به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه ۱

نام و نام خانوادگی	زهرا نصرالهی-فاطمه سلیقه
شماره دانشجویی	۸۱۰۱۹۸۳۰۶-۸۱۰۱۹۸۳۲۶
تاریخ ارسال گزارش	۹۹/۰۲/۳

فهرست گزارش

بخش تشریحی

(۱)

برای مسائل regression از mean error استفاده می کنیم که اختلاف بین مقادیر واقعی و مقادیر پیش بینی شده را محاسبه می کند و در اخر بین آنها میانگین می گیرد . اما مشکلی که وجود دارد این است که برخی از این اختلاف ها دارای مقادیر منفی و برخی مثبت هستند و جمع آن ها باعث خنثی شدن می شود و حتی ممکن است خطا را صفر کند .بنابراین از دو روش Mean Squared Error و Mean Absolute Error استفاده می کنیم . در MSE خطا ها را به توان دو می رسانیم اما در MAE قدر مطلق می گیریم . از MSE معمولا در مواردی استفاده می کنیم که داده دارای داده پرت باشد زیرا در MSE خطا ها که بیشتر از یک هستند وقتی به توان دو می رسند مقدار خطا را بیشتر نشان می دهند و اگر خطا کم باشد یعنی زیر بین ۰و۱ آنگاه خطا کوچکتر می شود . اما در حالتی که داده پرت نداریم بهتر است از همان MAE استفاده کنیم .

در مسائل classification از روش cross entropy استفاده می کنیم . در این روش اگر توزیع احتمال واقعی کلاس ها به صورت [x1,x2,…,xN] باشد و توزیع احتمال پیش بینی شده به صورت [y1,y2,…,yN] باشند آنگاه cross entropy به صورت زیر محاسبه می شود :

$$cross\ entropy = -(x1 \log y1 + \cdots + xN \log yN)$$

که برای هر ورودی محاسبه می شود . حال در صورتی که چند کلاس داشته باشیم از categorical cross entropy و در صورتی که یک خروجی داشته باشیم از binary cross entropy استفاده می کنیم .

$$categorical\ cross\ entropy = \frac{sum\ of\ cross\ entropy\ for\ N\ data}{N}$$

(۲)

روش های بهینه سازی محاسباتی ، از مجموعه شرایط لازم و کافی که در جواب مسأله بهینه‌سازی صدق می‌کند، استفاده می‌کنند. وجود یا عدم وجود محدودیت‌های بهینه‌سازی در این روش‌ها نقش اساسی دارد. به همین علت، این روش‌ها به دو دسته روش‌های با محدودیت و بی‌محدودیت تقسیم می‌شوند.

روش‌های بهینه‌سازی بی‌محدودیت با توجه به تعداد متغیرها شامل بهینه‌سازی توابع یک متغیره و چند متغیره می‌باشند.

روش‌های بهینه‌سازی توابع یک متغیره، به سه دسته روش‌های مرتبه صفر، مرتبه اول و مرتبه دوم تقسیم می‌شوند. روش‌های مرتبه صفر فقط به محاسبه تابع هدف در نقاط مختلف نیاز دارد؛ اما روش‌های مرتبه اول از تابع هدف و مشتق آن و روش‌های مرتبه دوم از تابع هدف و مشتق اول و دوم آن استفاده می‌کنند. در روش مرتبه اول با شرط صفر شدن مشتق مرتبه یک، نقطه ی بهینه ماکزیمم یا مینیمم را به دست می آورند. در روش مرتبه دوم با شرط بزرگتر از صفر شدن مشتق مرتبه دوم، مینیمم موضعی یک نقطه را به دست می آورند. معمولا با استفاده از شرط لازم مرتبه اول، یک یا چند نقطه به عنوان کاندید جواب به دست می آید و با استفاده از شرط لازم مرتبه دوم، این جواب ها پالایش می شوند و مینیمم بون برخی از آن ها رد می شود. در نهایت برای نقاط باقی مانده، شرط کافی را بررسی کرده و نقاط مینیمم موضعی به دست می آید.

۳)

Overfitting زمانی اتفاق می افتد که مدل ما در هنگام یادگیری نسبت به زمان تست خیلی بهتر عمل کند . یعنی داده های آموزش را خوب یادگرفته اما اگر داده جدید به آن بدهیم نمی تواند خوب جواب دهد . روش های جلوگیری از overfitting :

۱) می توان از cross validation استفاده نمود به این صورت که با استفاده از k fold cross validation داده ها را به k بخش تقسیم می کنیم و k بار مدل را اجرا نموده و هر بار یکی از k بخش را به عنوان تست و k-1 بخش دیگر را برای آموزش استفاده می کنیم . در این صورت می توان کل داده ها را دید . اما اگر داده ای جدید ببیند که در داده های ما نباشد به مشکل می خوریم .

۲) می توان تعداد داده های آموزش را افزایش داد . با استفاده از نمونه گیری مجدد یا با استفاده از روش هایی چون Data Augmentation .

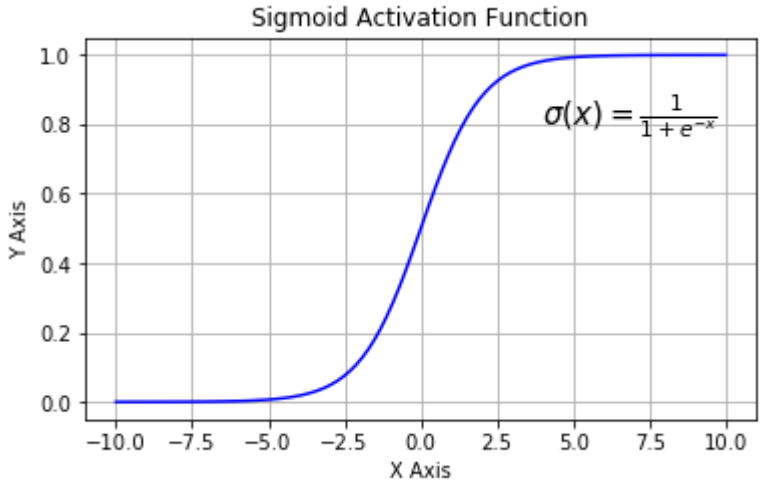
۳) روش دیگر استفاده از روش Early Stopping است . در نمودار loss مربوط به داده های تست و آموزش می بینیم که نمودار loss مربوط به آموزش به صورت کاهشی است و نمودار تست هم تا یک جایی کاهشی است ولی در یک نقطه شروع به افزایش می کند . می توان در همان نقطه یادگیری را پایان داد .

۴) می توان از dropout استفاده کرد . به این صورت عمل می شود که برخی نورون ها را با احتمال مشخص نادیده می گیرد .

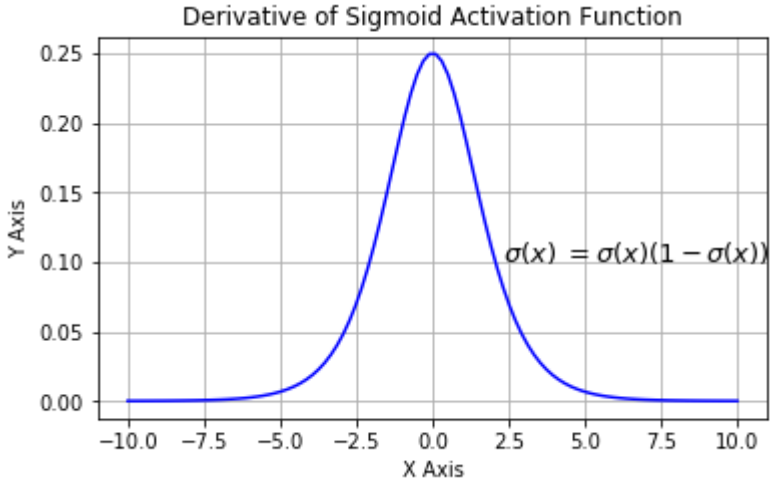
(۴

شبکه های عصبی برای پیاده سازی توابع پیچیده استفاده می شوند و توابع فعال سازی غیر خطی آنها را قادر می سازد تا توابع مختلط مختلط را تقریب آورند. با استفاده از توابع خطی و بدون تابع فعال سازی غیر خطی، چند لایه از یک شبکه عصبی معادل یک شبکه عصبی تک لایه است. در واقع ترکیب خطی چند خط، یک خط است. پس می توان لایه ها را با یکدیگر جمع کرد و یک لایه به دست آورد و همان طور که می دانیم شبکه عصبی تک لایه نمی تواند مسائل پیچیده را به خوبی طبقه بندی کند. در شبکه های پیچیده ما نیاز داریم که بین ورودی و خروجی یک تابع mapping غیر خطی داشته باشیم.

تابع sigmoid: این تابع هم یک ورودیِ منفیِ بینهایت تا مثبت بی‌نهایت را به یک بازه‌ی ۰ تا ۱ تبدیل می‌کند. این تابع غیرخطی، یکنواخت، پیوسته و مشتق پذیر است. این تابع خروجی آنالوگ تولید میکند و شیب نرمی دارد، از همین رو این تابع برای استفاده در مسائل طبقه بندی کاربرد دارد. این تابع به مقادیر بسیار کم یا بسیار زیادی ورودی واکنش درخوری نشان نمی دهد و شیب را از بین میبرد که به این مساله vanishing gradientنیز گفته می شود. به همین دلیل این تابع همگرایی کندی دارد.

