به نام خدا

گزارش فاز چهارم

پروژه هوش محاسباتی

تینا توکلی-محمدهادی امینی

هدف از انجام این پروژه ، اموزش مدلی برای تشخیص گل با استفاده از شبکه عصبی است که برای ان از دیتاست **Oxford Flowers** استفاده میکنیم.

Oxford Flowers:

دیتاست Oxford Flowers، که همچنین به عنوان دیتاست Fisher Iris شناخته می‌شود، یکی از دیتاست‌های معروف در حوزه یادگیری ماشینی است که شامل اطلاعاتی درباره 102 کلاس متفاوت از گل ها است.

هر نمونه در این دیتاست شامل چهار ویژگی است: طول و عرض گلبرگ‌های کاسبرگ و طول و عرض گلبرگ‌های تاج است. همچنین، هر نمونه دارای برچسب کلاس است که نشان می‌دهد که گل متعلق به کدام گونه است.

بخش اول:

در این بخش میخواهیم با در نظر گرفتن 80 کلاس اول(دیتا A) از دیتاست Oxford Flowers شبکه عمیق پیچشی با معماری درخواستی را اموزش دهیم که برای این کار از cross entropy وهمینطور 5لایه Convolutionalو 4 لایه pooling و یک لایه fully connected (البته بعد از flat کردن ویژگیها) استفاده می کنیم .

Cross Entropy:

در پایتورچ، یکی از مفاهیم مهم در حوزه یادگیری ماشین و شبکه‌های عصبی است. در واقع، یک معیار اندازه‌گیری برای مقایسه توزیع‌های احتمالاتی مختلف است.

وقتی مدل‌های آموزش داده شده بهره می‌بریم، ما می‌خواهیم برای اینکه مدل بهتر یاد بگیرد، توزیع احتمالاتی مدل واقعی (توزیعی که مدل پیش‌بینی می‌کند)، به توزیع احتمالاتی مورد انتظار (توزیع واقعی داده‌ها) نزدیک باشد و Cross Entropy یک معیار است که این نزدیکی را اندازه‌گیری می‌کند.

در پایتورچ، می‌توان از تابع torch.nn.CrossEntropyLoss برای محاسبه Cross Entropy استفاده کرد. این تابع به طور خودکار محاسبات مربوط به Cross Entropy را انجام می‌دهد و می‌توانید از آن برای آموزش شبکه‌های عصبی خود استفاده کنید.

Optimizer:

الگوریتم‌های بهینه‌سازی مانند Adam و SGD برای آموزش مدل‌های عمیق استفاده می‌شوند. هر دو الگوریتم بهینه‌سازی برای بهبود مدل و کاهش تابع هزینه (loss function) استفاده می‌شوند،اما با استراتژی‌های متفاوتی عمل می‌کنند

1) Adam (Adaptive Moment Estimation) :

یک الگوریتم بهینه‌سازی پیشرفته‌تر است که بر اساس ترکیبی از ایده‌های الگوریتم‌های Momentum و RMSpropساخته شده است .

از مزایای استفاده همزمان از دو مفهوم "Momentum" و "RMSprop" برای بهبود سرعت همگرایی و پیشرفت بهتر در فضای پارامتری استفاده می‌کندو همچنین عملکرد بهتری در مسائل ناهمگن و غیرخطی دارد و می‌تواند به جواب بهینه بهبود یافته در زمان کمتری همگرا شود. Adamقادر است نرخ یادگیری را برای هر پارامتر به صورت خودکار تطبیق دهد، که می‌تواند بهبودی در کارایی و سرعت آموزش مدل‌ها داشته باشد. البته که تعداد پارامترهای قابل تنظیم بیشتری دارد که باید تنظیم شوند و ممکن است در برخی موارد باعث overfit شود.

2) SGD (Stochastic Gradient Descent) :

یکی از اولین و ساده‌ترین الگوریتم‌های بهینه‌سازی است. در هر مرحله برای به روزرسانی وزن‌ها، یک نمونه تصادفی از داده‌ها را انتخاب کرده و یک گام کوچک در جهت کاهش تابع هزینه برمی‌دارد. همچنین، زمانی که مجموعه داده بسیار بزرگ است، استفاده از SGD می‌تواند منجر به سرعت بیشتر در آموزش شود ولی ممکن است در فضای پارامتری محلی گیر کند و نتواند به جواب بهینه همگرا شود.

با توجه به دلایل بالاو نوع دیتاست،ما در انجام پروژه از روش adam استفاده میکنیم .

مشخصات مدل براساس جدول زیر می باشد:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 64, 64]	1,792
BatchNorm2d-2	[-1, 64, 64, 64]	128
ReLU-3	[-1, 64, 64, 64]	0
Conv2d-4	[-1, 64, 64, 64]	36,928

