخاب تعداد لایه‌ها و پارامترهای شبکه محدود هستیم این است که با افزایش تعداد پارامترها، مدل پیچیده‌تر شده و از این رو احتمال فرابرازش آن زیاد می‌شود. همچنین موضوع دیگری که ما را در افزایش تعداد لایه‌ها محدود می‌کند این است که هرچه تعداد لایه‌های شبکه را زیاتر کنیم و عمق شبکه را افزایش دهیم، احتمال رخداد پدیده‌ی vanishing gradient زیاد می‌شود.

نکته دیگری که وجود دارد این است که افزایش تعداد پارامترهای مدل باعث افزایش پیچیدگی محاسباتی و طولانی شدن روند آموزش مدل خواهد شد و از این رو برای آموزش مدل ناچار به استفاده از سخت‌افزارهای خیلی قوی خواهیم بود.

۵.۱ قسمت پنجم

یکی از مهم‌ترین نکاتی که در انتخاب مقدار اولیه برای وزن‌های شبکه وجود دارد این است که باید تقارن بین نورون‌های مختلف را از بین ببریم. اگر دو نورون با تابع فعال‌ساز یکسان داشته باشیم که به ورودی‌های یکسانی متصل باشند(دو نورون در یک لایه مخفی) این نورون‌ها باید پارامترهای اولیه متفاوتی را در آغاز فرآیند آموزش داشته باشند. اگر این نورون‌ها مقدارهای اولیه یکسانی برای وزن‌ها داشته باشند، الگوریتم یادگیری deterministic که بر روی یک تابع هزینه و مدل deterministic اعمال می‌شود، موجب خواهد شد تا هر دوی این نورون‌ها به صورت یکسانی آموزش بیینند.

حتی اگر مدل یا الگوریتم یادگیری قابلیت استفاده از stochasticity به منظور آپدیت متفاوت نورون‌های متفاوت را داشته باشد(برای مثال استفاده از dropout) باز هم این خطر وجود دارد که نورون‌ها به نحو مشابهی بروزرسانی شوند. در واقع بهتر است که مقداردهی اولیه را به گونه‌ای انجام دهیم تا هر نورون تابع متفاوتی را نسبت به بقیه محاسبه کند از این رو باید مقدارهای اولیه وزن‌های هر نورون را متفاوت از بقیه انتخاب کنیم.

۶.۱ قسمت ششم

همانطور که در بخش‌های قبل توضیح دادیم، vanishing gradient در روند back propagation و رسیدن به لایه‌های ابتدایی، خیلی کوچک می‌شوند و از این رو در محاسبه گرادیان لایه‌های ابتدایی شبکه طبق قانون زنجیره‌ای، گرادیان این لایه‌های نزدیک به صفر خواهد شد و از این رو یادگیری در لایه‌های ابتدایی خیلی آهسته انجام می‌شود. در نتیجه مدل در حال یادگیری است اما از سرعت یادگیری کاسته می‌شود تا جایی که دیگر یادگیری صورت نمی‌گیرد.

اما در exploding gradient مساله به این صورت است که در یک شبکه n لایه، برای محاسبه وزن‌ها باید طبق قانون زنجیره‌ای، n مشتق را در هم ضرب کنیم و در صورتی که این گرادیان‌ها مقداری بزرگی باشند، نتیجه‌ی این ضرب خیلی بزرگ خواهد بود و نتیجتاً این گرادیان خیلی بزرگ، گام بزرگی را در روند بروزرسانی وزن‌ها موجب می‌شود که می‌تواند مدل را دچار واگرایی کند. در این حالت مدل فاقد پایداری است و در هر epoch تغییرات زیادی را در مقدار تابع هزینه خواهد داشت.

از این رو در exploding gradient برخلاف vanishing gradient یادگیری دچار واگرایی می‌شود و مدل حتی در جهت کمینه تابع هزینه حرکت نمی‌کند، زیرا گام بزرگ در بروزرسانی وزن‌ها می‌تواند شبکه را از نقطه بهینه دور کند.

۲ بخش دوم

۱.۲ مشخصات شبکه عصبی

۱. اندازه‌ی پنجره کانولوشن برابر با ۳ انتخاب شده است و همچنین به منظور یکسان بودن خروجی کانولوشن و ورودی آن، stride برابر با ۱ و padding برابر با ۱ انتخاب شده است. همچنین تعداد فیلترهای لایه اول ۳۲، لایه دوم ۶۴ و لایه سوم برابر با ۱۲۸ انتخاب شده است. (هر لایه از دو کانولوشن با اندازه کرنل ۳ در ۳ تشکیل شده است، برای مثال لایه اول شامل دو کانولوشن هر کدام با ۱۶ فیلتر می‌باشد که مجموعاً ۳۲ فیلتر لایه اول را تشکیل می‌دهد).
۲. در بخش کانولوشنی این شبکه از تابع فعال‌ساز ReLU و تابع فعال‌ساز لایه خروجی نیز softmax است تا خروجی‌های شبکه بیان احتمالاتی داشته باشند.
۳. در بخش fully connected شبکه از ۲ لایه مخفی با تعداد ۱۰۲۴ و ۵۱۲ نورون استفاده شده است. با توجه به اینکه در این بخش از دیتابست cifar-10 استفاده کرده‌ایم، تعداد نورون‌های لایه خروجی برابر با ۱۰ می‌باشد.
۴. با توجه به ذات مساله طبقه‌بندی چند کلاسه، تابع هزینه‌ی categorical cross entropy را انتخاب کرده‌ایم و از روش Adam به منظور بروزرسانی وزن‌ها استفاده می‌کنیم.
۵. اندازه batch استفاده شده در آموزش این شبکه برابر با ۱۲۸ انتخاب شده است.

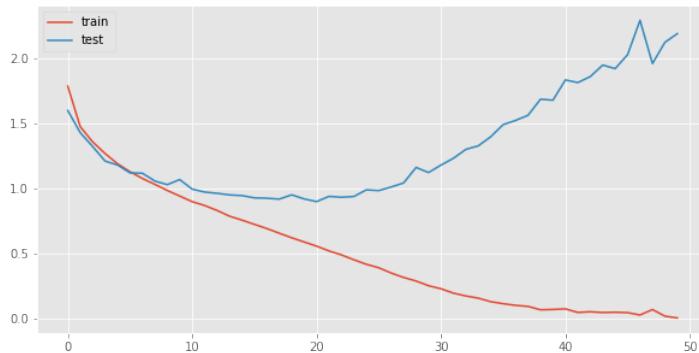
ساختار این شبکه به صورت زیر است:

Model: "MyConvNet"		
Layer (type)	Output Shape	Param #
conv1 (Conv2D)	(None, 32, 32, 16)	448
conv12 (Conv2D)	(None, 32, 32, 16)	2320
pooling1 (MaxPooling2D)	(None, 16, 16, 16)	0
conv21 (Conv2D)	(None, 16, 16, 32)	4640
conv22 (Conv2D)	(None, 16, 16, 32)	9248
pooling2 (MaxPooling2D)	(None, 8, 8, 32)	0
conv31 (Conv2D)	(None, 8, 8, 64)	18496
conv32 (Conv2D)	(None, 8, 8, 64)	36928
pooling3 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
FC1 (Dense)	(None, 1024)	1049600
FC2 (Dense)	(None, 512)	524800
FC3 (Dense)	(None, 10)	5130
<hr/>		
Total params: 1,651,610		
Trainable params: 1,651,610		
Non-trainable params: 0		

شکل ۱: ساختار شبکه

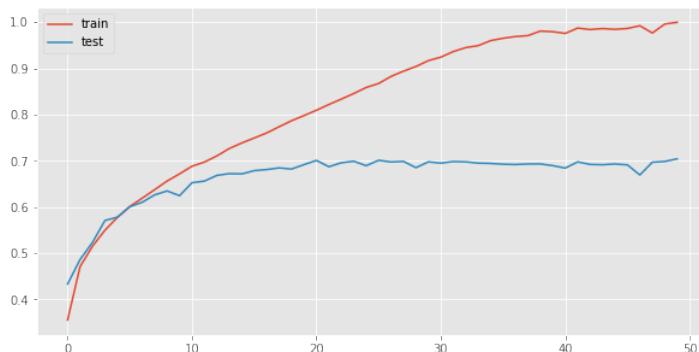
۲.۲ آموزش شبکه

حال در این قسمت شبکه را برای epoch ۵۰ آموزش می‌دهیم و نمودار مقدار تابع هزینه و دقت شبکه را برای دیتای آموزشی و تست بررسی می‌کنیم.



