Optimizers: 5 epoch

Augmentation = - Dropouts = 0.05, 0.1, 0.1 Regularization = -	Train accuracy	Test accuracy
SGD(lr = 0.01, momentum=0.9)	60%	58%
SGD(lr = 0.001, momentum=0.9 nesterov = True)	91%	80%
Adam(lr= 0.0001)	93%	81%
Adam(lr= 0.001) lr_scheduler.StepLR(step_size=2, gamma=0.1)	87%	80%

آموزش این قسمت در 5 ایپاک انجام شد و بهترین نتیجه برای adam به دست آمد. با توجه به اینکه adam از مومنتوم و rms به صورت ترکیبی استفاده میکند انتظار میرفت که نتیجه بهتری داشته باشد که مطابق با نتایج به دست آمده میباشد.

برای sgd با نرخ یادگیری 0.01 نتایج مناسبی به دست نیامده است که بنظر میرسد دلیل آن واگرایی به دلیل انتخاب نرخ یادگیری نسبتا بالا است اما با انتخاب نرخ یادگیری 0.001 و اعمال nag حدود 22 درصد بهبود در نتایج تست داشتیم.

آخرین ردیف جدول مربوط به adam با scheduler.StepLR میباشد. طبق پارامتر هایی که تنظیم کردم بعد از هر 2 ایپاک نرخ یادگیری ضربدر ضریب gamma میشود و کاهش میابد. با اینکار میخواستم در آخرین ایپاک ها نرخ یادگیری بسیار کم شود تا رفتار واگرا نشان ندهد و به مینیمم سراسری همگرا شود. با توجه به اسکرین شاتی که در صفحه بعد قرار داده ام بنظر میرسد از همان ابتدا loss کم میباشد(0.7) اما سرعت کاهش آن مانند adam نیست و در آخرین ایپاک به loss برابر با adam میرسد.

در ادامه برای تمام تغییراتی که ایجاد کردم از بهینه سازی adam با نرخ یادگیری 0.0001 استفاده کردم.

منبع:https://pytorch.org/docs/stable/optim.html

Adam with scheduler

```
[1,  2000] loss: 0.704
[1,  4000] loss: 0.737
[1,  6000] loss: 0.765
[1,  8000] loss: 0.743
[1, 10000] loss: 0.753
[1, 12000] loss: 0.730
Epoch-1 lr: 0.001

[2,  2000] loss: 0.666
[2,  4000] loss: 0.697
[2,  6000] loss: 0.679
[2,  8000] loss: 0.696
[2, 10000] loss: 0.709
[2, 12000] loss: 0.704
Epoch-2 lr: 0.0001
[3,  2000] loss: 0.525
[3,  4000] loss: 0.496
[3,  6000] loss: 0.470
[3,  8000] loss: 0.448
[3, 10000] loss: 0.444
[3, 12000] loss: 0.447
Epoch-3 lr: 0.0001

[4,  2000] loss: 0.404
[4,  4000] loss: 0.406
[4,  6000] loss: 0.404
[4,  8000] loss: 0.396
[4, 10000] loss: 0.396
[4, 12000] loss: 0.386
Epoch-4 lr: 1e-05
[5,  2000] loss: 0.356
[5,  4000] loss: 0.350
[5,  6000] loss: 0.337
[5,  8000] loss: 0.350
[5, 10000] loss: 0.361
[5, 12000] loss: 0.357
Epoch-5 lr: 1e-05
```

Adam

```
[1,  2000] loss: 1.781
[1,  4000] loss: 1.430
[1,  6000] loss: 1.236
[1,  8000] loss: 1.104
[1, 10000] loss: 0.982
[1, 12000] loss: 0.909
[2,  2000] loss: 0.801
[2,  4000] loss: 0.777
[2,  6000] loss: 0.740
[2,  8000] loss: 0.728
[2, 10000] loss: 0.694
[2, 12000] loss: 0.665
[3,  2000] loss: 0.567
[3,  4000] loss: 0.552
[3,  6000] loss: 0.559
[3,  8000] loss: 0.572
[3, 10000] loss: 0.531
[3, 12000] loss: 0.557
[4,  2000] loss: 0.407
[4,  4000] loss: 0.425
[4,  6000] loss: 0.425
[4,  8000] loss: 0.419
[4, 10000] loss: 0.440
[4, 12000] loss: 0.432
[5,  2000] loss: 0.314
[5,  4000] loss: 0.322
[5,  6000] loss: 0.311
[5,  8000] loss: 0.329
[5, 10000] loss: 0.343
[5, 12000] loss: 0.343
Finished Training
```

Dropouts: 10 epoch

Optimizer = Adam Augmentation = - Regularization = -	Train accuracy	Test accuracy
Dropouts = [0.05, 0.1, 0.1]	97٪	81٪
Dropouts = [0.07, 0.2, 0.2]	94٪	80٪
Dropouts = [0.2, 0.7, 0.7]	79٪	74٪
Dropouts = [0.7, 0.2, 0.2]	86٪	77٪
Dropouts = [0.03, 0.08, 0.08]	97٪	80٪
Dropouts = [0.1, 0.7, 0]	86٪	80٪
Dropouts = [0, 0.7, 0.1]	88٪	80٪
Dropouts = [0.5, 0.5, 0.5]	83٪	77٪
Dropouts = [0, 0, 0]	98٪	82٪
Dropouts = [0.5, 0.5, 0]	86%	78%
Dropouts = [0.7, 0.7, 0]	73%	75%

اولین سطر این جدول مربوط حالت پایه محسوب میشود و برای مقایسه های بعدی حتی برای دیگر جدول ها از آن استفاده میکنیم(دقت آموزش: 97% و دقت تست 81%).