BatchNorm2d-5	[-1, 64, 64, 64]	128
ReLU-6	[-1, 64, 64, 64]	0
Conv2d-7	[-1, 64, 64, 64]	36,928
BatchNorm2d-8	[-1, 64, 64, 64]	128
ReLU-9	[-1, 64, 64, 64]	0
Conv2d-10	[-1, 64, 64, 64]	36,928
BatchNorm2d-11	[-1, 64, 64, 64]	128
ReLU-12	[-1, 64, 64, 64]	0
Conv2d-13	[-1, 64, 64, 64]	36,928
BatchNorm2d-14	[-1, 64, 64, 64]	128
ReLU-15	[-1, 64, 64, 64]	0
MaxPool2d-16	[-1, 64, 32, 32]	0
Conv2d-17	[-1, 96, 32, 32]	55,392
BatchNorm2d-18	[-1, 96, 32, 32]	192
ReLU-19	[-1, 96, 32, 32]	0
Conv2d-20	[-1, 96, 32, 32]	83,040
BatchNorm2d-21	[-1, 96, 32, 32]	192
ReLU-22	[-1, 96, 32, 32]	0
Conv2d-23	[-1, 96, 32, 32]	83,040
BatchNorm2d-24	[-1, 96, 32, 32]	192
ReLU-25	[-1, 96, 32, 32]	0
Conv2d-26	[-1, 96, 32, 32]	83,040
BatchNorm2d-27	[-1, 96, 32, 32]	192
ReLU-28	[-1, 96, 32, 32]	0
MaxPool2d-29	[-1, 96, 16, 16]	0
Conv2d-30	[-1, 128, 16, 16]	110,720
BatchNorm2d-31	[-1, 128, 16, 16]	256
ReLU-32	[-1, 128, 16, 16]	0
Conv2d-33	[-1, 128, 16, 16]	147,584
BatchNorm2d-34	[-1, 128, 16, 16]	256
ReLU-35	[-1, 128, 16, 16]	0
Conv2d-36	[-1, 128, 16, 16]	147,584
BatchNorm2d-37	[-1, 128, 16, 16]	256
ReLU-38	[-1, 128, 16, 16]	0
Conv2d-39	[-1, 128, 16, 16]	147,584
BatchNorm2d-40	[-1, 128, 16, 16]	256

ReLU-41	[-1, 128, 16, 16]	0
MaxPool2d-42	[-1, 128, 8, 8]	0
Conv2d-43	[-1, 256, 8, 8]	295,168
BatchNorm2d-44	[-1, 256, 8, 8]	512
ReLU-45	[-1, 256, 8, 8]	0
Conv2d-46	[-1, 256, 8, 8]	590,080
BatchNorm2d-47	[-1, 256, 8, 8]	512
ReLU-48	[-1, 256, 8, 8]	0
Conv2d-49	[-1, 256, 8, 8]	590,080
BatchNorm2d-50	[-1, 256, 8, 8]	512
ReLU-51	[-1, 256, 8, 8]	0
Conv2d-52	[-1, 256, 8, 8]	590,080
BatchNorm2d-53	[-1, 256, 8, 8]	512
ReLU-54	[-1, 256, 8, 8]	0
MaxPool2d-55	[-1, 256, 4, 4]	0
Flatten-56	[-1, 4096]	0
Linear-57	[-1, 80]	327,760
==========		
Total params: 3,405,136		
Trainable params: 3,405,136		
Non-trainable params: 0		

Input size (MB): 0.05		
Forward/backward pass size (MB): 44.31		
Params size (MB): 12.99		
Estimated Total Size (MB): 57.35		

همانطور که در صورت پروژه گفته شده است پس از بررسی بسیاری از learning rate وepoch و batch_size های متفاوت ،در اخر بهترین دقت تست وترین روی دیتاA با پارامترهای زیر به دست امده است:

learning rate	0.003
epoch	40
batch_size	128

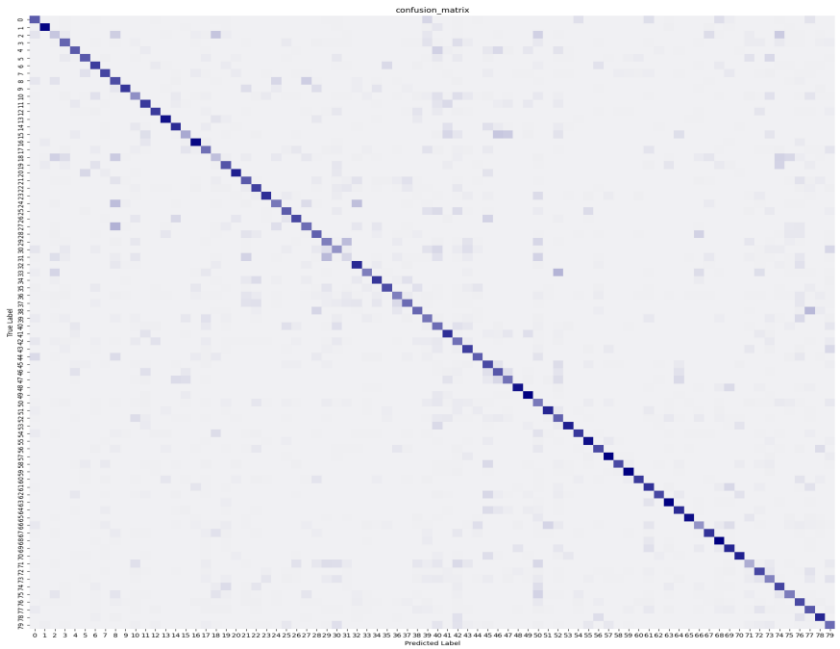
جدول1- بهترین هایپر پارامترهای به دست امده

train_acc	99.91336822509766
test_acc	73.21195983886719
train_loss	0.013617317330696294
test_loss	1.0926333488180087

جدول2- بهترین معیارهای ارزیابی به دست امده

ماتریس گمراهی(Confution_matrix):