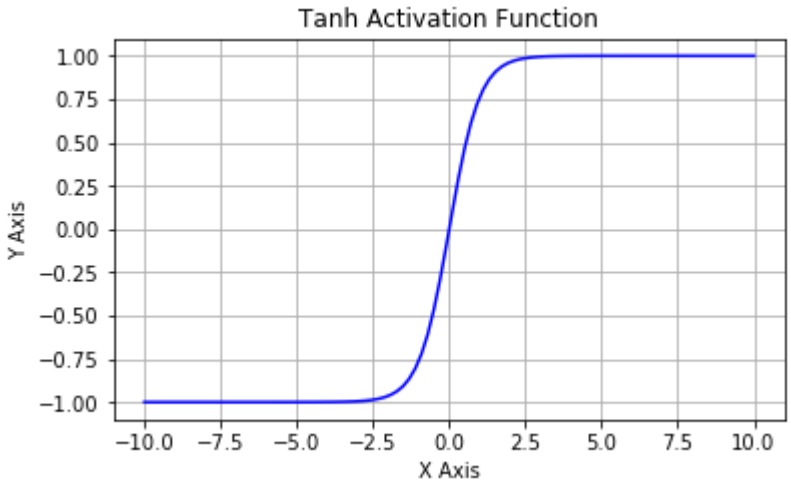


شکل ۱. منحنی **sigmoid**

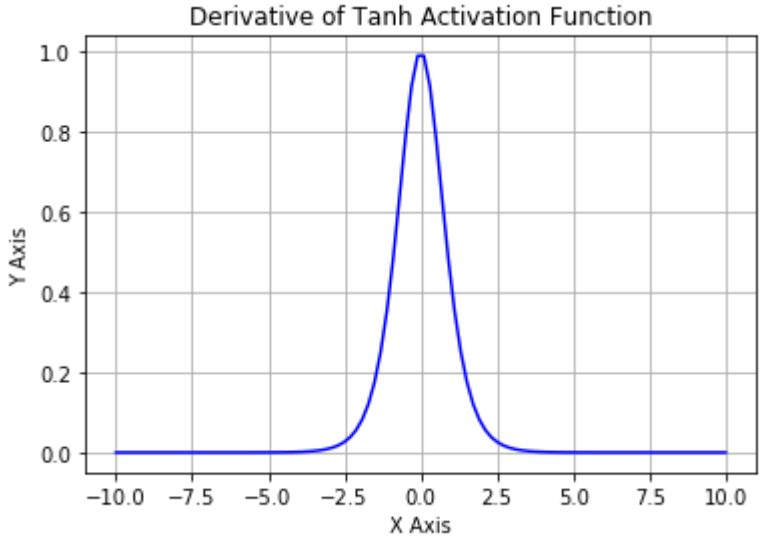


شکل ۲. منحنی مشتق **sigmoid**

تابع tanh: شیب های تندتر مشتقات آن و همچنین خروجی های این تابع مرکزیت ۰ دارند بر تابع قبلی مزیت دارد که باعث می شود برای مدل هایی که ورودیها مقادیر مثبت یا منفی شدیدی دارند، مناسب تر باشد. این تابع و مشتقاتش هردو یکنواخت هستند. تصمیم برای انتخاب بین sigmoid و tanh به میزان قوی بودن شیب در تابع بسته به مسئله ی پیش روی ما بستگی دارد. بر خلاف سیگموئید، خروجی های tanhصفر محور هستند، زیرا محدوده بین -۱ و ۱ است. یک تابع tan به عنوان دو سیگموئید با هم قرار گرفته اند. در عمل، tanh بیش از sigmoid ترجیح داده است. ورودی های منفی که به عنوان منفی بسیار منفی، ارزش ورودی صفر به صفر نزدیک می شوند و ورودی های مثبت در نظر گرفته شده اند هم چنین دارای مشکل vanishing gradient است.

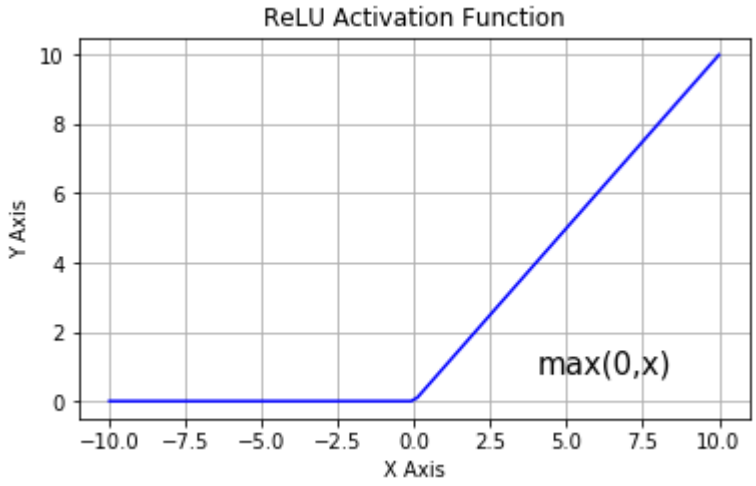


شکل ۳.منحنی **tanh**

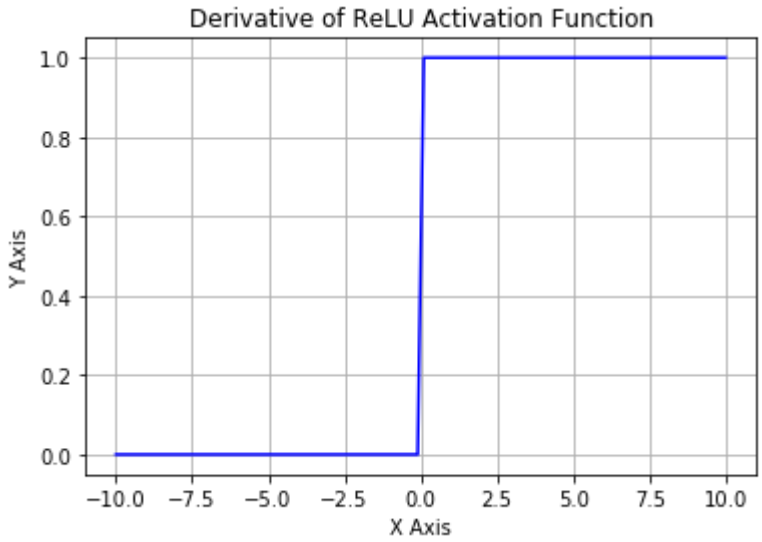


شکل ۴.منحنی مشتق **tanh**

تابع relu: این تابع برای داده های منفی ، عدد صفر رو به عنوان خروجی بر می گرداند (به عبارتی از نورون عبور نمی کنند) و برای داده های مثبت دقیقا خود عدد را بر می گرداند. این تابع هزینه ی محاسباتی کمی دارد و همچنین مشکل vanishing gradient را ندارد.



شکل ۵. منحنی **Relu**



شکل ۶. منحنی مشتق **Relu**

نقاط ضعف ReLU وجود دارد:

صفر محور: خروجی ها صفر محور مانند تابع فعال سازی sigmoid نیستند.

مسئله دیگر با ReLU این است که اگر $x < 0$ در طول پاس رو به جلو، نورون غیر فعال باقی بماند و در طول گذر عقب، گرادیان را از بین می برد. بنابراین وزن ها به روز نمی شوند و شبکه یاد نمی گیرد.

هنگامی که $x = 0$ ، شیب در آن نقطه تعریف نشده است، اما این مشکل در حین اجرای با انتخاب چارچوب یا شیب سمت راست مورد توجه قرار گرفته است.

۵)

با استفاده از Data Augmentation می توان تعداد داده های آموزش را افزایش داد که این کار با استفاده از ویرایش داده های آموزش انجام می شود .در واقع می توان با تغییر ویژگی های هر داده داده های جدید تولید کرد . استفاده از روش های augmentation باعث ایجاد یک مدل قوی می شود که به دلیل مشاهده داده های زیاد امکان generalization را فراهم می کند و درصورت مشاهده داده جدید بهتر می تواند عمل کند . augmentation می تواند به روش هایی مانند CNNکمک کند تا ویژگی هایی را که تنوع کمی دارند را بهتر یاد بگیرد . در نتیجه به generalization کمک می کند .

۶)

Batch Normalization: می توان تکنیکی برای بهبود عملکرد و پایداری شبکه های عصبی دانست.

ایده این کار نرمال سازی ورودی هر لایه با کمک میانگین و واریانس می باشد. به شکلی که میانگین اعداد بردار نهایی برابر با صفر و واریانس آن ها برابر با یک شود.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

نرمال سازی ورودی شبکه به Learn شدن شبکه کمک می کند. شبکه یک سریال از لایه ها می باشد و خروجی هر لایه هم ورودی لایه بعدی می باشد، پس هر لایه می تواند برای ما حکم لایه اول یک زیر شبکه کوچک تر را داشته باشد.

مزایای استفاده از Batch Normalization

تجمع داده ها در مبدا :

با فرمول ذکر شده میانگین داده های هر مینی بچ را صفر میکنیم، با اینکار داده ها به سمت مبدا حرکت می کنند. همچنین تغییر واریانس داده ها به یک هم موجب توزیع یکنواخت داده ها در همان قسمت می شود. تصاویر زیر نحوه تغییر ماتریس داده ها را بعد از اعمال میانگین گیری و سپس واریانس، نمایش میدهد:

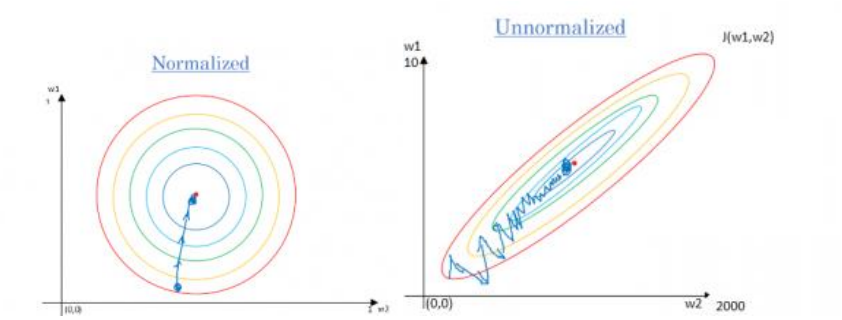
با نرمال سازی داده ها، Lrدر جهت همه ویژگی ها برابر می شود پس در نتیجه یادگیری و کم کردن نرخ خطا با سرعت بیشتری انجام می شود.

افزایش سرعت شبکه :

در حالی که training iteration به علت اضافه شدن سربار محاسباتی نرمال سازی مدت زمان بیشتری را صرف می کند، اما چون به طور معمول به همگرایی زودتر شبکه کمک می کند، پس در مجموع موجب افزایش سرعت خواهد شد.

ممکن شدن انتخاب Learning Rate بزرگتر:

با توجه به نمودار ها، با نرمال سازی می توانیم گام های بزرگتری به سمت هدف برداریم



شکل ۷. **unnormalized & normalized**

نحوه Learn شبکه به کمک Batch Normalization و هایپر پارامتر های اضافه شده:

با اضافه شدن پارامتر های گاما و بتا، شبکه می تواند هر پارامتر ورودی را با مقدار قبل نرمال سازی بازیابی کند. در هر لایه می توان نرمالسازی را روی خروجی لایه قبل انجام داد و در لایه بعد، به عنوان ورودی از آن استفاده کرد.

