دانشکده مهندسی کامپیوتر

مباحث ویژه ۱ (یادگیری عمیق)

تمرین سری دهم

علی صداقی

۹۷۵۲۱۳۷۸

۱ سوال اول

ابتدا زمان‌بند Learning rate را حذف کردیم زیرا در تعداد اپک پایین نتیجه مطلوبی نداشت و سریعا نرخ یادگیری را کوچک می‌کرد. سایر هایپرپارامترهای مشترک به صورت زیر می‌باشد:

- Batch size = 32
- Loss function = Cross entropy
- Optimizer = Adam
- Learning rate = 0.001
- Epochs = 15

با توجه به زمان‌بر بودن آموزش و همچنین محدودیت کولب در استفاده از GPU مجبور به آموزش در 15 اپک شدیم. بدیهی است آموزش در بازه بیشتر می‌توانست نتیجه بهتری داشته باشد.

الف) در این قسمت هنگام لود کردن مدل ResNet آرگومان pretrained را برابر False قرار دادیم تا مدل وزن‌های تصادفی داشته باشد. در قسمت دسته‌بند هم از یکی شبکه FC استفاده کردیم که تعداد 196 نورون دارد. بدیهی است مدل نیازمند مدت زمان زیادی برای آموزش است.

تعداد پارامترهای مدل به صورت زیر است:

- Number of parameters: 23909636
- Number of trainable parameters: 23909636

در اپاک اول مدل به صورت زیر عمل کرد که مشاهده می‌شود اصلا عملکرد جالبی نبوده است. زیرا مدل از نقطه‌ای کاملا تصادفی شروع به آموزش می‌کند.

```
Epoch 0/14
----------
Iterating through data...
train Loss: 173.7111 Acc: 0.5392
Iterating through data...
val Loss: 169.3775 Acc: 1.0789
```

دقت در آموزش برابر 0.53 درصد و در تست برابر 1 درصد بوده که با توجه به وجود 196 کلاس امری طبیعی است:

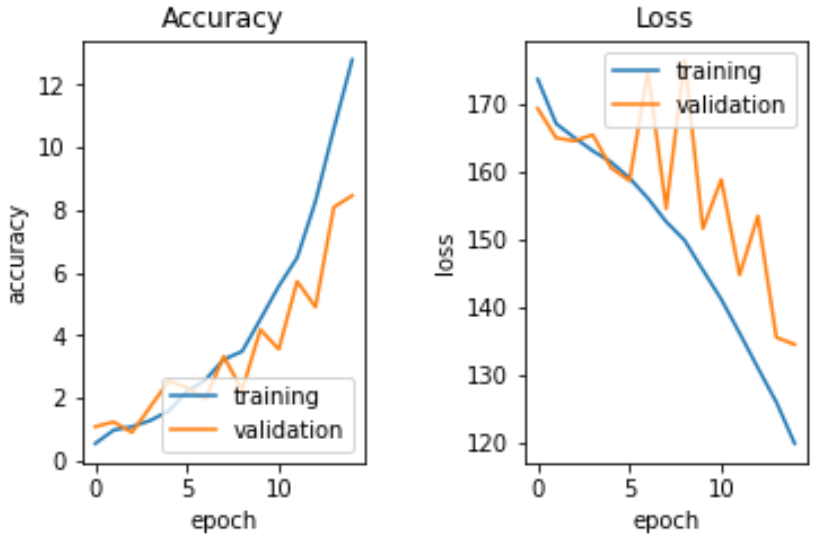
$$\frac{1}{196} \times 100 = 0.51\ \%$$

عملکرد مدل در پایان آموزش نیز به صورت زیر است:

```
Epoch 14/14
----------
Iterating through data...
train Loss: 119.7486 Acc: 12.7941
Iterating through data...
val Loss: 134.4328 Acc: 8.4573

Training complete in 75m 40s
Best val Acc: 8.457342
```

دقت مدل در پایان آموزش روی داده آموزشی برابر 13 درصد و روی داده تست برابر 8.45 درصد بوده است که اصلا نتیجه جالبی نیست.



