



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیووتر
شبکه های عصبی و یادگیری عمیق

مینی پروژه اول

زهرا موسوی موحد، امیرمحمد رنجبر پازکی	نام و نام خانوادگی
۸۱۰۱۹۹۳۴۰، ۸۱۰۱۹۶۶۲۹	شماره دانشجویی
۱۴۰۰/۲/۲۸	تاریخ ارسال گزارش

فهرست گزارش سوالات

3.....	سوال ۱ – مفاهیم تئوری
7.....	CNN – ۲
17.....	سوال ۳ Data Augmentation – ۳
21.....	سوال ۴ Transfer Learning – ۴

سوال ۱ – مفاهیم تئوری

-۱

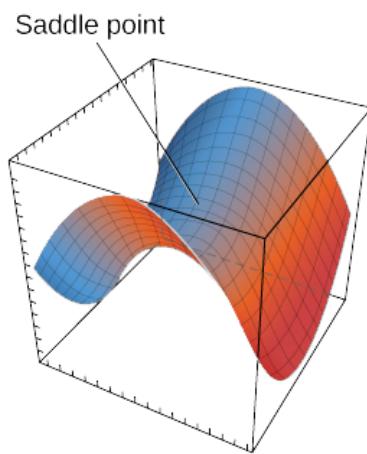
(آ)

با استفاده از این روش رسیدن به نقطه کمینه می‌تواند زمان بر باشد و باید نرخ آموزش به درستی تنظیم شود. در حقیقت، تنظیم نرخ آموزش یکی از سختی‌های دیگر این روش است. اگر نرخ آموزش خیلی کوچک باشد، خیلی طول می‌کشد تا به پاسخ برسیم. اگر خیلی بزرگ باشد، ممکن است از روی جواب بپریم و نتوانیم به سمت آن میل کنیم. اگر در مسئله ما چندین نقطه کمینه محلی وجود داشته باشد، این روش هیچ تضمینی برای پیدا کردن کمینه مطلق و جهانی نمی‌دهد.

یکی دیگر از مشکلاتی که این روش دارد مشکل vanishing gradient است. این مشکل زمانی رخ می‌دهد که لایه‌های شبکه زیاد باشد و با مشتق گیری مقدار مشتق خیلی کوچک شود و عملیات یادگیری رخ ندهد.

یک مشکل دیگر این روش مشکل exploding gradient است. این مشکل زمانی رخ می‌دهد که گرادیان بسیار بزرگ می‌شود و به این ترتیب، فرآیند بهروزرسانی ادامه می‌یابد و هیچ وقت به نقطه کمینه مطلق نمی‌رسد.

یک مشکل دیگر این روش که در ابعاد بالاتر به چشم می‌خورد مشکل saddle point است. نقطه‌ای از یک طرف کمینه و از یک طرف بیشینه است. در نتیجه امکان کاهش از طرفی دارد ولی از بعد دیگر گرادیانش صفر شده و متوقف می‌شود و نقطه‌ای که الگوریتم ما به آن رسیده است، در واقعیت، جواب بهینه نیست.



شکل ۱ - saddle point

(ب)

- روش momentum برای گرادیان کاهشی:
در این روش گرادیان‌های گذشته را نیز برای بهروزرسانی اثر می‌دهیم و به صرف گرادیان همین نقطه عمل نمی‌کنیم.

این روش برای آن طراحی شده است تا فرآیند بهینه سازی را سرعت ببخشد. این روش مشکل تناوب کردن در حوالی نقطه کمینه را حل می کند. به این صورت که با در نظر گرفتن شکل کلی با توجه به تاریخچه به روز رسانی، به روز رسانی را تعدیل می کند تا از روی نقطه کمینه نپرداز و با سرعت بیشتری به آن میل کند.

• **روش Adam**

این روش با استفاده از تغییر adaptive نرخ یادگیری با توجه به میانگین ممان مرتبه دوم گرادیانها که همان واریانس است، سعی بر کنترل بهتر یادگیری دارد چراکه استفاده از یک نرخ یادگیری در تمام طول مسیر ممکن است باعث مشکلات همگرایی شود. همچنین، این روش برای هر پارامتر یک نرخ یادگیری به خصوص دارد تا بتوان یادگیری بهتری برای هر پارامتر داشت. این روش نسبت به گرادیان کاهش عادی بسیار به صرفه تر از نظر زمانی و محاسباتی است. همچنین، پارامترهاییش شهود دارند و نیاز به tuning کمی دارد.

• **روش AdaDelta**

روش AdaGrad یک روش برای adaptive کردن نرخ یادگیری برای پارامترهای است. به این معنی که به روز رسانی های کوچک تری برای پارامترهای مرتبط با ویژگی های پر تکرار و به روز رسانی های بزرگ تری برای پارامترهای مرتبط با ویژگی های کم تکرار در نظر می گیرد. این روش میزان robustness SGD را افزایش می دهد.

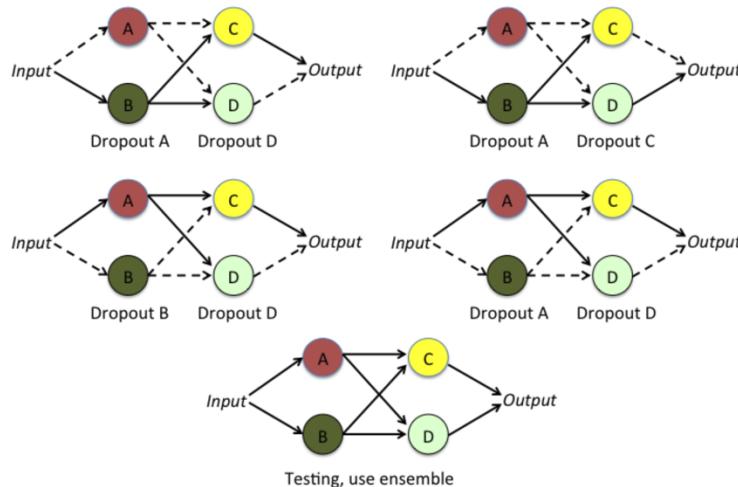
روش AdaDelta یک تعمیم از AdaGrad است که رفتار تهاجمی و اکیدا نزولی نرخ یادگیری در آن را کاهش داده است. این روش به جای در نظر گرفتن تمام گرادیان های گذشته با یک پنجره ای آن را در نظر می گیرد. در حقیقت، در هر بار مقدار گذشته در یک نرخ کاهشی ضرب می شوند و تاثیر داده می شوند. با استفاده از این روش نیازی به قراردادن یک نرخ یادگیری پیش فرض نیست چراکه نرخ یادگیری از معادلات به روز رسانی حذف می شود و گرادیان ها در طول زمان با ضریب کاهشی برای هر کدام در به روز رسانی اثر می گذارند.

-۲

بیش برآش یا همان overfitting زمانی اتفاق می افتد که مدل ما عملکرد بسیار خوبی روی داده های آموزش داشته باشد در حالی که عملکرد آن بر روی داده های آزمون مناسب نیست. در واقع در مسیر آموزش مدل نقطه ای هست که از آن به بعد مدل جز ویژگی های مهم و عمومی داده آموزش شروع به یادگیری ویژگی های خاص آن داده ها و حتی نویز های موجود می کند و این باعث می شود عملکرد آن روی بقیه ست های داده مثل داده آزمون نزولی شود. در حقیقت، در اینجا دچار خطای variance می شویم.

• **Dropout**

یکی از تکنیک هایی است که برای کاهش احتمال overfitting استفاده می شود. به این صورت که در طول آموزش و در هر epoch و یا هر iteration، به صورت رندوم تعدادی از نورون ها و اتصالات آنها را حذف می کنیم و در واقع یک نمونه از شبکه اصلی می گیریم. این موضوع شبیه آن است که ما میانگین عملکرد شبکه های عصبی مختلف را به عنوان نتیجه در نظر بگیریم. همانطور که می دانیم در فرایند آموزش شبکه ما سعی می کند تا در هر مرحله وزن ها را طوری به روز رسانی کند که نتیجه بهبود پیدا کند و مقدار loss کاهش یابد. این به روز رسانی تا حدی وابسته به سایر نورون هاست و می تواند موجب آن شود که مدل هماهنگی هایی که تنها در داده های آموزش وجود دارند را نیز آموزش ببیند. Dropout باعث می شود که هر نورون برای نتیجه خود روی سایر نورون ها حساب کمتری باز کند و در نتیجه بر روی داده های دیده نشده عملکرد بهتری نشان دهد.