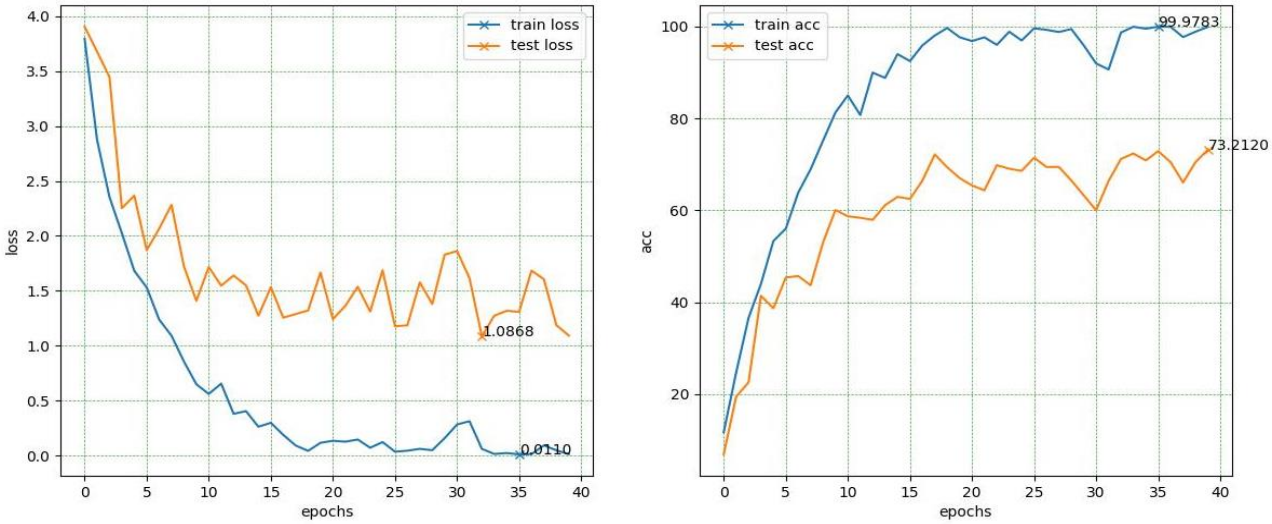
در این ماتریس لیبل های حقیقی را با لیبل های پیش بینی شده توسط مدلی که اموزش داده ایم برای دیتاست A ،مقایسه میکنیم و همانطور که مشاهده میشود پررنگ بودن قطر اصلی نشان ازتشخیص درست و دقت بالای مدل اموزش دیده دارد



تصویر1 ماتریس گمراهی برای دیتاA

نمودارهای منحنی یادگیری و دقت :

به طور کلی هرچه دقت train بیشتر شود ، اگر مدل به درستی اموزش داده شده باشد همراه با ان دقت test هم باید زیاد شود و مشابه همین رویه در loss_ train و loss_test نیز باید مشاهده شود که با گذشت زمان و بیشتر شدن تعداد epoch ،هر دومورد همراه با هم کم شود که این روند در نمودار نیز مشهود است.



تصویر2- نمودار acc و loss مدل نسبت به تعداد epochها

بخش دوم:

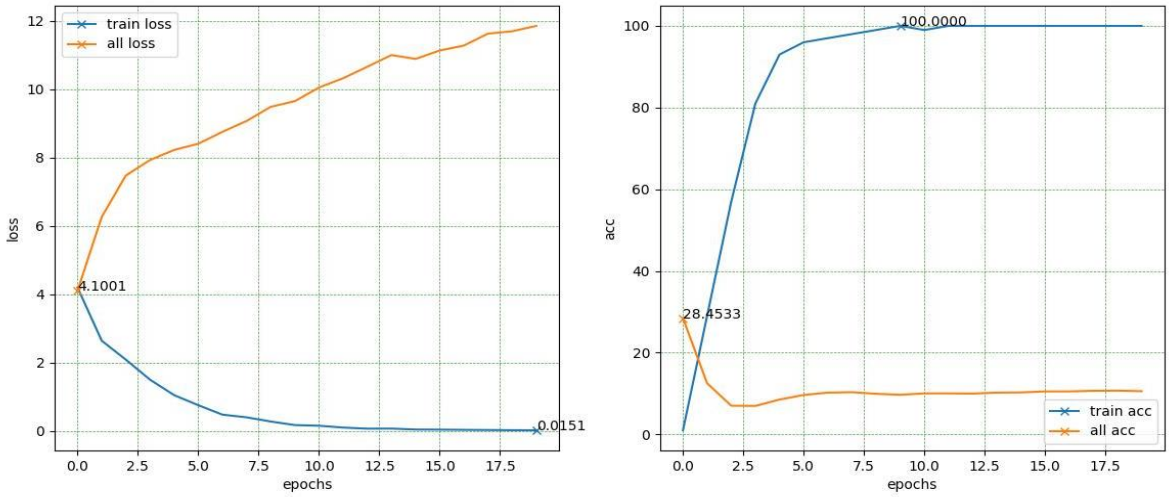
در این بخش میخواهیم با در نظر گرفتن 20 کلاس دوم(دیتا B) از دیتاست Oxford Flowers ،با همان معماری که در بخش قبل توضیح داده شده ،مدلی برای 100 کلاس اول دیتاست اموزش دهیم اما به دلیل تعداد کم داده در دیتا B نتیجه خوبی به دست نمی اوریم به همین علت از مدل اموزش دیده قبلی استفاده میکنیم و 3 روش را بررسی میکنیم تا بهترین نتیجه را به دست اوریم .

