به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه سری 1

نام و نام خانوادگی	مهسا مسعود - امید واهب
شماره دانشجویی	۸۱۰۱۹۶۶۳۵ - ۸۱۰۱۹۶۵۸۲
تاریخ ارسال گزارش	۱۴۰۰/۰۲/۲۴

فهرست گزارش سوالات

سوال ۱ – مفاهیم تئوری ۳

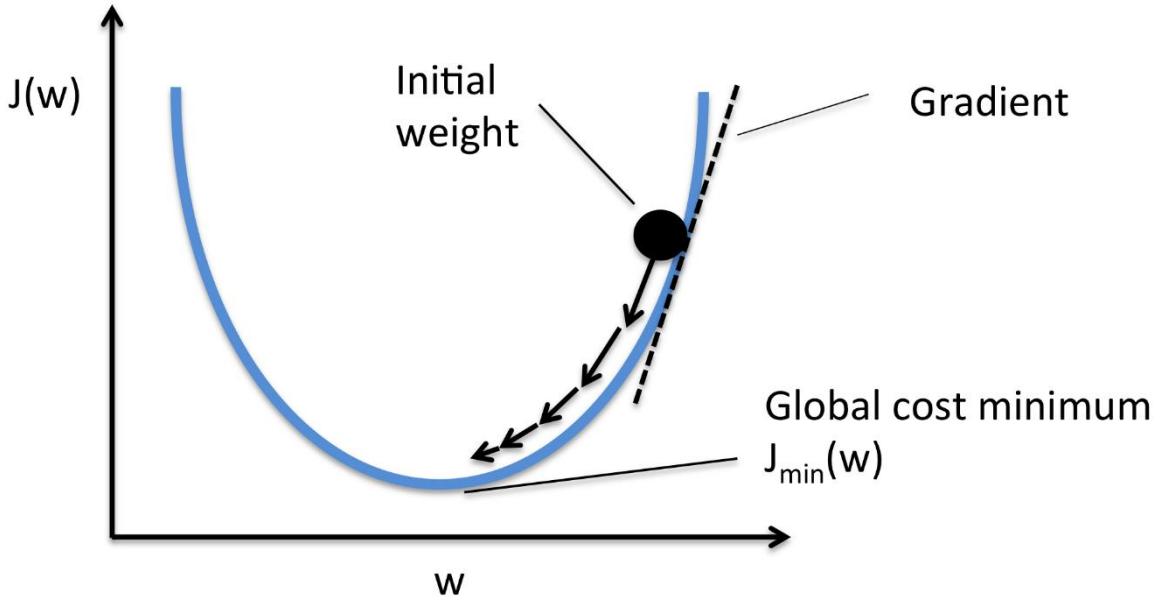
سوال ۲ – CNN ۱۰

سوال ۳ – Data Augmentation ۲۸

سوال ۴ – Transfer Learning ۴۲

سوال ۱ – مفاهیم تئوری

۱. در این سوال به بررسی مشکلات گرادیان کاهشی یا Gradient Descent می پردازیم سپس روش ها و متد هایی را برای حل این مشکلات ارایه می کنیم. همانطور که قبلا حین تدریس بحث شده است، گرادیان کاهشی روشی است برای یافتن نقطه بهینه که در شبکه های عصبی بسیار استفاده می شود و اساس کار آن حرکت در راستای عکس مشتق تابع است تا به نقطه ی بهینه کلی یا local برسیم.



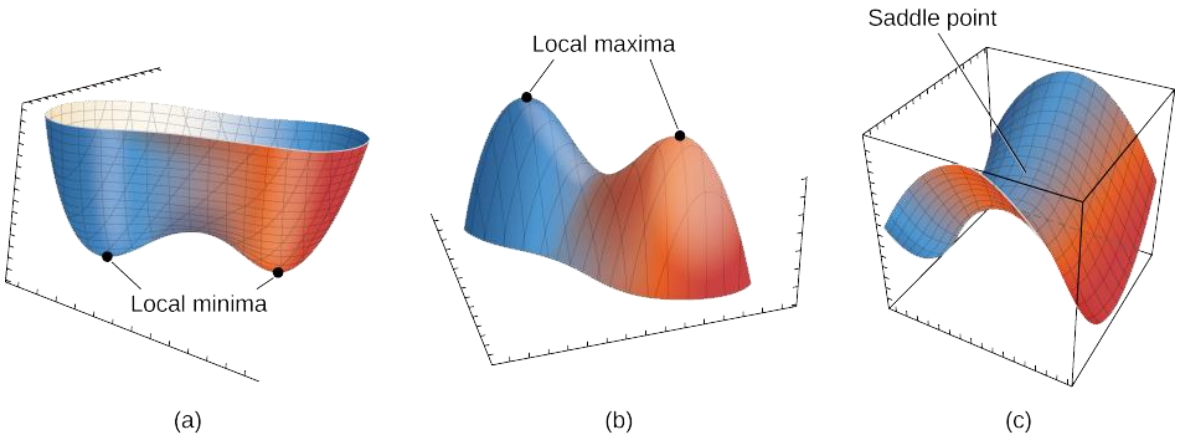
شکل ۱-۱ مثالی از عملکرد Gradient Descent

آ) اولین مشکل استفاده از این روش Vanishing Gradient و Exploding Gradient هستند. Vanishing Gradient حالتی است که در بعضی شبکه های عصبی، مقدار گرادیان بسیار کوچک می شود طوری که دیگر وزن ها آپدیت نمی شوند و مقدار تغییرشان به حدی کوچک است که می توان گفت یادگیری انجام نمی شود و اگر زمانی که این اتفاق می افتد در نقطه ی بهینه نباشیم، تعداد زیادی ایپاک برای خروج از این حالت نیاز می شود و حتی ممکن است که این کار عملی نباشد. برای روشن تر شدن موضوع شکل ۲ را مشاهده کنید. فرض کنید این نمودار تابعی است که قصد بهینه سازی آن را در محور Y داریم. می توان دید که در قسمت آبی شیب یا مشتق تابع بسیار کوچک است و اگر شبکه ما در همچین نقطه ای بیافتد، خروج از آن بسیار سخت خواهد بود.

شکل ۱-۲ نمودار تابعی که قصد بهینه سازی اش را داریم

مشکل دیگر Exploding Gradient است که برعکس Vanishing Gradient در این حالت مقدار گرادیان خیلی بزرگ است و هنگام آپدیت کردن وزن ها تغییرات بزرگی می کنند و ممکن است این تغییرات بزرگ منجر به واگرایی شوند.

یکی دیگر از مشکلات Gradient Descent مشکل Saddle Point ها یا MiniMax Point ها است زیرا این نقاط در بعضی ابعاد مینیمم و در بعضی ابعاد ماکسیمم و ممکن است مقدار مشتق در آن نقطه منجر به دور شدن از نقطه ی همگرایی شود و به جواب بهینه نرسیم یا دیرتر برسیم. در شکل ۳ انواع مختلف نقاط بحرانی که گرادیان صفر دارند را مشاهده می کنیم:



شکل ۱-۳ انواع مختلف نقاط بحرانی

اگر بخواهیم مختصر عواملی که موجب این مشکلات می شوند را نام ببریم می توان به Learning rate کوچک و بعضی مشکلات ناشی از طراحی اشتباه از جمله استفاده از activation function های tanh و sigmoid اشاره کرد.

ب) حال روش های موجود برای حل مشکلات Gradient Descent را بررسی می کنیم:

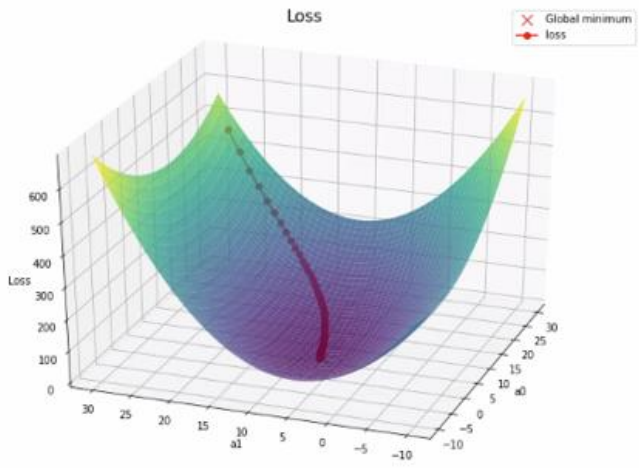
روش Momentum برای گرادیان کاهشی: قانون بروزرسانی وزن ها در Gradient Descent به صورت $W_{new} = W_{old} - \frac{\alpha dj}{dW}$ بود. حال در روش Momentum عبارت جدیدی با نام Velocity(t) که مشتق تابع Loss است را معرفی می کنیم که به کمک فرمول زیر بروزرسانی می شود:

$$V_t = \gamma V_{t-1} + \alpha \frac{dJ}{dW}$$

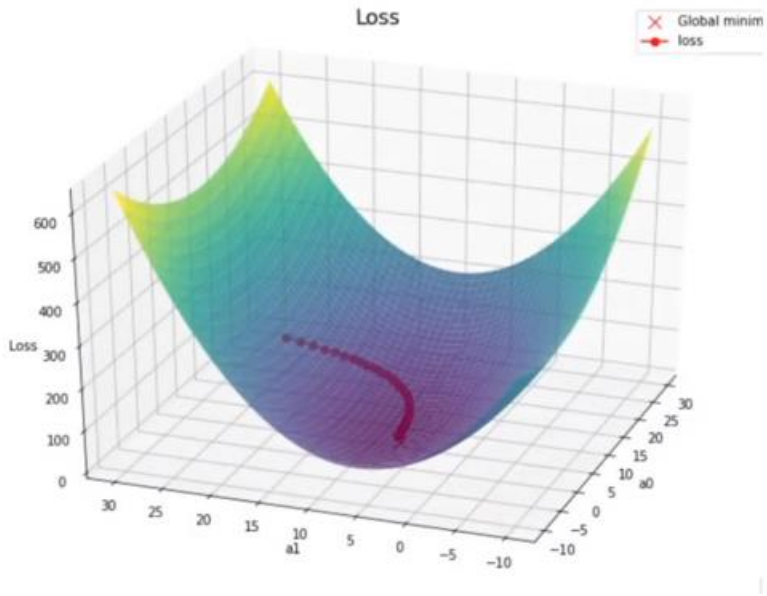
پس قانون آپدیت وزن ها به صورت زیر تغییر می کند:

$$W_{new} = W_{old} - V_t$$

این ترم کمک می کند که نرخ یا سرعت مناسب حرکت به سمت نقطه بهینه را وارد فرمول کنیم و بهتر به سمت بهینه شدن حرکت کنیم. شکل ۴ با Gradient Descent خالی بهینه سازی را انجام داده و در شکل ۵ با Momentum که به کمک این روش به جای ۲۰۰ ایپاک ۱۰۰ ایپاک نیاز شده است.



شکل ۱-۴ بهینه سازی به کمک Gradient Descent



شکل ۱-۵ بهینه سازی به کمک Gradient Descent و Momentum

می توان دید که این روش تعداد ایپاک ها را نصف کرده است و علت این کاهش هم حل مشکل Vanishing Gradient است. در مسیر حرکت هم می توان دید اثر Momentum را به صورتی که در نقاط با گرادیان کوچک گیر نکرده و سریعتر خارج شده است زیرا ترم اضافه تر آن در این نقاط کوچک بودن مقدار گرادیان را جبران کرده است پس اگرچه momentum سریعتر و بهتر converge می کند ولی همه ی مشکلات گرادیان کاهشی را حل نمی کند.