شکل ۲: مقدار تابع خطا

همانطور که مشاهده می‌شود، پس از گذشت تقریباً epoch ۲۲ شبکه دچار فربرازش شده است و مقدار تابع خطا برای دیتای تست افزایش می‌یابد.



شکل ۳: دقت شبکه

بیشترین دقت شبکه بعد از epoch ۲۰ برای دیتای آموزشی برابر با 99.94 درصد و برای دیتای تست 70.42 درصد است. همچنین مدل پس از epoch ۵۰ به بهترین دقت برای دیتای تست، یعنی 70.42 درصد می‌رسد. البته epoch ای که در آن بیشتر دقت رخ می‌دهد می‌تواند از ۲۰ تا ۵۰ متغیر باشد، این موضوع کاملاً وابسته به مقدار دهی اولیه است. اما نکته‌ی اصلی آن است که در تمامی دفعات اجرای یادگیری شبکه، دقت شبکه تقریباً بین ۷۰ تا ۷۲٪ متغیر است.

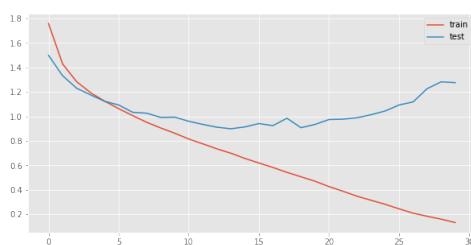
۳.۲ تعداد لایه‌های متفاوت

حال در این بخش می‌خواهیم عملکرد شبکه را به ازای تعداد لایه‌های متفاوت بررسی کنیم. برای این منظور شبکه را به ازای ۱، ۲ و ۳ لایه کانولوشنی مورد بررسی قرار می‌دهیم.

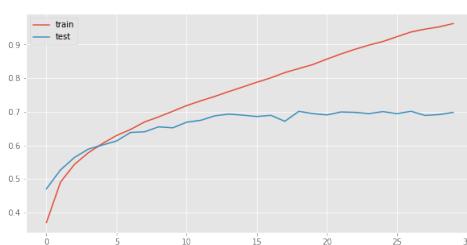
۱.۳.۲ دو لایه

در این شبکه تنها از دو لایه کانولوشنی اول شبکه اصلی استفاده کردیم. لایه اول مشتمل بر ۲ کانولوشن، هر کدام با ۱۶ فیلتر ۳ در ۳ و لایه دوم با دو کانولوشن، هر کدام ۳۲ فیلتر ۳ در ۳.

در این سوال ما از ۳ لایه که هر کدام متشکل از تعدادی فیلتر و یک max pooling بود استفاده کردیم. هر کدام از این لایه‌ها این امکان را به ما می‌دهند تا اولاً با استفاده از فیلتر کانولوشنی، الگوهای بیشتر تکرار شده را در دیتا یاد بگیریم و سپس با استفاده از max pooling ویژگی‌ها به تدریج نسبت به موقعیت، اندازه، رنگ و ... invariant می‌شوند.



شکل ۵: مقدار تابع خطا

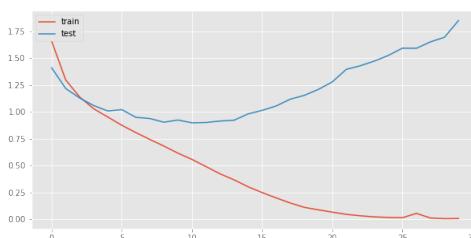


شکل ۶: دقت مدل با دو لایه کانولوشنی

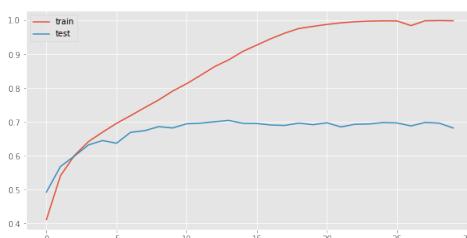
همانطور که مشاهده می‌شود با استفاده از دو لایه کانولوشنی هم تا حد خوبی، ویژگی‌های مناسبی برای طبقه‌بندی تصاویر یاد گرفته می‌شوند و از این رو عملکرد شبکه تقریباً مشابه شبکه سه لایه است. اما در این حالت به دلیل کاهش پیچیدگی مدل، شبکه فرابرازش کمتری را تجربه می‌کند.

۲.۳.۲ یک لایه

در این شبکه تنها از اولین لایه شبکه کانولوشنی مطرح شده در بخش قبل استفاده می‌کنیم.



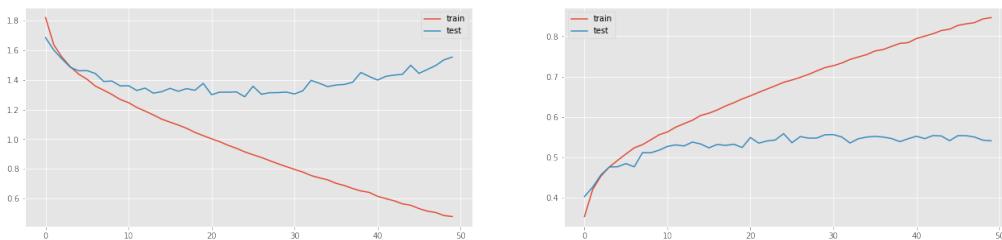
شکل ۷: مقدار تابع خطا



شکل ۸: دقت مدل با یک لایه کانولوشنی

۳.۳.۲ صفر لایه

در این شبکه از هیچ فیلتر کانولوشنی استفاده نمی‌کنیم و پیکسل‌های تصاویر را مستقیماً به طبقه‌بند fully connected تحويل می‌دهیم.



شکل ۹: مقدار تابع خطا

شکل ۸: دقت مدل با صفر لایه کانولوشنی

در این حالت کاهش قابل توجهی را در دقت شبکه شاهد هستیم. زیرا ویژگی‌های دیتا دارای همبستگی زیادی هستند و نویز زیادی را به همراه دارند، اما ما بدون فیلتر کردن و کاهش همبستگی این ویژگی‌ها و انتخاب ویژگی‌ها و الگوهای متدال، اقدام به طبقه‌بندی کردایم. مشاهده می‌شود که شبکه همچنان توانایی آن را دارد تا برای دیتای آموزشی به دقت زیادی برسد، زیرا پیچیدگی مدل به اندازه‌ای هست تا بتواند میانگ ورودی و خروجی را بدست بیاورد، اما دقت برای دیتای تست خیلی کم خواهد بود. زیرا مدل ویژگی‌های مناسبی را برای طبقه‌بندی یاد نگرفته است. در حالی که با استفاده از فیلترهای کانولوشنی و استفاده از max pooling می‌توانیم invariancy را زیادتر کنیم.

۴.۲ بررسی توابع فعال‌سازی

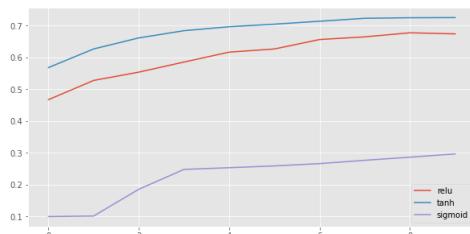
حال می‌خواهیم اثر توابع فعال‌سازی متفاوت را در آموزش این شبکه کانولوشنی ساده مورد بررسی قرار دهیم.