شکل – ۲ Drop out

• Norm penalty

یکی از تکنیک‌هایی است که برای کاهش احتمال overfitting استفاده می‌شود. وقتی شبکه‌ها پیچیده‌تر می‌شوند امکان دارد که مقدار وزن‌های شبکه بالا برود. هر قدر وزن‌ها بیشتر باشند باعث می‌شوند تا تغییرات به وجود آمده در ورودی تاثیر بیشتری در خروجی بگذارند و این موضوع می‌تواند باعث overfitting شود. برای جلوگیری از این اتفاق نیاز است تا با دخیل کردن اندازه وزن‌ها در فرایند آموزش سعی کنیم تا آنها را کوچک‌تر نگه‌داریم. همانطور که می‌دانیم در آموزش مدل، تلاش بر حداقل کردن مقدار loss است. پس اگر بتوانیم اندازه وزن‌هارا نیز به تابع loss اضافه کنیم، مدل ما سعی می‌کند تا آنها را حداقل نگه دارد. برای اضافه کردن وزن‌ها دو روش وجود دارد. یکی L_1 که برابر با مجموع قدر مطلق وزن‌هاست و یکی L_2 که برابر با مجموع مربعات کامل وزن‌هاست. همچنین می‌توانیم میزان تاثیر اندازه این پارامترها را نیز با یک هایپر پارامتر مشخص کنیم. در نتیجه استفاده از این متاد باعث می‌شود که شبکه سعی بر کمینه کردن وزن‌ها داشته باشد و احتمال رخ دادن overfitting به علت بزرگی وزن‌ها کم شود.

• Early stopping

همانطور که گفتیم overfitting زمانی رخ می‌دهد که از نقطه‌ای به بعد عملکرد شبکه ما روی داده‌های آموزش بهبود یافته و اما روی داده‌های دیده نشده بهتر نمی‌شود. پس می‌توانیم با تشخیص آن نقطه آموزش را تنها تا همان epoch ادامه دهیم و از overfitting جلوگیری کنیم. برای این کار نیاز است تا بخشی از داده‌های خود را به عنوان داده ارزیابی در نظر بگیریم و در طی فرایند آموزش نقطه‌ای که در آن مقدار loss داده ارزیابی کمتر نمی‌شود و یا دقیق آن بهتر نمی‌شود را پیدا کرده و بعد از آن epoch آموزش را ادامه ندهیم.

-۳

در این شرایط به دنبال ایجاد ویژگی‌های دیگری به جز تخمین تابع در شبکه هستیم. ویژگی‌هایی مانند:

• کاهش ابعاد برای شکاندن correlation های داخلی داده‌ها (Dimensionality reduction)

با اضافه کردن لایه‌هایی شبکه‌های عصبی خود می‌توانند وابستگی داخلی داده‌ها را شناسایی کرده و با بردن آن‌ها به ابعاد پایین‌تر، ضمن سریع‌تر کردن فرآیند آموزش قدرت تعمیم شبکه را بالاتر ببرند.

• فیلتر و scale کردن داده‌ها:

داده‌ها معمولاً تمیز نیستند و حاوی noise هستند. به صورتی که تا حدی کار یادگیری تابع را مختل می‌کنند. با اضافه کردن لایه‌های اضافه‌تر می‌توان این نویزها را حذف کرد تا شبکه بتواند به درستی عمل کند.

-۴

زیرا با زیاد کردن این پارامترها مسئله ما می‌تواند دچار مسئله over parametrization شود. دلیل این موضوع آن است که تعداد پارامترهای مسئله از پیچیدگی آن بسیار بیشتر است و به همین دلیل، overfitting رخ می‌دهد و قدرت تعمیم شبکه برای نمونه‌هایی که ندیده است، کاهش می‌یابد.

اگر پارامترها را از حدی کمتر کنیم، مسئله دچار under parametrization می‌شود. به نوعی انگار پیچیدگی شبکه برای یادگیری کافی نیست و خطای زیادی پیدا می‌کند. این نوع خطأ به عنوان under fitting یا خطای bias شناخته می‌شود و به عنوان فاصله شبکه از هدف موردنظر و عدم توانایی شبکه است.

به همین دلیل، در انتخاب تعداد لایه‌ها و یا پارامترهای هر لایه محدود هستیم.

-۵

مقداردهی اولیه تمامی وزن‌ها به مقدار یکسان باعث می‌شود که نورون‌های عصبی در هر epoch عملکرد مشابهی داشته باشند. چرا که مقدار اولیه آنها یکسان است و مقادیر مشتق محاسبه شده نیز یکسان خواهد شد و در نتیجه وزن‌ها هربار به مقدار مشابهی به روزرسانی می‌شوند. در واقع در این حالت همه نورون‌ها ویژگی‌های یکسانی را آموزش می‌بینند و باعث می‌شود شبکه ما عملکردی که انتظار می‌رود را انجام ندهد. برای جلوگیری از این مشکل وزن‌های اولیه را به صورت رندوم مقداردهی می‌کنیم.

-۶

همانطور که می‌دانیم هنگام آموزش یک شبکه عصبی برای به روز رسانی وزن‌ها از مشتق زنجیره‌ای استفاده می‌کنیم و با پیمایش شبکه از انتهای تا وزن مورد نظرمان مشتقات جزئی را به دست می‌آوریم.

• Vanishing gradient

در صورتی که مشتقات جزئی خیلی کوچک باشند باعث می‌شود که حاصل ضرب آنها به طور تصاعدی کاهش پیدا کرده و از بین برود. این مقادیر کم باعث می‌شود که وزن‌ها به طور موثری به روز نشوند و آموزش شبکه با مشکل مواجه شود. همچنین در بدترین حالت ممکن است حاصل ضرب مشتق‌ها صفر شود و در نتیجه وزن‌ها دیگر به روز نشوند.

• Exploding gradient

در صورتی که مشتقات جزئی خیلی بزرگ باشند باعث می‌شود که حاصل ضرب آنها به طور تصاعدی افزایش پیدا کند و به اصطلاح منفجر شود. مقادیر خیلی زیاد وزن‌ها باعث می‌شود که شبکه ما عملکرد ناپایداری داشته باشد و در صورت ادامه باعث می‌شود که به علت overflow وزن‌ها NaN شوند و دیگر قابل به روزرسانی نباشند.

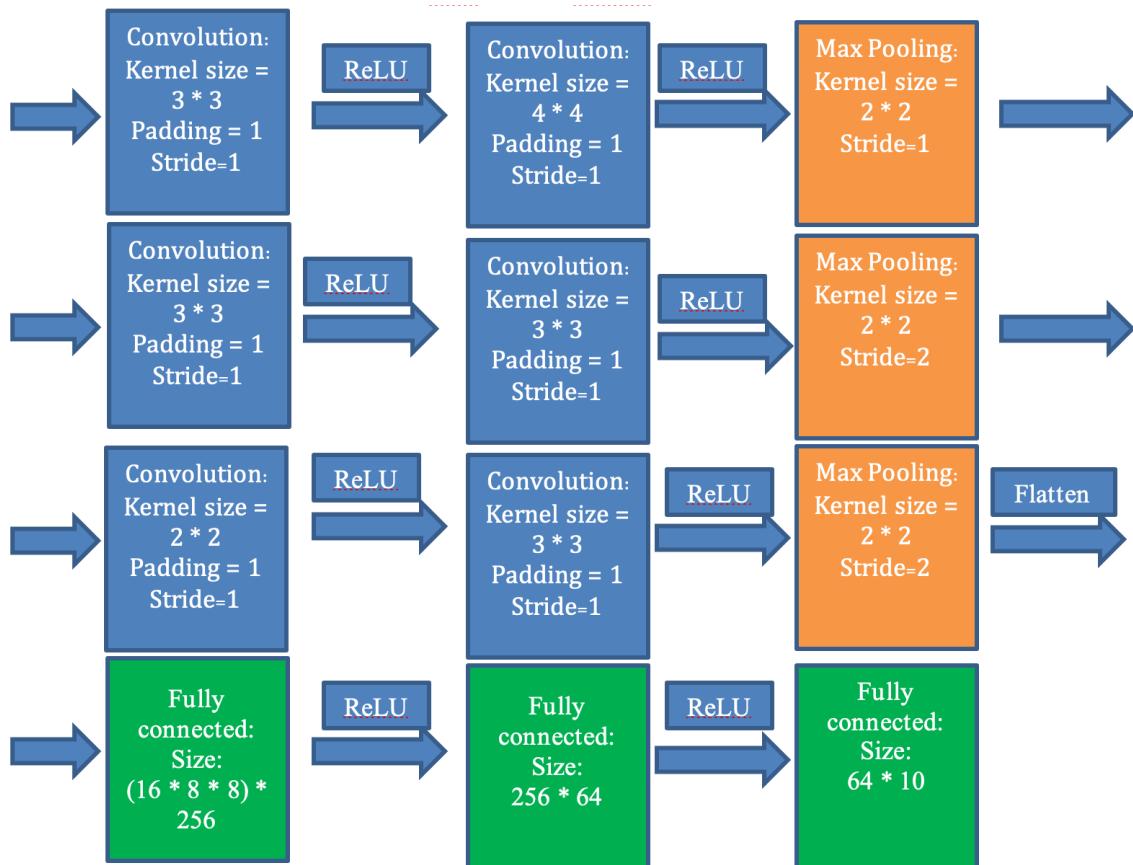
CNN – ۲

کد و نتایج این سوال در فایل Q2_۸۱۰۱۹۶۶۲۹_۸۱۰۱۹۹۳۴۰.ipynb آمده است.