روش Adam:

روش دیگر روش Adam است که با روش کلاسیک Gradient Descent تفاوت دارد و برای هر پارامتر مقدار learning rate متفاوتی را بر حسب momentum اول و momentum دوم تعیین می کند و از فرمول های $W_{t+1} = W_t - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$ که $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ و $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ استفاده می کند و همچنین برای محاسبه این مقادیر داریم $m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial W_t}$ و $v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial W_t}\right)^2$

در مسایل روزمره بسیار از این روش بهینه سازی که پایه اش از گرادیان کاهشی است، استفاده می شود مخصوصا در مسایل non-convex زیرا پیاده سازی فرمول های آن راحت است و از نظر محاسباتی و حافظه بهتر عمل می کند. در مقابل noise و rescaling و ابعاد بالا مقاومت بیشتری دارد. می توان گفت

که Adam سرعت را بیشتر می کند ولی برای حل مشکلات Gradient Descent طراحی نشده است اگرچه با تنظیم learning rate تا حدی کنترل می کند ولی کاملا موفق نیست.

روش AdaDelta:

در این روش روی AdaGrad بهینه سازی انجام شده است و با تغییرات نرخ یادگیری سعی در بهبود بهینه سازی می کند. فرمول آپدیت کردن این روش به شرح زیر است(نیازی به مقداردهی اولیه برای نرخ یادگیری نیست):

$$W_{t+1} = W_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t} + \epsilon} \frac{\partial L}{\partial W_t}$$

همچنین می دانیم که $D_t = \beta D_{t-1} + (1-\beta)(\Delta W_t)^2$ و $v_t = \beta v_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial W_t}\right)^2$

همانند Adam این روش نیز برای حل این مشکلات طراحی نشده اگرچه تنظیم نرخ یادگیری ای که در آن انجام می شود تا حدی کمک می کند ولی خیلی مناسب نیست و بهتر است از روش های مرسوم که برای حل این مشکلات استفاده می شود استفاده کنیم مثل: کاهش تعداد لایه ها و ساده تر کردن مدل، استفاده از شبکه LSTM، Gradient Clipping، نرمال کردن وزن ها، استفاده نکردن از Sigmoid و ...

۲. بیش برازش یا overfitting حالتی است که مدل روی داده های training بیش از حد آموزش دیده است یا پیچیدگی آن از پیچیدگی مساله خیلی بیشتر است. در این حالت نتیجه و دقت روی داده های آموزش خیلی خوب است ولی مدل Generality ندارد و روی داده هایی که قبلا ندیده است مثل train یا validation عملکردی ضعیف تر خواهد داشت و یادگیری بیش از حد ما نه تنها مفید نبوده بلکه عملکرد را بدتر کرده است.

Dropout:

در این روش در حین یادگیری به صورت تصادفی تعدادی از نورون ها را ignore می کنیم و در نتیجه در هر مرحله از یادگیری از زاویه ی دیدی متفاوت به نورون ها و داده های ورودی آن لایه نگاه می کنیم در نتیجه overfitting انجام نمی شود یا کمتر می شود.

Norm Penalty:

در ای روش به تابع loss ضریبی از نرم وزن ها یعنی $(\sum_{i=0}^{n}||w_i||^2)\frac{\gamma}{2m}$ را اضافه می کنند که m تعداد ورودی ها و گاما پارامتر تنظیم است. در این روش چون نرم وزن ها در تابع هزینه می آید وقتی عمل یادگیری بیش از حد انجام می شود و overfitting اتفاق می افتد چون مقادیر وزن ها ممکن است بی رویه بزرگ شوند، مدل به آن سمت حرکت نخواهد کرد و از بیش برازش جلوگیری می شود. توجه شود که این ترم روی بایاس ها اثر نمی گذارد و فقط در وزن ها می آید زیرا بایاس به تعداد داده کمتری برای تعیین شدن نیاز دارد.

Early Stopping:

این روش شایع ترین روشی است که همیشه در مدل های iterative استفاده می شود به این صورت که طبق توضیح بالاتر مدل پس از چند مرحله آموزش روی داده های آموزش بیش از حد fit می شود و خاصیت general بودن خود را از دست می دهد پس برای جلوگیری از این اتفاق از داده های validation استفاده می کنیم و در هر ایپاک یا iteration عملکرد مدل را روی داده های validation چک می کنیم و اگر مشاهده شد که با ادامه ی یادگیری روی train بهبودی روی validation صورت نگرفته است و یا حتی بدتر شده است متوجه می شویم که باید در مرحله ی قبل آموزش متوقف می شده است و اصطلاحا باید early stopping می کردیم.

۳. می دانیم که افزایش تعداد لایه ها و تعداد نورون ها، موجب افزایش تعداد پارامترها و در نتیجه افزایش پیچیدگی مسئله می شود. در حین تدریس دیدیم که شبکه با یک لایه مخفی از پس داده های غیر خطی بر می آید اما برای pattern های non-convex خیلی مناسب نیست و تضمین جواب خوب ندارد اما هنگامی که از دو لایه مخفی استفاده کردیم در یادگیری این نوع داده هم موفق عمل کردیم. مشاهده کردیم که دو لایه مخفی خاصیت general function approximator را دارا است ولی دو محدودیت دارد: یکی اینکه در مسایل دنیای واقعی مثل تصویر، صدا، سری های زمانی و ... پیچیدگی مسایل خیلی بالا است و ساختارهای ما با وجود توانایی بالایشان در هر مرحله زمان زیادی زمان نیاز دارند و بهینه سازی طولانی خواهد شد همچنین احتمال overfitting زیاد می شود. مشکل دیگر distortion و نویز هایی است که گاها روی داده می افتند و نیاز به فیلترها و اسکیل های مختلفی برای حل آنها است که در دو لایه مخفی ممکن نیست.

۴. معمولا مسایل در دنیای واقعی از پیچیدگی و سختی بسیار بالایی برخودار هستند و نیاز به مدلی با پارامترهای فراوان دارند اما واضح است که افزودن پیچیدگی و پارامتر به مساله بدون مشکل نخواهد بود و مشکلات حافظه ای و زمانی گریبان گیر ما خواهند شد. پس باید tradeoff بین بهبود مدل با افزودن پیچیدگی و افزوده شدن زمان و حافظه ی مورد نیاز و همچنین سخت تر شدن همگرایی را رعایت کنیم. در بعضی مسایل به دلیل سختی و پیچیدگی ذات مساله نیاز به افزودن پیچیدگی مدل حس می شود ولی این کار محدودیت دارد و نمی توان به سادگی تعداد پارامترها را زیاد کرد و این کار را تنها تا جایی می توان ادامه داد که زمان و حافظه ما اجازه می دهد و معمولا شبکه ای را انتخاب می کنیم که پیچیدگی بیش از حد نداشته باشد و درصد قابل قبولی به ما بدهد. فرمولی برای تعیین تعداد لایه های مورد نیاز برای یک مساله وجود دارد که می توان از آن استفاده کرد:

$$Number\ of\ Layers = \frac{Number\ of\ Training\ Samples}{\alpha(Number\ of\ Input\ Neurons + Number\ of\ output\ Neurons)}$$

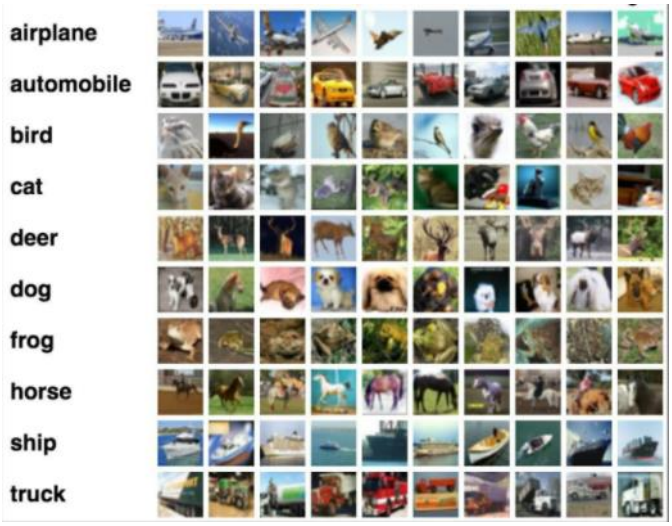
که α در این فرمول ضریبی است بین ۱ تا ۱۰ و می توان دید که به سادگی تعداد لایه ها تعیین می شود. برای تعداد نورون هر لایه هم معمولا به دانش قبلی و ساختار کلی شبکه روی آورده می شود مثلا ابتدا پله پله در چند لایه ابتدایی تعداد نورون ها را تا چندین برابر نورون های ورودی زیاد می کنیم سپس کاهش می دهیم تا به نورون های لایه آخر برسیم.

۵. معمولا توصیه می شود که مقادیر اولیه وزن ها و بایاس ها را تصادفی انتخاب کرده و از مقداری ثابت برای همه ی آنها استفاده نکنیم مثلا فرض کنید مقدار ثابت a را برای همه ی وزن ها در نظر بگیریم در ای صورت خروجی همه ی نورون های یک لایه(اگر fully-connected باشد) یکسان شده و برابر a ضرب در مجموع ورودی ها از لایه قبل می شوند و این اتفاق یادگیری را دچار مشکل می کند و تعداد زیادی ایپاک طول خواهد کشید تا از این وضعیت خارج شویم علت این امر هم یکسان بودن نورون ها در ایپاک های اول است که این امر منجر می شود ما از تعداد نورون های بالا و پیچیدگی که به مساله دادیم بهره ای نبریم. همچنین می دانیم برای ReLU مقادیر تصادفی مناسب ترند. اگر همه ی وزن ها صفر انتخاب شوند هم یادگیری تقریبا انجام نمی شود و مقادیر نورون ها ثابت خواهند بود و در نتیجه تنها اولین مرحله از یادگیری انجام خواهد شد پس بهتر است که مقادیر وزن ها را توسط یک توزیع گوسی با میانگین صفر و واریانس محدود مشخص کنیم.

۶. در سوال اول این بخش به این موضوع پرداخته شد علی ای حال دوباره موارد مهم را ذکر می کنیم: Exploding Gradient زمانی اتفاق می افتد که مقدار گرادیان بزرگ است و در آپدیت کردن وزن ها این مقادیر بزرگ موجب تغییرات بیش از حد می شوند و ممکن است سبب واگرایی شوند و بهینه سازی به درستی انجام نشود. Vanishing Gradient حالت برعکس است به این صورت که مقدار گرادیان کوچک می شود و در آپدیت کردن وزن ها تغییرات اینقدر کوچک می شوند که در آپدیت کردن اثر بزرگی نمی گذارد و می توان گفت که یادگیری انجام نمی شود و مدل ما ممکن است برای خارج شدن از این نقطه تعداد زیادی ایپاک نیاز داشته باشد یا اصلا نتواند نقطه ی بهینه ای را بیابد. تفاوت این دو مشکل در بزرگی و کوچکی مقدار گرادیان است همچنین در یکی ممکن است واگرا شویم و در دیگری ممکن است در ماکسیمم یا مینیمم محلی گیر کنیم و یا تعداد زیادی ایپاک با تغییرات خیلی کوچک داشته باشیم.