نکته: با توجه به این که در ساختار قسمت‌های قل در هر لایه از دو کانولوشن استفاده کردیم، مدل تا حدی عمیق شده است و این موضوع باعث شد تا به ازای تابع فعال‌سازی مثل sigmoid هیچ یادگیری صورت نگیرد. زیرا این تابع فعال‌ساز، ناحیه اشباع زیادی دارد و از این رو یادگیری و بروزرسانی وزن‌های عملکرد لایه‌های ابتدایی انجام نمی‌شود. از این رو در هر لایه تنها از یک کانولوشن استفاده می‌کنیم و اثر این تابع فعال‌ساز را مورد بررسی قرار می‌دهیم.

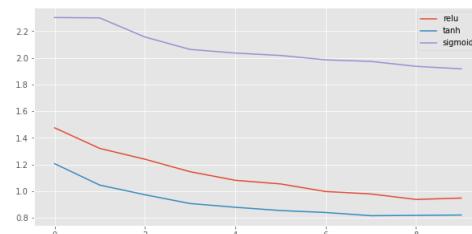
شبکه مورد استفاده برای بررسی اثر تابع فعال‌ساز به صورت زیر است:

Model: "MyConvNet"		
Layer (type)	Output Shape	Param #
conv1 (Conv2D)	(None, 32, 32, 32)	896
pooling1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2 (Conv2D)	(None, 16, 16, 64)	18496
pooling2 (MaxPooling2D)	(None, 8, 8, 64)	0
conv3 (Conv2D)	(None, 8, 8, 128)	73856
pooling3 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
FC1 (Dense)	(None, 1024)	2098176
FC2 (Dense)	(None, 512)	524800
FC3 (Dense)	(None, 10)	5130
<hr/>		
Total params: 2,721,354		
Trainable params: 2,721,354		
Non-trainable params: 0		

شکل ۱۰: ساختار شبکه



شکل ۱۲: دقت شبکه

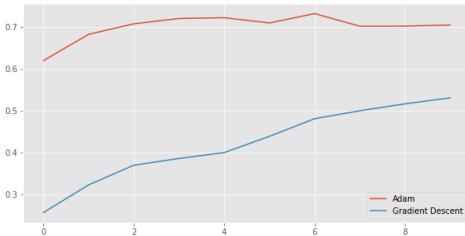


شکل ۱۱: مقدار تابع خطأ

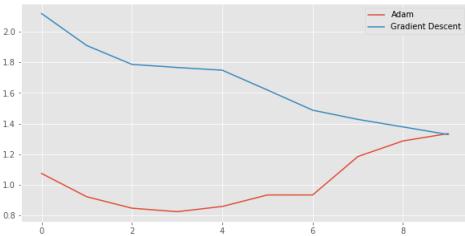
همانطور که در نمودار بالا مشاهده می‌شود، عملکرد شبکه به ازای دو تابع فعال ساز ReLU و \tanh تقریباً مشابه است و به ازای استفاده از تابع sigmoid مدل یادگیری خیلی ضعیفتری را دارد. این موضوع می‌تواند به این دلیل باشد که این تابع ناحیه اشباع زیادی دارد و از این رو احتمالاً مدل دچار پدیده‌ی محوشدنگی گردایان شده و وزن‌ها به خوبی بروزرسانی نمی‌شوند.

بر اساس این تست، دو تابع ReLU و \tanh عملکرد مناسبی داشته‌اند و برای آموزش می‌توانیم از این دو تابع استفاده کنیم. اما با توجه به این موضوع که تابع \tanh دارای ناحیه اشباع برای مقادیر مثبت است، این امکان وجود دارد که در ادامه یادگیری، محوشدنگی گردایان رخ دهد. همچنین چون مقادیر ورودی نورون‌ها همگی اعدادی مثبت هستند، می‌توانیم از تابع ReLU استفاده کنیم که در قسمت مثبت ناحیه اشباع ندارد و سرعت یادگیری مناسبی را در این بررسی، از خود نشان داده است.

۵.۲ بررسی روش بهینه‌سازی



شکل ۱۴: دقت شبکه

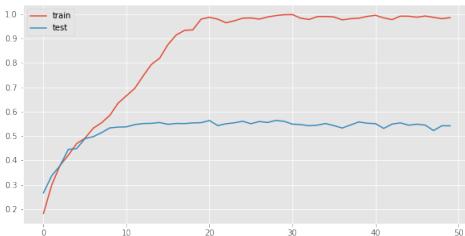


شکل ۱۳: مقدار تابع خطا

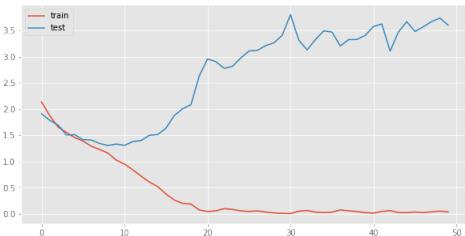
در این بخش همان مدل بخش ۴.۲ را به ازای دو روش بهینه‌سازی، آموزش می‌دهیم. اولین روش، مدل ساده‌ی گرادیان کاهشی است و روش دوم نسخه بهبود یافته‌ی آن یعنی Adam است. همانطور که مشاهده می‌شود، همگرایی روش Adam نسبت به روش معمولی گرادیان کاهشی، سریع‌تر است و از این رو در تعداد epoch کمتری به یک نقطه کمینه محلی می‌رسد. مشاهده می‌شود که مدل به ازای این روش بروزرسانی، دچار فرابرازش شده‌است. می‌توان انتظار داشت که فرابرازش در epoch های آینده، برای روش گرادیان کاهشی نیز رخ دهد، زیرا روش Adam مدل را خیلی سریع‌تر به نقطه کمینه محلی رسانده است و به دلیل یادگیری سریع‌تر، مدل خیلی زودتر به داده‌های آموزشی چسبیده‌است و احتمالاً اگر یادگیری را ادامه دهیم تا به ازای گرادیان کاهشی، به این کمینه محلی برسیم، احتمالاً همین اتفاق مجدداً برای مدل می‌افتد.

۶.۲ کاهش حجم دیتا

حال در این بخش می‌خواهیم حجم دیتاست را به ۶۰۰۰ دیتا کاهش دهیم. انتظار داریم که با این کاهش حجم دیتا، مدل هم به دقت کمتری برسد و هم فرابرازش بیشتری را تجربه کند. زیرا با کاهش حجم دیتای آموزشی مدل، عملاً پیچیدگی الگوهایی که مدل می‌تواند با استفاده از آن‌ها دیتای آموزشی را تفکیک کند، کاهش پیدا می‌کند و از این رو مدل استعداد بالاتری را برای چسبیدگی به این حجم دیتای اندک دارد. زیرا پیچیدگی مدل نسبت به حجم دیتای آموزشی بسیار زیاد است.