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

بخش عملی

الف)

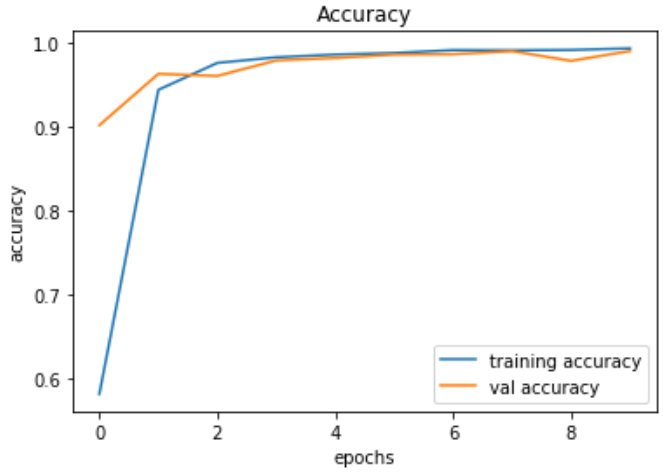
مشخصات شبکه

تعداد فیلتر در لایه ۱	۳۲
اندازه پنجره conv در لایه ۱	۳*۳
اندازه stride در لایه ۱	1*1
تابع فعال ساز در لایه ۱	relu
Pooling در لایه ۱	Max_pooling با سایز ۲*۲
تعداد فیلتر در لایه۲	۶۴
اندازه پنجره conv در لایه ۲	۳*۳
اندازه stride در لایه ۲	1*1
تابع فعال ساز در لایه ۲	relu
Pooling در لایه ۲	Max_pooling با سایز ۲*۲
تعداد فیلتر در لایه ۳	۶۴
اندازه پنجره conv در لایه ۳	۳*۳
اندازه stride در لایه ۳	1*1
تابع فعال ساز در لایه ۳	relu
Pooling در لایه ۳	Max_pooling با سایز ۲*۲
تعداد لایه های fully connected	۲
تعداد نرون ها در لایه ۱	۲۵۶
تابع فعال ساز در لایه ۱	relu
تعداد نرون ها در لایه ۲	تعداد کلاس ها (43)
تابع فعال ساز در لایه ۲	softmax
تابع هزینه	categorical_crossentropy

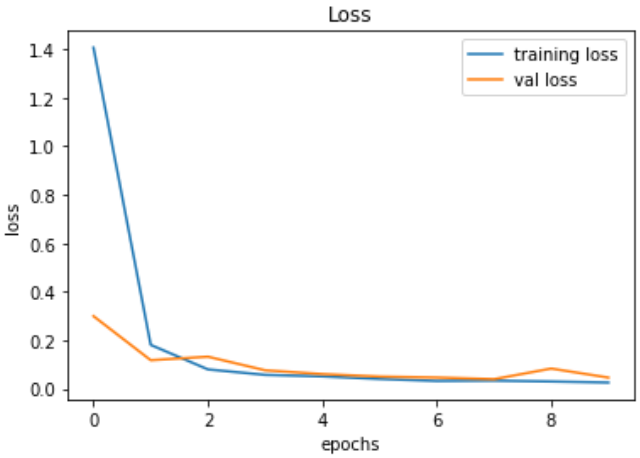
تابع بهینه ساز	Adam
اندازه mini batch	۳۲
تعداد epoch	۱۰

ب)

نمودار Accuracy و loss برای داده های train و validation:



شکل ۸. نمودار تغییرات دقت شبکه در هر اپاک



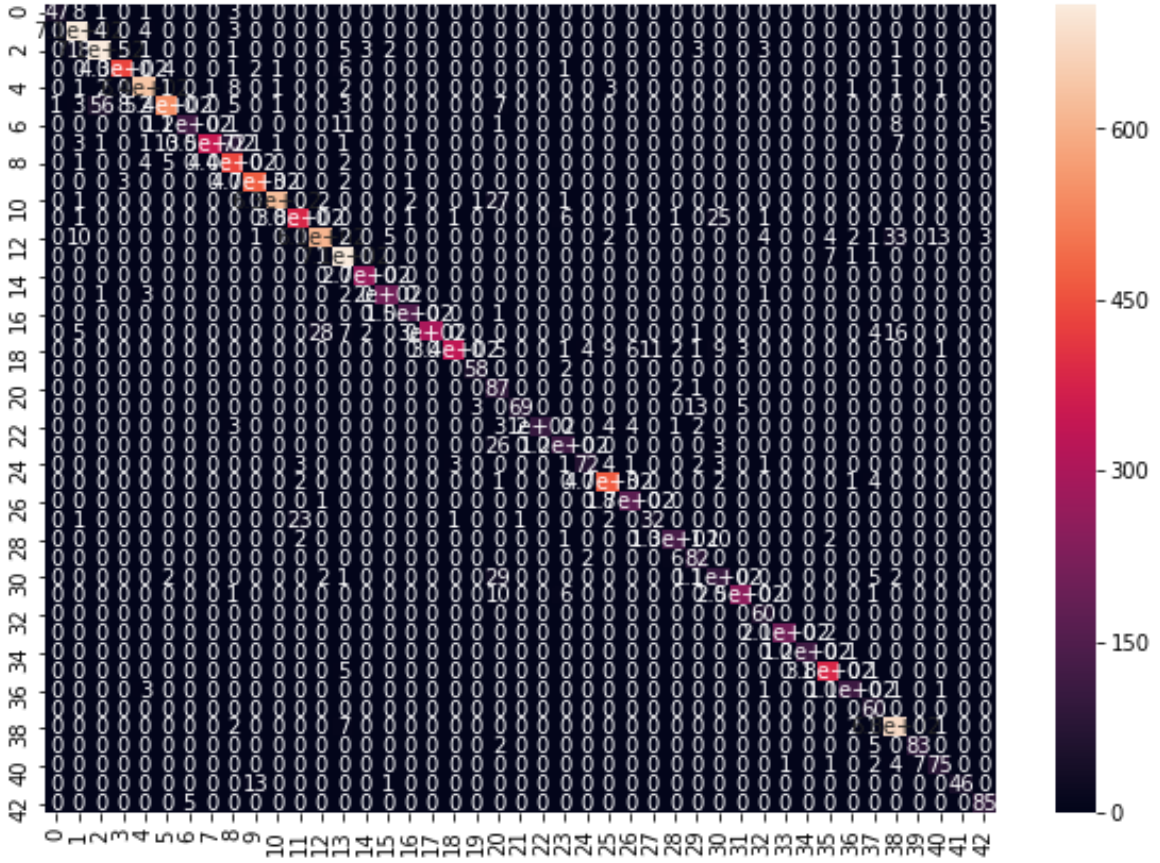
شکل ۹. نمودار تغییرات خطا شبکه در هر اپاک

مقادیر accuracy و loss به دست آمده برای داده های تست:

```
loss = 0.40612689601721996 acc = 0.93056214
```

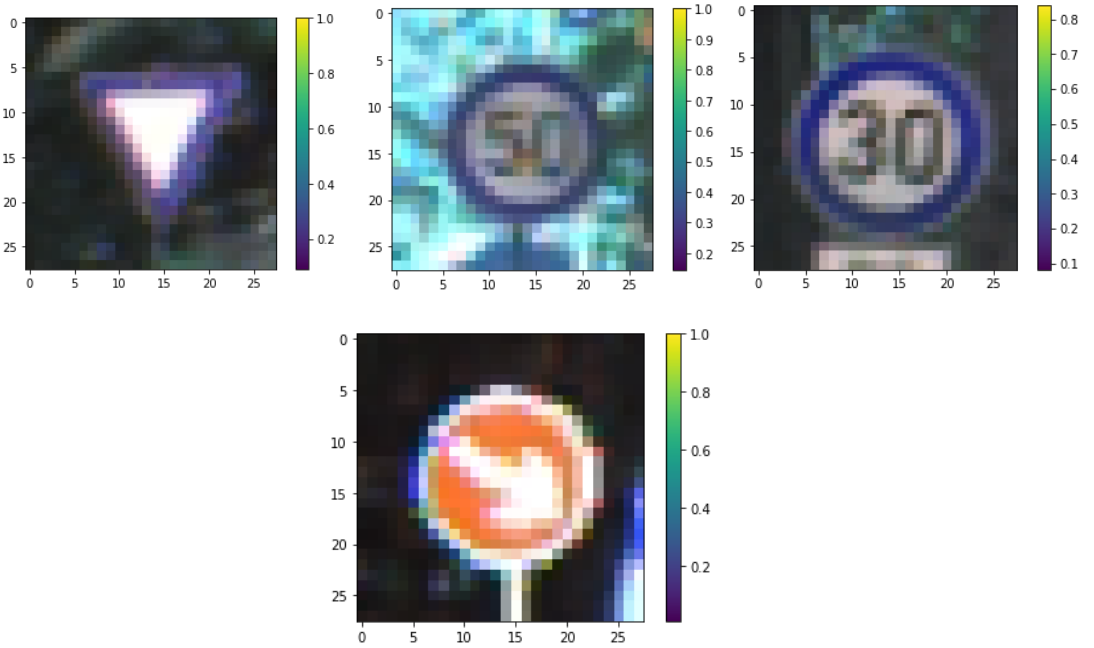
پ)

ماتریس آشفتگی به صورت زیر است . همان طور که می بینیم مواردی که به درستی تشخیص داده شده است بالا می باشد و دقت 0.92 را دارد .

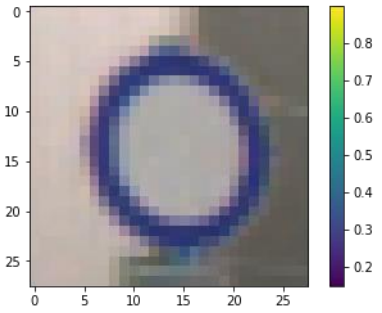


شکل ۱۰. ماتریس آشفتگی

همانطور که می بینیم تابلوهایی که درست تشخیص داده موارد زیر هستند:



اما از مواردی که به خوبی تشخیص داده نشده :



یکی از دلایل آن است که موارد بالا که خوب تشخیص داده شده بودند تعداد بیشتری نسب به تابلویی که خوب تشخیص داده نشده است را داشتند . یکی دیگر هم آن است که مثلا تابلوی مثلثی انواع کمتری دارند . یعنی بیشتر تابلو ها دایره هستند .