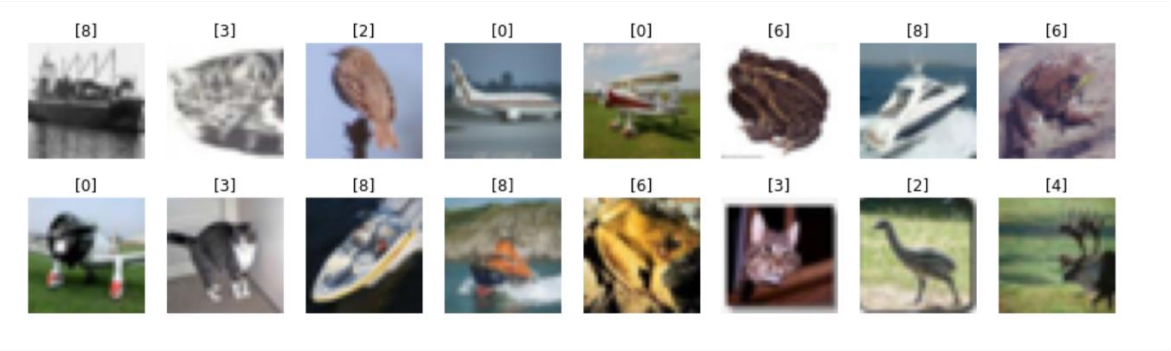
سوال ۲ – CNN

در این سوال از داده های Cifar10 قرار است استفاده شود، که شامل داده هایی با ۶۰۰۰۰ تصویر ۳۲*۳۲ هستند که در ۱۰ کلاس مختلف Labeled شده اند و هر کلاس ۶۰۰۰ نمونه دارد. از این ۶۰۰۰۰ داده، ۵۰۰۰۰ داده برای یادگیری در نظر گرفته شده است و ۱۰۰۰۰ داده برای آزمون در نظر گرفته شده است.



شکل ۲-۱ نمونه کلاس های مجموعه داده Cifar10

داده ها به پنج Batch که هر کدام ۱۰۰۰۰ داده(تصویر) در خود جای داده اند برای تمرین و یک Batch که ۱۰۰۰۰ داده(تصویر) در خود جای داده است (که دقیقا از هر کلاس ۱۰۰۰ عکس به صورت تصادفی در آن انتخاب شده است) برای آزمون تقسیم شده است.



شکل ۲-۲ نمونه ای از داده ها با برچسبشان

داده های هر کدام از این Batch ها 10000*3072 میباشد که ۱۰۰۰۰ تا که نمونه های ما هستند و ۳۰۷۲ ستون به ترتیب ۱۰۲۴ ستون آن کانال قرمز آن تصویر ۱۰۲۴ ستون بعد آن کانال سبز آن تصویر و ۱۰۲۴ ستون بعد آن کانال آبی آن تصویر میباشد.

هدف این تمرین پیاده سازی شبکه عصبی کانوولوشنی CNN سه لایه ای که در هر لایه چند فیلتر و یک pool و یک لایه dense موجود است میباشد که هدف آن Classify کردن داده ها است.

۱: مشخصات شبکه عصبی

۱- اندازه پنجره های Convolution اندازه Stride و تعداد filter ها:

مدل اول:

- برای انجام عملیات کانوولوشن، اندازه پنجره ها 3*3 تعریف شده است.
- Stride آن برابر با ۱ میباشد یعنی یک cell هر بار به جلو میرود و پرش بیشتر از آن نمیکند.
- تعداد filter ها برای انجام این مدل، در لایه اول و لایه کانوولوشنی اول ۱۶ در نظر گرفته شده، گویی ۱۶ ویژگی استخراج میشود که هر کدام از پنجره های ۳*۳ تشکیل شده اند ، دومین لایه کانولوشنی این لایه با ۳۲ فیلتر است، و در لایه دوم به ترتیب هر کدام از Conv2D ها ۳۲ و ۶۴ عدد فیلتر و در لایه سوم ۶۴ و ۱۲۸ فیلتر در نظر گرفته شده است.
- خروجی کانوولوشن را به صورتی Padding میکنیم که هم سایز با ورودی یعنی ۳۲*۳۲ شود.
- دو لایه dense نیز در انتها اضافه شده است که اولی تعداد پارامتر ها را پس از عبور از این لایه کاهش میدهد و سپس به ۱۰ ویژگی مپ میکند (اکتیویشن فانکشن به کار رفته در لایه dense آخر، softmax میباشد.)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 407,178
Trainable params: 407,178
Non-trainable params: 0
```

شکل ۲-۳ مدل اول

مدل دوم:

- برای انجام عملیات کانوولوشن، اندازه پنجره ها 3*3 تعریف شده است.
- Stride آن برابر با ۱ میباشد یعنی یک cell هر بار به جلو میرود و پرش بیشتر از آن نمیکند.
- تعداد filter ها برای انجام این مدل، در لایه اول و لایه کانوولوشنی اول۳۲ در نظر گرفته شده، یعنی ۳۲ ویژگی استخراج میشود که هر کدام از پنجره های ۳*۳ تشکیل شده اند ، دومین لایه کانولوشنی این لایه با ۶۴ فیلتر است، و در لایه دوم به ترتیب هر کدام از Conv2D ها ۶۴و ۱۲۸ عدد فیلتر و در لایه سوم ۱۲۸ و ۲۵۶ فیلتر در نظر گرفته شده است.

- خروجی کانوولوشن را به صورتی Padding میکنیم که هم سایز با ورودی یعنی ۳۲*۳۲ شود.
- - دو لایه dense نیز در انتها اضافه شده است که اولی تعداد پارامتر ها را پس از عبور از این لایه کاهش میدهد و سپس به ۱۰ ویژگی مپ میکند (اکتیویشن فانکشن به کار رفته در لایه dense آخر، softmax میباشد.)

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_30 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 4, 4, 256)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               1048832
_________________________________________________________________
dense_11 (Dense)             (None, 10)                2570
=================================================================
Total params: 1,624,330
Trainable params: 1,624,330
Non-trainable params: 0
_________________________________________________________________
```

شکل ۲-۴ مدل دوم

۲- توابع فعال سازی :

برای فعال سازی نورون ها تابع ReLU در نظر گرفته شده است که عملکرد بسیار خوبی در شبکه های عصبی کانوولوشنی دارد

۳- اندازه لایه های Fully Connected:

در مدل اول: این عدد برابر با ۴*۴*۱۲۸ برابر با ۲۰۴۸ است

در مدل دوم: این عدد برابر با ۴*۴*۲۵۶ برابر با ۴۰۹۶ است

۴- تابع Loss مورد استفاده و روش بهینه سازی:

تابع Loss برای هر دو مدل برای هر دو مدل Categorical_crossentropy انتخاب شده است

۵- اندازه Mini-Batch مورد استفاده:

در مدل اول: ۳۲

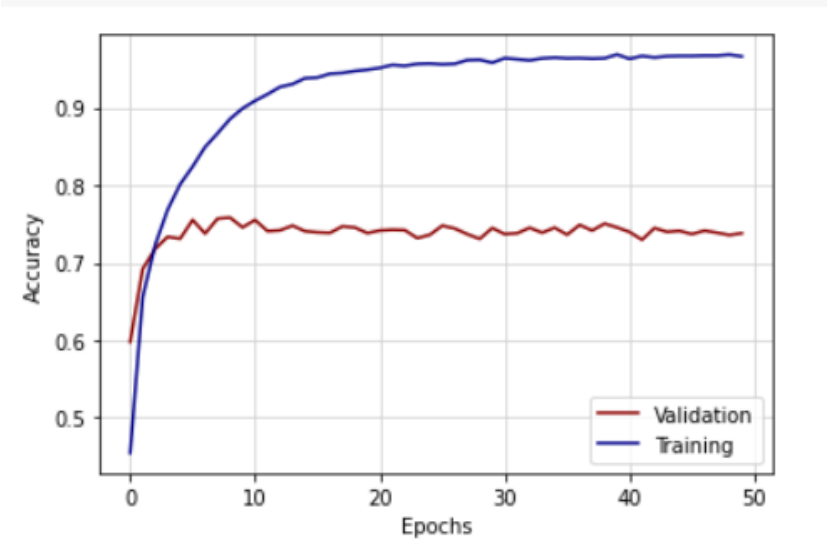
در مدل دوم: ۱۲۸

۲: عملکرد شبکه عصبی

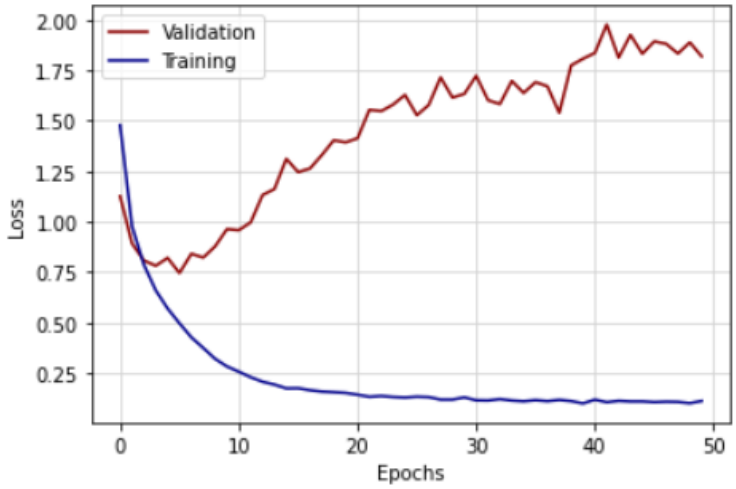
در این بخش دقت عملکرد شبکه عصبی آموزش داده شده برای داده های آموزش و تست، نمایش داده میشود:

مدل اول:

مدل پردازش شده با ۵۰ ایپاک:



شکل ۲-۵ دقت مدل ۱



شکل ۲-۶ loss برای مدل ۱

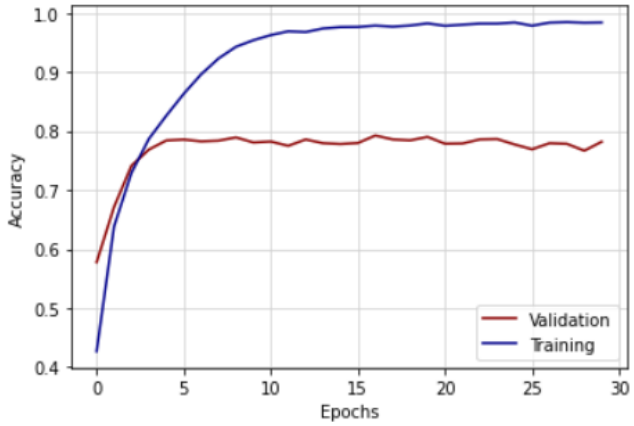
بهترین دقت برای داده های آموزش: ۰.۹۷۳ بعد از ۵۰ ایپاک

بهترین دقت برای داده های ارزیابی: ۰.۷۵ بعد از ۱۱ ایپاک

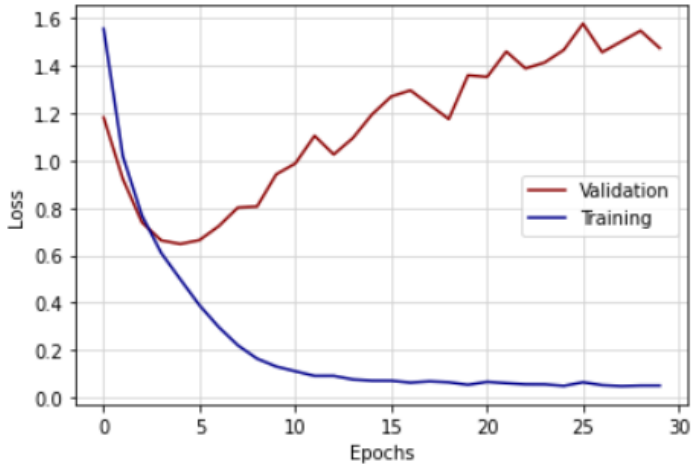
دقت برای داده های تست: ۰.۷۲

مدل دوم:

مدل پردازش شده با ۳۰ ایپاک:



شکل ۲-۷ دقت مدل دوم



شکل ۲-۸ loss مدل دوم

بهترین دقت برای داده های آموزش: ۰.۹۸ بعد از ۳۰ ایپاک

بهترین دقت برای داده های ارزیابی: ۰.۷۹ بعد از ۲۰ ایپاک

دقت برای داده های تست: ۰.۷۸

نتیجه گیری: مدل دوم دقت بهتری برای داده های تست داشت و از همین مدل استفاده میشود.