شرح مقادیری که انتخاب کردم:

```
dropouts = [[0.05, 0.1, 0.1], مقادیر پیش فرض
[0.07, 0.2, 0.2], افزایش مقادیر پیش فرض به مقدار کم
[0.2, 0.7, 0.7], افزایش مقادیر پیش فرض به مقدار زیاد
[0.7, 0.2, 0.2], چینش برعکس مقادیر بالا
[0.03, 0.08, 0.08], کاهش مقادیر پیش فرض به مقدار کم
[0.1, 0.7, 0], صفر کردن احتمال لایه آخر
[0, 0.7, 0.1], صفر کردن احتمال لایه اول
[0.5, 0.5, 0.5], 0.5 مقادیر یکسان
[0, 0, 0]] مقادیر یکسان 0 و بی اثر کردن دراپ اوت
```

بهترین حالت برای زمانی است که هیچ dropout ای نداشته باشیم. یکی از دلایل بدتر شدن نتیجه که با توجه به منبع مطالعه شده به آن پی بردم استفاده از dropout قبل از لایه آخر میباشد. چون شبکه قابلیت تصحیح خطاهای به وجود آمده ناشی از dropout قبل از دسته بندی را ندارد. با توجه به نتایج مشاهده شده در جدول این گفته میتواند تایید شود چون با $p = 0.7$ قبل از لایه آخر به بدترین دقت که 74% میباشد رسیده ایم اما با $p = 0$ قبل از لایه آخر به دقت 80% الی 82% رسیده ایم. همچنین چینش دو حالت `[0.2, 0.7, 0.7]` و `[0.7, 0.2, 0.2]` نیز اختلاف 3 درصدی را نشان میدهد و میتواند دلیل ذکر شده را توجیه کند.

با توجه به `[0.5, 0.5, 0]` و `[0.7, 0.7, 0]` انتخاب اعداد بزرگ برای لایه های داخلی دقت داده های تست را کاهش میدهد.

افزایش و کاهش کمِ مقادیر پیش فرض باعث کاهش 1 درصدی دقت شده اما این کاهش چشمگیر نیست.

منبع:https://stats.stackexchange.com/questions/299292/dropout-makes-performance-worse/299305

Augmentation: 5 epoch

Optimizer = Adam Dropouts = 0.05, 0.1, 0.1 Regularization = -	Train accuracy	Test accuracy
Normalize(mean = 0 , std = 1)	92%	80%
RandomGrayscale(p=0.3) Normalize(mean = 0.5 , std = 0.5)	91%	79%
RandomGrayscale(p=0.5) Normalize(mean = 0.5 , std = 0.5)	92%	79%
RandomHorizontalFlip() RandomVerticalFlip() Normalize(mean = 0.5 , std = 0.5)	79%	75%
RandomHorizontalFlip(p=0.5) RandomCrop(32, padding=4) Normalize(mean = 0.5 , std = 0.5)	81%	79%

هیچ یک از تبدیل های بالا نسبت به حالتی که هیچ آگمنتیشنی نداشتیم بهتر نشدند.

من در ابتدا از تبدیل های flip افقی و عمودی استفاده کردم اما وقتی دیدم بهبودی ایجاد نشده است تصمیم گرفتم تا confusion_matrix را چاپ کنم و با مشاهداتی که انجام دادم تصمیم گرفتم تبدیل grayscale را بزنم زیرا مثلا در این ماتریس مشاهده کردم 52 عدد از پیش بینی های اشتباه کلاس هواپیما مربوط به کشتی است. یا اینکه 58 عدد از کشتی ها هواپیما تشخیص داده شدند. ازین تعداد اشتباه به این فکر رسیدم که شاید شباهت رنگ در هواپیما و کشتی و همچنین آسمان و دریا باعث میشود مدل نتواند این دو را از هم تشخیص بدهد.