نمودار Loss و Accuracy نشان می‌دهد که مدل دچار Underfit شدید است که دلیل اصلی آن کم بودن میزان آموزش یعنی تعداد Epoch کم است. در واقع هنگامی که می‌خواهیم در تعداد Epoch کم به نتیجه خوب برسیم بهتر است از مدلی از پیش آموزش داده شده (Pretrained) استفاده کنیم (بخش ب) و تنها لایه‌های انتهایی را Fine tune کنیم (بخش ت)

ب) در این بخش از مدل از پیش آموزش داده شده ResNet به عنوان مدل پایه استفاده کردیم. برای تحقق این امر در هنگام لود کردن مدل ResNet آرگومان pretrained را برابر True قرار دادیم. پس از انجام این کار وزن‌های شبکه از جای دیگری دانلود شدند که حجمی حدود 98MB داشت.

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100% 97.8M/97.8M [00:02<00:00, 46.0MB/s]
```

با توجه به این که در این بخش Fine tune روی مدل پایه ResNet نداشتیم، این مدل را Freeze کردیم. برای تحقق این امر مقدار requires_grad را برای لایه‌های مدل پایه برابر False قرار دادیم.

همچنین از یک دسته‌بند مبتنی بر شبکه عصبی استفاده کردیم که دارای 196 نورون است. این دسته‌بند دارای وزن‌های تصادفی است و نیازمند آموزش است. پس مقدار requires_grad را برای این لایه برابر True قرار دادیم.

تعداد پارامترهای مدل به صورت زیر است:

- Number of parameters: 23909636
- Number of trainable parameters: 401604

همانطور که مشاهده می‌شود تعداد پارامترهای قابل آموزش در این بخش بسیار کم‌تر است. به همین دلیل زمان اجرای مدل نیز کاهش می‌یابد (بدلیل کمتر شدن عملیات Backpropagation)

در اپاک اول نتیجه از بخش الف بسیار بهتر بود زیرا وزن‌های مدل پایه ResNet تصادفی نیستند و تنها وزن‌های لایه آخر تصادفی است.

```
Epoch 0/14
----------
Iterating through data...
train Loss: 156.0890 Acc: 7.8309
Iterating through data...
val Loss: 126.3226 Acc: 16.8899
```

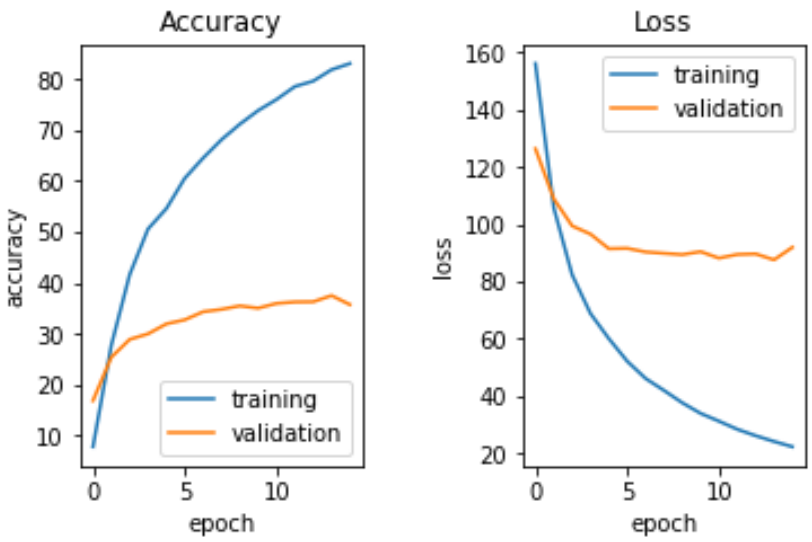
دقت در آموزش برابر 7.83 درصد و در تست برابر 16.88 درصد بوده که از حالت الف بسیار بهتر است.

عملکرد مدل در پایان آموزش نیز به صورت زیر است:

```
Epoch 14/14
----------
Iterating through data...
train Loss: 22.2534 Acc: 83.1127
Iterating through data...
val Loss: 91.8879 Acc: 35.7515