روش کپی کردن مدل اموزش دیده در بخش اول :

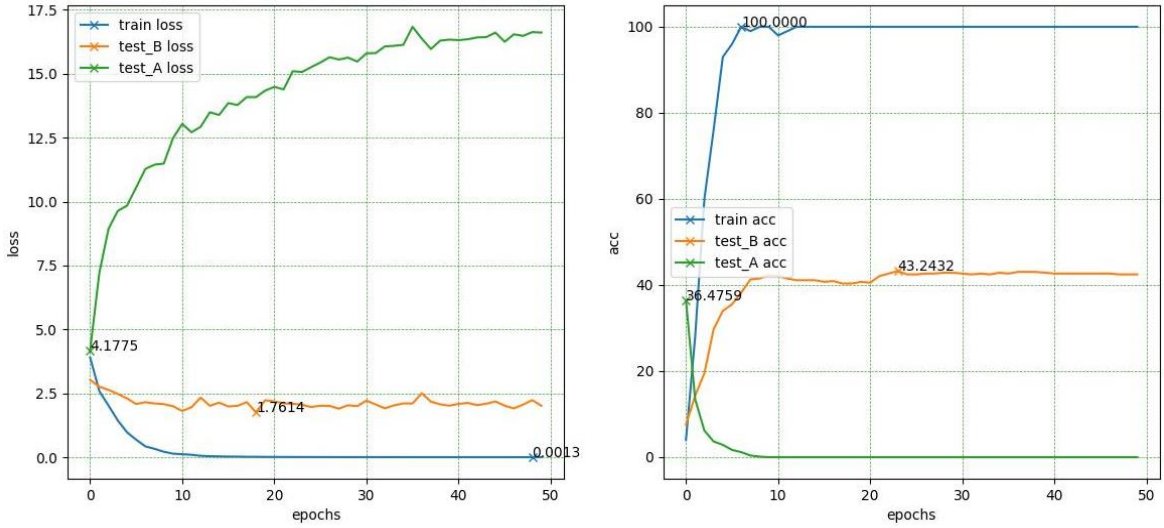
با استفاده از () torch.save مدل را سیو کرده و سپس برای استفاده مجدد از این مدل و تغییرلایه اخر ان، ابتدا با استفاده از ()model.load_state_dict مدل را لود میکنیم سپس لایه اخر ان را عوض میکنیم (به جای 80 خروجی ،100 خروجی میگذاریم) و پس از ان وزن وbias مربوط به 80 کلاس اول را برابر با مقادیر ان در مدل قبلی قرار میدهیم .

روش اول:

پس از کپی کردن مدل اموزش دیده که شامل تمام وزنها و پارامترهای قابل یادگیری است و همانطور که گفته شده است تنها تفاوتی که ایجاد میشود در لایه اخر (لایه fully connected) شبکه است که به جای 80 خروجی ،100 خروجی گذاشته ایم و سپس مدل جدید را با دیتاB اموزش میدهیم

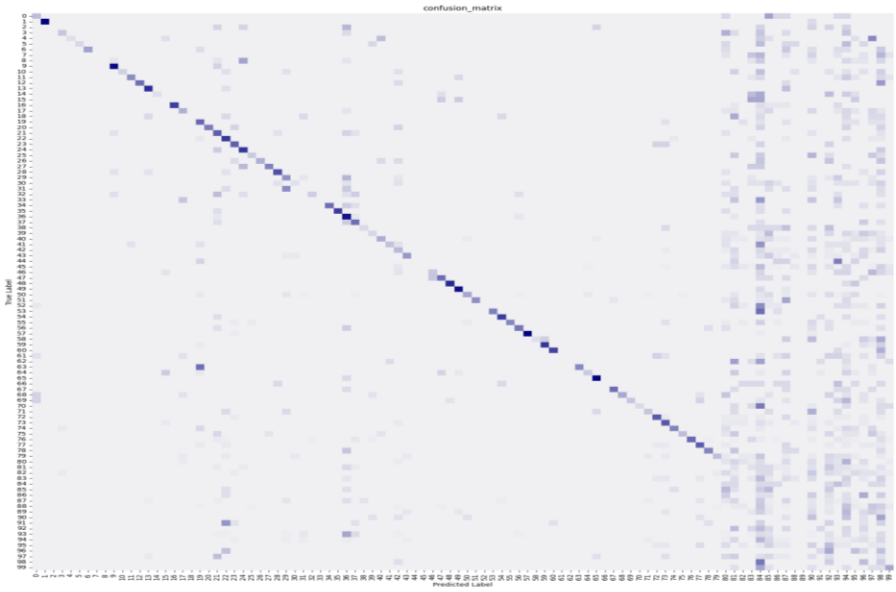


تصویر 1 - منحنی یادگیری و دقت روش اول برای test all (کل دیتا)