شکل ۱۶: دقت شبکه



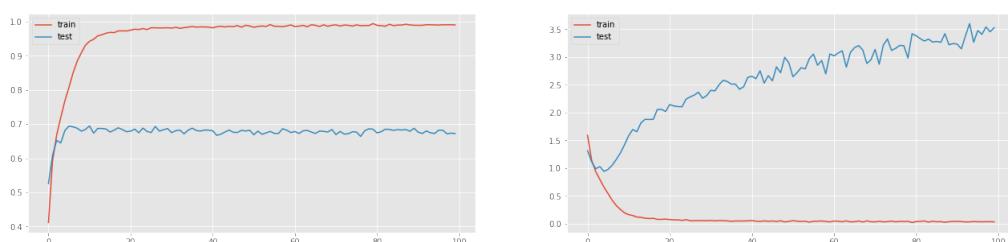
شکل ۱۵: مقدار تابع خطا

همانطور که انتظار داشتیم، مدل به ازای این دیتاست کم‌حجم، فرابرازش زیادی را تجربه می‌کند و در مدت زمان epoch ۲۰ دقت مدل برای دیتای آموزشی به ۱۰۰ درصد رسیده در حالی که دقت دیتای تست، از دقت به دست آمده در بخش ۴.۲ نیز کمتر است. کمتر بودن دقت مدل می‌تواند به این دلیل باشد که شبکه با این حجم دیتای نتوانسته است

الگوهای مناسبی را برای تفکیک دیتای کلاس‌های مختلف یاد بگیرد و از این رو قابلیت تعمیم کمتری برای طبقه‌بندی دیتاهای دیده نشده دارد.

۷.۲ استفاده از کرنل بزرگ تر

در تمامی بخش‌های قبل در هر لایه از مدل کانولوشنی، از دو کانولوشن با اندازه کرنل ۳ در ۳ استفاده کرده بودیم. اما در این بخش می‌خواهیم این دو کانولوشن را با یک کانولوشن، متنهای با کرنل بزرگ‌تر جایگزین کنیم. از این رو از کرنل با اندازه ۷ در ۷ استفاده کرده و عملکرد مدل را در این حالت بررسی می‌کنیم:



شکل ۱۸: دقت شبکه

شکل ۱۷: مقدار تابع خطأ

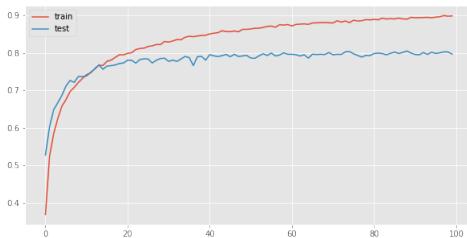
در نگاه اول، این افزایش اندازه‌ی کرنل، باعث بیشتر شدن فرابرازش شده است، زیرا تعداد پارامترهای مدل را افزایش داده ایم و از این رو پیچیدگی مدل زیادتر شده است. اما دقت مدل برای دیتای تست چندان دچار تغییر نشده است.

در حالت کلی جایگزین کردن کرنل با سایز بزرگ، با چند کرنل کوچک‌تر می‌تواند حجم محاسباتی و پیچیدگی مدل را کاهش دهد. برای مثال در مدل Inception که در بخش چهارم به آن می‌پردازیم، در نسخه‌های ابتدایی آن، در هر مازول Inception چندین فیلتر با اندازه‌های مختلف وجود دارد. یکی از این فیلترها یک فیلتر ۵ در ۵ است. اما در نسخه‌های ۲ و ۳ به این موضوع اشاره شده است که برای کاهش پیچیدگی محاسباتی و زمانی مدل، این فیلتر را با دو فیلتر ۳ در ۳ جایگزین کرده‌اند.

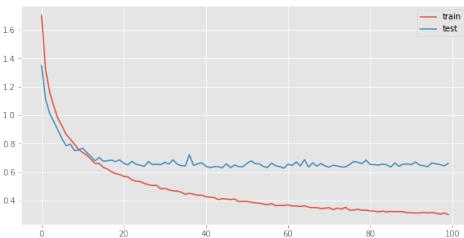
اما اندازه‌ی کرنل وابسته به دیتایی که می‌خواهیم بررسی کنیم می‌تواند متغیر باشد. گاهی الگوهایی که می‌خواهیم آن‌ها را بررسی کنیم، خیلی الگوهای محلی هستند و بررسی آن‌ها در دیتای ورودی، نیازمند استفاده از یک کرنل کوچک است که به ما این امکان بررسی محلی را بدهد. در حالی که اگر از کرنل بزرگ‌تر استفاده کنیم، با دید بزرگ‌تری الگوهای موجود در دیتا را بررسی می‌کنیم و ممکن است بعضی از این الگوها را از دست بدهیم.

Dropout ۸.۲

در تمامی قسمت‌های گذشته، مشاهده کردیم که استعداد مدل برای فرابرازش زیاد است و مدل در مدت کوتاهی، چسبندگی زیادی را به دیتای آموزشی پیدا می‌کند. از این رو می‌توانیم از روش‌های ارائه شده که در سوال اول به آن‌ها پرداختیم، استفاده کنیم. یکی از این روش‌ها dropout نام دارد که در سوال اول مفصلًا در مورد آن صحبت کردیم. حال پس از هر یک از لایه‌های شبکه، یک لایه dropout قرار می‌دهیم و سپس عملکرد مدل را در این حالت بررسی می‌کنیم:



شکل ۲۰: دقت شبکه



شکل ۱۹: مقدار تابع خطا

پس از بررسی‌های زیاد، درصد dropout را برای لایه‌های کانولوشنی برابر با ۲۰ درصد و برای لایه‌های flatten و FC1 برابر با ۳۰ درصد و برای لایه‌ی FC2 برابر با ۴۰ درصد انتخاب کردیم. همانطور که در نمودار بالا مشاهده می‌شود، مدل برخلاف بخش‌های قبل، دچار فراپرازش نشده است و روند کاهش خطای دیتای آموزشی و تست، شیاهت بیشتری به هم دارند. همچنین دقت مدل در این حالت به ۸۰ درصد رسیده است که نشان‌دهنده این موضوع است که با مجبور کردن نورون‌های غیرفعال به یادگیری از طریق غیرفعال کردن تصادفی نورون‌های هر لایه، توانستیم شبکه را به سمتی سوق دهیم تا الگوهای بهینه‌تری را از دیتا یاد بگیرد. این موضوع به این دلیل است که روش dropout با جلوگیری شبکه از چسبیدگی به دیتا، این امکان را فراهم کرد تا یادگیری بیشتری داشته باشد و به الگوهای بهتری برای طبقه‌بندی برسد.

همچنین ما در این بخش برای بررسی اضافه، لایه‌ی Batch Normalization را نیز به شبکه اضافه کردیم و مشاهده شد که روند یادگیری سرعت بیشتری پیدا می‌کند و دقت مدل در مواردی حتی به ۸۸ درصد نیز می‌رسد. دلیل بهبود شبکه در این حالت این است که در فرآیند آموزش، بعضی از نورون‌ها خیلی فعال می‌شوند و بعضی دیگر نقش کمتری در یادگیری الگوها دارند. اما با استفاده از این روش و غیرفعال کردن تصادفی نورون‌ها، نورون‌های غیرفعال را قادر می‌کنیم تا در روند آموزش نقش بیشتری داشته باشند. همچنین از زاویه‌ی دیگری می‌توان به موضوع نگاه کرد. به این صورت که در هر epoch ما عملاً ساختار شبکه‌ی جدیدی را آموزش می‌دهیم و این موضوع چسبندگی مدل به دیتای آموزشی را کاهش می‌دهد.

۳ بخش سوم

۱.۳ Augmentation

در روند آموزش مدل‌های یادگیری عمیق، فراوانی و تنوع دیتا از اهمیت بالایی برخوردار است. به این دلیل که مدل باید بتواند انواع مختلفی از دیتا را مشاهده کند و در نهایت ویژگی‌هایی را از آن استخراج کند که با استفاده از آن‌ها بتوان دیتاهای دیده نشده را نیز به درستی طبقه‌بندی کرد. نکته مهم این است که گاهی حجم دیتا در دسترس کافی نیست و همچنین تنوع کافی را ندارد. از این رو ممکن است مدل به یادگیری ویژگی‌هایی پردازد که شاید عمومیت کافی را برای تشخیص دیتاهای جدید نداشته باشد. بنابراین سته به نوع دیتا بی که در شبکه استفاده می‌کنیم، می‌توانیم تبدیل‌هایی را بر روی دیتا اعمال کرده و دیتاهای جدیدی را ایجاد کنیم.