Training complete in 62m 11s
Best val Acc: 37.524803
```

دقت مدل در پایان آموزش روی داده آموزشی برابر 83 درصد و روی داده تست برابر 37 درصد بوده است که نسبت به حالت الف نتیجه بهتری است اما همچنین نتیجه بدی است. در این بخش دچار Overfit شدید شده ایم و اختلاف دقت در فاز آموزش و تست برابر 46 درصد است.



نمودار Loss و Accuracy نیز نشان دهنده این Overfit شدید است. تقریبا از ایپاک 13 مدل شروع به Overfit می‌کند و خطای Validation شروع به افزایش می‌کند.

دلیل این Overfit شدید ثابت و Freeze بودن وزن‌های پایه شبکه می‌باشد زیرا مدل تنها می‌تواند وزن‌های لایه دسته‌بند را تغییر دهد و به قدری وزن‌های این لایه را تغییر می‌دهد که داده آموزشی حفظ می‌شود. طبیعتا اگر وزن‌های بیشتری قابل آموزش بودند مدل می‌توانست ویژگی‌های کلی‌تر و Generalتری آموزش ببیند و کمتر دچار مشکل Overfit شویم (بخش د).

نتیجه این بخش نسبت به بخش الف بهتر است زیرا شبکه پایه از پیش آموخته رزنت به دلیل وجود لایه‌های Conv قادر است ویژگی‌های مشترکی مانند لبه‌ها، رنگ‌ها و ... را سریعا تشخیص دهد. دلیل عملکرد بد آن نیز به این دلیل است که ویژگی‌های ثابت استخراج شده در لایه‌های آخر (ویژگی‌های سطح بالا) مربوط به مسئله ImageNet است و نمی‌توان در این مسئله استفاده کرد.

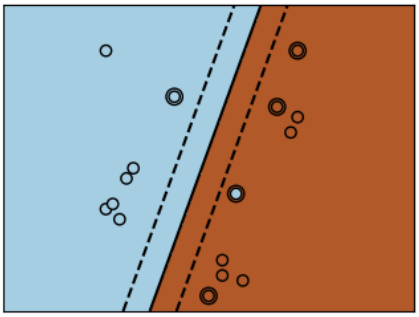
پ) قسمت ب که در آن از شبکه پایه از پیش آموخته (Pretrained) ResNet50 استفاده کرده بودیم را این بار با یک دسته‌بند SVM آموزش می‌دهیم. در واقع به جای اینکه در لایه آخر بخش ب یک دسته‌بندی مبتنی بر شبکه عصبی قرار دهیم یک SVM قرار می‌دهیم. برای انجام این کار مراحل زیر را طی می‌کنیم:

۱. ابتدا داده‌ها را در Batchهای 32تایی به مدل پایه رزنت می‌دهیم سپس ویژگی‌های استخراج شده از این بسته‌ها را درون یک لیست کلی ذخیره می‌کنیم. در واقع تنها از بخش Forward مدل رزنت استفاده می‌کنیم.
۲. کار بالا را برای داده‌های تست نیز انجام می‌دهیم.
۳. ویژگی‌های به دست آمده را به عنوان ورودی به شبکه SVM می‌دهیم تا آموزش ببیند.

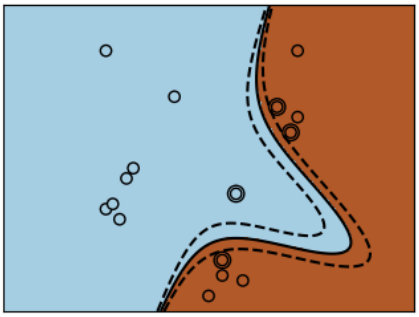
در واقع آموزش SVM به صورت Batch می‌باشد و کل داده‌ها را با هم می‌بیند.

برای SVM کرنل‌های متفاوتی وجود دارد که برخی از آن‌ها را معرفی می‌کنیم:

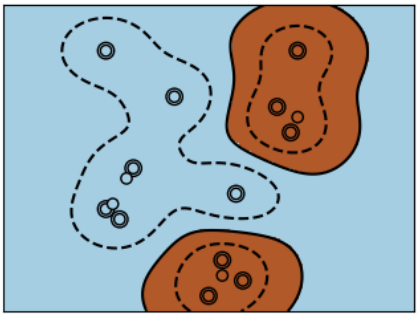
- Linear: برای داده‌های جداپذیر با خط (Linearly separable) مناسب است و مناسب این مسئله نیست.



- Polynomial: از یک چندجمله‌ای به عنوان کرنل SVM استفاده می‌کند و برای این مسئله مناسب است.



- RBF: از کرنل‌های Radial استفاده می‌کند که دارای توزیع گاوسی هستند. برای این مسئله مناسب است.



- Sigmoid: از تابع Sigmoid یا Tanh استفاده می‌کند که برای این مسئله مناسب نیست.

با امتحان کردن کرنل‌های مختلف کرنل Poly به عنوان بهترین کرنل شناخته شد. نتایج کرنل RBF بسیار نزدیک به Poly بود و دقت Poly تنها یک درصد بهتر از RBF بود.

با استفاده از توابع fit و score به آموزش و ارزیابی مدل SVM می‌پردازیم. نتایج به صورت زیر است:

```
Train accuracy: 100.0 %
Test accuracy: 34.6723044397463 %
```

همانطور که مشاهده می‌شود در این قسمت نیز دچار Overfit شدید هستیم. در واقع دقت آموزش برابر 100 درصد شده است و اختلاف دقت آموزش و تست برابر 66 درصد است. دلیل این Overfit مشابه با قسمت ب است. یعنی مشکل ثابت و Freeze بودن وزن‌های پایه شبکه می‌باشد زیرا مدل تنها می‌تواند وزن‌های لایه SVM را تغییر دهد و به قدری وزن‌های این لایه را تغییر می‌دهد که داده آموزشی حفظ می‌شود. طبیعتا اگر وزن‌های بیشتری قابل آموزش بودند مدل می‌توانست ویژگی‌های کلی‌تر و Generalتری آموزش ببیند و کمتر دچار مشکل Overfit شویم (بخش د).

ت) در این قسمت از مدل آموزش داده شده در قسمت ب یک کپی می‌گیریم تا وزن‌های لایه دسته‌بند نیز تصادفی نباشد و کمی آموزش دیده شده باشد. سپس آخرین لایه مدل پایه رزنت یعنی features[7] را قابل آموزش می‌کنیم. لایه دسته‌بند نیز از مدل قبلی قابل آموزش است.

تعداد پارامترهای مدل به صورت زیر است:

- Number of parameters: 23909636
- Number of trainable parameters: 15366340

اعداد بالا نشان می‌دهد که چگالی پارامترها در بلاک آخر مدل پایه رزنت یعنی features[7] بسیار زیاد است.

این بلاک از ۳ بلاک Bottleneck تشکیل شده است که هر یک دارای ۳ یا ۴ لایه Conv2D است.

در صفحه بعد ساختار این بلاک آورده شده است.

```
(7): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
```

در اپیاک اول نتیجه بسیار خوب است زیرا هم وزن‌های اولیه دسته‌بند و هم وزن‌های اولیه بلاک آخر شبکه رزنت از پیش آموخته هستند و مدل آموزش را از نقطه خوبی شروع می‌کند. همچنین با توجه به قابل آموزش بودن پارامترهای بیشتر مدل توانسته در جهت بهتر حرکت کند.

```
Epoch 0/14
----------
Iterating through data...
train Loss: 76.5675 Acc: 40.6250
Iterating through data...
val Loss: 59.8551 Acc: 49.1071
```

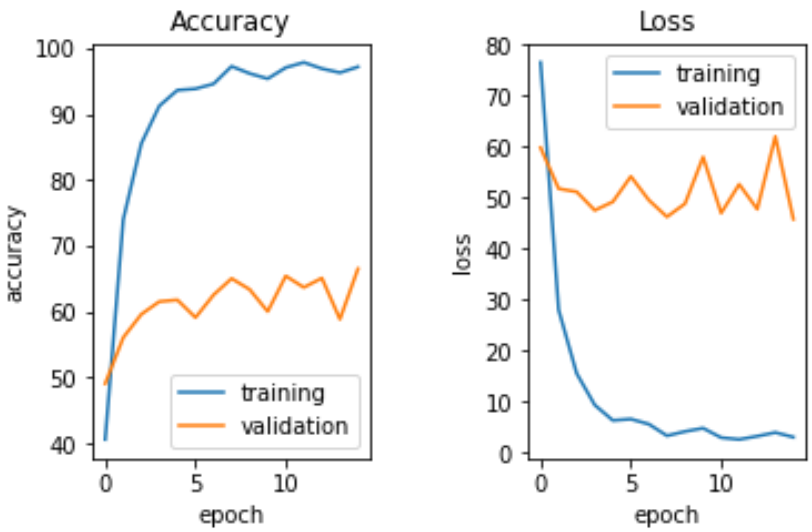
دقت در آموزش برابر 40.62 درصد و در تست برابر 49.10 درصد بوده که برای شروع بسیار مناسب است.

عملکرد مدل در پایان آموزش نیز به صورت زیر است:

```
Epoch 14/14
----------
Iterating through data...
train Loss: 3.1025 Acc: 97.2181
Iterating through data...
val Loss: 45.8368 Acc: 66.5551

Training complete in 64m 14s
Best val Acc: 66.555061
```

دقت مدل در پایان آموزش روی داده آموزشی برابر 97 درصد و روی داده تست برابر 66 درصد بوده است که نتیجه بسیار بهتری نسبت حالت‌های قبلی است. همچنان مدل دارای مشکل Overfit است که به دلیل استفاده نکردن از منظم‌سازی و لایه Dropout رخ داده است.

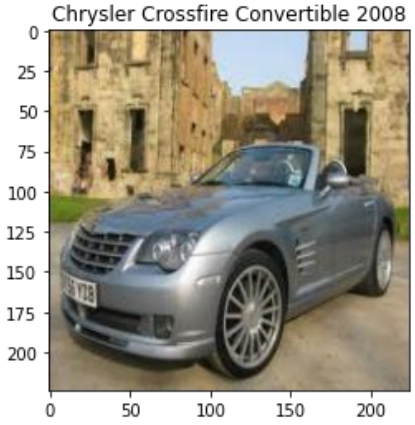


نمودار Loss و Accuracy نشان می‌دهد علارقم اختلاف زیاد فاز آموزش و تست، مدل همچنان در جهت بهبود عملکرد روی داده Validation در حال حرکت بوده است و اگر مدت زمان بیشتری آموزش را پیش می‌بدیم دقت Validation بیشتر نیز می‌شد.

عملکرد مدل در این قسمت از همه حالات بهتر شد زیرا وظیفه استخراج اطلاعات سطح پایین را به مدل پایه از پیش آموخته رزنت سپردیم و دیگر تغییری روی وزن‌های لایه‌های ابتدایی انجام ندادیم زیرا این لایه‌های ابتدایی ویژگی‌های سطح پایینی مانند لبه‌ها، رنگ‌ها و ... که در میان تمام تسک‌های پردازش تصویر مشترک هستند را استخراج می‌کنند و بدون نگرانی می‌توان دانش از پیش آموخته آن‌ها در تسکی دیگر منتقل کرد.

عامل دیگر موثر در عملکرد این مدل قابل آموزش قرار دادن لایه‌های انتهایی مدل بود که باعث شد ویژگی‌های سطح بالا مخصوص این مسئله استخراج شوند نه مسئله ImageNet (حذف باياس و ذهنیت). در واقع با Fine tune کردن این لایه‌ها علاوه بر اینکه از یک نقطه خوب شروع کردیم امکان حرکت به نقطه مناسب‌تر را نیز ایجاد کردیم.

ث) تصویر زیر را به عنوان ورودی برای تست این قسمت استفاده می‌کنیم:



عملکرد مدل بخش‌های الف، ب، ت به صورت زیر است:

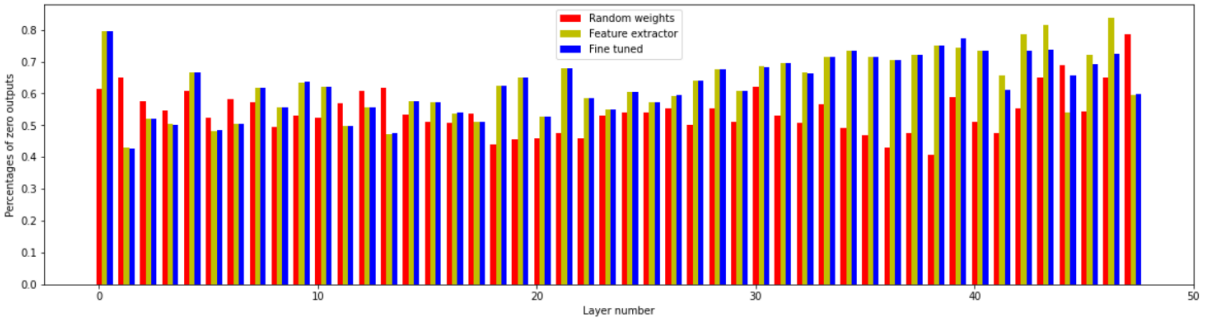
```
Model A predicted class: Acura TL Type-S 2008
Confidence: 2.920487642288208