۳: کارایی شبکه عصبی با ۰، ۱ و ۲ لایه مخفی:

شبکه عصبی برای هر سه حالت گفته شده بررسی میشود:

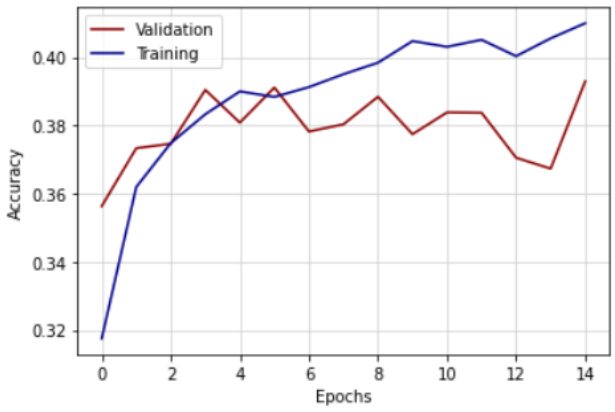
۱- شبکه طراحی ۰ لایه مخفی:

شبکه عصبی با صفر لایه مخفی به صورت مقابل به دست آمد:

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_23 (Flatten)         (None, 3072)              0
_________________________________________________________________
dense_42 (Dense)             (None, 10)                30730
=================================================================
Total params: 30,730
Trainable params: 30,730
Non-trainable params: 0
_________________________________________________________________
```

شکل ۲-۹مدل ۰ لایه مخفی

پس از Fit کردن مدل، بعد از۱۵ ایپاک گراف Accuracy حاصل گشت:



شکل ۲-۱۰ دقت مدل ۰ لایه مخفی

همچنین مقدار دقت این شبکه برای داده های تست برابر با ۰.۳۹۲ شد.

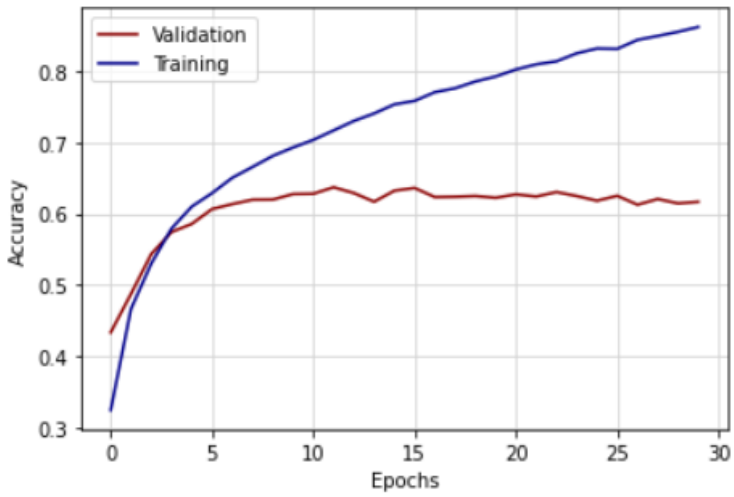
۲- شبکه طراحی ۱ لایه مخفی:

شبکه عصبی با یک لایه مخفی به صورت مقابل به دست آمد:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_5 (Dense)              (None, 16)                262160
_________________________________________________________________
dense_6 (Dense)              (None, 10)                170
=================================================================
Total params: 281,722
Trainable params: 281,722
Non-trainable params: 0
_________________________________________________________________
```

شکل ۲-۱۱ مدل ۱ لایه مخفی

پس از Fit کردن مدل، بعد از ۱۵ ایپاک گراف Accuracy حاصل گشت:



شکل ۲-۱۲ دقت مدل ۱ لایه مخفی

همچنین مقدار دقت این شبکه برای داده های تست برابر با ۰.۶۱۶ شد.

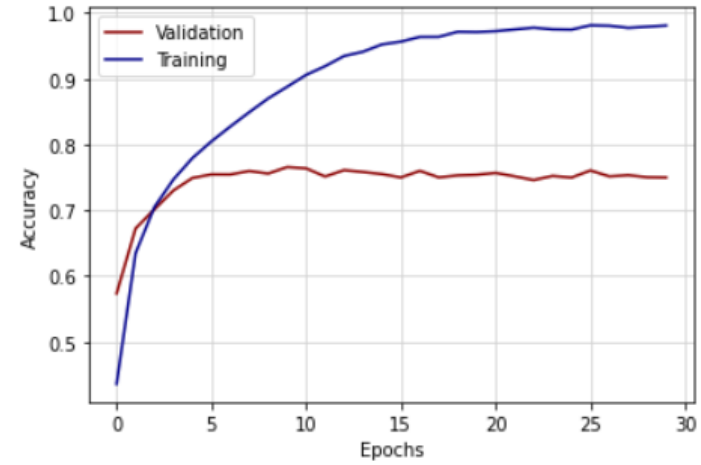
۳- شبکه طراحی ۲ لایه مخفی:

شبکه عصبی با دو لایه مخفی به صورت مقابل به دست آمد:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                262176
_________________________________________________________________
dense_8 (Dense)              (None, 10)                330
=================================================================
Total params: 392,682
Trainable params: 392,682
Non-trainable params: 0
_________________________________________________________________
```

شکل ۲-۱۳ مدل ۲ لایه مخفی

پس از Fit کردن مدل، بعد از ۱۵ ایپاک گراف Accuracy حاصل گشت::

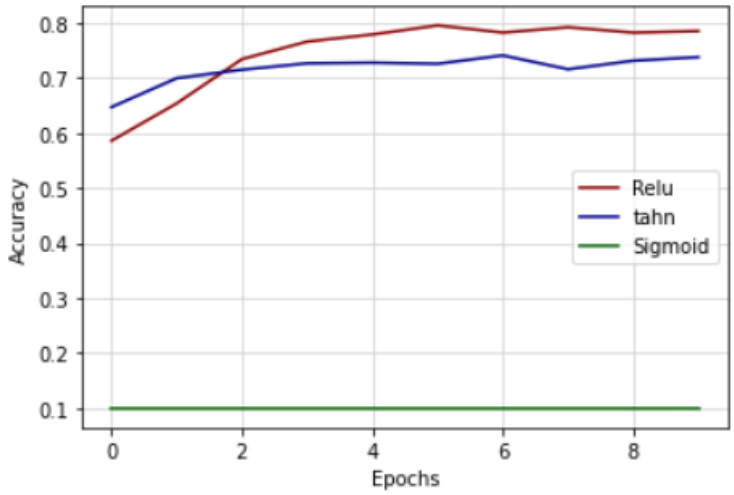


شکل ۲-۱۴ دقت مدل ۲ لایه مخفی

همچنین مقدار دقت این شبکه برای داده های تست برابر با ۰.۷۴۹ شد.

می توان مشاهده کرد که با دو لایه مخفی بهترین دقت را به دست آوردیم.

۴: کارایی شبکه عصبی با توابع فعالساز ReLU, tanh, sigmoid:

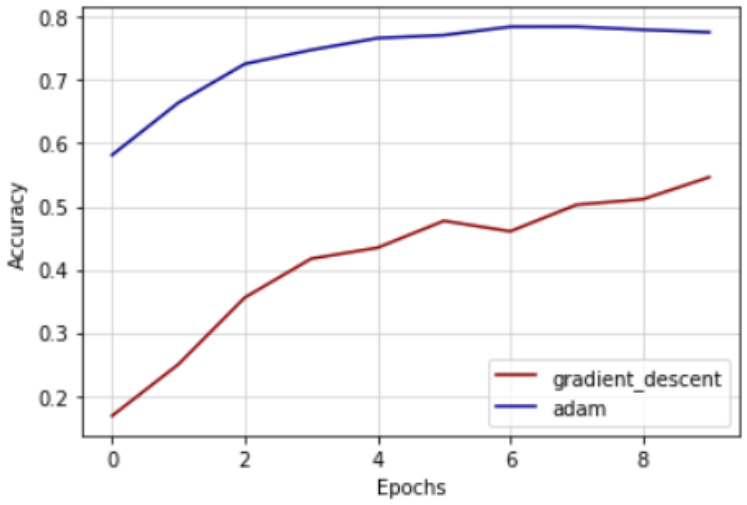


شکل ۲-۱۵ دقت مدل در ۱۰ ایپاک برای activation function های مختلف

در این بخش تاثیر توابع فعالساز بررسی شده که البته در لایه آخر هر ۳ مدل از تابع softmax استفاده گردیده است چون توابع relu و tanh برای لایه های میانی مناسب هستند و تابع sigmoid فقط برای حالتی که دو خروجی داریم(نه ۱۰) مناسب است.

همانطور که از شکل مشخص است دقت مدل برای تابع فعالساز relu از tanh و sigmoid بهتر است و دلیل این امر این است که relu نسبت به sigmoid همگرایی بهتری دارد و همچنین به مشکل gradient saturation دچار نمی شود. مزیت دیگر relu محاسبات بسیار ساده تر آن نسبت به دو تابع دیگر است که میتواند به سادگی با یک ماتریس threshold تمام صفر پیاده سازی شود. خروجی relu ، یک خروجی zero-centered نیست و به کارایی شبکه عصبی آسیبی نمیرساند (دلیل دقت کم sigmoid وجود این مساله است) همچنین مساله vanishing gradient کاملا در تابع sigmoid اثر گذار است و تابع tanh نیز این مساله را دارد به عبارت دیگر با کوچک شدن گرادیان آنها، به روز شدن پارامتر ها متوقف میگردد.

۵: کارایی شبکه عصبی با روش های بهینه سازی gradient descent و adam :



شکل ۲-۱۶ دقت مدل در ۱۰ ایپاک برای روش های بهینه سازی مختلف

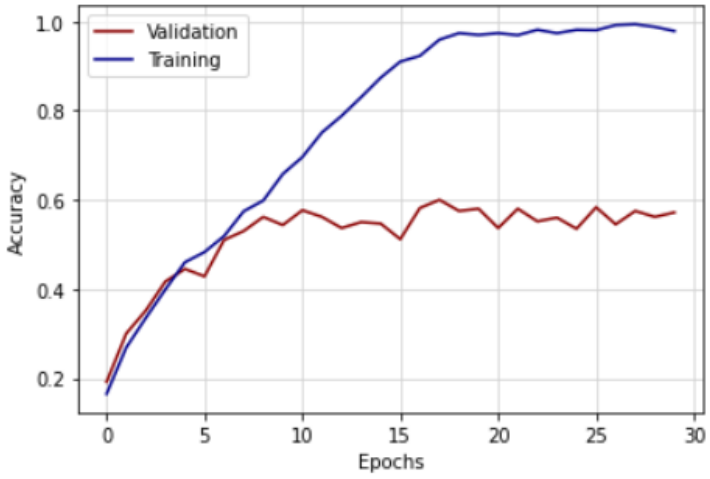
روش بهینه سازی adam از SGD بهتر عمل کرده و حدود ۲۵ درصد accuracy بالاتری در انتهای ۱۰ ایپاک به ما می دهد. دلیل این امر این است که adam قابل اتکا تر برای رسیدن به نقطه مینیمم در هنگام minimizeکردن تابع هزینه میباشد همچنین سریع تر هم هست.

۶: کارایی شبکه عصبی با کاهش حجم داده ها:

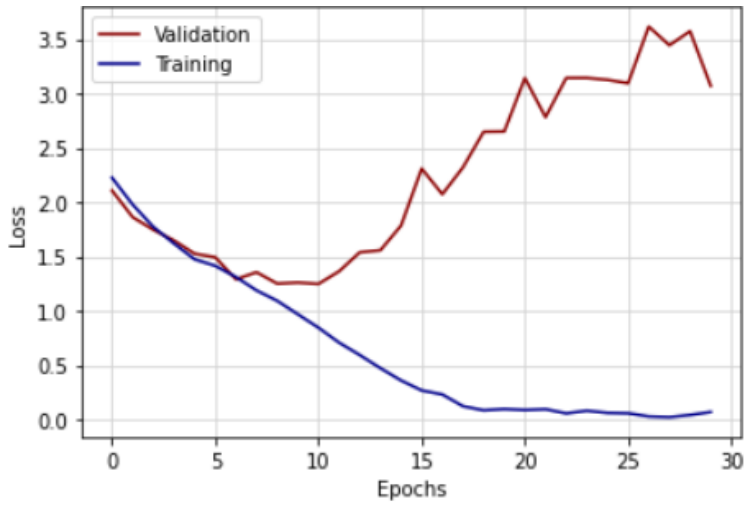
همانطور که گفته شد دیتا ست **cifar10** ، از ۶۰۰۰۰ داده ی یادگیری در ۱۰ کلاس تشکیل شده است. در این بخش هر کلاس را به ۶۰۰ داده تقلیل داده و بدیهتا در نهایت ۶۰۰۰ داده خواهیم داشت.

برای این کاهش از شمارنده هایی برای هر کلاس استفاده شده تا دیتا ها دقیقا ۶۰۰ تا باشند و از روی label ای که در داده های تست داریم این مقادیر را به کلاس های صحیح متناظر نسبت می دهیم.

در نهایت یک دیتاست جدید با ۶۰۰۰ داده داریم که شبکه را روی آن پیاده سازی کرده و پس از ۵۰ ایپاک به training accuracy ۹۹ درصد و حدود ۶۰ درصد هم دقت برای داده های validation می رسیم (لازم به ذکر است که ۱۰ درصد از ۶۰۰۰داده های تولیدی را به عنوان داده های validation در نظر گرفتیم).



شکل ۲-۱۷ دقت مدل در ۱۰ ایپاک برای داده های کاهش داده شده



شکل ۲-۱۸ lossمدل در ۱۰ ایپاک برای داده های کاهش داده شده

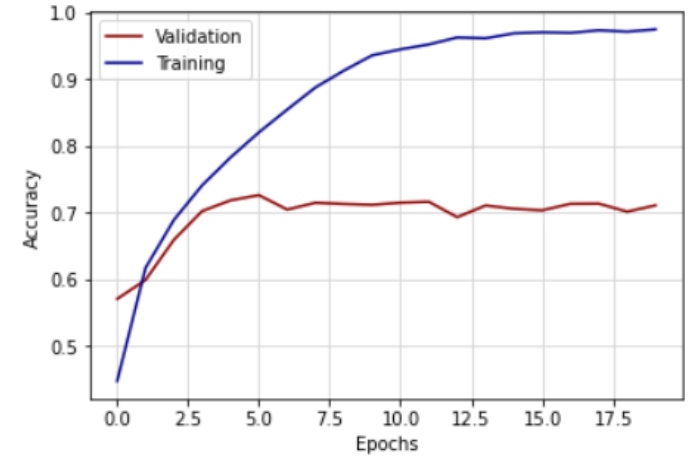
همانطور که مشخص است به دلیل کمتر شدن تعداد داده های آموزشی، مدل به خوبی گذشته کار نمی کند و دقتی حدود ۶۰ درصد برای داده های تست میگیریم. همچنین تعداد ایپاک بهینه برای این مدل ۱۳ است و از آن به بعد دچار overfitting می شویم که منطقی هم هست زیرا با داده ی کمتر زودتر به بیش برازش می رسیم.

۷: کرنل بزرگتر و یک عمل کانولوشن:

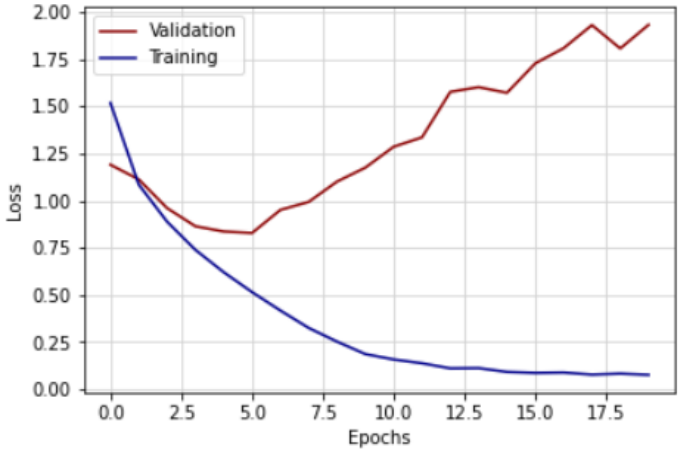
در این قسمت ، در هر لایه از یک لایه conv2D استفاده میکنیم ولی سایز کرنل را از (۳و۳) به (۶و۶) افزایش می دهیم.

مشاهده می شود که تعداد ایپاک لازم برای بهینه شدن شبکه به ۶ ایپاک کاهش میابد و دلیل این امر این است که پنجره ای که روی مدل کشیده میشود، سایزش بزرگ شده و اطلاعات بیشتری از عکس در هر ایپاک وارد شود. البته از آنجایی که یک لایه کانولوشنی در هر لایه کم شده، تعداد پارامترها کمتر میشود و پیچیدگی مدل نیز کم میشود.

در نتیجه دقت روی داده های تست به ۰.۷۰۸ تقلیل می یابد.



شکل ۲-۱۹دقت مدل در ۲۰ ایپاک با سایز کرنل بزرگتر

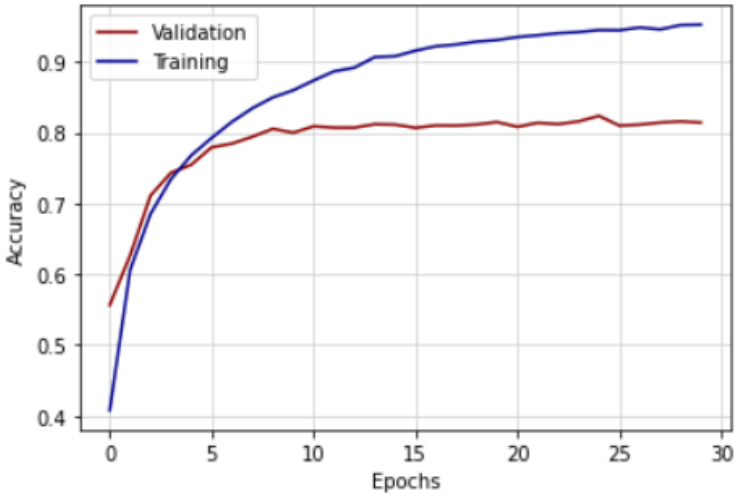


شکل ۲-۲۰ loss مدل در ۲۰ ایپاک با سایز کرنل بزرگتر

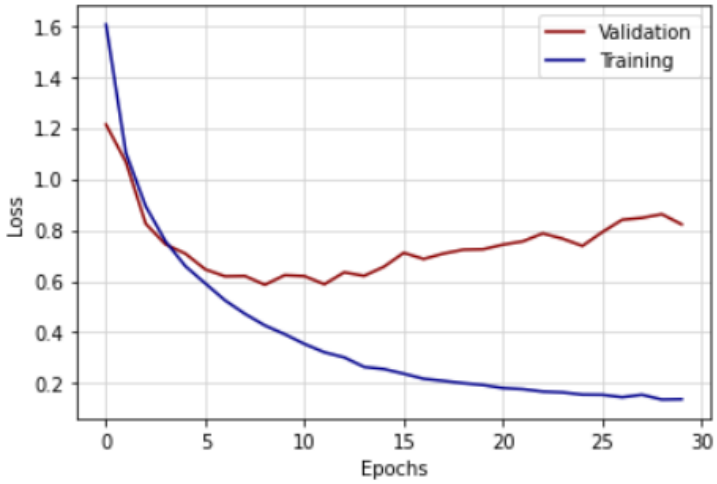
۷: تاثیر افزودن dropout به شبکه :

لایه dropout به منظور جلوگیری از overfitting زودهنگام گذاشته میشود. به این صورت که یک ورودی دارد که یک عدد بین ۰ تا ۱، که در واقع یک احتمال است را می گیرد و به صورت تصادفی و با احتمال ورودی اش، تعدادی از لایه های شبکه را غیر فعال یا به اصطلاح drop می کند.

ابتدا این احتمال را ۰.۲۵ گذاشتیم که موجب افزایش validation loss در ایپاک سیزدهم شد و در این ایپاک به دقت ۸۰ درصد روی داده های validation و دقت ۸۹ درصد روی داده های آموزشی رسیدیم.

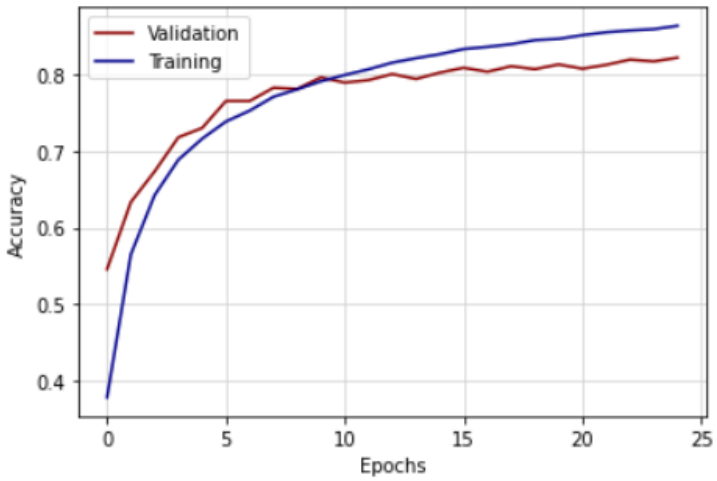


شکل ۲-۲۱ دقت مدل با dropout در ۳۰ ایپاک

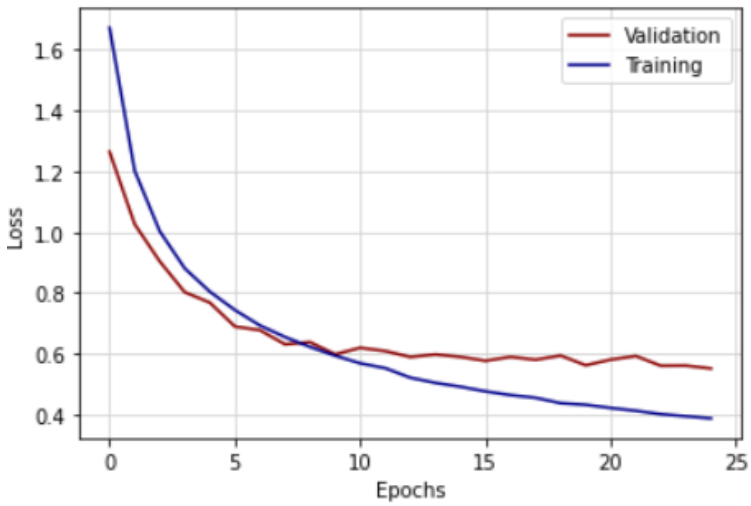


شکل ۲-۲۲ lossمدل با dropout در ۳۰ ایپاک

سپس به مقدار بهینه $p = 0.5$ برای تابع dropout مان رسیدیم زیرا با این مقدار در ۲۵ ایپاک دچار overfit نمیشویم و دقت داده های تست تا ۸۲.۷ درصد افزایش می یابد.



شکل ۲-۲۳ دقت مدل با dropout در ۲۵ ایپاک



شکل ۲-۲۴ loss مدل با dropout در ۳۰ ایپاک

سوال ۳ – Data Augmentation

۱: تاثیر data augmentation روی شبکه:

Data augmentation روشی است برای افزایش داده ها برای این منظور که overfitting با جبران دیتا های کم توسط این روش (افزودن دیتا های مصنوعی جدید با تفاوت به نسبت دیتای اولیه)، از بین برود. این روش رابطه نزدیکی با oversampling دارد.

هدف این است که از دیتا های موجود، دیتا های جدیدی که دچار تغییراتی شده اند را تولید کنیم که این امر باعث مقاومت بیشتر شبکه به distortion ها و افزایش generalization خواهد شد.

این روش تبدیل های مختلفی را جهت دستکاری عکس به منظور تولید دیتا های جدید شامل میشود.

Scaling:

در این روش اسکیل عکس میتواند به مقدار دلخواه resize شود و عکس جدیدی تولید گردد.

شکل ۳-۱ scaling در عکس با data augmentation

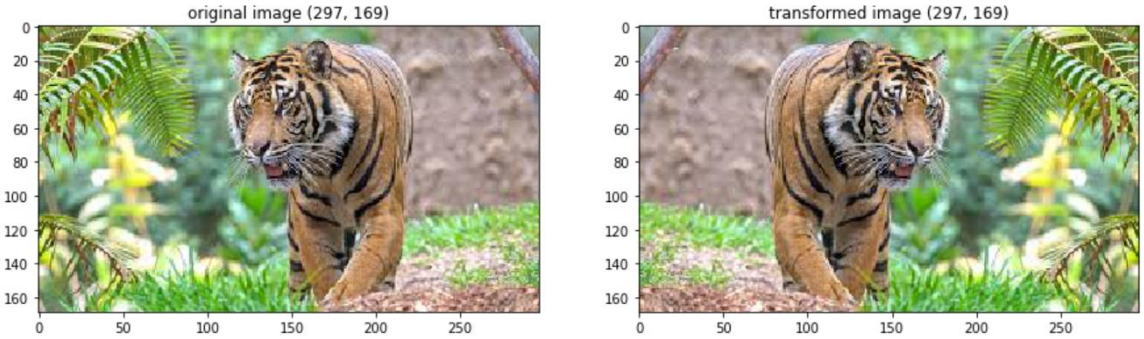
Cropping :

در این تبدیل قسمتی از عکس برداشته میشود.

شکل ۳-۲ cropping در عکس با data augmentation

: Flipping

این تبدیل در دو جهت vertical و horizontal میتواند انجام پذیرد که در شکل زیر در تبدیل horizontal مشاهده می شود:



شکل ۳-3 flipping در عکس با data augmentation

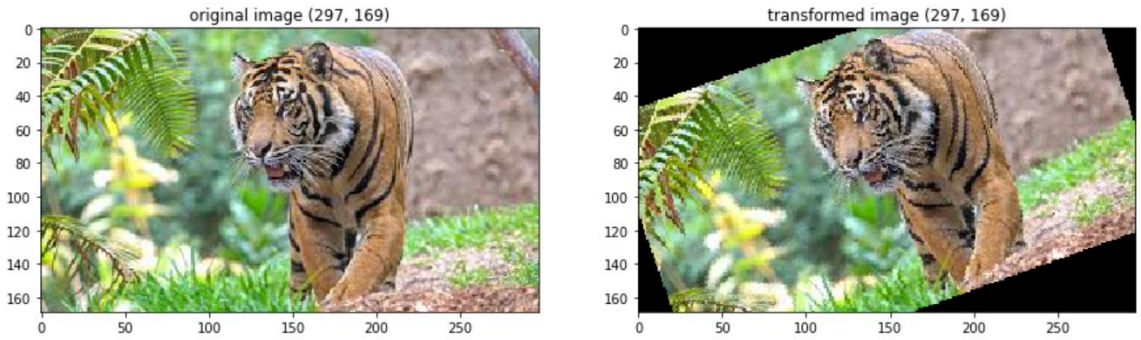
: Padding

یک border با سایز دلخواه به عکس اضافه میشود:

شکل ۳- 4 padding در عکس با data augmentation

: Rotation

چرخش عکس به مقدار زاویه دلخواه:



شکل ۳-۵ padding در عکس با data augmentation

همچنین میتوان ، روشنایی، رنگ و کیفیت تصویر را تغییر داد و دیتای جدیدی تولید کرد:

شکل ۳-۶ تغییر در hue, saturation ,brightness , color در عکس با data augmentation

از آنجایی که هدف data augmentation ، افزایشgeneralization و سنجیدن مدل در زمانی است که دیتاهای واقعی با error همراه هستند، این عمل فقط روی داده های آموزشی (و نه test) انجام میشود. اگر این کار روی داده های تست انجام شود، نوعی معرفی ارور به مدل به عنوان دیتای درست است که باعث ایجاد اشکال می شود. مثلا اگر عکس عدد ۶ در دیتاست MNIST را rotate کنیم به ۹ تبدیل میشود که در نتیجه حتی دیتا های درست قبل از augmentation نیز، احتمال خطا در تشخیص خواهند داشت.

۲ : تولید ۱۰ عکس مصنوعی با data augmentation:

برای اینکار از تابع ImageDataGenerator از کتابخانه keras استفاده شده است :