برای مثال در طبقه‌بندی‌های مبتنی بر تصویر، تبدیل‌هایی نظیر چرخش، تغییر مقیاس و بزرگنمایی، flip و ... بر روی دیتاهای اعمال می‌شود. این کار علاوه بر آنکه می‌تواند حجم دیتا را افزایش دهد، موجب می‌شود که مدل دریابد تغییرات اندکی که به واسطه این تبدیل‌ها انجام شده‌است، تغییری در مفهوم تصویر ایجاد نمی‌کند و از این رو مدل نسبت به این ویژگی‌ها حساس نخواهد شد. مثلاً اگر در فرآیند Augmentation اندکی نور تصویر را تغییر دهیم، مثلاً مدل نسبت به تغییر نور در تصاویر ماشین حساسیت کمتر خواهد داشت و می‌تواند تصاویر ماشین را در شرایط نوری متنوع تری تشخیص دهد.

ذکر این نکته حائز اهمیت است که این روش در منابع مختلف به نحو مختلف بیان می‌شود. اولین نوع آن است که از دیتای محدود موجود استفاده می‌کنیم و با اعمال تبدیل‌هایی تصاویر جدیدی را ایجاد کرده و دیتاست را گسترش می‌دهیم. در این روش باید توجه داشت که به هر حال دیتاست پایه‌ای ما همچنان محدود است و ممکن است حتی با اعمال این تبدیل‌ها هم نتوانیم جامعیت مدل را افزایش دهیم.

برای مثال فرض کنید که تنها یک تصویر از ماشین داریم و با استفاده از Augmentation می‌خواهیم ۱۰۰۰ نمونه از آن را ایجاد کنیم. ولی مدل در نهایت تنها یک نمونه از ماشین را در این تصاویر دیده است و نتوانسته ویژگی‌های کلی که یک ماشین را مثلاً از هوایپما تمایز می‌کند تشخیص دهد. اما بیان دومی از Augmentation که متدال‌تر است به این صورت است که در هر epoch در فرآیند یادگیری، به صورت تصادفی تبدیل‌هایی را بر روی دیتا ایجاد کنیم تا دیتا با variation های جدیدی از دیتا مواجه شود و در نهایت به هدف خود که بالاتر بردن قدرت تعمیم مدل بود برسیم. در این حالت، در هر epoch، شبکه تنها تبدیل یافته‌ی دیتا را مشاهده می‌کند و دیتای اصلی به شبکه داده نمی‌شود.

نکته‌ی مهم این است که روش برای دیتای آموزشی مورد استفاده قرار می‌گیرد زیرا هدف کلی این است که مدل را با انواع جدیدی از دیتای اصلی (که مفهوم اصلی خود را حفظ کرده‌اند و دارای sub-pattern های متدال آن کلاس هستند) مواجه کنیم تا مدل بتواند قدرت تعمیم بالاتری را داشته باشد و بتواند با وجود این تغییرات اندک (مثلاً تغییرات در شدت نور تصویر یا مقیاس آن) دیتاهای جدید را طبقه‌بندی کند. از این رو استفاده از این روش بر روی دیتای تست عملاً مفهوم و توجیهی ندارد.

۲.۳ ایجاد نمونه مصنوعی

در این بخش می‌خواهیم با استفاده از تابع `ImageDataGenerator` ۱۰ نمونه مصنوعی را از تصویر داده شده ایجاد کنیم.

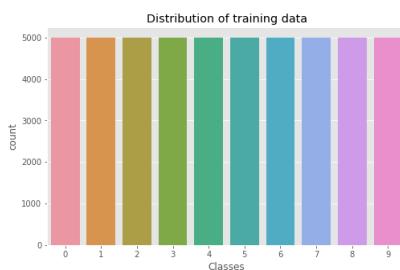


شکل ۲۱: تصاویر مصنوعی ایجاد شده

همانگونه که در این تصاویر مشاهده می‌شود، با اعمال تبدیل‌هایی بر روی تصویر اولیه، تصاویر جدیدی را ایجاد کرده‌ایم که هنوز هم ویژگی‌های اساسی پرنده بودن را داراست. اما اگر در هر epoch یکی از این تصاویر را به مدل نشان دهیم، مدل در می‌یابد که این جایه‌جایی یا چرخش و تغییر مقیاس در تصویر پرنده باعث نمی‌شود که مفهوم کلی عکس عوض شود و از این رو به دنبال یادگیری ویژگی‌هایی می‌رود که جامعیت بیشتری داشته باشد.

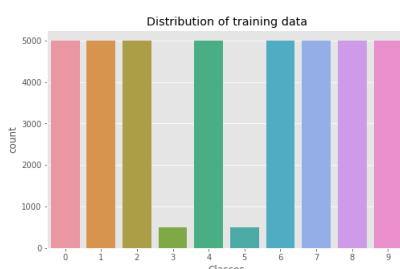
۳.۳ کاهش دیتا

در این بخش می‌خواهیم به صورت عامدانه تعداد زیادی از دیتاهای دو کلاس گربه و سگ را در دیتاست cifar10 حذف کرده و به این صورت عدم توازنی را در دیتاست ایجاد کنیم. قبل از حذف دیتا، توزیع دیتاهای هر کلاس در این دیتاست به صورت زیر است:



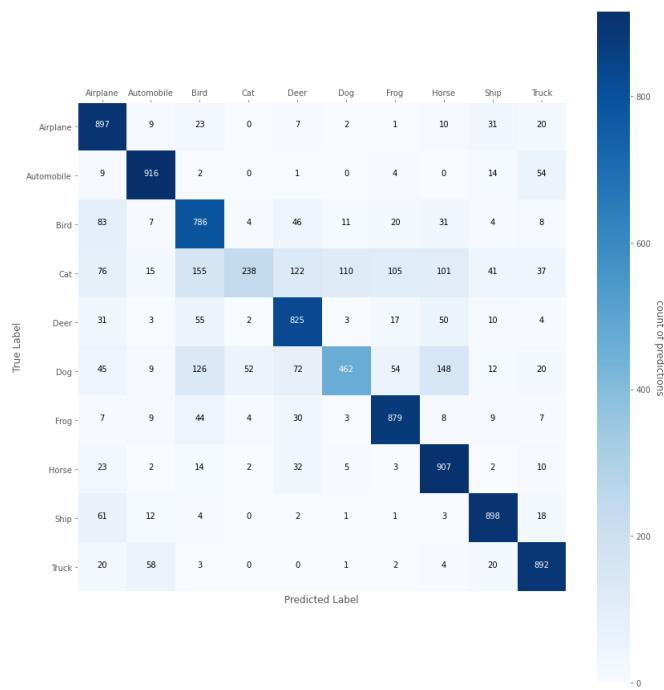
شکل ۲۲: توزیع دیتاهای هر کلاس در دیتاست اصلی

حال به صورت تصادفی، ۴۵۰۰ دیتا را از هر یک از دو کلاس‌های ۳ و ۵ حذف می‌کنیم. توزیع دیتا در نهایت به صورت زیر خواهد بود:



شکل ۲۳: توزیع دیتاهای هر کلاس پس از کاهش دیتا

اکنون بهترین مدل از سوال دوم را با استفاده از این دیتاست جدید آموزش می‌دهیم. دقت مدل در بهترین حالت برابر با ۷۷ درصد و همچنین ماتریس آشفتگی برای این طبقه‌بندی به صورت زیر است:



شکل ۲۴: ماتریس آشفتگی برای یادگیری با دیتای نامتوازن

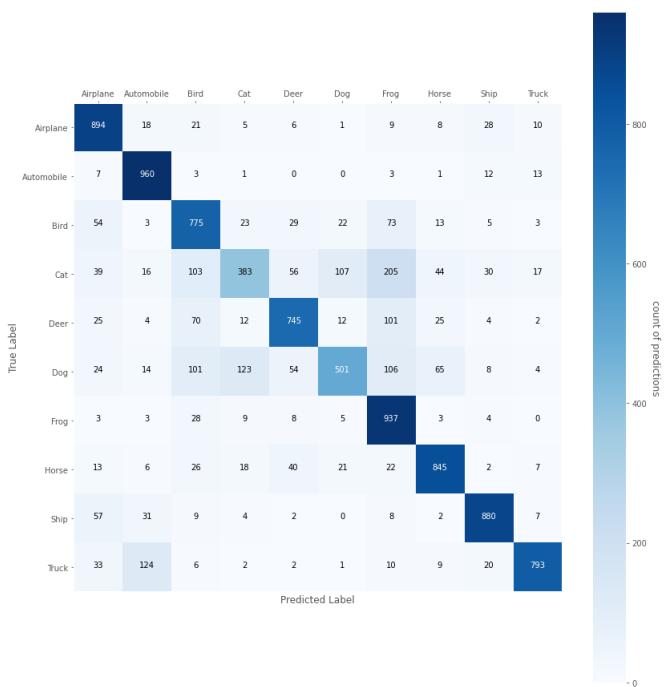
بر اساس این ماتریس، دیتای تمامی کلاس‌ها با دقت تقریباً خوبی طبقه‌بندی شده‌اند به جز کلاس‌های ۳ و ۵. این موضوع به این دلیل است که مدل تعداد کافی نمونه را از این دو کلاس مشاهده نکرده‌است و از این رو نتوانسته است ویژگی‌هایی را استخراج کند که بینگر خصوصیات اصلی گریه یا سگ باشد بنابراین مدل قابلیت تعمیم کافی برای تشخیص دیتاهای جدید و دیده‌نشده از این دو کلاس را ندارد.

۴.۳ حل مشکل عدم توازن دیتاست

فرض را بر این می‌گذاریم که دیتاهای دو کلاس ۳ و ۵ کاملاً مفقود بوده و قابل جایگزینی نیستند. از این رو برای حل مشکل عدم توازن دیتای کلاس‌های مختلف دیتاست می‌توانیم با استفاده از روش Augmentation دیتاهای مصنوعی ImageDataGenerator متعلق به کتابخانه keras استفاده می‌کیم. برای این کار از تبدیلهایی نظیر چرخش، بزرگنمایی، شیفت در عرض و ارتفاع و flip استفاده می‌کیم.

با استفاده از این روش، ۴۵۰۰ دیتای جدید را بر اساس ۵۰۰ دیتای موجود ایجاد می‌کنیم. اما همانطور که قبلاً ذکر شد دیتای ایجاد شده از این روش غنای اطلاعاتی کافی را ندارد و اصلاً جایگزین مناسبی برای دیتای واقعی نیست. در واقع ما تنها با تغییرات اندکی در دیتاست کوچک خود دیتاهایی را ایجاد کردہ‌ایم که همبستگی زیادی با دیتاهای دیتاست کوچک دارند. در واقع از نظر من این روش استفاده از Augmentation چندان کارایی مناسبی ندارد زیرا منطق اصلی این روش این است که ما مدل را با variation های جدید از دیتا مواجه کنیم و از این رو قدرت تعمیم را بالا ببریم،

اما اگر در همان epoch هم دیتای اصلی و هم تمامی تبدیل یافته‌های همان تصویر را به مدل نشان دهیم، مدل عملاً نمی‌تواند ویژگی‌های مناسبی را از این تصاویر با همبستگی بالا، استخراج کند. از این رو من احساس می‌کنم هنوز هم گسترش دیتاست از طریق جمع‌آوری دیتای جدید بهترین راهکار است هرچند بسیار پرهزینه می‌باشد. حال برای بررسی عملکرد شبکه بر روی این دیتاست گسترش یافته با دیتای مصنوعی، می‌توانیم ماتریس آشفتگی زیر را بررسی کنیم.



شکل ۲۵: ماتریس آشفتگی برای یادگیری با دیتاست گسترش یافته

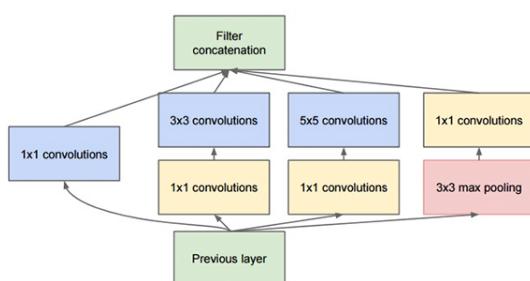
همانطور که مشاهده می‌شود بهبود محسوسی در طبقه‌بندی دیتاهای دو کلاس ۳ و ۵ رخ داد، هر چند که هنوز هم تشخیص دیتاهای این دو کلاس نسبت به باقی کلاس‌های ضعیفتر انجام می‌شود. حتی این موضوع باعث شده است این بار در موارد بیشتری دیتاهای دو کلاس سگ و گربه یا یکدیگر اشتباه گرفته شوند. زیرا همانطور که گفته شد در نهایت این دیتاست بر اساس یک دیتاست بسیار کوچک است و حتی با اعمال تغییر بر دیتا و گسترش دیتاست نمی‌توانیم نمونه‌های کافی را برای یادگیری مدل ایجاد کنیم، هرچند که بهبود واضحی نسبت به یادگیری بر اساس دیتاست نامتوازن را شاهد بودیم.

۴ بخش چهارم

با توجه به شماره دانشجویی اعضاي گروه در اين بخش از مدل Inception استفاده می‌شود.

۱.۴ مدل Inception

اين مدل يك شبکه کانولوشنی عميق است که به واسطه تعریف يك ماژول به نام Inception به اين نام معروف شده است. در لاييه‌های مختلف اين شبکه، از اين ماژول استفاده شده:



شکل ۲۶: ماژول Inception

هدف اين ماژول اين است تا استخراج و پرگی چند سطحی را در يك ماژول از شبکه انجام دهد و اين کار را با انجام کانولوشن با فیلترهای $1 * 1$ ، $1 * 3$ و $5 * 5$ انجام می‌دهد همچنین در کنار آن‌ها، يك max pooling نیز بر روی ورودی انجام می‌شود. سپس خروجی اين فیلترها concatenate شده و به لایه بعدی شبکه داده می‌شود. طبقه‌بندی تصاویر عموماً با مشکلاتی همراه است، برای مثال ممکن است جسمی که می‌خواهیم در تصویر تشخیص دهیم، نسبت‌های متفاوتی را اشغال کرده باشد. برای مثال در تشخیص یک گل، ممکن است، گل تمامی تصویر را اشغال کرده باشد یا تنها بخش کوچکی از آن باشد، برای اینکه بتوان این تنوع را پوشش داد، ایده‌ی استفاده از چند فیلتر با اندازه‌های متفاوت در این شبکه داده شده است.

اولین نسخه از اين ماژول فاقد فیلترهای $1 * 1$ می‌باشد، اما در نسخه‌های بعدی برای کاهش هزینه محاسباتی، این فیلترها اضافه شده‌اند تا بدون تغییر در ابعاد ورودی، تنها عمق ورودی را کاهش دهند.

اولین نسخه این شبکه که با نام GoogLeNet شناخته می‌شود، مشتمل بر ۹ ماژول Inception است که به صورت خطی پشت سر هم قرار گرفته‌اند و درنتیجه از ۲۷ لایه تشکیل شده است. (با در نظر گرفتن لایه‌های pooling) باید توجه داشت که این شبکه يك شبکه کانولوشنی عميق است و بسیار مستعد برای vanishing gradient می‌باشد. از این رو دو طبقه‌بند کمکی در میانه‌ی این شبکه اضافه شده است که مقادیر خطای آن‌ها با وزن کمتری در خطای اصلی شبکه نقش داده می‌شود. هدف از این کار این است که خطای میانی شبکه را نیز در فرآیند یادگیری اثر داده و از غیرفعال شدن شبکه به دلیل محو شدگی گرادیان، جلوگیری شود. بدینهی است که این طبقه‌بندی‌های کمکی تنها در روند آموزش شبکه مورد استفاده قرار می‌گیرند و در روند استنتاج نقشی ندارند.