ت)

شبکه طراحی شده را برای سه تابع فعال‌ساز Relu و tanh و sigmoid و در ۱۵ اپاک اجرا نموده و نتایج برای داده های تست به صورت زیر است :

تابع فعال ساز RELU :

```
loss =  0.44762764169827 36 acc =  0.92335707
```

تابع فعال ساز tanh :

```
loss =  0.20616927139985694 acc =  0.9524149
```

تابع فعال ساز sigmoid :

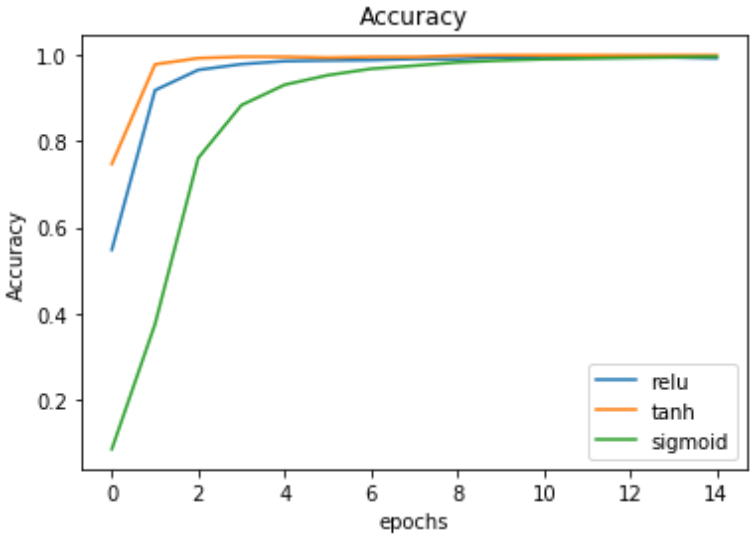
```
loss =  0.3902015986276447 acc =  0.9055424
```

نمودارهای زیر تغییرات Accuracy و lossرابرای هر سه تابع فعال‌ساز در حالت آموزش و validation نشان می دهد . همان طور که از نمودار val_Accuracy مشاهده می کنیم تابع tanh از همه بهتر است سپس Relu و پس از آن sigmoid است . tanh و Relu بسیار نزدیک به هم عمل می کنند اما sigmoid در چند epoch اول بدتر عمل می کند و تقریبا از اپاک ۹ به آنها نزدیک می شود. در مسائل مختلف معمولا تابع Relu بهتر عمل می کند . زیرا در شبکه های خیلی عمیق در هنگام back propagation گرادیان ها در لایه های اولیه به صفر میل می کنند و یادگیری در لایه های اولیه کم می شود . چون Relu این مشکل را حل می کند بهتر عمل می کند اما در شبکه ی ما چون تعداد لایه ها کم است Relu نسبت به بقیه برتری ندارد .

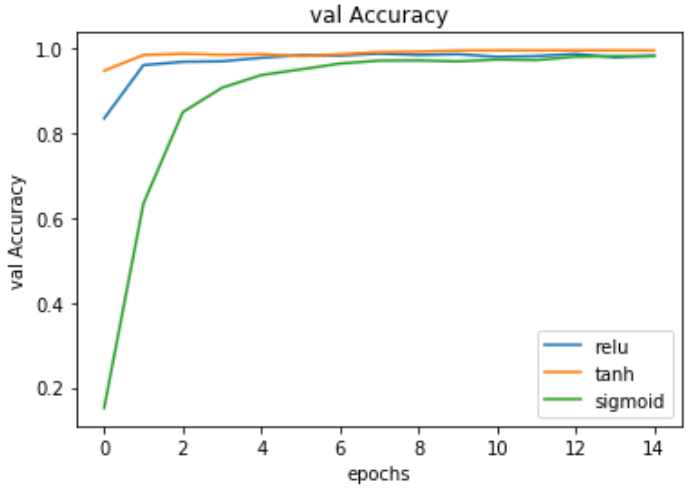
هم چنین موارد زیر نیز ممکن است به وجود آمده باشد که tanh کمی بهتر عمل کند :

۱. ممکن است که وزن ها در هر ایپاک مثبت بشن و relu ان ها را عبور بدهد و هر بار این وزن ها بزرگ و بزرگتر میشوند و این باعث میشه تغییرات وزن ها زیاد باشه و شبکه به خوبی همگرا نشه.
۲. ممکنه وزنا توی آپدیت کوچیک بشن و کلا نورون خاموش بشه و چون بیشتر از صفرو عبور نمیده و تا آخر دیگه این نورون خاموش می ماند.

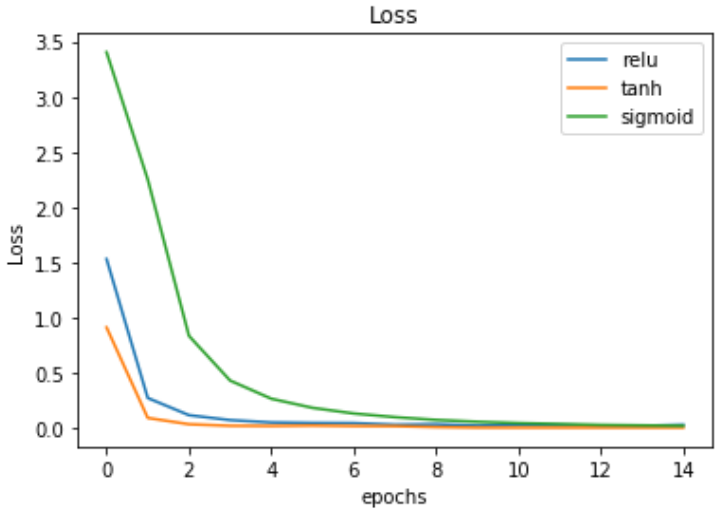
دلیل اینکه sigmoid نسبت به tanh دیرتر همگرا می شود این است که مشتق های تابع tanh بزرگتر از sigmoid هستند .



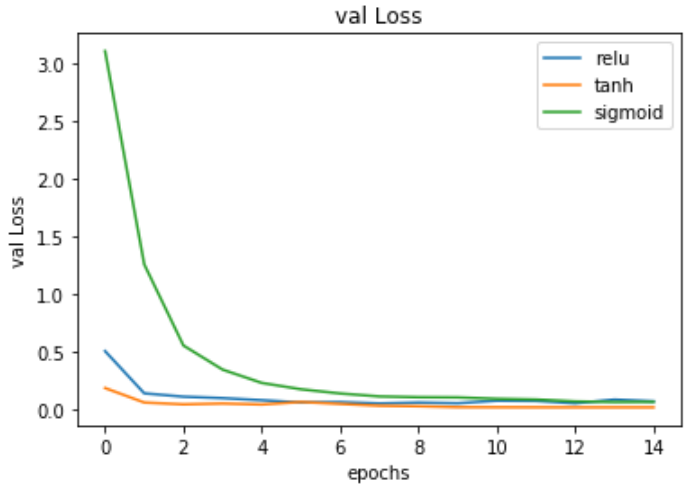
شکل ۱۱.منحنی تغییرات دقت شبکه در داده های **train** با سه نوع تابع فعال ساز



شکل ۱۲. منحنی تغییرات دقت شبکه در داده های **validation** با سه نوع فعال ساز



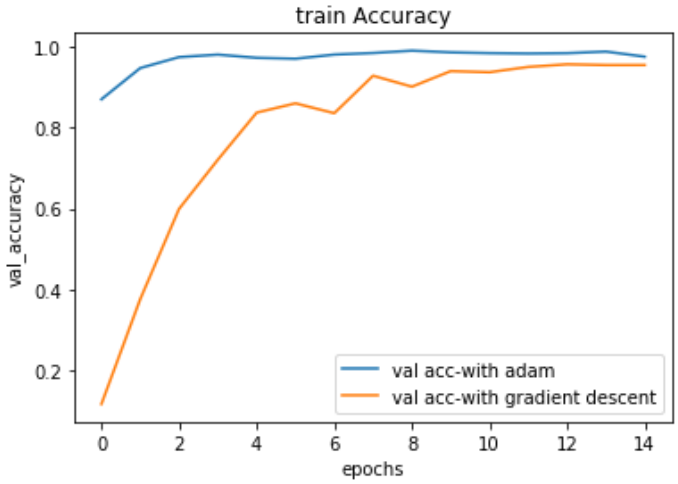
شکل ۱۳. منحنی تغییرات خطای شبکه در داده های **train**، با سه نوع تابع فعال ساز



شکل ۱۴. منحنی تغییرات شبکه در داده های **validation**، با سه نوع تابع فعال ساز

ث)

در ۱۵ اپاک شبکه را با بهینه ساز adam و بهینه ساز gradient descent به صورت مجزا آموزش دادیم و نمودار تغییرات آن ها در هر اپاک، برای داده های validation رسم کردیم که به صورت زیر می باشد:

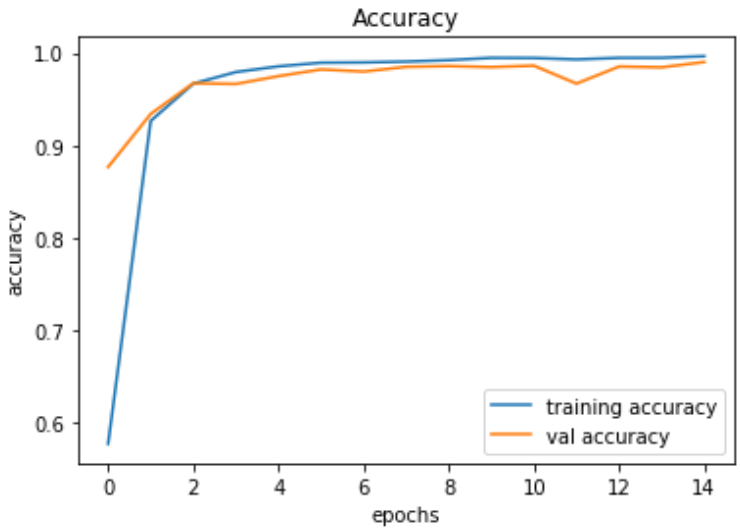


شکل ۱۵. منحنی تغییرات دقت شبکه در داده های **validation**، با بهینه ساز **adam** و **gradient descent** در هر اپاک

همان طور که در شکل دیده می شود، دقت در شبکه با بهینه ساز adam بهتر از بهینه ساز gradient descent بوده است. در منحنی gradient descent، در بعضی نقاط دچار نوسان شده است و چندین اپاک طول کشیده تا به دقت مطلوب برسد، در واقع شبکه دیرتر از شبکه با بهینه ساز Adam یاد گرفته است. اگر نرخ یادگیری را تغییر دهیم و زیاد تر کنیم، این نوسانات بیشتر شده و اگر کوچک تر کنیم، نیاز داریم که در اپاک های بیشتری شبکه را آموزش دهیم.