-۱

مشخصات شبکه ابتدایی استفاده شده به صورت زیر است:

شبکه از سه قسمت فیلتر و سه لایه fully connected تشکیل شده است. هر قسمت فیلتر، از دو فیلتر convolutional و یک پول تشکیل شده است. معماری لایه ها به صورت زیر است:



شکل ۳ – شماتیک لایه های شبکه CNN به همراه پارامترهای آنها

در شکل بالا، سه ردیف اول مربوط به قسمت فیلترینگ و ردیف انتهایی مربوط به قسمت classification می شوند. جعبه های آبی رنگ نماینده فیلترهای convolutional، جعبه نارنجی رنگ نشان دهنده pooling و سبزها نشان دهنده لایه های fully connected هستند.

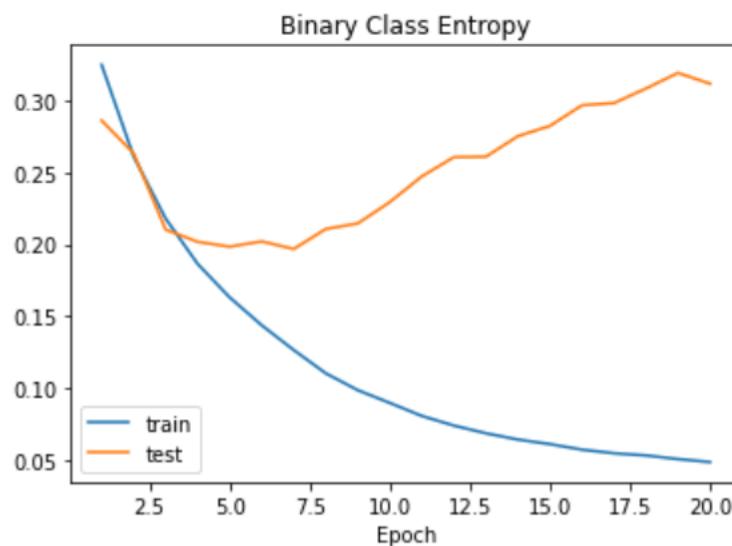
تابع فعالسازی مورد استفاده پس از فیلترهای convolutional و دو لایه اول fully connected تابع ReLU است. لایهی اخر این تابع را ندارد تا برای هر کلاس عددی را پیشنهاد کند.

تابع loss مورد استفاده تابع CrossEntropyLoss است که در Stochastic Gradient Descent قرار می گیرد. روش بهینه سازی مورد استفاده در شبکه پایه روش SGD می باشد.

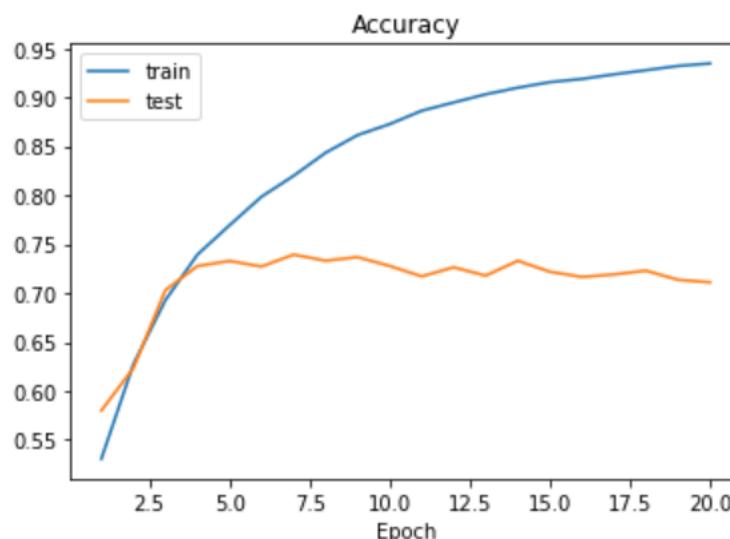
اندازه mini-batch مورد استفاده ۴ بوده است.

-۲

شبکه‌ی موردنظر بر روی داده‌ی آموزش دیده است و بر روی تست نیز در انتهای هر epoch مورد آزمون قرار گرفته است. نمودار دقت و خطا برای داده‌ی آموزش و آزمون در هر epoch به صورت زیر به دست آمده است.



شکل ۴- مقدار LOSS داده‌ی آموزش و آزمون در هر epoch

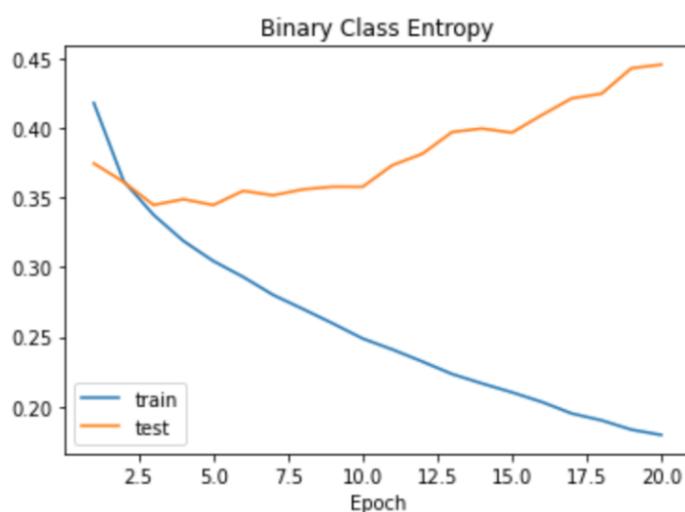


شکل ۵- مقدار دقت داده‌ی آموزش و آزمون در هر epoch

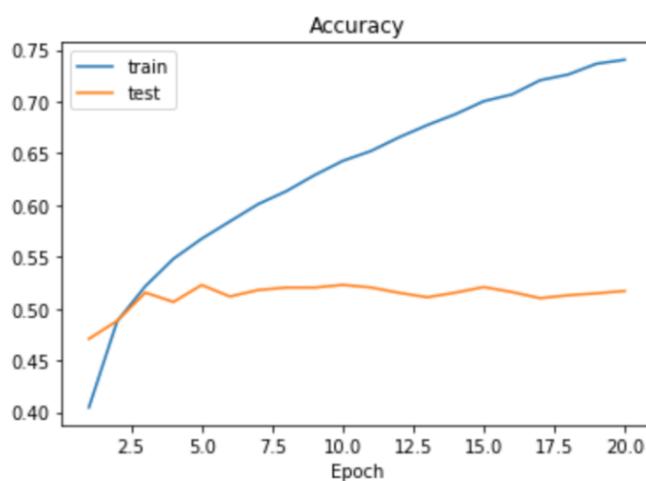
همان‌طور که از نمودار بر می‌آید، بهترین دقت به دست‌آمده برای داده‌ی آموزش و آزمون به ترتیب ۹۳ درصد و ۷۴ درصد است که به ترتیب در epoch شماره ۲۰ و ۷ به دست‌آمده است. داده‌ی آموزش با بیشتر کردن تعداد epoch‌ها می‌تواند به دقت بالاتری برسد اما همانطور که مشاهده می‌کنید بعد از epoch شماره ۷ به نوعی overfitting رخداده است. چراکه دقت داده‌ی آموزش بالا رفته اما دقت داده‌ی آزمون بهبود نداشته است.

-۳

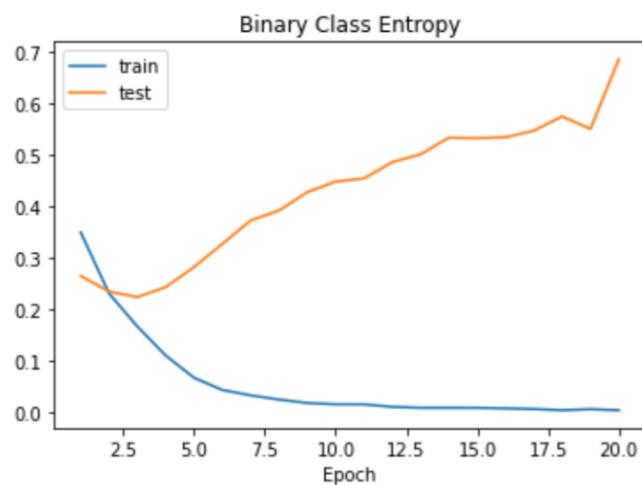
شبکه‌هایی با صفر، یک و دو لایه‌ی مخفی با لایه‌های مشابه شبکه اصلی مورد آموزش و آزمون قرار گرفته‌اند که نمودار دقت و خطای آن‌ها در زیر قابل مشاهده است.