Model B predicted class: Chrysler Crossfire Convertible 2008
Confidence: -2.653465509414673

Model D predicted class: Chrysler Crossfire Convertible 2008
Confidence: 12.835813522338867
```

همانطور که مشاهده می‌شود عملکرد مدل Fine tune شده یعنی بخش ت از همه بهتر بوده است.

سپس نسبت خروجی‌های صفر به کل خروجی‌ها در هر لایه را در ۳ مدل بالا به دست می‌آوریم و نمودار میله‌ای مربوط به آن را رسم می‌کنیم:



همانطور که از نمودار پیداست تعداد صفرها در لایه‌های ابتدایی دو مدل Feature extractor و Fine tuned یکسان است زیرا هر دو مدل وزن‌های لایه‌های ابتدایی را Freeze می‌کنند بنابراین خروجی در آن لایه‌ها یکسان می‌شود.

در مقایسه دو مدل Feature extractor و Fine tuned در لایه‌های انتهایی می‌توان گفت تعداد صفرها در مدل Feature extractor بیشتر است. دلیل آن این است که تعداد کلاس‌ها در مسئله ImageNet بسیار زیاد است بنابراین ویژگی‌های زیادی باید صفر باشند تا انحصار ویژگی‌های برای کلاس‌ها حفظ شود. اما در مسئله ما تعداد کلاس‌ها کمتر است در نتیجه می‌توان ویژگی‌های مشترک‌تری میان کلاس‌ها پیدا کرد و انحصار ویژگی‌ها کمتر است.

درباره تعداد صفرهای مدل Random weights می‌توان گفت چون مدل هنوز به نقطه مناسبی از جهت دقت نرسیده اکثر وزن‌ها همچنان دارای ویژگی تصادفی بودن هستند بنابراین خروجی لایه‌ها نیز از این تصادفی بودن برخوردار است. برای مثال در لایه اول تعداد صفرها نسبت به دو مدل دیگر کمتر است اما در ۳ لایه بعدی تعداد صفرها از دو مدل دیگر است. این نامنظمی تا انتها ادامه می‌یابد. تقریبا از لایه ۱۵ تا لایه انتهایی تعداد صفرها در این مدل نسبت به دو مدل دیگر کمتر است زیرا مدل همچنان نیاموخته که ویژگی‌های سطح بالا برای هر کلاس چه هستند و نمی‌تواند بگوید که این عکس دارای این ویژگی‌های سطح بالا نیست.

مقایسه نتایج

Model	زمان آموزش وابسته به تعداد پارامتر قابل آموزش	Train	Test
Random weights	خیلی زیاد	loss: 119.7486 acc: 12.7941	loss: 134.4328 acc: 8.4573
Feature extractor + NN	متوسط	loss: 22.2534 acc: 83.1127	loss: 91.8879 acc: 35.7515
Feature extractor + SVM	کم	acc: 100.0	acc: 34.67
Fine tuned	زیاد	loss: 3.1025 acc: 97.2181	loss: 45.8368 acc: 66.5551

منابع

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

https://www.pyimagesearch.com/2021/10/11/pytorch-transfer-learning-and-image-classification/

https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html