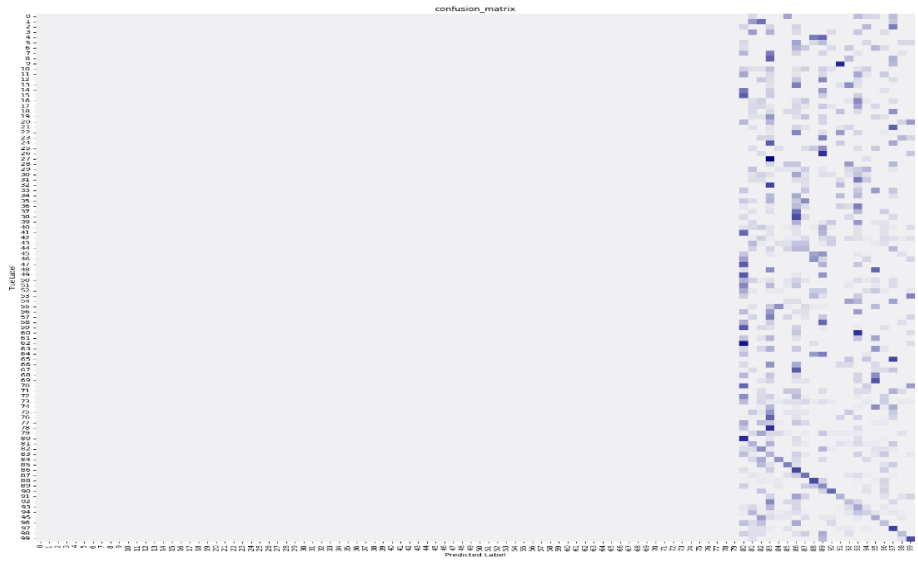


تصویر 2 - منحنی یادگیری و دقت روش اول برای دیتاست B

هرچه تعداد epoch بیشتر میشود به علت اینکه شبکه فقط داده های مربوط به دیتاست B را میبیند بنابراین پارامترها را با توجه به این دیتاست اموزش میدهد که باعث افزایش دقت دیتاست B و افت دقت تست برای دیتاست A میشود و به دلیل اینکه پارامترها به درستی تغییر نکرده ،دقت دیتاست all نیز کاهش می یابد(در همان epoch اول مشهود است)



تصویر 3 - ماتریس گمراهی روش اول در epoch=1



تصویر 4 - ماتریس گمراهی روش اول در epoch آخر

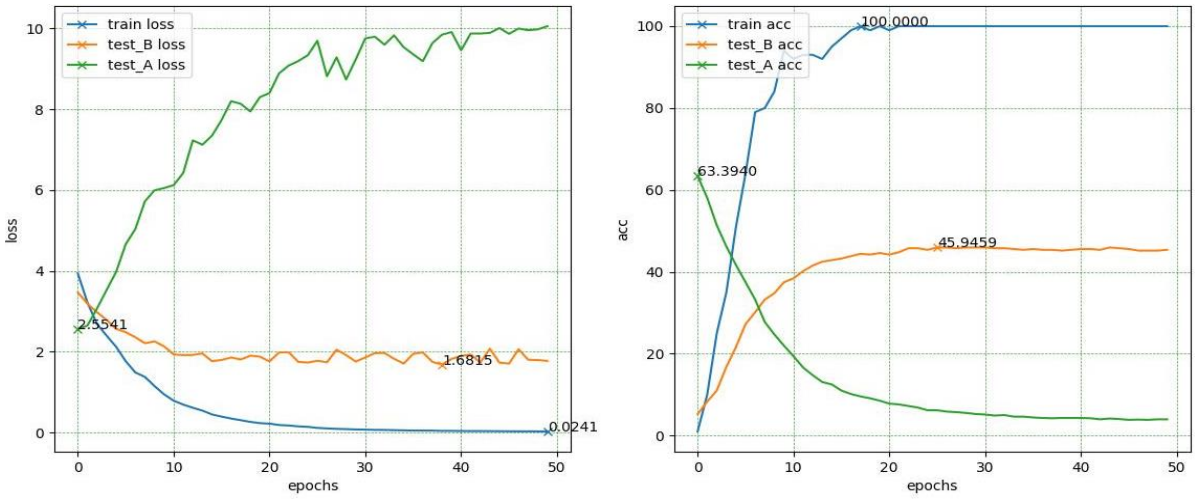
همانطور که در بالا نیز توضیح داده شده به دلیل اینکه تمام وزنها و biasهای 80 کلاس اول کپی شده است ، در اولین ماتریس گمراهی به دست امده داده های 80 کلاس اول به خوبی تشخیص داده شده اند اما ادامه اموزش باعث فراموشی وزنهای مربوط به 80 کلاس اول و overfit شدن مدل براساس 20 کلاس دیگر میشود.

روش دوم:

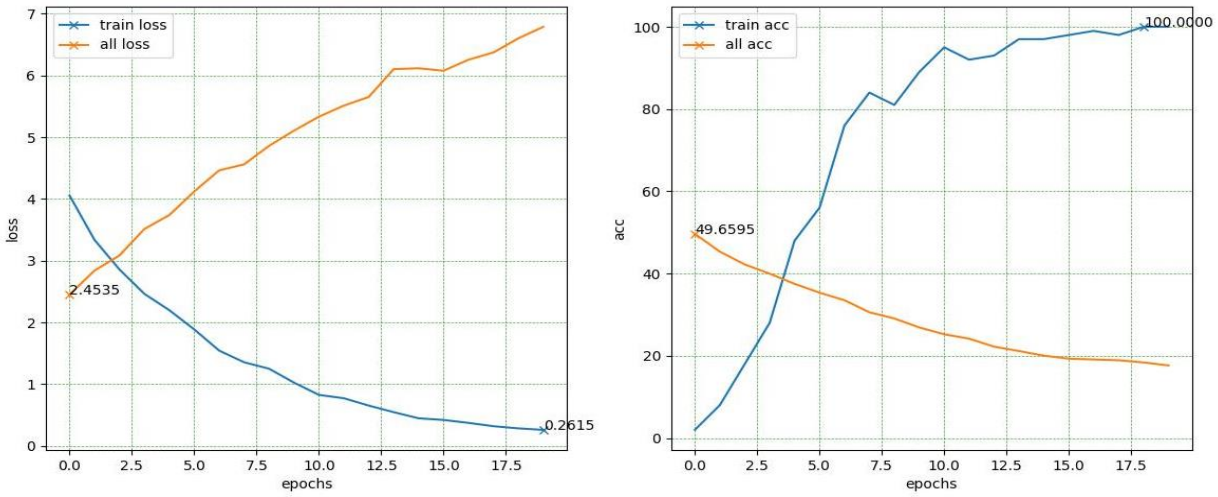
در این روش همانند روش اول مدل را کپی کرده و تغییر میدهیم سپس تمام لایه های موجود در مدل را غیر از لایه اخر فریز کرده تا بخشی از پارامترها را ثابت نگه داشته و سپس مدل را اموزش میدهیم.

نحوه فریز کردن لایه در مدل:

با استفاده از دستور requires_grad = False ، که برای غیرفعال کردن محاسبات مشتق‌پذیر برای یک پارامتر در شبکه عصبی است، دیگر مشتق را نسبت به آن پارامتردر مدل محاسبه نمی‌کند و این به معنای عدم به‌روزرسانی آن پارامتر در طول فرآیند یادگیری است که سبب بهبود سرعت محاسبات نیز میشود.

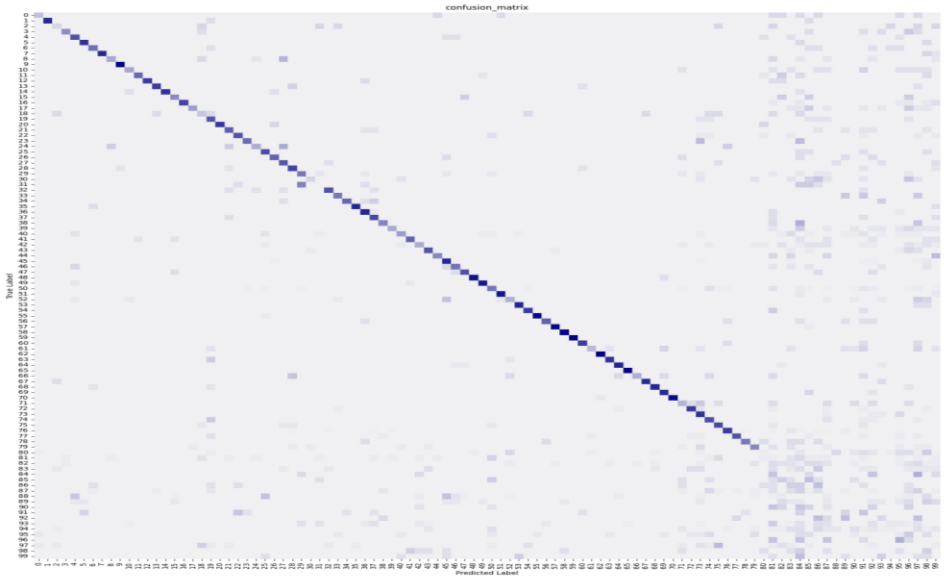


تصویر5 منحنی یادگیری و دقت روش دوم برای دیتاست B

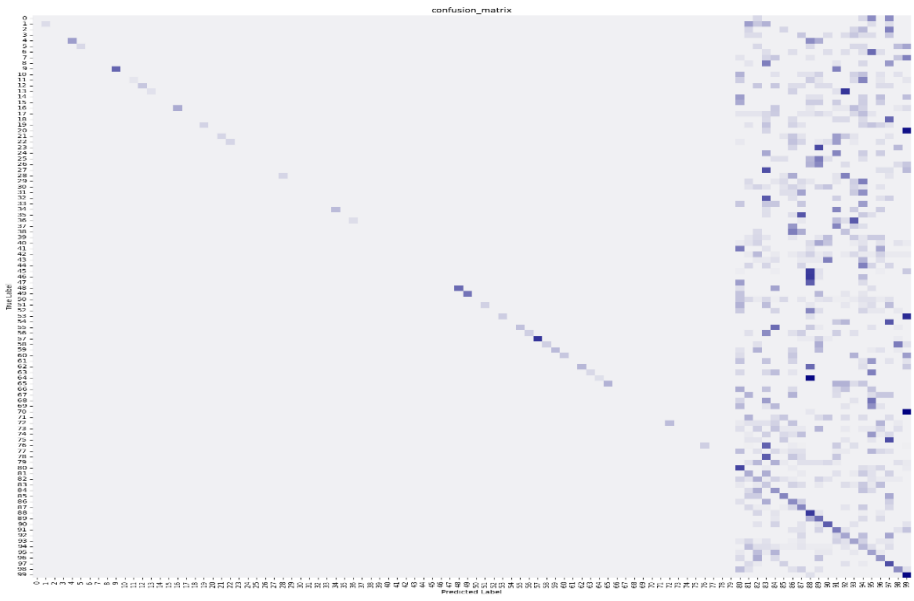


تصویر 6 - منحنی یادگیری و دقت روش دوم برای test all (کل دیتا)