در نسخه‌های بعدی اين ماژول به منظور کاهش هزینه محاسباتی، فیلترهای بزرگ مثل فیلتر $5 * 5$ با دو فیلتر کوچک‌تر جایگزین شده‌اند و ترکیب‌های متفاوتی از فیلترهای متوالی به منظور کاهش هزینه محاسبات در عین حفظ عملکرد مورد بررسی قرار گرفته است. ساختار اين شبکه در نسخه دوم و سوم مطابق شکل زير است:

type	patch size/stride or remarks	input size
conv	$3 \times 3 / 2$	$299 \times 299 \times 3$
conv	$3 \times 3 / 1$	$149 \times 149 \times 32$
conv padded	$3 \times 3 / 1$	$147 \times 147 \times 32$
pool	$3 \times 3 / 2$	$147 \times 147 \times 64$
conv	$3 \times 3 / 1$	$73 \times 73 \times 64$
conv	$3 \times 3 / 2$	$71 \times 71 \times 80$
conv	$3 \times 3 / 1$	$35 \times 35 \times 192$
$3 \times$ Inception	As in figure 5	$35 \times 35 \times 288$
$5 \times$ Inception	As in figure 6	$17 \times 17 \times 768$
$2 \times$ Inception	As in figure 7	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

شکل ۲۷: ساختار Inception v2

۱.۱.۴ ورودی شبکه

ورودی شبکه مطابق با آن چیزی که در شکل ۲۷ نشان داده شده است، یک تصویر رنگی ۲۹۹ در ۲۹۹ پیکسل با ۳ کanal رنگی است.

۲.۱.۴ پیش پردازش‌های لازم

بله، تصاویر قبل از ورود به این شبکه باید پیش پردازش شوند. مهم‌ترین نکته در پیش‌پردازش تصاویر برای این شبکه آن است که مقدار پیکسل‌های تصویر باید بین ۰ و ۱ باشد. همچنین اندازه‌ی تصویر باید برابر با اندازه‌ی ورودی شبکه باشد. غیر از این دو مورد پیش پردازش خاصی برای تصاویر نیاز نیست.

۳.۱.۴ خروجی شبکه

خروجی بخش کانولوشنی شبکه، یک بردار ۲۰۴۸ عضوی است که مشتمل بر ویژگی‌هایی است که بخش استخراج ویژگی شبکه بدست آورده است و شامل الگوهای پر تکرار در تصاویر است. همچنین در نسخه اصلی شبکه که بر روی دیتاست ImageNet آموزش دیده است، یک بردار ۱۰۰۰ عضوی است که هر نورون در این لایه، احتمال حضور یک object خاص را در لیست هزاراتی اشیایی که در دیتاست حضور دارند، بیان می‌کند.

۲.۴ Transfer Learning

گاهی اوقات حجم دیتاست مربوط به یک پروژه محدود است و با توجه به هدفی که داریم، مجبور به استفاده از یک مدل پیچیده هستیم. در چنین مواردی یکی از راه‌های مناسب، استفاده از روش Transfer Learning است. به این صورت که یک مدل از پیش آموزش دیده را استفاده می‌کنیم. این مدل می‌تواند بر روی حجم زیادی از دیتا، طی ساعتها و با استفاده از GPU های قوی آموزش دیده باشد که عملاً دست‌یابی به همچنین حجم دیتاستی برای هر پروژه به منظور آموزش مدل عمیق مناسب نیست. از این رو بخش‌هایی از این مدل را با استفاده از وزن‌های آموزش دیده در شبکه‌ی خود استفاده می‌کنیم.

برای مثال می‌خواهیم برای تشخیص یک مفهوم در تصویر، از شبکه کانولوشنی عمیق استفاده کنیم، اما به حجم کافی دیتا برای آموزش کامل این شبکه دسترسی نداریم. در این صورت می‌توانیم بخش‌های ابتدایی شبکه را که وظیفه

فیلتر کردن و استخراج ویژگی را بر عهده دارند، به صورت از پیش آموزش دیده استفاده کنیم و تنها چند لایه را به آن اضافه کرده تا با آموزش این لایه‌های نهایی که وظیفه طبقه‌بندی را بر عهده دارند، بتوانیم مساله جدید را حل کنیم.

برای مثال در این بخش از پروژه، ما می‌خواهیم از شبکه‌ی Inception به منظور تشخیص ۵ گونه متفاوت گل استفاده کنیم، در حالی که تنها حدود ۳۰۰۰ تصویر از این گل‌ها را در اختیار داریم. آموزش چنین مدل عمیقی با تقریباً ۲۰ میلیون پارامتر، احتیاج به حجم زیادی دیتا دارد و آموزش مدل به صورت کامل، غیرممکن به نظر می‌رسد. از این رو بخش کانولوشنی مربوط به استخراج ویژگی را از شبکه آموزش دیده شده با دیتاست ImageNet جدا می‌کنیم و وزن‌های آن را نگه می‌داریم و سپس طبقه‌بند جدیدی را مطابق با مساله خود، به آن اضافه می‌کنیم. در این صورت می‌توانیم وزن‌های بخش کانولوشنی را freeze کرده و تنها وزن‌های طبقه‌بند را در روند آموزش، بروزرسانی کنیم.

۳.۴ انجام Transfer Learning

۱.۳.۴ تعریف مدل

ابتدا می‌خواهیم مدل را تعریف کنیم. برای این منظور بخش کانولوشنی را از شبکه Inception بر می‌داریم. این شبکه در حالت عادی از دو بخش کانولوشنی به عنوان استخراج کننده ویژگی و یک بخش fully connected به عنوان طبقه‌بند تشکیل شده‌است. طبقه‌بند این شبکه همانگونه که ذکر شد توانایی تشخیص ۱۰۰۰ Object را دارد. سپس دو لایه مخفی و یک لایه خروجی را به عنوان طبقه‌بند جدید برای این مساله تعریف می‌کنیم. دو لایه خروجی به ترتیب ۱۰۲۴ و ۵۱۲ نورون دارند و تابع فعال‌سازی آن‌ها ReLU استفاده شده‌است. همچنین لایه خروجی شامل ۵ نورون است، زیرا در این دیتاست جدید می‌خواهیم ۵ نوع گل مختلف را شناسایی کنیم. تابع فعال‌ساز این لایه خروجی softmax است. ساختار این شبکه در شکل زیر نشان داده شده‌است:

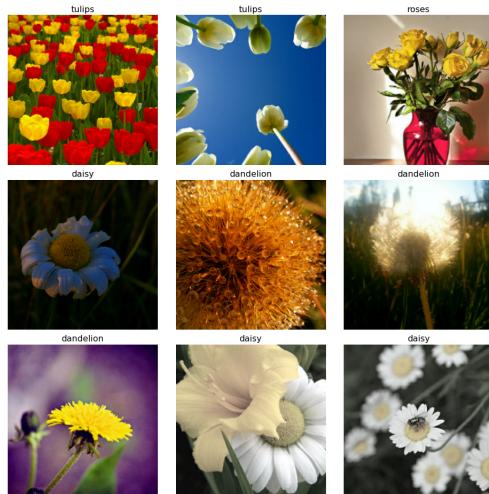
Model: "myInception"		
Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 8, 8, 2048)	21802784
global_average_pool (GlobalA)	(None, 2048)	0
dropout1 (Dropout)	(None, 2048)	0
FC1 (Dense)	(None, 1024)	2098176
dropout2 (Dropout)	(None, 1024)	0
FC2 (Dense)	(None, 512)	524800
dropout3 (Dropout)	(None, 512)	0
output (Dense)	(None, 5)	2565
<hr/>		
Total params: 24,428,325		
Trainable params: 2,625,541		
Non-trainable params: 21,802,784		

شکل ۲۸: ساختار شبکه

بعد از هر لایه طبقه‌بند به منظور جلوگیری از فرابرازش، از تکنیک dropout استفاده کرده‌ایم. همانطور که مشاهده می‌شود پارامترهای بخش کانولوشنی non trainable هستند و در فرآیند آموزش مدل بروزرسانی نمی‌شوند.