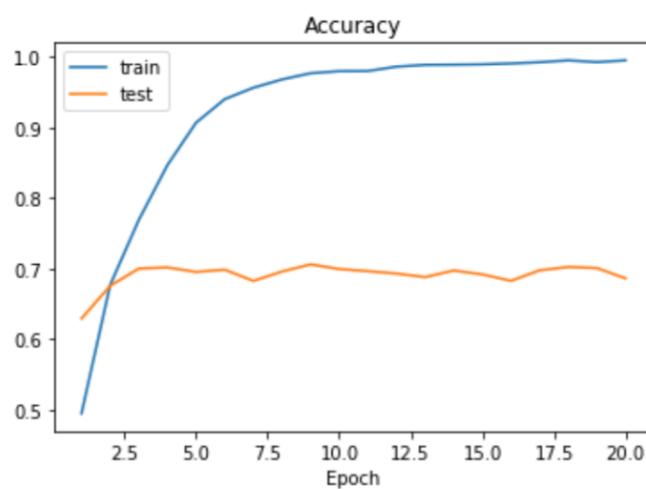
شکل ۶- مقدار داده‌ای آموزش و آزمون در هر epoch برای شبکه با ۰ لایه مخفی



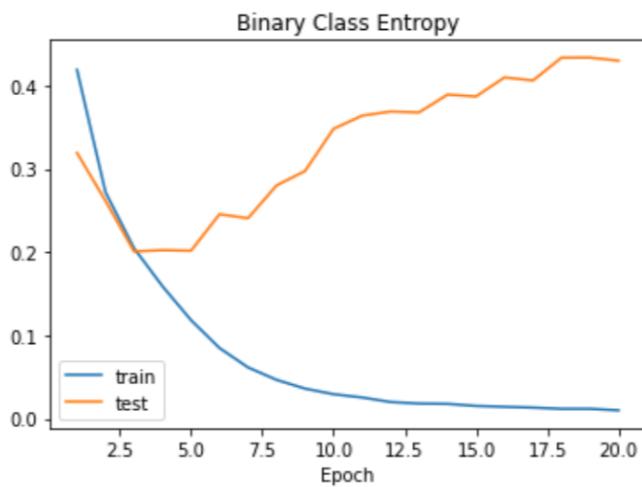
شکل ۷- مقدار دقت داده‌ای آموزش و آزمون در هر epoch برای شبکه با ۱ لایه مخفی



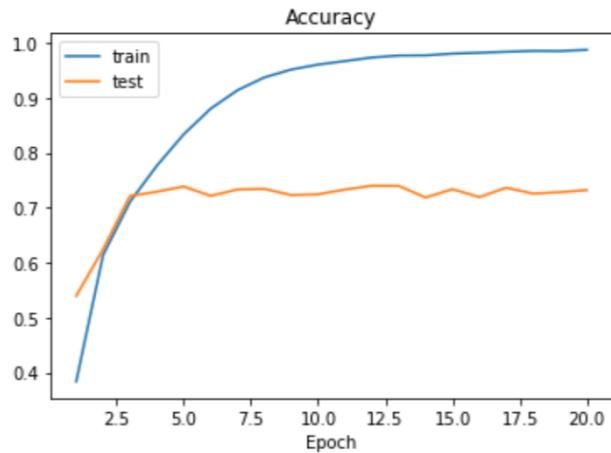
شکل ۸- مقدار دادهای آموزش و آزمون در هر epoch برای شبکه با ۱ لایه مخفی



شکل ۹- مقدار دقیقت دادهای آموزش و آزمون در هر epoch برای شبکه با ۱ لایه مخفی



شکل ۱۰- مقدار دادهای آموزش و آزمون در هر epoch برای شبکه با ۲ لایه مخفی

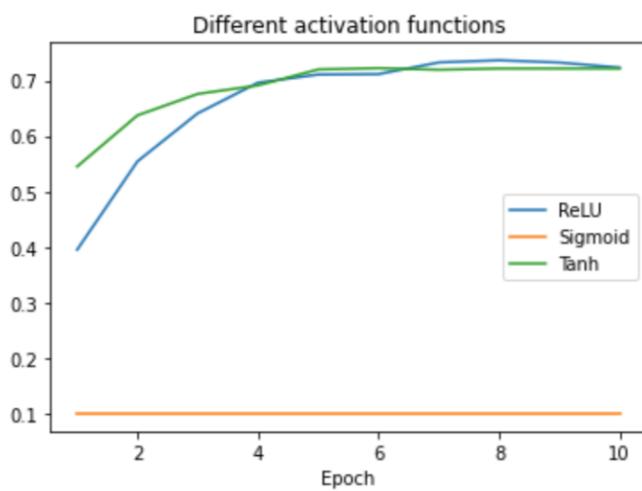


شکل ۱۱- مقدار دقت دادهای آموزش و آزمون در هر epoch برای شبکه با ۲ لایه مخفی

از مقایسه این سه شبکه می‌توان دریافت که شبکه‌ی دو و سه لایه بهترین عملکرد را داشته‌اند. دلیل این موضوع آن است که شبکه‌های صفر و یک لایه توانایی حذف نویز و اعوجاج‌های داده را ندارد و می‌توان گفت در گیر خطا bias هستند. به این معنی که این شبکه بیش از حد نیاز ساده هستند. اما شبکه‌ی دو و سه لایه می‌توانند نویزها را به خوبی حذف کنند و پیچیدگی کافی را دارند.

-۴

سه تابع فعالسازی ReLU، sigmoid و tanh به عنوان توابع فعالسازی مورد آزمون قرار گرفته‌اند. برای هر شبکه ۱۰ epoch آموزش رخ داده است و سپس، بر روی داده‌ی آزمون مورد امتحان قرار گرفته‌اند. دقت هر یک از این شبکه‌ها بر حسب epoch در نمودار زیر قابل مشاهده است.



شکل ۱۲- مقدار دقت به دست آمده با استفاده از توابع فعالساز مختلف در هر epoch

برای این منظور تابع Sigmoid گزینه مناسبی نیست. اما توابع ReLU و tanh در انتهای آموزش عملکرد مشابهی داشته‌اند. تابع ReLU کمی عملکرد بهتری در epoch‌های بالاتر داشته است.

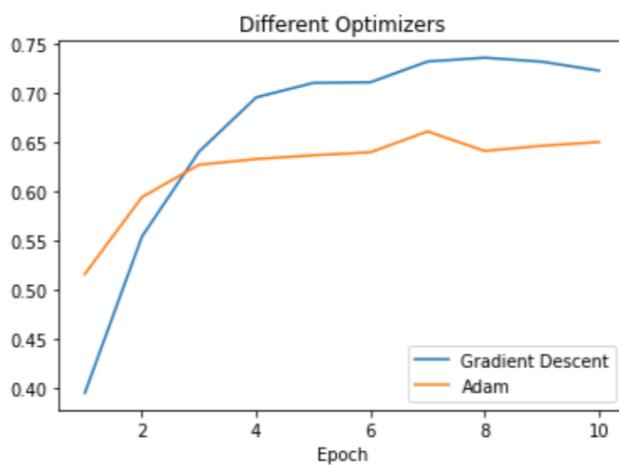
توابع tanh و Sigmoid مشکل vanishing gradient دارند ولی ReLU این مشکل را ندارد. در بین این توابع tanh متقارن و با مرکز صفر است که برای ما کاربرد ندارد. در تابع ReLU تمام مقادیر منفی صفر می‌شوند. دقت و بازدهی ReLU بیشینه است چراکه از طرف مثبت حد ندارد و به همین دلیل، به مشکل vanishing gradient برنمی‌خورد.

در نتیجه انتخاب نهایی ما، ReLU است.

-۵

دو روش بهینه‌سازی gradient descent و adam مورد آزمون قرار گرفته‌اند.

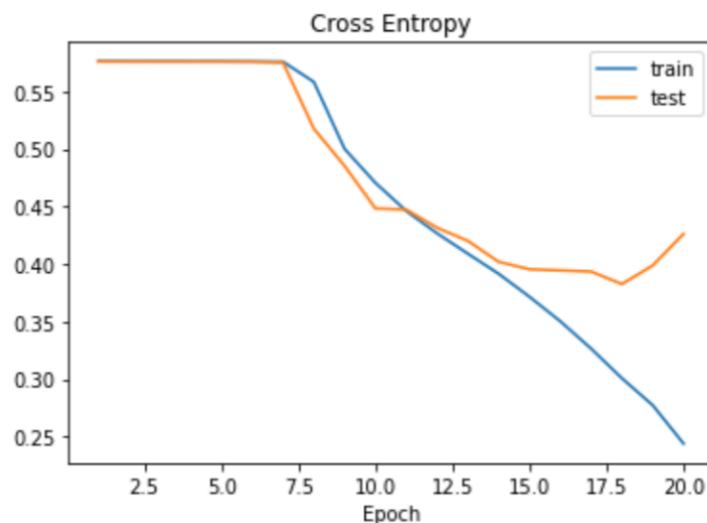
برای هر شبکه ۱۰ epoch آموزش رخ داده است و سپس، بر روی داده‌ی آزمون مورد امتحان قرار گرفته‌اند. دقت هر یک از این شبکه‌ها بر حسب epoch در نمودار زیر قابل مشاهده است.