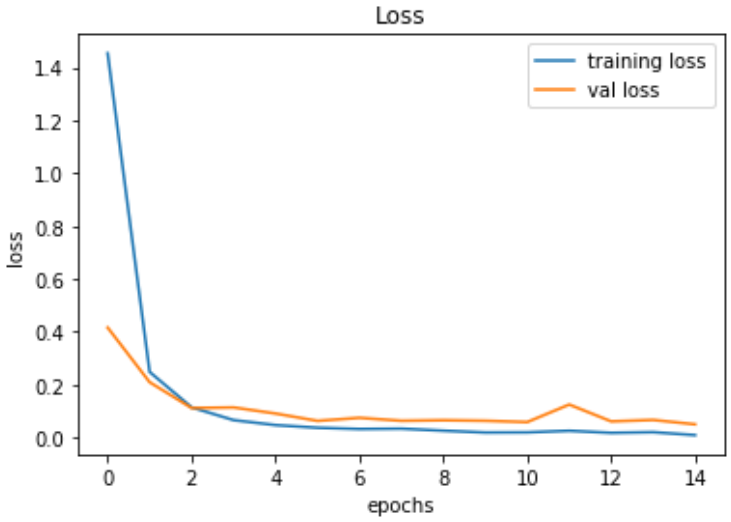
نمودار های زیر، تغییرات دقت و خطا را به صورت مجزا برای هر شبکه نشان داده است:

نمودار دقت با بهینه ساز adam در ۱۵ اپاک:



شکل ۱۶. نمودار تغییرات دقت با بهینه ساز **adam** در هر اپاک

نمودار خطا با بهینه ساز adam در ۱۵ اپاک:

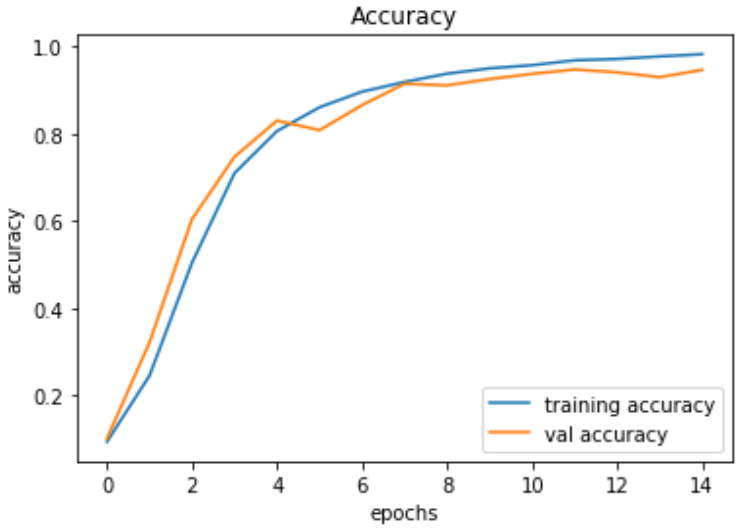


شکل ۱۷. نمودار تغییرات خطا با بهینه ساز **adam** در هر اپاک

دقت و خطای داده های تست با بهینه ساز adam :

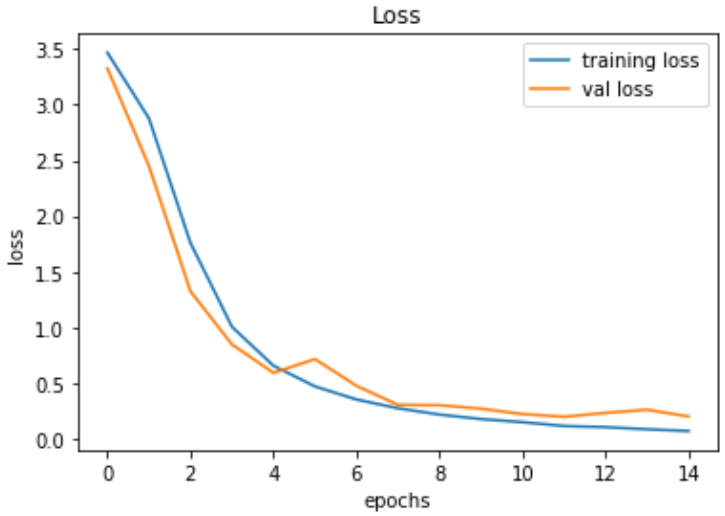
```
loss = 0.38829237675283773 acc = 0.94196355
```

نمودار دقت با بهینه ساز Gradient descent در ۱۵ اپاک:



شکل ۱۸. نمودار تغییرات دقت با بهینه ساز **gradient descent** در هر اپاک

نمودار خطا با بهینه ساز Gradient descent در ۱۵ اپاک:



شکل ۱۹. نمودار تغییرات خطا با بهینه ساز **gradient descent** در هر اپاک

دقت و خطای داده های تست با بهینه ساز Gradient descent:

```
loss = 1.045471629526345 acc = 0.8199525
```

در داده های تست نیز، بهینه ساز adam عملکرد بهتری نسبت به gradient descent داشته است.

در روش gradient descent با وجود این که الگوریتم یادگیری کم هزینه است اما مشکلات زیر را دارد:

- میزان نرخ یادگیری باید درست انتخاب شود، اگر خیلی کوچک باشد در local opt گیر می کند و اگر خیلی بزرگ باشد، دچار نوسانات می شود.
- مقدار نرخ یادگیری در تمام آپدیت های وزن ها ثابت است و تغییری نمی کند. در واقع تمام پارامتر ها مقدار دهی یکسانی دارد.
- ممکن است در نقاط زین اسبی گیر کند.
- به طور کلی شبکه های عصبی شامل توابع با پیچیدگی های مختلف هستند که شامل تعداد زیادی تبدیلات غیر خطی هستند .تابع loss یک نوع منحی محدب است که این امکان را دارد که شامل چندین نقطه Min باشد. با این شرایط با توجه به وجود چندین نقطه Min که شیب آن ها صفر است ، اگر با استفاده از روش Gradient Decent به یکی از این Minimum ها رسیده باشیم امکان دارد در این نقطه در Local Minimum بیفتیم .

ما در این پروژه از max pooling استفاده کرده ایم و در واقع در هر ناحیه افزار واحد با بیشترین فعالیت را استخراج می کند. جایی که نویز با طیف فرکانسی باز، با فراوانی و دامنه زیاد تمام سیگنال را در برخی لایه ها فراگرفته باشد ممکن است روش گرادیان فیلتر در کاهش هزینه،

بصورت محلی، دچار اشتباه شود و در یک بهینه محلی گیر کند. برای رفع این مشکل می توان از mean pooling استفاده کرد تا حساسیت نسبت به نویز از بین برود.

بهینه ساز adam مدل گسترش یافته ای از stochastic gradient descent است که مزایای زیر را داراست:

- کارآمدی محاسباتی دارد
- حافظه ی کمی را نیاز دارد.
- برای مسائلی که نوع داده ها زیاد است و یا پارامترهای زیادی دارد، مناسب می باشد
- مناسب برای اهداف غیر ثابت
- مناسب برای داده های noisy و یا گرادیان های خلوت
- در adam برای هر پارامتر، نرخ یادگیری مجزا دارد که کمک می کند در گرادیان های خلوت، عملکرد ارتقاع یابد.

تقریبا در بیشتر مسائل Adam نتایج مطلوب تری را نسبت به سایر بهینه سازها ایجاد می کند. قانون به روز رسانی برای Adam بر اساس برآورد لحظه اول (میانگین) و دومین لحظه خام از شیب های پیشین (در پنجره زمانی اخیر از طریق میانگین حرکت نمایی) تعیین می شود.

ج)

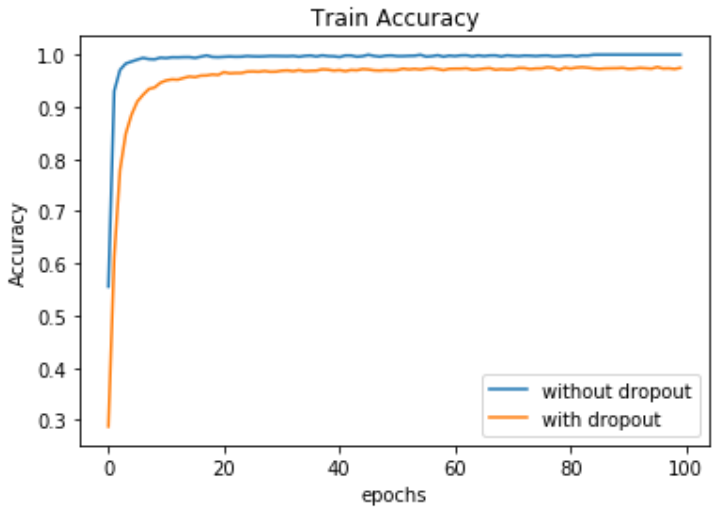
برای جلوگیری از overfit شدن شبکه های عمیق از Dropout استفاده می شود. به این صورت که برخی از نورون ها در هر اپاک با یک احتمالی نادیده گرفته می شوند و در واقع در یادگیری تاثیری نخواهند داشت .

همان طور که در نمودار های به دست آمده (شکل ۲۰) مشاهده می کنیم دقت در زمان یادگیری در شبکه بدون dropout بهتر از شبکه با dropout است. ازآنجایی که لایه های convolution تعداد پارامترهای کمی دارند نیاز کمی به regularization وجود دارد و هم چنین رابطه ای که بین نگاشت ویژگی ها کدشده است باعث میشود تا activation ها به شدت به هم وابسته شوند . بنابراین استفاده از dropout در لایه های convolution مناسب نیست و بهتر است در لایه های fully connected استفاده شود .