۲.۳.۴ دیتاست جدید

حال قبل از آموزش مدل ابتدا دیتاست جدیدی را که می‌خواهیم مدل را با آن آموزش دهیم بررسی می‌کنیم. این دیتاست مشتمل بر ۳۶۷۰ تصویر از ۵ نوع گل می‌باشد. این ۵ دسته عبارتند از ۱. رز ۲. لاله ۳. آفتاب‌گردان ۴. قاصدک ۵. گل مینا تعدادی از نمونه‌های این دیتاست در تصویر زیر نمایش داده شده است:



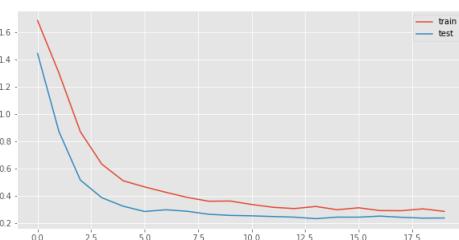
شکل ۲۹: نمونه دیتاهای دیتاست

این دیتاست را به دو بخش آموزشی و تست تقسیم می‌کنیم و از ۲۹۹۰ دیتا برای آموزش مدل و از ۶۸۰ دیتا باقی‌مانده برای تست عملکرد آن بهره می‌گیریم.

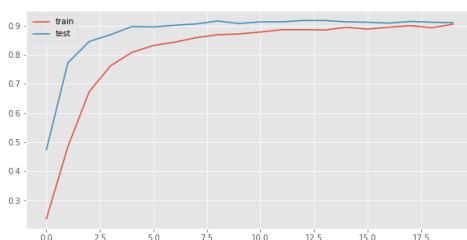
۳.۳.۴ آموزش مدل

اکنون می‌خواهیم مدل جدید تعریف شده را با این دیتاست آموزش دهیم، برای این منظور ازتابع هزینه‌ی cross entropy و روش بروزرسانی adam استفاده شده است. همچنین اندازه‌ی batch برابر با ۲۳ و تعداد epoch ۲۰ در نظر گرفته شده است.

برای آموزش این مدل از روش نرخ یادگیری متغیر استفاده می‌کنیم تا ابتدا با اندکی افزایش نرخ یادگیری، توانایی مدل را برای جستجو در فضای تابع هدف افزایش دهیم تا بتوانیم سریع‌تر به نقطه بهینه برسیم و سپس با کاهش نرخ یادگیری تلاش می‌کنیم تا به نقطه بهینه همگرا شده و دچار نوسان نشویم. دقت شبکه پس از ۲۰ epoch برابر با ۹۱.۱۵ درصد می‌باشد و نمودار دقت و مقدار تابع خطا در شکل‌های زیر نمایش داده شده‌اند.



شکل ۳۱: مقدار تابع خطا



شکل ۳۰: دقت مدل

همانطور که مشاهده می‌شود با استفاده از روش Transfer Learning توانستیم یک مدل عمیق و پیچیده را با دیتاست کوچکی آموزش دهیم و این آموزش بدون فرآیند اتفاق افتاده است و به دقت مناسبی رسیده است.

۴.۴ اشیای قابل تشخیص توسط شبکه

همانطور که ذکر شد، با استفاده از Transfer Learning مدل Inception که در ابتدا روی دیتابست ImageNet آموزش دیده بود، توانستیم طبقه‌بندی طراحی کنیم که ۵ نوع گل را می‌تواند شناسایی کند.

۱. رز
۲. لاله
۳. آفتاب‌گردان
۴. قاصدک
۵. گل مینا

۴.۵ بررسی عملکرد شبکه

حال می‌خواهیم عملکرد این شبکه را با استفاده از تعدادی تصویر بررسی کنیم. برای این منظور از ۳ تصویر که همگی با استفاده از دوربین Canon EOS 700D و در موقعیت‌های متفاوت توسط آقای علیرضا محمدی تهیه شده‌است، استفاده می‌کنیم. پس از انجام پیش‌بینی توسط مدل، ۳ شی با بالاترین احتمال را که توسط شبکه پیش‌بینی شده‌اند، به ترتیب احتمال، نشان می‌دهیم.

برای استفاده از این تصاویر در پیش‌بینی مدل، باید ابتدا پیش‌پردازش‌هایی را بر روی تصاویر اعمال کنیم. نکته اول این است که اندازه‌ی تصاویر باید مطابق با ورودی شبکه، یعنی 299×299 باشد. همچنین مقادیر پیکسل‌ها باید مقداری بین ۰ و ۱ باشد، از این رو با اعمال تبدیل‌های لازم، اندازه تصویر را به گونه‌ای تنظیم می‌کنیم تا در شبکه قابل استفاده باشد و مقادیر پیکسل‌ها از ۰ تا ۲۵۵ به مقداری بین ۰ تا ۱ تصویر می‌کنیم.

۱.۵.۴ تصویر اول



شکل ۳۲: تصویر نمونه اول (baghe خونه :)

پیش‌بینی مدل برای این تصویر به صورت زیر است:

```
[INFO] loading and pre-processing image...
[INFO] classifying image with 'Inception'...
1. roses: 85.91%
2. tulips: 14.07%
3. sunflowers: 0.01%
```

شکل ۳۳: نتیجه پیش‌بینی مدل

۲.۵.۴ تصویر دوم



شکل ۳۴: تصویر نمونه دوم (پارک آب و آتش)

پیش‌بینی مدل برای این تصویر به صورت زیر است:

```
[INFO] loading and pre-processing image...
[INFO] classifying image with 'Inception'...
1. tulips: 99.80%
2. sunflowers: 0.11%
3. dandelion: 0.04%
```

شکل ۳۵: نتیجه پیش‌بینی مدل

۳.۵.۴ تصویر سوم



شکل ۳۶: تصویر نمونه سوم (باغ ارم شیراز)

پیش‌بینی مدل برای این تصویر به صورت زیر است:

```
[INFO] loading and pre-processing image...
[INFO] classifying image with 'Inception'...
1. roses: 76.22%
2. tulips: 23.70%
3. sunflowers: 0.08%
```

شکل ۳۷: نتیجه پیش‌بینی مدل

تست عملکرد شبکه

به منظور تست عملکرد شبکه، همانطور که در صورت پروژه ذکر شده که کد باید قابل اجرا باشد، کد `testNetwork.py` را پیاده‌سازی کرده‌ایم. با نصب نیازمندی‌های این کد که در فایل `requirements.txt` اعلام شده‌اند و اجرای کد، وزن‌های آموزش دیده‌ی شبکه و تصویر نمونه به منظور تست شبکه (همان تصویری که در ضمیمه نیز ارسال شده‌است) به صورت خودکار از روی گوگل درایو دانلود شده و فرآیند بازیابی وزن‌ها و پیش‌بینی برچسب تصویر انجام می‌شود. این کد ابتدا وزن‌های شبکه و تصویر نمونه را دانلود می‌کند، پس در این مرحله حتماً به یک اتصال اینترنت احتیاج دارد. سپس شبکه ساخته شده و وزن‌ها لود می‌شوند. نهایتاً تصویر به شبکه داده شده و پیش‌بینی شبکه نمایش داده می‌شود.