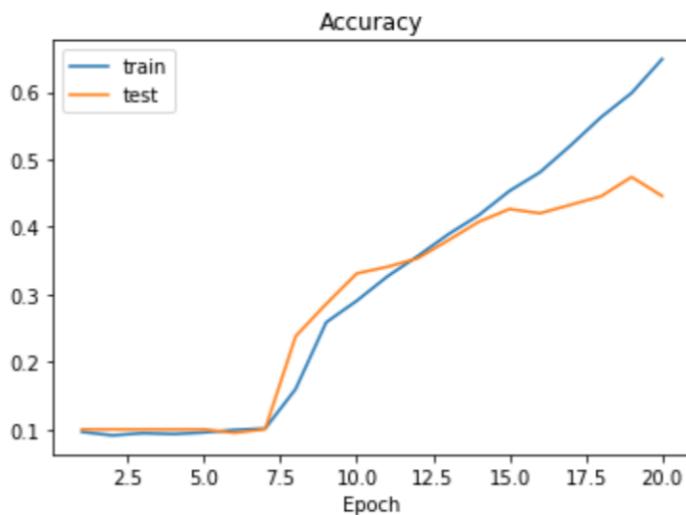
شکل ۱۳- مقدار دقت به دست آمده با استفاده از روش‌های بهینه‌سازی مختلف در هر epoch

همانطور که در نمودار مشاهده می‌کنید، روش Gradient Descent عملکرد بهتری داشته است. دلیل این موضوع آن است که روش گرادیان کاهش با ثابت نگهداشتن نرخ یادگیری امکان گذر از local minimum را داده است و احتمالاً قسمت‌های بیشتری توسط مدل دیده شده است. به همین دلیل، قدرت تعیین این بهینه‌ساز بیشتر بوده است و توانسته به دقت بالاتری برای داده‌ی آزمون بررسد. اما روش Adam احتمالاً به مشکل overfitting نسبی دچار شده است که با کاهش نرخ یادگیری نتوانسته از آن گذر کند.

داده‌های هر کلاس به ۶۰۰ عدد کاهش داده شدند و سپس، آموزش بر روی آن‌ها انجام شد. نمودار خطای دقت در برای داده‌های آموزش و آزمون در شکل زیر بر حسب epoch آمده است.



شکل ۱۴- مقدار Loss داده‌های آموزش و آزمون در هر epoch بعد از کاهش داده

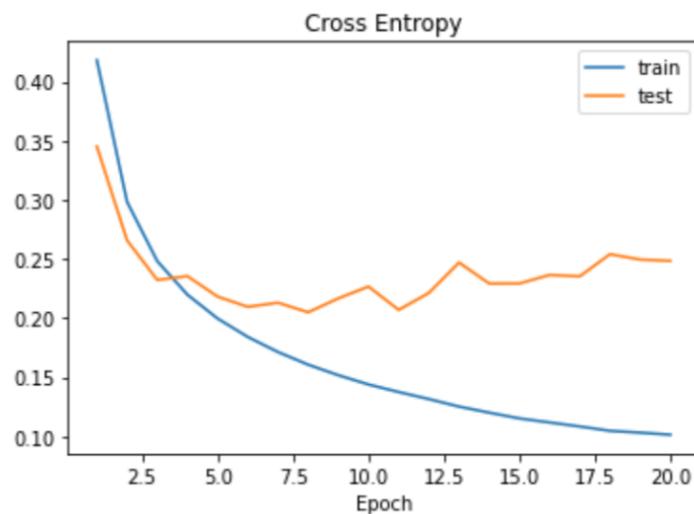


شکل ۱۵- مقدار دقت داده‌های آموزش و آزمون در هر epoch بعد از کاهش داده

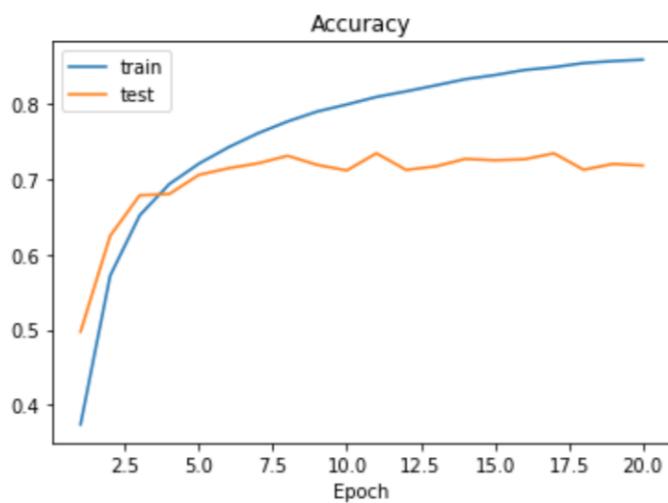
همانطور که مشاهده می‌کنید، دقت افت محسوس داشته است. دلیل آن است که شبکه‌های عصبی با عماری گفته شده پارامترهای زیادی دارد که برای یادگیری آن‌ها به داده‌ی زیاد نیاز دارد. از طرفی هر چقدر تعداد داده‌ها و تنوع آن‌ها در هر کلاس بیشتر باشد، شبکه‌ی ما به تغییرات مقاوم‌تر می‌شود و آن‌ها را بهتر یاد می‌گیرد و قدرت تعمیم بالاتری خواهد داشت.

با کاهش تعداد داده‌های هر کلاس، شبکه نمونه‌ها متنوع کمتری می‌بیند و در نتیجه، قدرت تعمیم آن برای داده‌ها ندیده کاهش می‌یابد. گویی شبکه اطلاعات کمتری برای یادگیری داشته است.

به جای دو عمل کانولوشن در هر لایه یک کانولوشن با سایز پنجره بزرگ‌تر قرار گرفت و سپس، آموزش با شبکه‌ی جدید انجام شد. نمودار خطا و دقت در برای داده‌های آموزش و آزمون در شکل زیر بر حسب epoch آمده است.



شکل ۱۶- مقدار Loss داده‌های آموزش و آزمون در هر epoch با کرنل بزرگتر



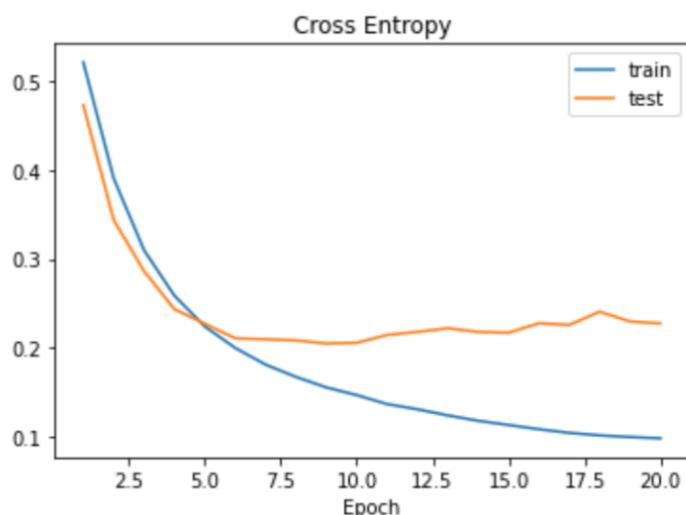
شکل ۱۷- مقدار دقت داده‌های آموزش و آزمون در هر epoch با کرنل بزرگتر

همانطور که در نمودار بالا مشاهده می‌کنید، دقت کمی افت کرده است و عملکرد شبکه بدتر شده است. دلیل این موضوع آن است که با بزرگ‌تر شدن اندازه پنجره، تعداد پارامترها به صورت quadratic زیاد می‌شود. به همین دلیل، آموزش شبکه با سختی رو برو می‌شود. چرا که به مشکل curse of dimensionality می‌شود.

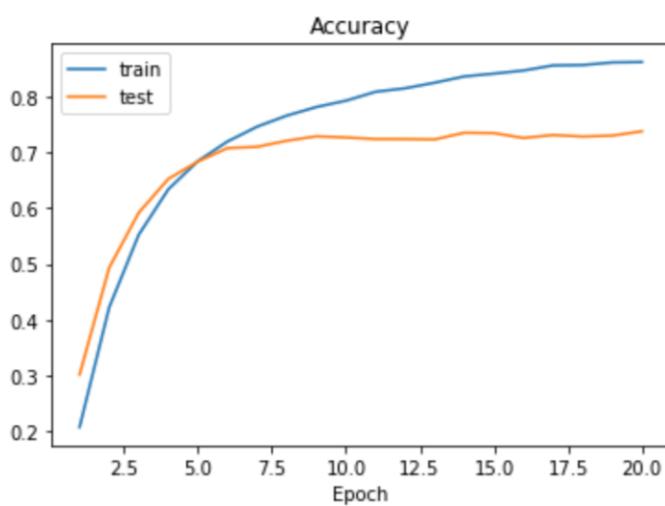
بر می‌خوردیم و نیاز به داده‌های بسیار بیشتری برای آموزش خواهیم داشت. در نتیجه، با این تعداد داده آموزش به صورت مطلوبی رخ نمی‌دهد و دقت افت می‌کند. جایگزینی این کرنل بزرگتر با چند لایه با کرنل‌های کوچک‌تر باعث می‌شود به این مشکل بر نخوریم و ویژگی‌های موردنظر با همین تعداد داده به خوبی استخراج شود و به دقت بالاتری برسیم.