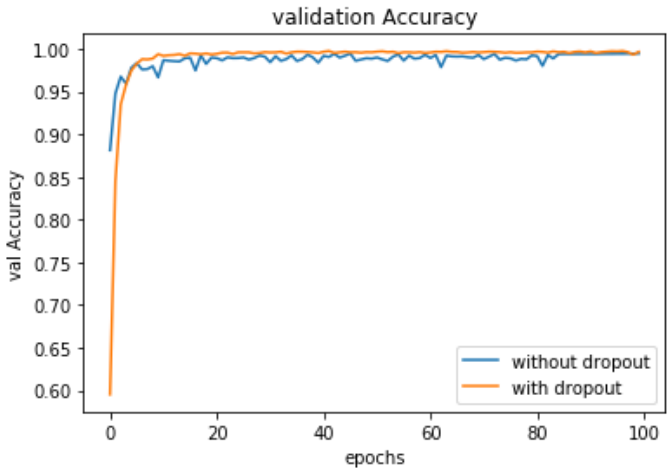
هم چنین بهتر است برای regularization از batch normalization استفاده کنیم .

اما در نمودار validation (شکل ۲۱) همان طور که مشاهده می کنیم دقت هر دو شبکه یکسان است زیرا در زمان validation تمام نورون ها موجود هستند و در واقع هیچ نورونی حذف نمی شود و هر دو

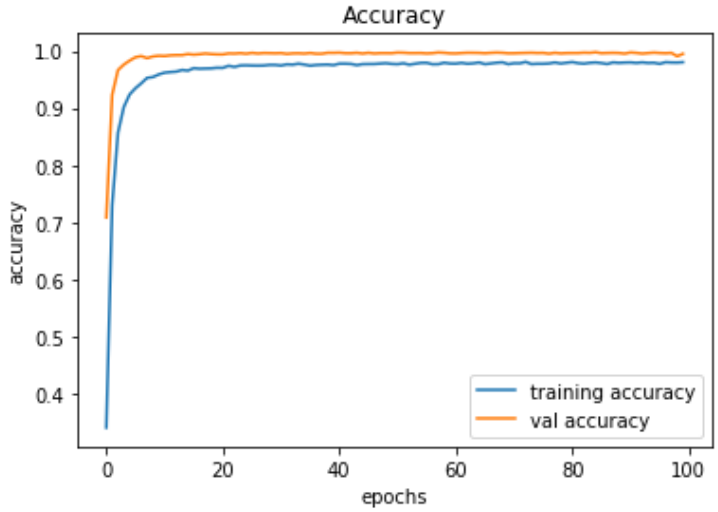
شبکه یکی هستند. و همان طور که در نمودار Accuracy (شکل ۲۲) می بینیم دقت validation بالاتر از train است و در نمودار loss (شکل ۲۳) هم می بینیم که loss داده validation کمتر از trainاست.



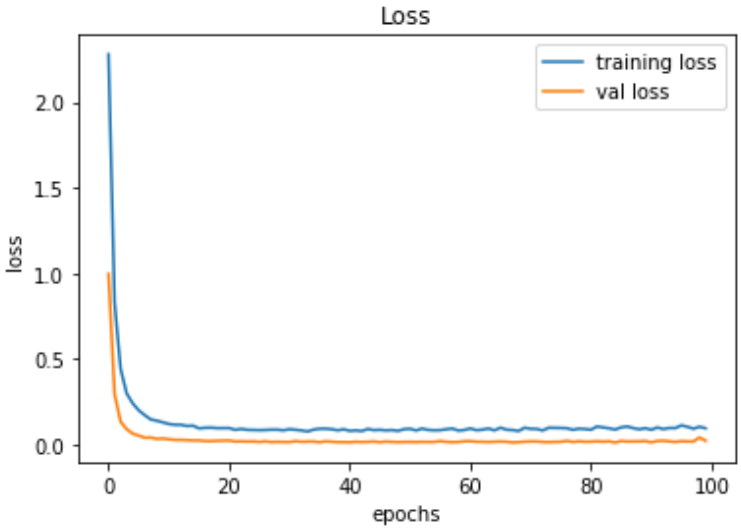
شکل ۲۰.نمودار دقت داده های **train** در حالت **dropout** و بدون آن در هر اپاک



شکل ۲۱. نمودار خطا داده های **train** در حالت **dropout** و بدون آن در هر اپاک

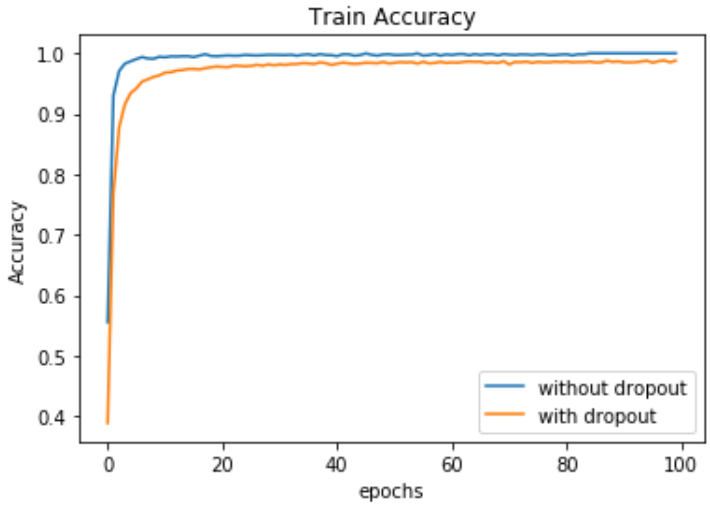


شکل ۲۲. نمودار تغییرات دقت در شبکه همراه با **dropout** در هر اپاک



شکل ۲۳. نمودار تغییرات خطا در شبکه همراه با **dropout** در هر اپاک

نکته دیگری که در ارتباط با dropout قبل از لایه خروجی با activation function از نوع softmax وجود دارد این است که وجود این dropout باعث میشود تا دقت یادگیری کمتر شود که این مورد نیز بررسی شد و نمودار زیر حاصل از شبکه بدون اخرین dropout اجرا شد :

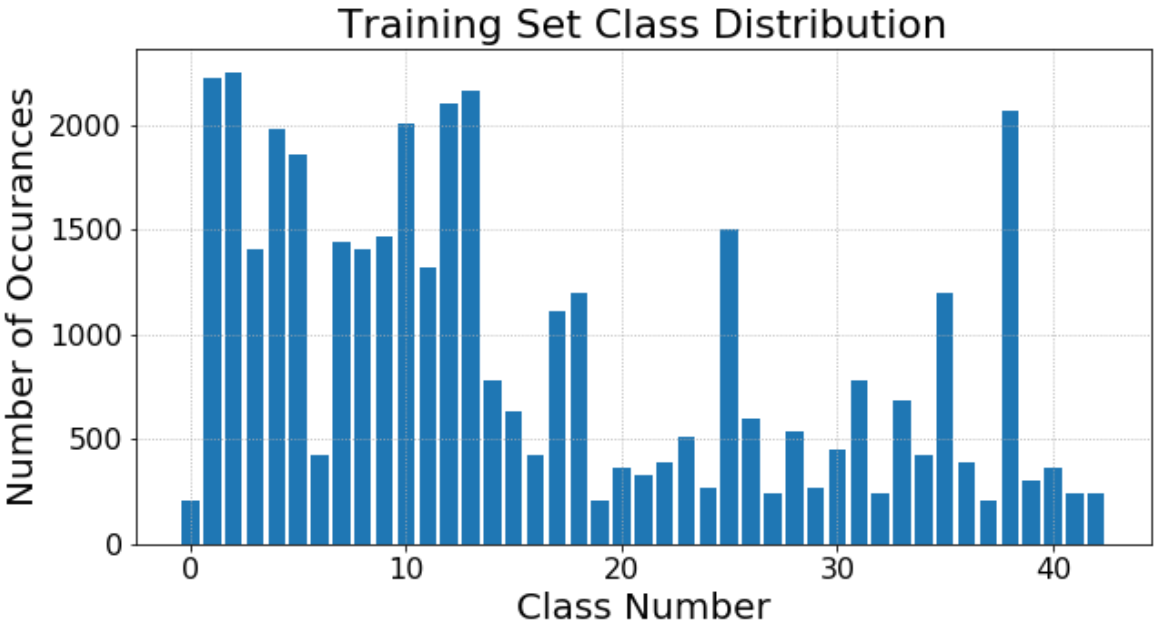


شکل ۲۴. نمودار دقت داده های **train** در حالت **dropout** (به جزء لایه ی آخر) و بدون آن در هر اپاک

همان طور که از نمودار بالا می بینیم تفاوت دو نمودار کمتر شده است .

ج)

در دیتاست آموزش، داده های هر دسته با یکدیگر متفاوت اند و بعضی از دسته ها تعداد بسیار بیشتری نسبت به دسته های دیگر دارند. در واقع Imbalanced اند. توزیع هر دسته در داده های Train به صورت زیر است:



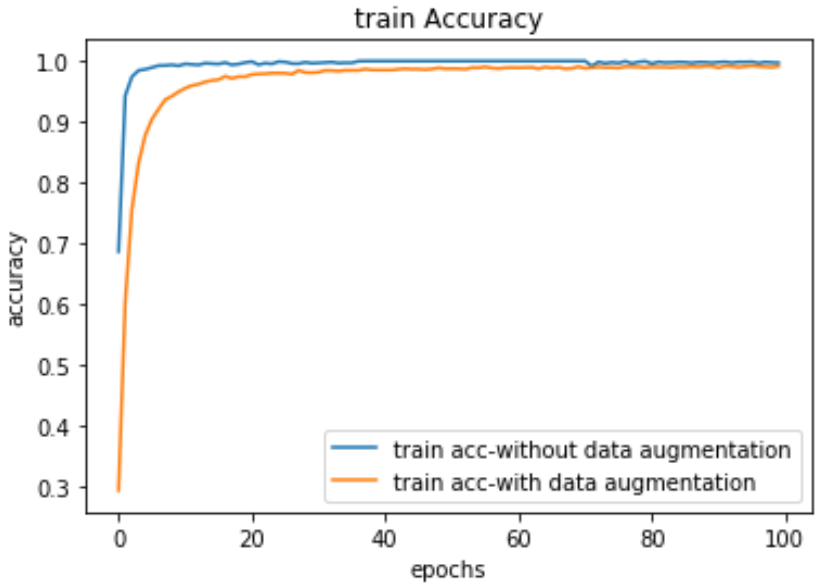
شکل ۲۵. نمودار توزیع داده های هر کلاس

برای این که توزیع داده ها را به صورت balance دربیاوریم تا طبقه بندی بهبود پیدا کند با استفاده از data augmentation دسته ها را balance کرده و تعداد داده های آموزش را افزایش می دهیم.