```
datagen = ImageDataGenerator(    rotation_range=15,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.4,1.5],
    zoom_range=0.3,
    width_shift_range=0.1,
    height_shift_range=0.1)
```

در این کد، rotation ،flip افقی و عمودی (که به طور رندوم اعمال می شوند)، تغییر در روشنایی عکس، زوم کردن روی عکس و شیفت عکس اعمال شده است.

شکل ۳-۷ تصویر original

شکل ۳-8تصویر با data augmentation اول

شکل ۳-۹ تصویر با data augmentation دوم

شکل ۳-۱۰ تصویر با data augmentation سوم

شکل ۳-۱۱ تصویر با data augmentation چهارم

شکل ۳-۱۲ تصویر با data augmentation پنجم

شکل ۳-۱۳ تصویر با data augmentation ششم

شکل ۳-۱۴ تصویر با data augmentation هفتم

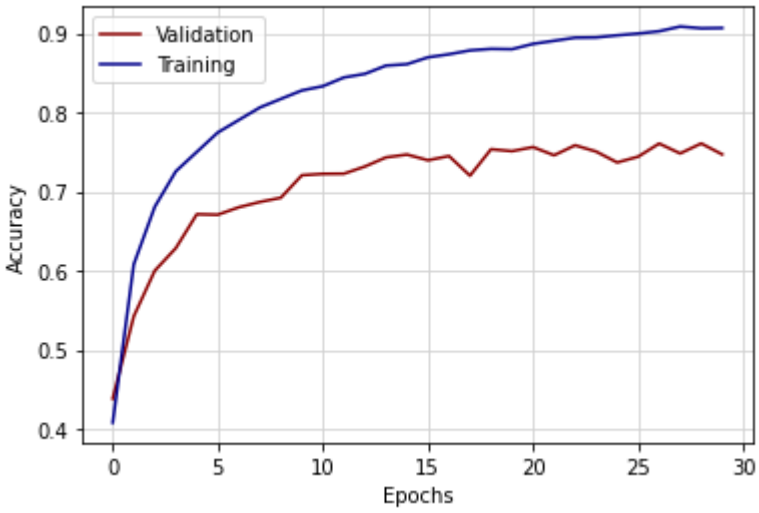
شکل ۳-۱۵ تصویر با data augmentation هشتم

شکل ۳-۱۶ تصویر با data augmentation نهم

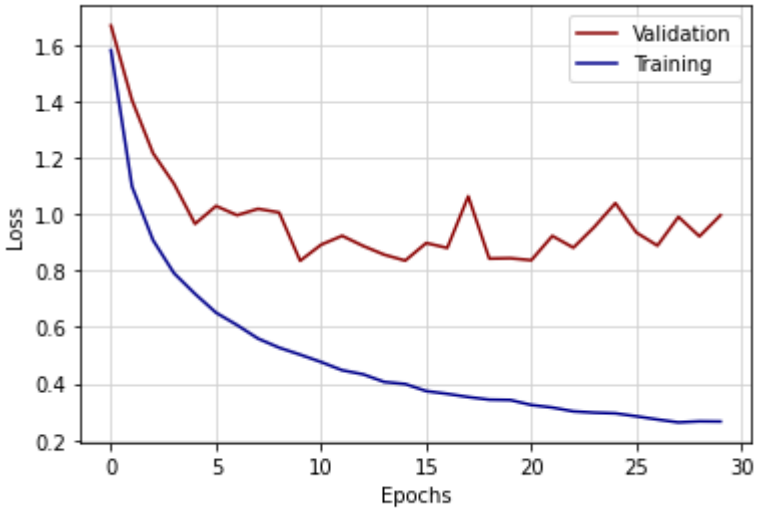
شکل ۳-۱۷ تصویر با data augmentation دهم

۳ : آموزش مدل با داده های کمتر:

حال در این قسمت داده های دو کلاس سک و گربه در دیتاست CIFAR10 را دستخوش تغییر می دهیم و ۹۰ درصد از داده های آموزش هر یک را تصادفی حذف می کنیم تا کل داده های آموزش ما برابر ۴۱۰۰۰ به جای ۵۰۰۰۰ شود و داده های تست نیز برابر ۱۰۰۰۰ باقی می مانند. دیتاست تغییر یافته را توسط بهترین مدل سوال دوم که مدل با dropout برابر ۰.۵ آموزش می دهیم. می توان دید که دقت مدل بیشتر از ۱۰ درصد کاهش می یابد و به ۷۴ درصد روی داده های تست می رسد.



شکل ۳-۱۸ دقت مدل روی داده های کاهش یافته



شکل ۳-۱۹ خطا مدل روی داده های کاهش یافته

همچنین دقت مدل نیز به شرح زیر است:

```
Test Loss 0.9968368411064148
Test Accuracy 0.7468000054359436
```

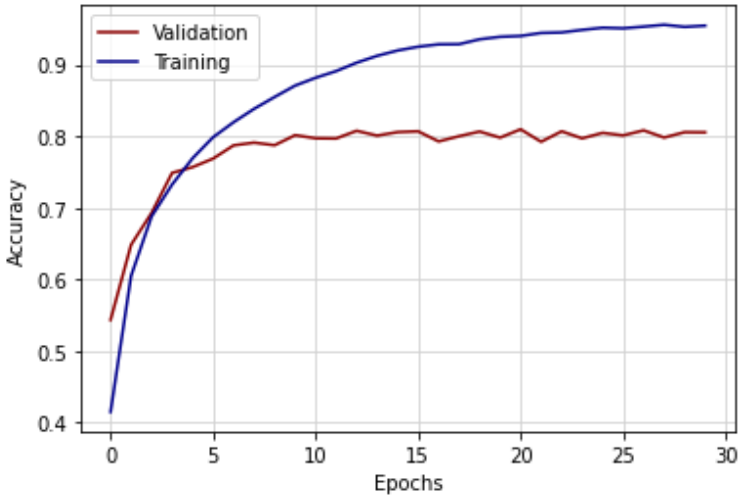
با دقت در confusion matrix مشاهده می شود که مدل در شناخت این دو کلاس به مشکل بزرگی خورده است و علت اصلی کاهش عملکرد نیز همین مشکل است.