-۸

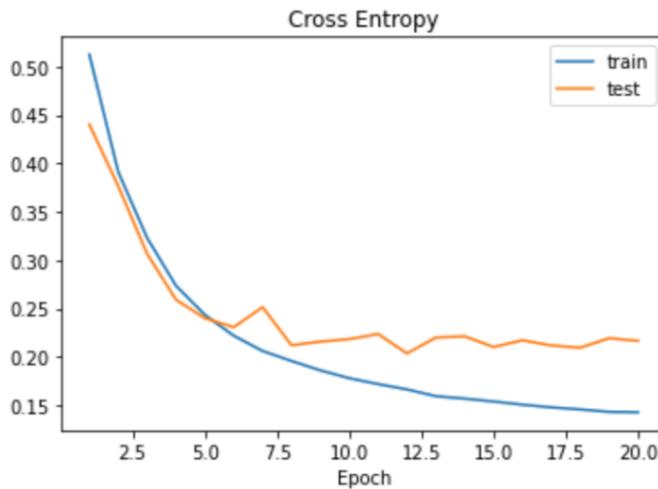
در این سوال پس از هر تابع فعالساز **dropout** قرارداده شده است. یک بار **10 dropout** درصدی و یک بار **20 dropout** درصدی مورد آزمون قرار گرفته‌اند تا مقدار بهینه آن به دست آید. در شکل زیر نمودار خطای دقت برای داده‌ها آموزش و آزمون بر حسب epoch برای هر یک از دو شبکه نمایش داده شده است.



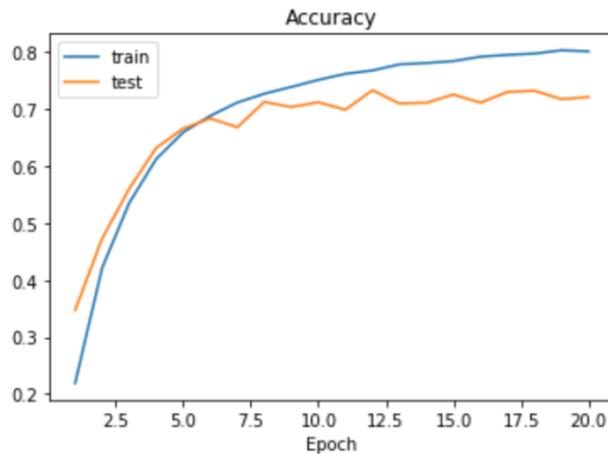
شکل ۱۸- مقدار Loss داده‌های آموزش و آزمون در هر epoch پس از افزودن ۱۰ درصد



شکل ۱۹- مقدار دقت داده‌های آموزش و آزمون در هر epoch پس از افزودن ۱۰ dropout درصد



شکل ۲۰- مقدار Loss داده‌های آموزش و آزمون در هر epoch پس از افزودن ۲۰ dropout ۲۰ درصد



شکل ۲۱- مقدار دقت داده‌های آموزش و آزمون در هر epoch پس از افزودن ۲۰ dropout ۲۰ درصد

همانطور که در نمودارها مشاهده می‌کنید، ۲۰ dropout درصدی عملکرد بهتری از ۱۰ درصدی داشته است. اما دقت آن بر خلاف انتظار چندان متفاوت با دقت شبکه‌ی پایه نشده است.

انتظار بهبود در عملکرد شبکه داشتیم. زیرا با قرار dropout بخشی از نورون‌های لایه غیرفعال می‌شوند و با این کار قدرت تعمیم شبکه بالاتر می‌رود. چراکه با این کار پدیده‌ی overfitting رخ نمی‌دهد.

دلیل بهبود پیدا نکردن در مثال ما ممکن است این باشد که شبکه مناسب بوده است و دچار overfitting نشده است و به همین دلیل، حالت پایه و این حالت تفاوت چندانی با یکدیگر ندارند. البته لازم به ذکر است این پدیده در اینجا رخ نداده است. در نمودارهای قبلی، از جایی به بعد دقت train بالاتر می‌رود در صورتی که دقت آزمون کمتر می‌شود. اما در اینجا دو نمودار به نرمی با هم بالا رفته‌اند.

سوال ۳ – Data Augmentation

کد و نتایج این سوال در فایل **Q۳_۸۱۰۱۹۹۳۴۰.ipynb** آمده است.

-۱

در یادگیری به روش supervised دقت به دست آمده از مدل، وابستگی زیادی با تعداد داده هایی که برای آموزش به مدل داده ایم دارد. در شرایطی که به دلایل مختلفی همچون کم بودن داده های موجود و عدم امکان به دست آوردن داده جدید، تعداد داده های موجود کمتر از نیاز و پیچیدگی مدل ماست، و یا در شرایطی که کلاس های داده تناسب تعداد ندارند و ... می توانیم به ساخت داده های جدید با استفاده از داده های موجود اقدام کنیم که به این کار Data Augmentation می گوییم. وابسته به نوع داده تکنیک های استفاده شده برای افزودن داده متفاوت است. مثلا در حالتی که مانند این سوال ورودی شبکه ما عکس است می توانیم از تغییراتی چون چرخاندن عکس در زوایای مختلف، انتقال آن در راستای عمودی و یا افقی، زوم کردن و یا کراپ کردن عکس، تغییر میزان روشنایی، تغییر میزان وضوح، افزودن نویز به عکس ، ترکیب این موارد و ... استفاده کنیم.

برای داده متنی روش های متفاوتی وجود دارد. یکی از این روش ها، روش Thesaurus است که می گوید روی جمله های موجود کلمه ها را با هم معنی هایشان جایگزین کنیم. روش دیگر روش word embeddings است که با استفاده از knn و پیدا کردن مشابه ها، کلمات مشابه را جایگزین می کند. در این مورد می توانیم با توجه به context متن این کار را انجام دهیم که به آن contextualize word embedding می گویند. در این رویکرد با توجه به context جایگزینی تغییر می کند و همواره ثابت نیست. یک رویکرد دیگر text generation است که می توان با استفاده از مدل های این کار را انجام داد.

برای داده های عددی می توان از تکنیک های oversampling SMOTE استفاده کرد و یا با استفاده از یک شبکه GAN داده های اضافی نزدیک به واقعیت تولید کرد.

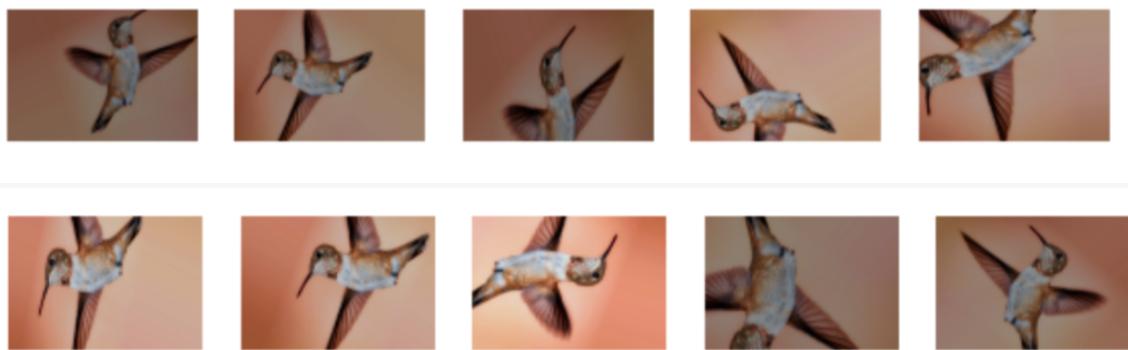
به طور کلی همانطور که گفتیم از این روش معمولا به دلیل کمبود داده موجود برای آموزش مدل استفاده می کنیم و چون آزمون مدل در محیط واقعی انجام می شود بهتر است که داده های تست تا جایی که امکان دارد به نمونه واقعی نزدیک باشند. به همین دلایل استفاده از این روش برای داده های آزمون مرسوم نیست. اما در شرایطی که مثلا داده های تست ما پوشش کافی از حالات مختلف را ندارند و به نمونه نزدیک تر به واقعیت دسترسی نداریم، وابسته به مورد می توان از این تکنیک برای افزودن داده های تست نیز استفاده نمود. در این موارد به دنبال سنجش مقاومت مدل نسبت به تغییرات هستیم.

-۲

برای این کار از keras ImageDataGenerator کتابخانه استفاده شده است. این ابزار این اجازه را می دهد تا تعدادی تبدیل برای آن تعریف شود. در اینجا تبدیل های چرخش، شیفت عمودی و افقی، زوم کردن، فلیپ عمودی و افقی، میزان روشنایی و کج کردن برای آن تعریف شده است. این ابزار روی یک عکس هر کدام از این تبدیل ها را به ترتیب و به مقداری تصادفی اعمال می کند. عکس اصلی و نمونه های تولید شده در زیر قابل مشاهده است.