هرچه تعداد epoch بیشتر میشود به علت اینکه شبکه فقط داده های مربوط به دیتاست B را میبیند بنابراین پارامترها را با توجه به این دیتاست اموزش میدهد که باعث افزایش دقت دیتاست B و افت دقت تست برای دیتاست A میشود و به دلیل اینکه پارامترهای لایه اخر به درستی تغییر نکرده ،دقت دیتاست all کاهش می یابد ،البته با توجه به فریز بودن لایه های میانی کاهش دقت برای دیتاست all با شیب بسیار کمتری نسبت به روش اول تغییر میکند و در نهایت دارای دقت بهتری از ان میباشد.



تصویر 7 ماتریس گمراهی روش دوم در اولین epoch



تصویر 8 ماتریس گمراهی روش دوم در اخرین epoch

همانطور که در بالا نیز توضیح داده شده به دلیل اینکه تمام وزنهاو biasهای 80 کلاس اول کپی شده است ، در اولین ماتریس گمراهی به دست امده داده های 80 کلاس اول به خوبی تشخیص داده شده اند وبرای 20 کلاس بعدی هنوز توانایی تشخیص درستی ندارد اما در ادامه اموزش برخلاف روش اول با توجه به ثابت بودن بخشی از پارامترها ،وزنهای مربوط به 80 کلاس اول اسیب کمتری نسبت به روش قبل می بیند که این باعث میشود دقت بهتری نیز نسبت به ان داشته باشد به طوریکه در اخرینepoch، قابلیت تشخیص بهتر بخشی از داده های مربوط به 80 کلاس اول را دارد وهمچنین پیش بینی درست در 20 کلاس بعدی نیز افزایش می یابد (اما باز هم روش درست وکاربردی نیست زیرا همانطور که در نمودار های بالا مشاهده میشود بخش بزرگی از دیتا را تشخیص نمیدهد)

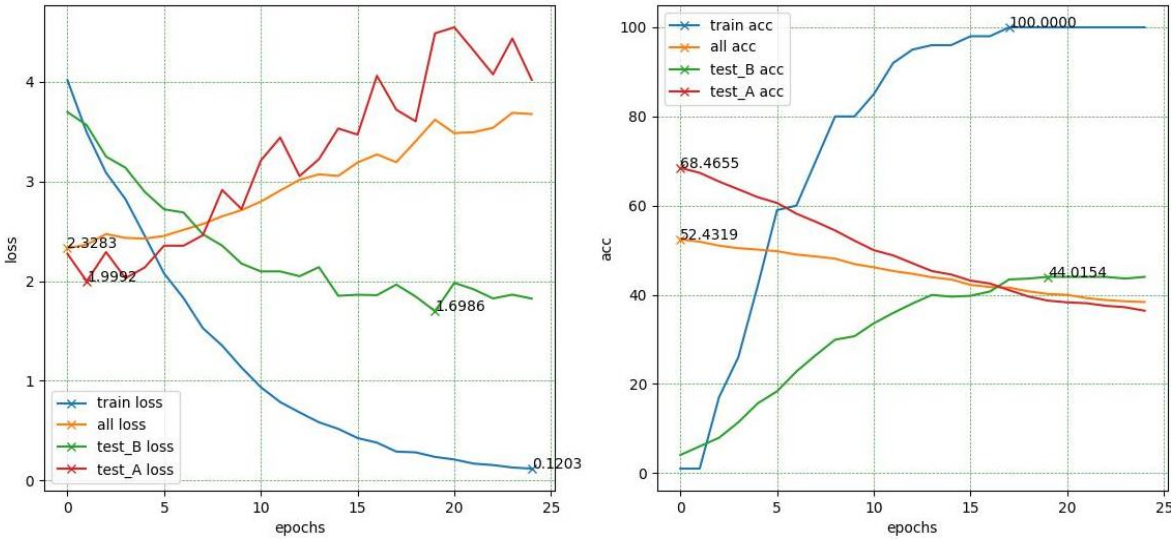
روش سوم:

همانند روش دوم مدل را تغییر میدهیم با این تفاوت که در لایه اخرنیز وزن و bias های مربوط به 80 کلاس اول را فریز میکنیم و سپس مدل را اموزش میدهیم.

نحوه فریز کردن بخشی از یک لایه در مدل:

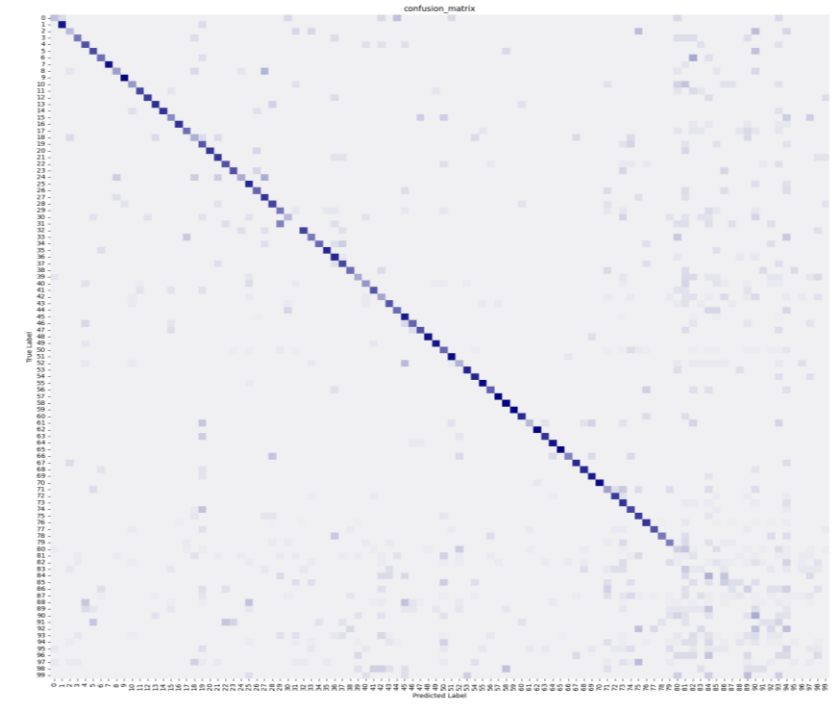
به دلیل محدودیت های pythorch در یک لایه نمیتوان همزمان requires_grad بخشی از ان را غیر فعال و بخش دیگری را فعال کرد بنابراین از روش دیگری استفاده میکنیم که به شرح زیر است:

با استفاده از register_hook بعد از محاسبه مشتق ،گرادیان مربوط به پارامترهای 80 کلاس اول را در 0 ضرب کرده که این سبب میشود ، پارامترها تغییر نکند .

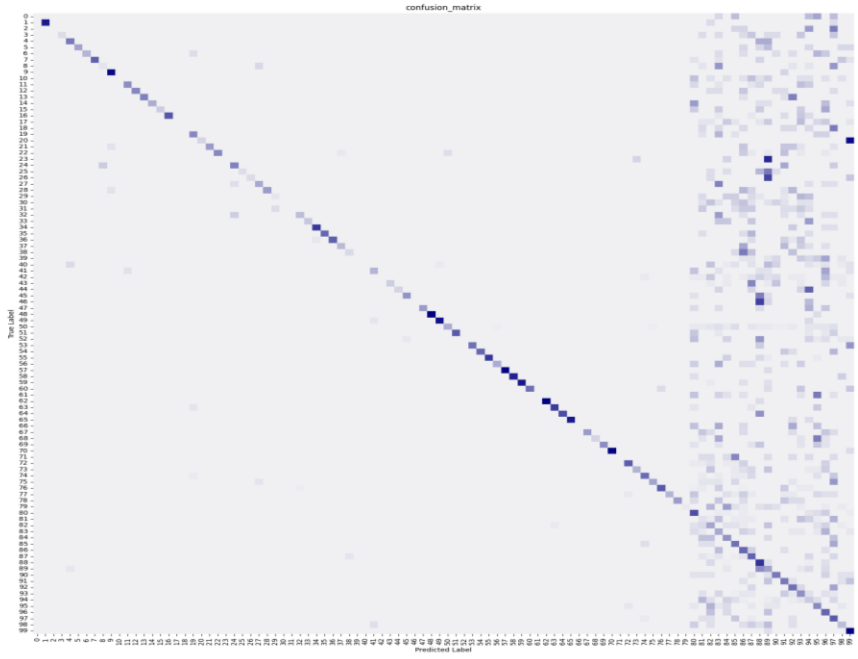


تصویر 9 منحنی یادگیری و دقت روش سوم

هرچه تعداد epoch بیشتر میشود به علت اینکه شبکه فقط داده های مربوط به دیتاست B را میبینند ، پارامترها را با توجه به این دیتاست اموزش میدهد که باعث افزایش دقت دیتاست B میشود و همچنین به علت اینکه لایه های میانی فریز هستند و در لایه اخرنیز گرادیان مربوط به 80 کلاس اول 0 میشود بنابراین تغییری در پارامترهای این 80 کلاس ایجاد نشده و دقت تست نسبت به روش دوم برای دیتاست A کمتر کاهش می یابد واین امر سبب میشود تا دقت all به طور چشمگیری با شیب کمتری کاهش می یابد و از روش های قبل بسیار بهتر است.



تصویر 10 ماتریس گمراهی روش سوم در اولین epoch



تصویر 11 ماتریس گمراهی روش سوم در اخرین epoch

همانطور که در بالا نیز توضیح داده شده به دلیل اینکه تمام وزنهاو biasهای 80 کلاس اول کپی شده است ، در اولین ماتریس گمراهی به دست امده داده های 80 کلاس اول به خوبی تشخیص داده شده اند ولی برای 20 کلاس بعدی هنوز توانایی تشخیص درستی ندارد ،در ادامه به علت فریز بودن لایه های میانی مدل و همچنین پارامترهای مربوط به این 80 کلاس در لایه اخر ، تشخیص مدل نیز تغییر بسیار کمی داشته به طوریکه در ماتریس های به دست امده برای این کلاسها تفاوت زیادی مشاهده نمیشود وهمچنین پیش بینی درست در 20 کلاس بعدی نیز نسبت به روش های قبل افزایش می یابد

در نهایت پس از بررسی دلایل و نتایج به دست امده، روش سوم نسبت به سایر روش ها مناسب تر است .