```
confusion matrix=
 [[809  12  61   5  31   0  13  12  34  23]
 [  9 904   1   1   4   3   8   2  18  50]
 [ 38   2 744   7  92  10  66  34   4   3]
 [ 28   2 123 259 166  64 237 101   9  11]
 [  9   0  28   8 876   4  47  24   3   1]
 [ 11   4 154  86 145 311 142 136   6   5]
 [  4   0  35   2  34   1 914   7   1   2]
 [  6   1  16  10  69   8  15 869   2   4]
 [ 56  10  12   2   5   0   9   4 886  16]
 [ 19  34   5   3   7   2  12   9  13 896]]
```

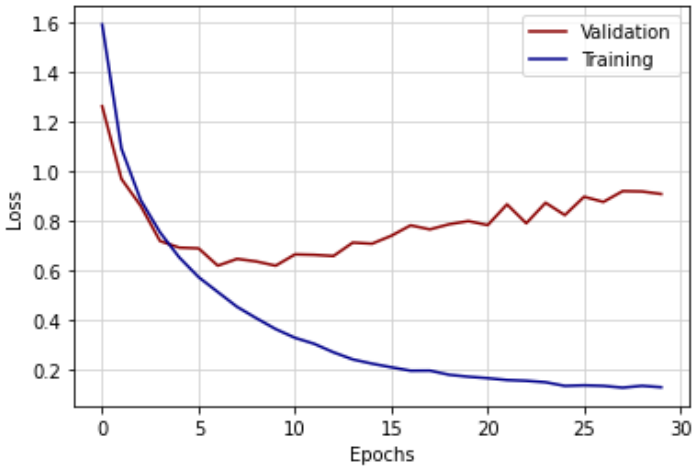
شکل ۳-۲۰ confusion matrix داده های تست

مشاهده می شود که تعداد داده هایی که کلاس گربه(۳) یا سگ(۵) تشخیص داده شده است کمتر است و بر خلاف بقیه که در حد ۸۰۰ و ۹۰۰ تا هستند در حد ۳۰۰ تا است که این نشان می دهد در تشخیص این مدل دچار مشکل شده ایم همچنین دیتاست ما biased است و تمرکز روی بقیه کلاس ها بیشتر است در نتیجه datapoint ها با لیبل گربه و سگ اکثرا لیبلی اشتباه خورده اند.

حال برای حل این مشکل به Data Augmentation توسط ImageDataGenerator روی می آوریم و با تعیین مرز و بازه تغییرات روی عکس ها تعدادی داده را به صورت شبیه سازی شده تولید می کنیم و بار دیگر بهترین مدل سوال ۲ را روی داده های جدید امتحان می کنیم. لازم به ذکر است که دوباره تعداد داده های آموزش برابر ۵۰۰۰۰ شده است و داده های تست نیز ۱۰۰۰۰ تا هستند. نتایج این عملیات به شرح زیر است:



شکل ۳-۲۱ دقت مدل روی داده های augmented



شکل ۳-۲۲ خطا مدل روی داده های augmented

همچنین دقت مدل نیز به شرح زیر است:

```
Test Loss 0.9105793237686157
Test Accuracy 0.8054999709129333
```

می توان دید که تا حدی حذف داده های train را جبران کردیم و به جای ۷۵ درصد دقت روی تست به ۸۰ درصد رسیدیم البته واضح است که این کار می تواند جای داده های واقعی را پر کند و دقت در مجموع کاهش داشته است ولی توانستیم با این تکنیک کنترل کنیم. با دقت در confusion matrix مشاهده می شود که مشکل مدل در مواجهه با دو کلاس سگ و گربه تقریبا رفع شده است اگرچه تعداد داده هایی مه لیبل ۳ و ۵ خورده اند هنوز کمتر از بقیه است اما به نسبت داده های بدون Augmentation پیشرفت بزرگی داشتیم و علت کاهش دقت نیز این است که با وجود اینکه داده های

موجود از نظر تعداد برابر باقی کلاس ها هستند اما کیفیت آنها کمتر است و information کمتری را ذخیره کرده اند.

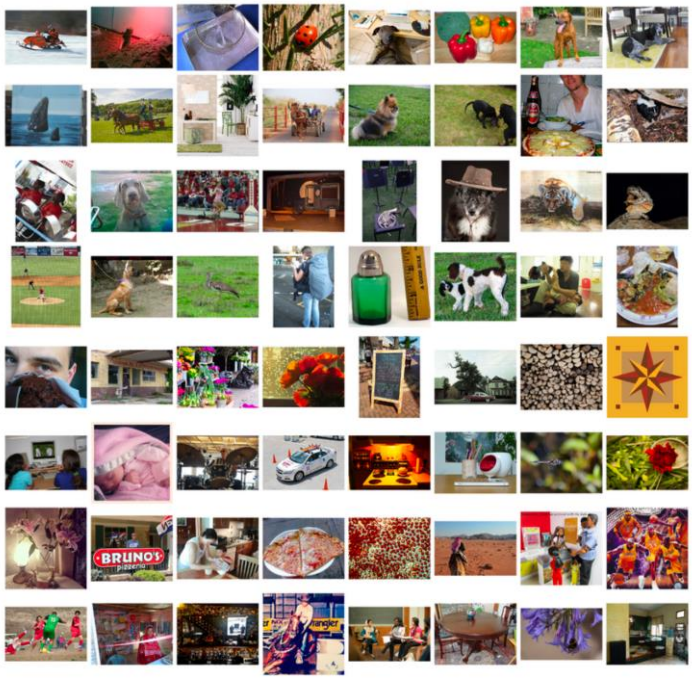
```
confusion matrix=
 [[809  12  61   5  31   0  13  12  34  23]
 [  9 904   1   1   4   3   8   2  18  50]
 [ 38   2 744   7  92  10  66  34   4   3]
 [ 28   2 123 259 166  64 237 101   9  11]
 [  9   0  28   8 876   4  47  24   3   1]
 [ 11   4 154  86 145 311 142 136   6   5]
 [  4   0  35   2  34   1 914   7   1   2]
 [  6   1  16  10  69   8  15 869   2   4]
 [ 56  10  12   2   5   0   9   4 886  16]
 [ 19  34   5   3   7   2  12   9  13 896]]
```

شکل ۳-۲۳ confusion matrix داده های تست

سوال ۴ – Transfer Learning

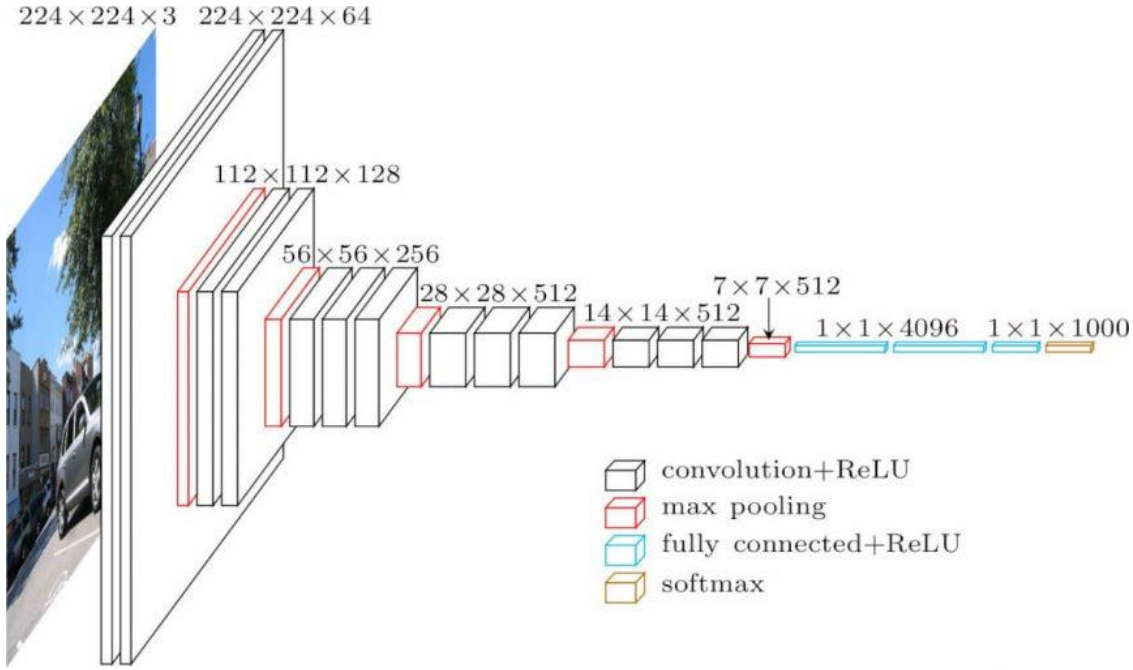
در این سوال قصد داریم transfer learning را به صورت عملی در یک مثال ببینیم. مدل استفاده شده VGG است که یک مدل پیشرفته برای پردازش و کلاس بندی تصویر است. کد این سوال در فایل Project1-Q4 موجود است. باید فایل cup.jpg را در کولب آپلود کرده و سپس اجرا کنید.

۱. مدل ما VGG است که یک مدل خیلی قدرتمند برای پردازش تصویر و کلاس بندی اشیا در تصاویر است. این مدل state of the art نیست و مدل های بسیار پیشرفته تر با لایه ها و پارامترهای بیشتری در طی سال ها معرفی شده اند اما همین الان هم از VGG استفاده می شود زیرا دقت قابل قبولی دارد و پیشرفت مدل ها معمولا در حد چند درصد و یا چند دهم درصد است. این مدل در سال ۲۰۱۴ برای مسابقه ILSVRC که روی دیتاست Imagenet بوده، معرفی شده است. Imagenet دیتاستی از بالای ۱۵ میلیون عکس باکیفیت است که از ۲۲۰۰۰ گروه شئ می باشند که توسط انسان لیبل زده شده اند. ILSVRC زیرمجموعه ای از این داده ها است که ۱۰۰۰ کلاس عکس دارد و در هر گروه هم تقریبا ۱۰۰۰ تا عکس وجود دارد و در مجموع ۱.۲ میلیون عکس با کیفیت برای یادگیری و ۵۰۰۰۰ تا برای validation و ۱۵۰۰۰۰ تا برای تست است. عکس ها down-sampled شده اند و ۲۵۶*۲۵۶ شده اند.



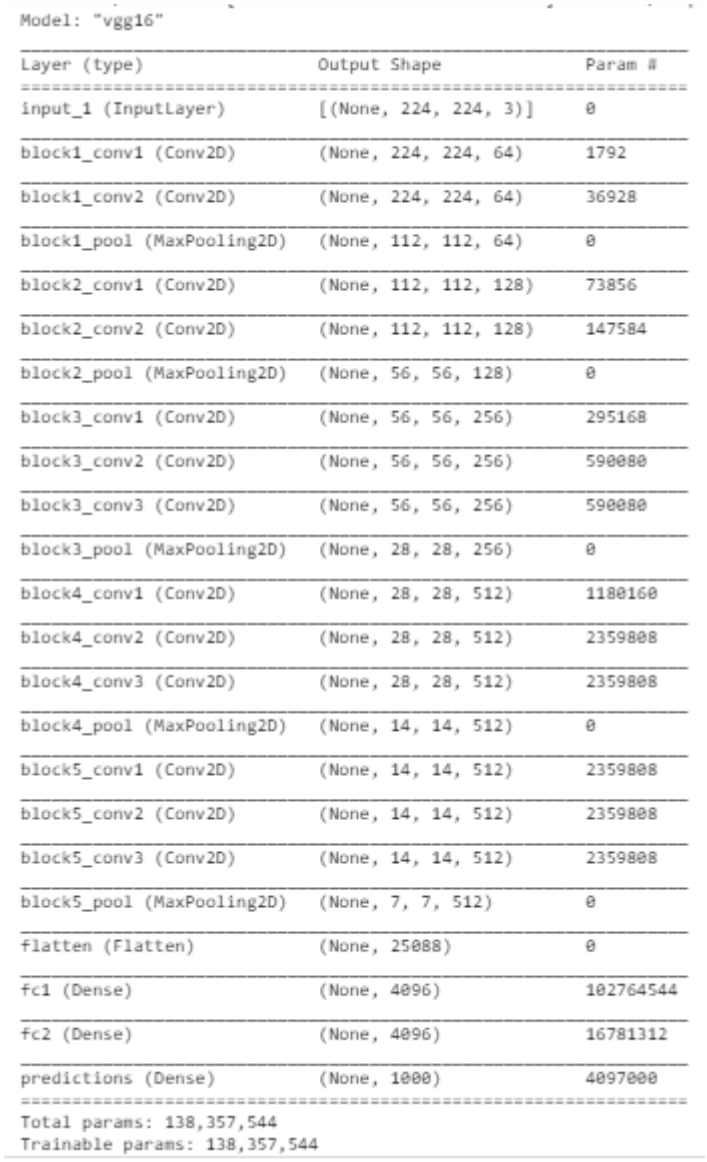
شکل ۴-۱ نمونه از عکس های موجود در ILSVRC

این مدل دو نوع دارد یک نوع با ۱۶ لایه و نوع دیگر با ۱۹ لایه که ما از ۱۶ لایه استفاده می کنیم.



شکل ۴-۲ معماری شبکه VGG16

همانطور که در شکل بالا مشخص است ورودی این شبکه به صورت عکس ۲۲۴*۲۲۴ با فرمت RGB است و معماری مشابه بالا دارد. فیلتر های لایه های convolution ۳*۳ اند(یک لایه ۱*۱) با stride یک و ۵ لایه max-pooling با پنجره های ۲*۲ داریم با stride دو و همچنین ۳ تا لایه fully-connected که به یک softmax ختم می شوند این شبکه را تشکیل می دهند. تابع فعال ساز نیز ReLU است و local normal response نیز نداریم. حجم این مدل بالای 500mb است زیرا تعداد پارامتر های آن زیاد است و مقادیر وزن ها نیز بزرگ اند همچنین زمان یادگیری این مدل خیلی طولانی است و می تواند بیشتر از یک هفته طول بکشد. به کمک دستور summary مشخصات مدل را بررسی می کنیم که در شکل زیر می توان این جزییات را دید:



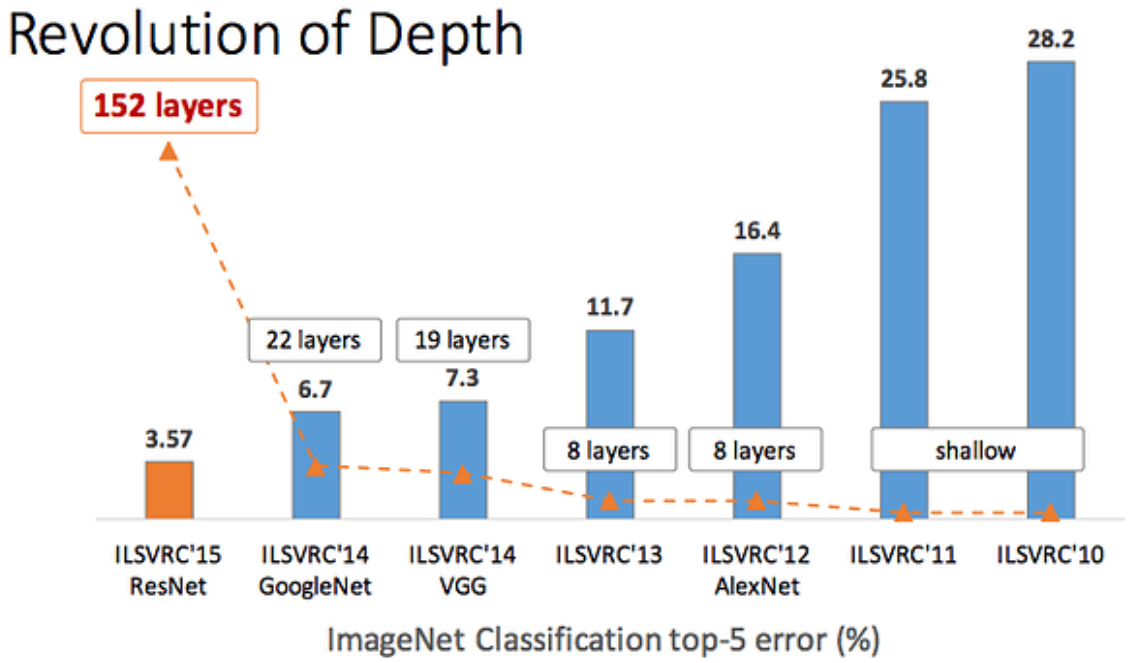
```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
```

شکل ۴-۳ جزییات مدل VGG16

می توان دید که این مدل ۱۳۸ میلیون پارامتر دارد که بزرگی و پیچیدگی آن را نشان می دهد همچنین می توان دید لایه خروجی ۱۰۰۰ تایی است که مقادیر احتمال برای ۱۰۰۰ دسته و category این دیتاست است و نشان می دهد که احتمال اینکه عکس ورودی لیبلی برابر هر یک از دسته ها داشته باشد چقدر است و به کمک دستور label می توانیم احتمال هر یک را چک کنیم. تصویر ورودی نیاز به پیش پردازش هایی دارد که اصلی ترین آن rescale کردن یا crop کردن تصویر اصلی و تبدیل آن به یک تصویر ۲۲۴*۲۲۴ است که ورودی استاندارد این شبکه است البته می توانستیم هنگام load کردن شبکه سایز ورودی را تغییر دهیم ولی پیش پردازش گفته شده روشی ساده تر و بهتر است. همچنین از preprocess_input که تابعی است از کتابخانه keras.applications نیز استفاده می کنیم که فرمت RGB تصویر را به BGR تبدیل کرده و همچنین مقدار هر کانال رنگ را طوری شیفت می دهد که zero-centered

باشند یعنی میانگین یک رنگ در عکس را کم می کند از کل پیکسل ها ولی اسکیل انجام نمی شود. علت این پیش پردازش این است که تصویر مدنظر را به شکل ورودی استانداردی که شبکه با آن آموزش دیده شده است در بیاوریم.

بزرگترین مزیت VGG16 این است که با اینکه جزو پیچیده ترین مدل های روز نیست اما درصد بسیار خوبی می گیرد که به معماری آن بر می گردد و پیاده سازی و استفاده از آن به عنوان مدلی برای transfer learning بسیار ساده است و به کمک keras می توان از فیلترهای آن استفاده کرد و با اعمال تغییرات لازم آن را متناسب با مساله موردنظر تغییر داد. همچنین خیلی preprocess خاصی نمی خواهد که مزیت بزرگی است ولی از معایب ان یادگیری طولانی و حجم بالای آن است که مشکل بزرگ شبکه های deep است و بزرگترین مانع برای بالا بردن پیچیدگی مدل های ما است. همچنین inference time آن زیاد است و به علت حجیم بودن کند است.



شکل ۴-۴ عملکرد برخی شبکه های عمیق به روز

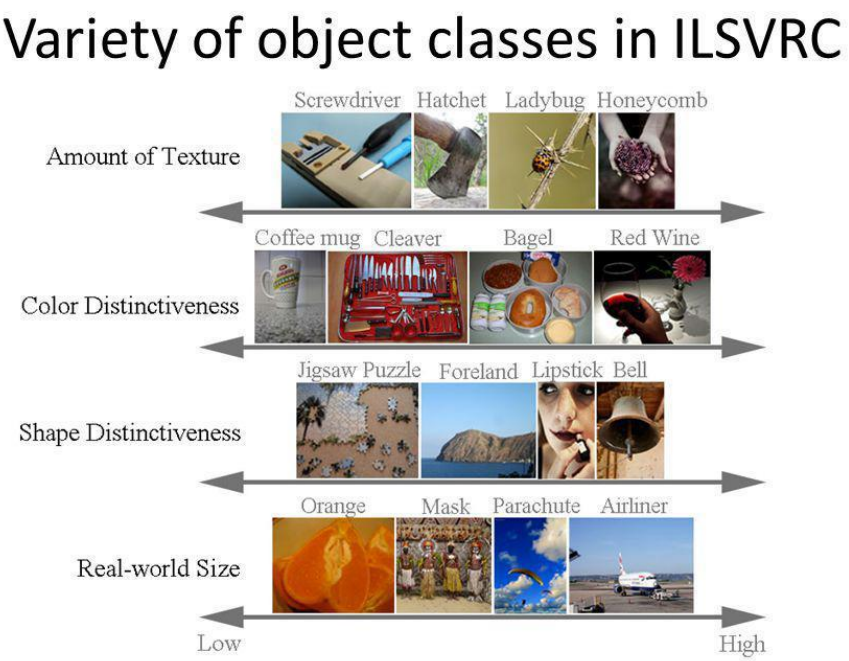
۲. عملیات transfer learning به پروسه ای گفته می شود که مدلی که برای یک مساله آموزش داده شده ست برای مساله ی دیگری استفاده می شود. این روش در deep learning به علت پیچیدگی، زمان بر بودن یادگیری و حجیم بودن مدل ها مرسوم است و از مدل هایی که روی مسایلی مشابه مساله فعلی train شده اند استفاده می شود. سود این روش زمان کمتر برای

training مساله دوم و کاهش خطای generalization است. سبک های مختلفی از transfer learning انجام می شوند از جمله کمک گرفتن از فیلترها و بعضی لایه های اولیه یا استفاده از وزن های مدل یادگیری شده به عنوان وزن های اولیه برای مدل جدید و ...

۳. از کتابخانه keras.applications مدل VGG16 را load می کنیم.

۴. مدل ما توانایی شناسایی انواع مختلفی از اشیا و حیوانات است که لیست کامل آنها در لینک زیر موجود است:

https://gist.github.com/xkumiyu/dd200f3f51986888c9151df4f2a9ef30



شکل ۴-۵ گوناگونی کلاس های مختلف ILSVRC

۵. حال عکسی به این مدل می دهیم تا شناسایی کندو عکس مورد نظر عکس یک ماگ است که جزو دسته های قابل تشخیص برای VGG است.

شکل ۴-۶ عکس ورودی مدل

قابل مشاهده است که عکس داده شده از زاویه ای نسبتا سخت گرفته شده و نورپردازی نیز اندکی مشکل را زیاد می کند همچنین نوشته های روی ماگ و سایه آن می توانند کار را سخت تر هم بکنند اما مدل از پس classification بر می آید و نتیجه پیش بینی آن به شرح زیر است:

```
cup (27.59%)
coffee_mug (27.45%)
measuring_cup (12.53%)
```

می توان دید که لیوان و ماگ قهوه به درستی بیشترین احتمال را گرفته اند و measuring cup نیز خیلی جواب بدی نبوده است و کلا مدل خیلی خوب عمل کرده است.