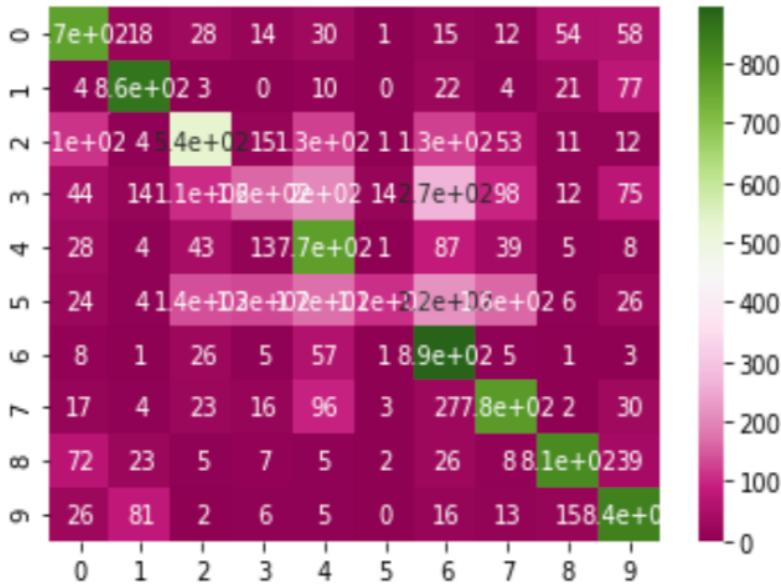
شکل ۲۲ – تصویر اصلی



شکل ۲۳ – تصاویر تولید شده در اثر Data augmentation

-۳

همانطور که در ماتریس آشفتگی و همچنین در مقدار دقیق پیش‌بینی مدل برای کلاس‌های مختلف می‌بینیم، کم بودن تعداد داده‌های دو کلاس گربه و سگ (۳ و ۵) به نسبت سایر کلاس‌ها باعث شده که مدل نتواند شناخت درستی از این دو به دست آورد و تعداد داده‌هایی از این دو کلاس که به درستی تشخیص داده شده‌اند به طور واضحی نسبت به سایر کلاس‌ها پایین است.



شکل ۲۴ – ماتریس آشفتگی داده‌ها تست پس از کاهش داده

```

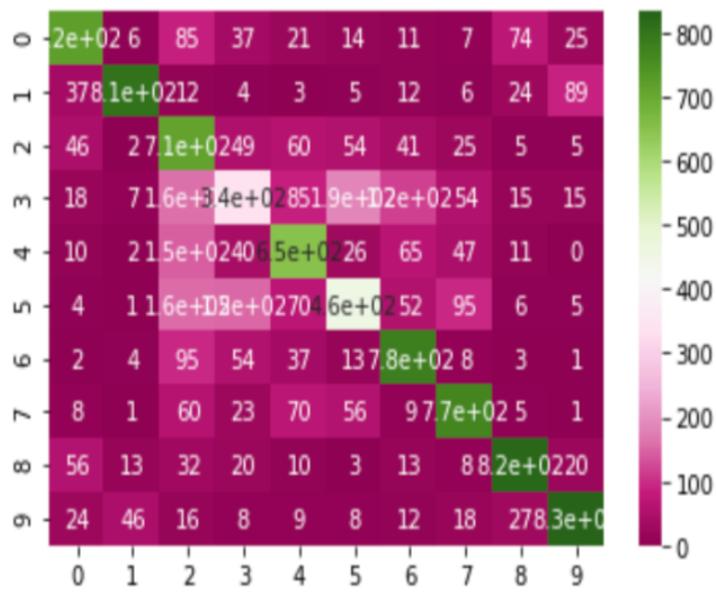
Class 0 accuracy: 0.77%
Class 1 accuracy: 0.86%
Class 2 accuracy: 0.54%
Class 3 accuracy: 0.16%
Class 4 accuracy: 0.77%
Class 5 accuracy: 0.11%
Class 6 accuracy: 0.89%
Class 7 accuracy: 0.78%
Class 8 accuracy: 0.81%
Class 9 accuracy: 0.84%

```

دقت کلاس‌های مختلف به صورت بالا به دست آمده است.

-۴

همانطور که در ماتریس آشفتگی و همچنین در مقدار دقث پیش‌بینی مدل برای کلاس‌های مختلف می‌بینیم، پس از افزودن داده به دو کلاس گربه و سگ تعداد داده‌هایی از این دو کلاس که به درستی تشخیص داده شده اند افزایش پیدا کرده است. دقث تشخیص برای کلاس گربه از ۱۶ به ۳۴ و برای کلاس سگ از ۱۱ به ۴۶ افزایش پیدا کرده است که مقدار قابل توجهی است. پس از افزودن داده‌ها دقث کلاس ۰ و ۴ کاهش چشم‌گیری داشته است که به نظر می‌آید دلیل آن از بین رفتن ویژگی‌های منحصر به فرد این دو کلاس با اضافه کردن عکس‌های جدید است. مورد دیگری که از مقایسه نتایج به دست می‌آید آن است که تعداد عکس‌هایی از کلاس ۳ که به اشتباه کلاس ۵ تشخیص داده شده اند و بالعکس افزایش داشته است. این مورد می‌تواند به دلیل استفاده از تبدیلات مشابه برای ساخت داده‌های جدید باشد.



شکل ۲۵ – ماتریس آشفتگی داده تست پس از Data augmentation

```

Class 0 accuracy: 0.72%
Class 1 accuracy: 0.81%
Class 2 accuracy: 0.71%
Class 3 accuracy: 0.34%
Class 4 accuracy: 0.65%
Class 5 accuracy: 0.46%
Class 6 accuracy: 0.78%
Class 7 accuracy: 0.77%
Class 8 accuracy: 0.82%
Class 9 accuracy: 0.83%

```

دقت کلاس‌های مختلف به صورت بالا به دست آمده است.

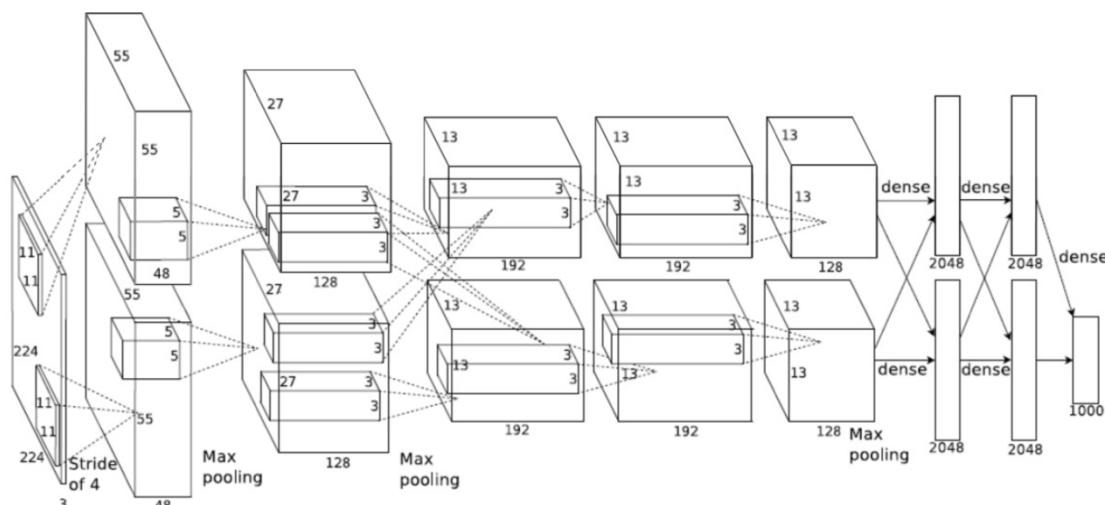
سوال 4 – Transfer Learning

کد و نتایج این سوال در فایل **Q4_۸۱۰۱۹۶۶۲۹_۸۱۰۱۹۹۳۴۰.ipynb** آمده است.

مجموع یکان شماره دانشجویی دو عضو گروه برابر $9+0 = 9$ است. به همین دلیل، شبکه‌ی کانولوشنی مورد مطالعه **AlexNet** است.

-۱

این شبکه دارای ۸ لایه (پنج لایه convolutional و سه لایه fully connected) است و در دسته‌ی شبکه‌های کم عمق (shallow) قرار می‌گیرد.



شکل ۲۶ - شماتیک معماری AlexNet

شکل بالا نمایانگر شبکه‌ی AlexNet است. در لایه اول یک فیلتر conv و یک max pooling رخ می‌دهد. لایه دوم مشابه لایه اول است البته با سایز متفاوت. لایه سوم و چهارم و پنجم فقط یک فیلتر conv هستند که در انتهای عملیات max pooling روی آن‌ها رخ می‌دهد. سپس، سه لایه fully connected قرار گرفته‌اند. نورون‌های لایه آخر نیز softmax Activation function است که احتمال هر کلاس را محاسبه می‌کند.

این شبکه برنده مسابقه ImageNet در سال ۲۰۱۲ بوده است. هدف این مسابقه دسته‌بندی تصویر در مقیاس وسیع می‌باشد. شبکه‌ها در این مسابقه باید بتوانند تصاویری با ۱۰۰۰ کلاس متفاوت را از هم تمیز دهند. این شبکه بهترین معیار خطرا برای ۵ کلاس برتر داشت که مقدار ۱۵.۴ درصد بود.

مزیت این شبکه بهتر کردن دقت به میزان ۳ درصد نسبت به سال گذشته بود. در این شبکه برای اولین بار از تابع ReLU و Dropout استفاده شد. استفاده از ReLU باعث شد زمان آموزش کاهش یابد. این شبکه اجازه استفاده از دو GPU را داد تا فرآیند آموزش سریع‌تر شود.

این شبکه از pooling با هم پوشانی استفاده کرده است که سخت‌تر overfit می‌شود و دقت را نیز ۰.۵ درصد بهبود بخشید.