نتیجه confusion_matrix در flip:

```
tensor([[782.,  10.,  50.,  26.,   7.,   3.,   5.,   7.,  52.,  58.],
        [ 13., 787.,   7.,   2.,   3.,   0.,  10.,   0.,  26., 152.],
        [ 51.,   2., 705.,  59.,  47.,  18.,  81.,  15.,  11.,  11.],
        [ 19.,   6., 104., 603.,  30.,  76., 105.,  28.,  11.,  18.],
        [  9.,   0., 103.,  63., 684.,  22.,  74.,  36.,   4.,   5.],
        [ 10.,   4.,  78., 248.,  32., 549.,  33.,  33.,   2.,  11.],
        [ 10.,   3.,  44.,  38.,  12.,   7., 880.,   1.,   2.,   3.],
        [ 10.,   1.,  51.,  64.,  38.,  30.,   8., 777.,   3.,  18.],
        [ 58.,  26.,  17.,   8.,   4.,   2.,  12.,   2., 820.,  51.],
        [  8.,  23.,   6.,  10.,   0.,   3.,   5.,   5.,  11., 929.]])
```

اما نتایج ماتریس در حالتی که grayscale را اعمال کردم از بهبود معناداری خبر نداد.

نتیجه confusion_matrix در grayscale(p=0.5) :

```
tensor([[885.,  11.,  16.,  14.,  13.,  12.,   2.,   7.,  22.,  18.],
        [ 12., 901.,   4.,   9.,   0.,   5.,   5.,   2.,  11.,  51.],
        [ 70.,   1., 640.,  52.,  80.,  96.,  33.,  16.,   7.,   5.],
        [ 19.,   5.,  36., 597.,  46., 224.,  34.,  21.,  13.,   5.],
        [ 18.,   0.,  36.,  57., 772.,  64.,  24.,  27.,   2.,   0.],
        [  8.,   0.,  14.,  95.,  27., 834.,   2.,  17.,   2.,   1.],
        [  6.,   4.,  24.,  48.,  24.,  53., 826.,   4.,  10.,   1.],
        [  7.,   3.,  11.,  41.,  35.,  87.,   1., 811.,   1.,   3.],
        [ 82.,  19.,   9.,  12.,   3.,   2.,   1.,   4., 849.,  19.],
        [ 22.,  53.,   3.,  20.,   3.,   6.,   4.,   5.,  20., 864.]])
```

تعداد هواپیماهایی که کشتی تشخیص داده شده اند به عدد 22 رسیده اند و کمتر از قبل شدند اما تعداد کشتی هایی که هواپیما تشخیص داده شده اند به 82 رسیدند. دلیل بهبود نیافتن دقت این تبدیل میتواند تاثیر رنگ در تشخیص و تمییز اشیایی باشد که با هم متفاوتند مانند تفاوت رنگ قورباغه و سگ. زیرا در ابتدا که grayscale را اعمال نکردیم تعداد قورباغه هایی که سگ تشخیص داده شده اند 7 تا بودند اما بعد از اعمال به 53 عدد رسیدند.

Augmentation: 10 epoch

Optimizer = Adam Dropouts = 0.05, 0.1, 0.1 Regularization = -	Train accuracy	Test accuracy
RandomGrayscale(p=0.5) RandomHorizontalFlip(p=0.5) RandomCrop(32, padding=4) Normalize(mean = 0.5 , std = 0.5)	86%	82%

برای حالتی که تعداد ایپاک را از 5 به 10 بردم حدود 1 درصد بهبود در داده های تست نسبت به حالتی که هیچ آگمنتیشنی نداشتیم مشاهده میشود. اگر چه دقت روی داده های آموزش حدود 11 درصد پایین آمده است اما بهبود روی داده های تست برای ما مهم است نه داده های آموزش.

منبع برای انتخاب حالت ترکیبی تبدیل ها:

https://medium.com/swlh/how-data-augmentation-improves-your-cnn-performance-an-experiment-in-pytorch-and-torchvision-e5fb36d038fb

L2: 10 epoch

Optimizer = Adam Augmentation = - Dropouts = 0.05, 0.1, 0.1	Train accuracy	Test accuracy
Weight_decay = 1e-4	96%	80%
Weight_decay = 1e-3	92%	82%
Weight_decay = 1e-2	81%	77%
Weight_decay = 1e-1	28%	29%

بهترین مقدار weight_decay = 1e-3 میباشد که باعث بهبود 1 درصدی رو نتیجه داده های تست شده است.

طبق مشاهده با افزایش این پارامتر به 0.1 مدل underfit میشود و جریمه بیش از اندازه شبکه برای وزن ها باعث میشود مدل قابلیت یادگیری خود را از دست بدهد.

منبع برای انتخاب رنج مقادیر: https://dejanbatanjac.github.io/2019/07/02/Impact-of-WD.html

بهترین حالت: 10 epoch

با انتخاب پارامتر هایی که بهترین نتایج را در جداول بالا دادند و ادغام آن ها نتیجه زیر به دست می آید:

	Train accuracy	Test accuracy
Optimizer = Adam(lr = 1e-4) Augmentation = [RandomGrayscale(p=0.5) RandomHorizontalFlip(p=0.5) RandomCrop(32, padding=4) Normalize(mean = 0.5 , std = 0.5)] Dropouts = [0, 0, 0] L2 = 1e-3	85%	83%

همانطور که مشاهده میشود اگرچه دقت ما روی داده های آموزشی از 97% به 85% رسیده است اما روی داده تست از 81% به 83% رسیده ایم که بهبود رو داده های تست مطلوب ما است.