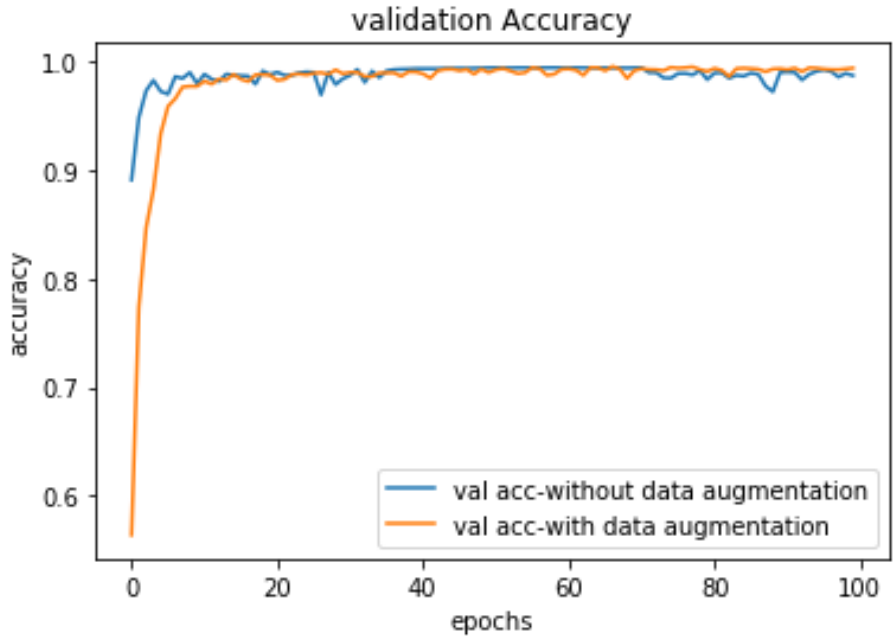
کمبود تعداد داده ها باعث ایجاد overfit در شبکه شود. برای جلوگیری از overfit شدن و به منظور بهبود generalization از data augmentation استفاده می کنیم. در واقع data augmentation با زیاد کردن داده های train، کمک می کند که تمام ویژگیهای انحصاری الگوهای کلاس با انواع به اندازه کافی بالا اعوجاج هندسی و فیزیکی داده ها ظاهر شوند تا همزمان با تضعیف ویژگیهای مشترک کلاسها و منابع مختلف اغتشاش و کاهش فضاهای پوچ، داده های یک کلاس در خوشه هایی متمرکز شوند و از باقی الگوهای باقی کلاسها جدا شوند.

در این پروژه تعداد داده ها به نسبت زیاد است (حدود ۴۰۰۰۰) اما به دنبال آن شبکه دارای ۲۱۵۰۰۰ پارامتر است (۵ برابر تعداد داده ها) و اگر تعداد داده ها افزایش پیدا کند، شبکه generalization بهتری پیدا می کند. در داده های اصلی، تعداد داده ها و هم چنین تنوع آن ها به اندازه کافی بالا بوده و داده های بین هر دسته، ویژگی مشترک کمی را دارند. این باعث می شود که شبکه بدون data augmentation نیز به خوبی تمام حالات را ببیند و دقت بالایی را در داده های آموزش به دست بیاورد و منحنی این دو تفاوت بسیار کمی با یکدیگر پیدا کند.

با استفاده از تابع ImageDataGenerator() عملکرد های rotation، shift عرضی و طولی را روی داده ها اعمال کرده و تصاویر جدیدی را به وجود آوردیم تا حجم داده های train را افزایش دهیم. سپس آن ها آموزش داده و با داده های validation قبلی تست کرده ایم. نمودار های زیر خروجی شبکه همراه با data augmentation و بدون آن را نشان داده است:



شکل ۲۶. نمودار تغییرات دقت داده های **train**، با **data augmentation** و بدون آن در هر اپاک

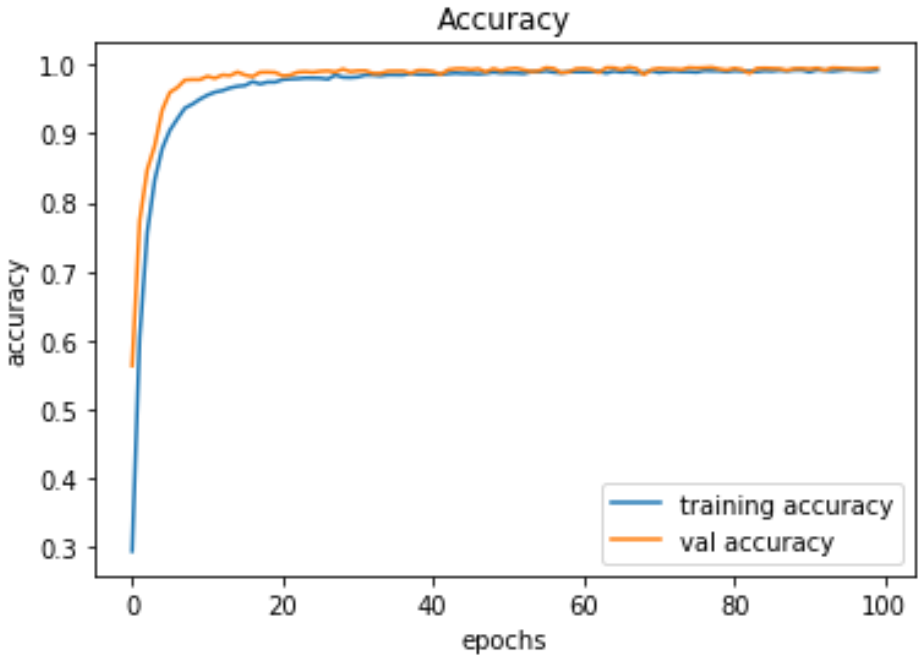


شکل ۲۷. نمودار تغییرات دقت داده های **validation**، با **data augmentation** و بدون آن در هر اپاک

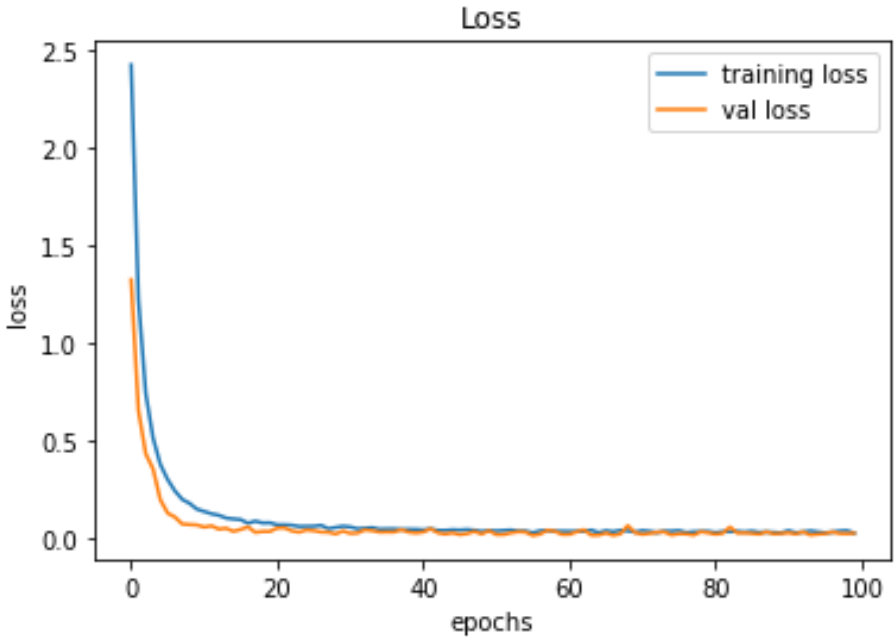
همان طور که دیده می شود، در داده های train، شبکه با ورودی داده های اصلی در اپاک های پایین تری یاد می گیرد و به دقت بالا می رسد و از یک جایی به بعد، میزان یادگیری هر دو تقریبا به یک اندازه بوده و حتی در data augmentation، دقت کمتری داشته اما در داده های validation دقت شبکه با data augmentation کمی بهتر شده است و به نوعی generation را بهبود داده و با یادگیری روی داده های

آموزش، در پیش بینی داده هایی که از قبل ندیده است، عملکرد بهتری را نشان دهد چرا که هر داده با تمرکز بالاتر در کلاس خودش وجود داشته و نسبت به داده های کلاس دیگر، وجه اشتراک را کمتر کرده است.

نمودار دقت و خطای داده های train و validation روی داده های augment شده:



شکل ۲۸. نمودار تغییرات دقت شبکه با **data augmentation** در هر اپاک

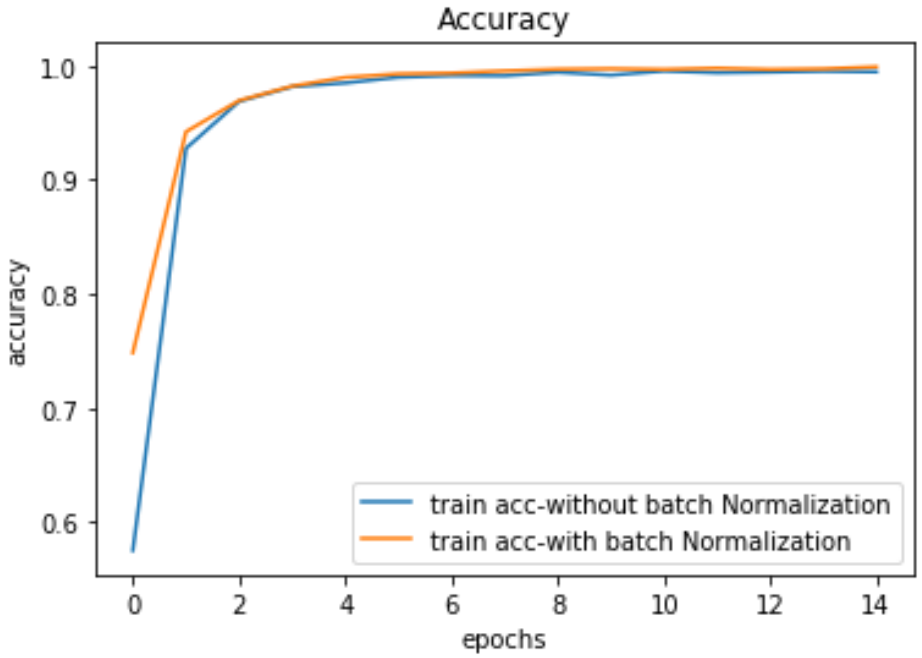


شکل ۲۹.مودار تغییرات خطا شبکه با **data augmentation** در هر اپاک

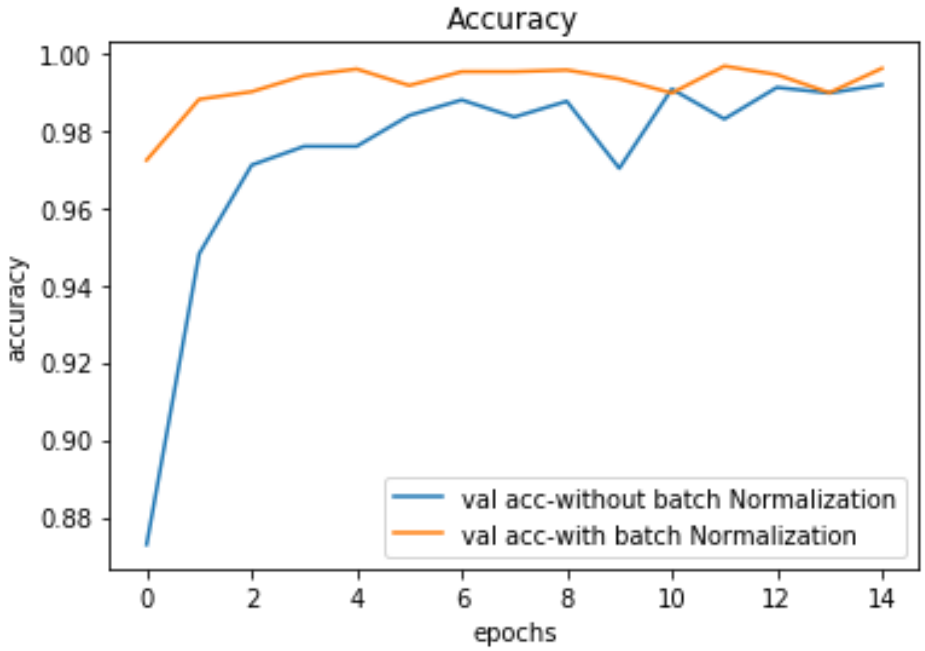
در نمودار های بالا نیز، با مقایسه منحنی داده های train و validation در حالت data augmentation و مشاهده تغییرات یکسان این دو داده، می توان به کارآمدی این روش پی برد.

ح)

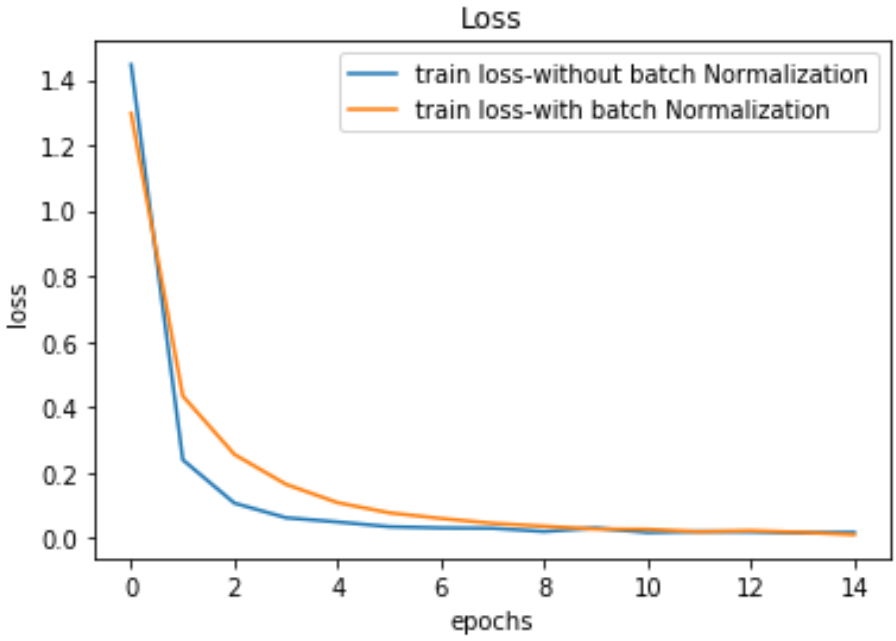
با استفاده از کتابخانه kerass به هر لایه batch_Normaliztion اضافه می کنیم و استفاده از bias را غیرفعال می کنیم. شبکه ایجاد شده را اجرا کرده و نمودار دقت و خطای آن را در مقایسه با مدل اولیه اجرا کرده ایم:



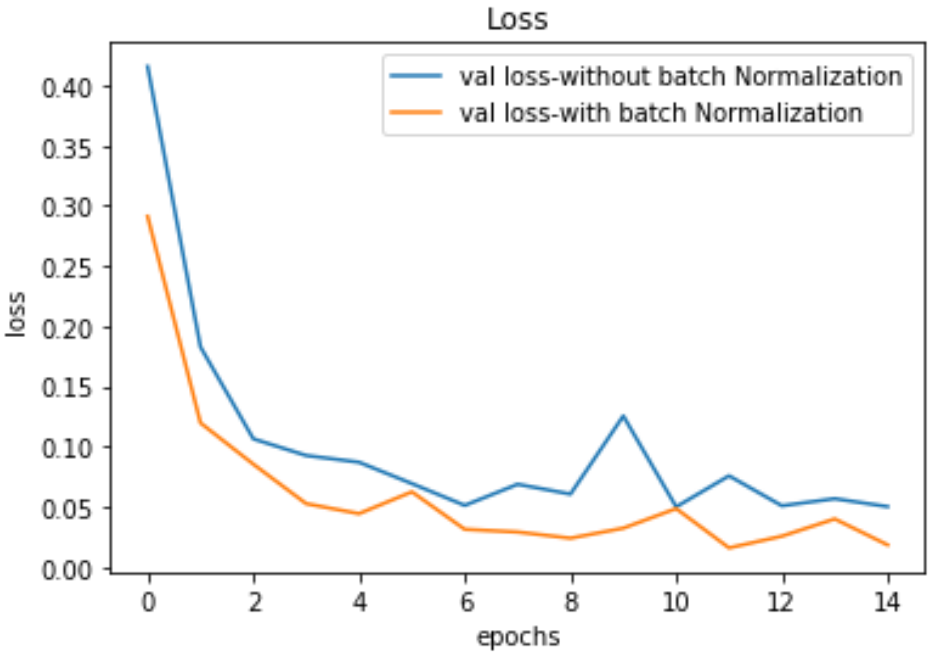
شکل ۳۰. نمودار تغییرات دقت داده های **train** همراه با **batch Normalization** و بدون آن در هر اپاک



شکل ۳۱. نمودار تغییرات دقت داده های **validation** همراه با **batch Normalization** و بدون آن در هر اپاک



شکل ۳۲. نمودار تغییرات خطا در داده های **train** همراه با **batch Normalization** و بدون آن در هر اپاک

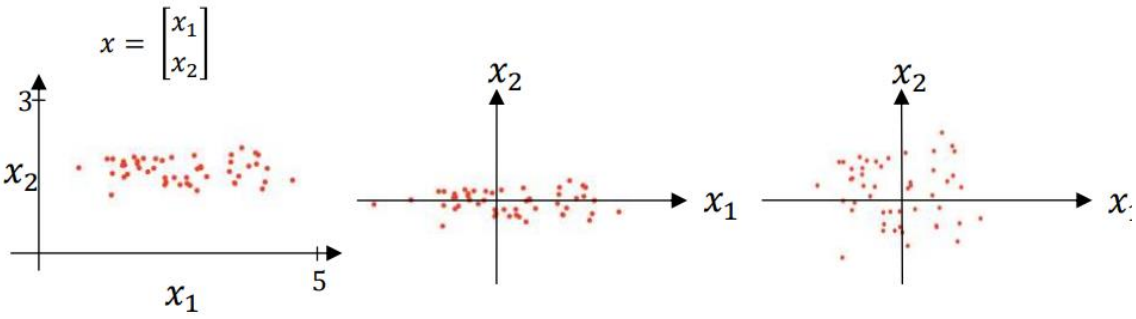


شکل ۳۳. نمودار تغییرات خطا در داده های **validation** همراه با **batch Normalization** و بدون آن در هر اپاک

نمودار ها نشان می دهند که در حالت batch Normalization، دقت بیشتر شده و خطا کم تر شده است. هم چنین منحنی smooth تری را داراست. زیرا داده های ورودی در هر لایه نرمالیزه شده اند و داده ها یکنواخت شده آموزش داده می شوند. با استفاده از نرمالیزه کردن در خروجی هر لایه میانگین اعداد بردار نهایی برابر با صفر و واریانس آن ها برابر با یک شود و این به عنوان ورودی لایه بعدی استفاده می شود.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

در واقع ورودی هر لایه به صورت سمت راست در می آید:



شکل ۳۴. مراحل **normalize** کردن ورودی ها

Train without batch Normalization:

```
Epoch 15/15 27446/27446 [==============================] -
27s 973us/sample - loss: 0.0172 - accuracy: 0.9948 -
val_loss: 0.0501 - val_accuracy: 0.9918
```

Train with batch Normalization:

```
Epoch 15/15 27446/27446 [==============================] -
32s 1ms/sample - loss: 0.0102 - accuracy: 0.9991 - val_loss:
0.0183 - val_accuracy: 0.9960
```

با اضافه کردن batch normalization سرعت یادگیری شبکه بیشتر شده و در واقع سریع تر همگرا می شود. مثلا با تابع بهینه ساز gradient descent برای این که شبکه همگرا شود، لازم است نرخ یادگیری پایینی انتخاب شود، اما با قرار دادن batch normalization می توان از نرخ یادگیری بزرگتر استفاده کرد و سرعت یادگیری را افزایش داد. مقدار دهی وزن ها را آسان تر انجام می دهد. عملکرد تابع فعال ساز را بهتر می کند زیرا مقداری که وارد تابع فعال ساز می شوند، منظم تر می شوند. در کل عملکرد بهتری را در شبکه ایجاد می کند.

همان طور که می بینیم در حالتی که batch Normalization در تمام لایه ها استفاده شده میزان دقت در آخرین اپاک ، بهتر است و خطای کمتری دارد .