سایز تصویر ورودی این شبکه ۲۲۴ در سه ۲۲۴ در سه رنگ است. تصویر اولیه نیاز به دو پیش پردازش دارد. اول normalize کردن عکس‌ها تا عکس‌ها هم مقیاس و هم توزیع شود تا بتوان به دقت بهتری برای مدل رسید و از overfitting جلوگیری کرد. همچنین، زمان یادگیری را نیز کاهش داد. دومین پیش پردازش مربوط به تبدیل سایز تصویر به ۲۲۴ در ۲۲۴ است تا بتوان آن را به عنوان ورودی به شبکه‌ی AlexNet داد.

سایز خروجی این شبکه ۱ (۱۰۰۰ عدد) است. به این معنا که عدد ۱ ام نشان‌دهنده احتمال تعلق عکس به کلاس ۱ ام است. این موضوع به دلیل استفاده از softmax است.

-۲

استفاده دوباره از یک مدل از قبیل آموزش دیده، برای حل یک مسئله جدید است. در واقع از دانش به دست آمده در مدل قبلی برای مسئله‌ای جدید اما مرتبط با مدل قبل استفاده می‌شود. از مزایای استفاده آن می‌توان به صرفه‌جویی در زمان آموزش، عملکرد بهتر شبکه و همینطور عدم نیاز به داده‌های زیاد اشاره کرد. در مواقعي که مدل ما نیاز به داده‌های زیاد برچسب‌خورده برای آموزش دارد که در دسترس ما نیست، در مواقعي که آموزش شبکه زمان زیادی می‌برد که برای ما امکان‌پذیر نیست، وقتی که شبکه‌ای وجود دارد که برای کار مشابه با مدل ما از قبیل روی داده‌های زیادی آموزش دیده و یا زمانی که مدل دیگری با ورودی‌های یکسان با مسئله ما از قبیل وجود دارد استفاده از Transfer Learning می‌تواند کمک کننده باشد.

-۳

برای پیاده‌سازی این شبکه با استفاده از torch کتابخانه Transfer Learning از hub pretrained شده روی مجموعه داده‌ی ImageNet از آن بارگیری شده است تا از آن استفاده شود.

-۴

این شبکه برای شرکت در مسابقه ImageNet ارائه شد و به همین دلیل، توانایی دسته‌بندی ۱۰۰۰ شی را که در این مجموعه داده بود، دارد. لیست این اشیا در بخش classes that can detect کد ضمیمه شده به طور کامل آمده است. بخشی از کلاس‌ها به صورت زیر است.

```

classes = {0: 'tench, Tinca tinca',
1: 'goldfish, Carassius auratus',
2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
3: 'tiger shark, Galeocerdo cuvieri',
4: 'hammerhead, hammerhead shark',
5: 'electric ray, crampfish, numbfish, torpedo',
6: 'stingray',
7: 'cock',
8: 'hen',
9: 'ostrich, Struthio camelus',
10: 'brambling, Fringilla montifringilla',
11: 'goldfinch, Carduelis carduelis',
12: 'house finch, linnet, Carpodacus mexicanus',
13: 'junco, snowbird',
14: 'indigo bunting, indigo finch, indigo bird, Passerina cyanea',
15: 'robin, American robin, Turdus migratorius',
16: 'bulbul',
17: 'jay',
18: 'magpie',
19: 'chickadee',
20: 'water ouzel, dipper',
21: 'kite',
22: 'bald eagle, American eagle, Haliaeetus leucocephalus',
23: 'vulture',
24: 'great grey owl, great gray owl, Strix nebulosa',
25: 'European fire salamander, Salamandra salamandra',
26: 'common newt, Triturus vulgaris',
27: 'eft',
28: 'spotted salamander, Ambystoma maculatum',
29: 'axolotl, mud puppy, Ambystoma mexicanum',
30: 'bullfrog, Rana catesbeiana',
31: 'tree frog, tree-frog',
32: 'tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui',
33: 'loggerhead, loggerhead turtle, Caretta caretta',
34: 'leatherback turtle, leatherback, leathery turtle, Dermochelys coriacea',
35: 'mud turtle',

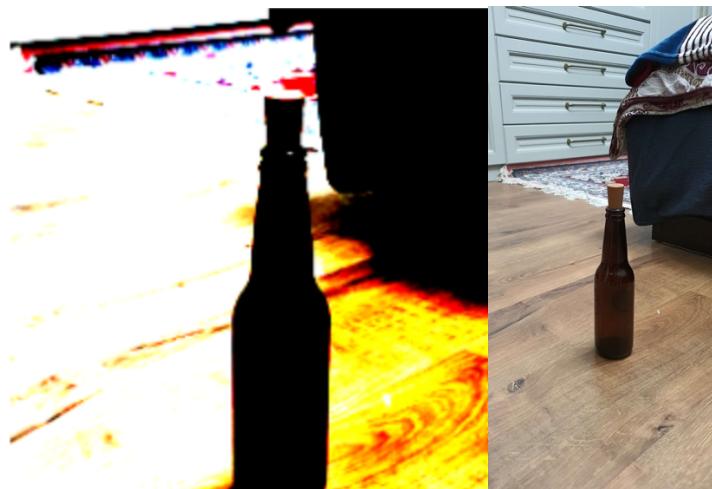
```

شکل ۲۷- برخی اشیای قابل تشخیص برای شبکه AlexNet

-۵

یک عکس رنگی از یک بطری دلستر گرفته شد. سپس، بر روی این عکس پیش‌پردازش‌های موردنیاز صورت گرفت. برای پیش‌پردازش ابتدا سایز عکس به ۲۵۶ در ۲۵۶ کاهاش یافت چراکه عکس گرفته شده بسیار بزرگ بود. سپس، ۲۲۴ در ۲۲۴ پیکسل وسط عکس crop شد تا این عکس مناسب ورودی دادن به شبکه‌ی AlexNet شود. این عکس به Tensor تبدیل شد تا بتوان از آنataloader ساخت و در انتهای عکس normalize شد تا بتوان به دقت بهتری برای مدل رسید و از overfitting جلوگیری کرد.

پس از پیش‌پردازش، ورودی موردنظر به شبکه داده شد و سه شی با بیشترین احتمال به همراه میزان احتمالشان در خروجی چاپ شدند. تصاویر و خروجی متناظر آن‌ها در زیر آمده است.



Top 3 predicted:
Detected object: parking meter
Probability: 12.13%

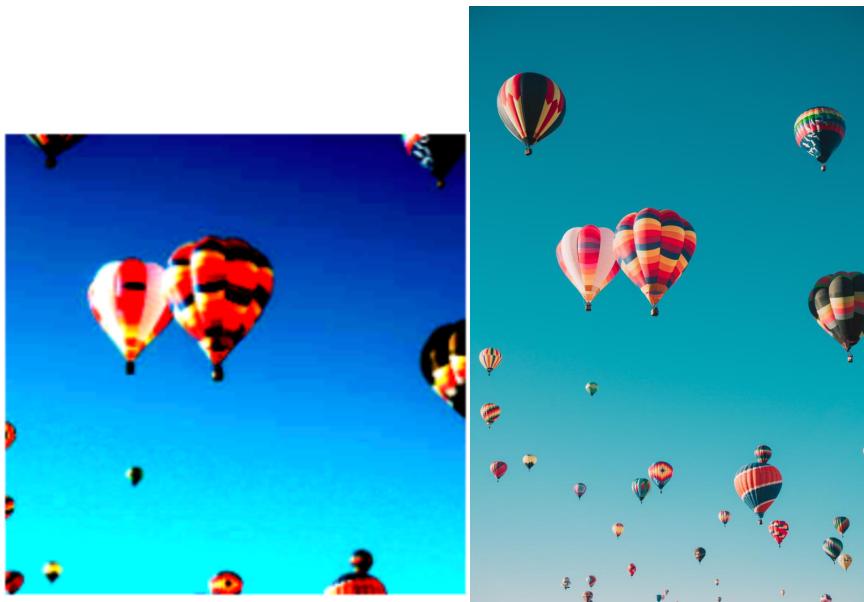
Detected object: beer bottle
Probability: 11.69%

Detected object: pop bottle, soda bottle
Probability: 11.60%

شکل ۲۸ – تصویر اصلی، تصویر پیش‌پردازش شده و خروجی شبکه اول

تصویر ورودی یک بطری دلستر است. همانطور که مشخص است، خروجی دوم و سوم به خوبی این موضوع را تشخیص داده‌اند.

یک تصویر دیگر نیز از اینترنت به شبکه داده شد. شبکه در این تصویر نیز عملکردی خوبی از خود نشان داده است.



Top 3 predicted:

Detected object: balloon

Probability: 15.51%

Detected object: jellyfish

Probability: 14.14%

Detected object: goldfish, Carassius auratus

Probability: 12.57%

شکل ۲۹ – تصویر اصلی، تصویر پیش‌پردازش شده و خروجی شبکه دوم

تصویر ورودی بالن است. همانطور که مشخص است، خروجی اول به خوبی این موضوع را تشخیص داده است. موضوع جالب توجه خروجی دوم است که احتمال عروس دریایی نیز به این تصویر داده است که به دلیل شباهت شکل کلی است.