به نام خدا

ملیکا احمدی رنجبر 97521036

1)

a. Train با استفاده از Random Weights: در این قسمت تنها کافی است Pre-Trained Attribute را از True به False تبدیل کنیم، تا مقادیر به صورت Random باشند. از طرفی تعداد Epoch را به 25 تغییر می‌دهیم. پس از 130 دقیقه آموزش، دقت آموزش به 7.88% و دقت Validation نیز در بهترین حالت به حدود 3.22% رسید. با توجه به این نکته که تعداد Classها در این مسئله بسیار

```
Epoch 24/24
----------
Iterating through data...
train Loss: 80.8459 Acc: 0.0788
Iterating through data...
val Loss: 155.3084 Acc: 0.0315

Training complete in 129m 4s
Best val Acc: 0.032210
```

زیاد است، میزان دقت کم با توجه به تعداد Epoch عجیب نیست. 196 Class برای Cars داریم و تنها در 25 Epochs آموزش دیدند، از طرفی تعداد Parameters نیز بسیار زیاد است که با این تعداد Epoch کم مقدار، Converge نمی‌کند. (این سوال پیش از انجام تغییرات بر روی کد ابتدایی انجام شد، بنابراین نکته مهم آن تمرکز بر روی میزان Loss پس از این تعداد Epoch است، که در صورت مقایسه با سوال بعد متوجه تغییرات مقدار Loss در انتها می‌شویم.)

تعداد Fully-Connected Layers را نیز تغییر ندادیم و همان است. و برای نتیجه‌گیری کلی از این قسمت باید گفت، Final Result به دلیل تعداد زیاد Parameters و تعداد کم Epoch مناسب نیس

b. در این قسمت ما Pre-Trained = True قرار داده، و تمامی Parameterهای Train شده را تعیین می‌کنیم که دیگر تغییر نکنند (Non-Trainable) شوند.

```
res_mod = models.resnet50(pretrained=True)
for p in res_mod.parameters():
    p.requires_grad = False
```

```
self.classifier = nn.Sequential(
    nn.Linear(2048, 512),
    nn.Linear(512, 256),
    nn.Linear(256, num_classes)
)
```

لایه انتهایی Fully-Connected، Trainable است. به عنوان یک شبکه ساده برای این مدل و همان را در طول مدت آموزش می‌دهیم (دارای 3 لایه).

```
params_to_update = []
for name,param in model.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

model = model.to(device)
criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.Adam(params_to_update, lr=0.001)
```

برای قسمت ورودی به Optimizer نیز تنها Parameterهایی را به آن می‌دهیم که Trainable باشند. به این ترتیب دوباره پس از گذشت 130 دقیقه نتایج نهایی حاصل از Train و Test به شکل زیر بود، که همانطور که می‌بینیم، بسیار بیشتر و بهتر از حالت Random است، چراکه مقادیر Weights با توجه به شبکه ResNet تا حدی آماده بوده‌است، و Featureهای داده‌های Train شده در آن شبکه را بر روی Cars Dataset پیاده‌سازی کرده، و ما فقط لایه نهایی را Train می کنیم. و برای Classify کردن نیز از همان تابع پیش‌فرض موجود استفاده کردم. به طور دقیق‌تر، یعنی ما تمام Featureهایی که در ResNet Model به دست آمده را بر روی این Dataset استفاده کردیم، و تنها لایه Classifier آن را که تشخیص نهایی را برای دسته‌بندی می‌دهد را Train کردیم و دقت به نسبت خوبی گرفتیم (اما توجه شود که Parameterهای مدل ResNet، Non-Trainable کردیم و تنها 3 لایه آخر قابل آموزش هستند).

تغییرات Loss و Accuracy را در زیر مشاهده می‌کنیم:

```
Epoch 0/24
----------
Iterating through data...
/usr/local/lib/python3.7/dist-package
  "https://pytorch.org/docs/stable/op
train Loss: 143.0815 Acc: 7.7941
Iterating through data...
val Loss: 120.2469 Acc: 15.2530
```

```
Epoch 24/24
----------
Iterating through data...
train Loss: 23.8730 Acc: 81.1152
Iterating through data...
val Loss: 79.6089 Acc: 41.3939

Training complete in 110m 39s
Best val Acc: 41.790676
```

نکته قابل توجه در این آموزش Converge شدن سریع‌تر است، و پس از Epoch = 15 تقریبا میزان دقت و Loss برای هر دو Train و Test پیشرفتی نمی‌کند.

c. در این مرحله نوبت تغییر نوع Classifier است و تمام مراحلی که در تابع Train_Model برای آموزش مدل استفاده شده با توجه به SVM کمی تغییر می‌یابد. به این ترتیب که تنها Feature های حاصل از شبکه ResNet را در نظر داریم و نگه می‌داریم، اما برای Optimize کردن و محاسبه Loss و ... از مدل جدید SVM استفاده می‌کنیم. مراحل تغییر داده شده را به ترتیب مشاهده خواهیم کرد. اگر در حال Train باشیم، SVM.fit را خواهیم داشت، و به Featureهای Extract شده از ResNet Model نیز توجه داریم. بنابراین به طور کلی تنها Features را از ResNet گرفته، سپس آن را به Fit کرده، و در نهایت Accuracy را محاسبه خواهیم کرد. به شکل زیر توجه کنید:

```
for inputs, labels in dataloaders[phase]:
    inputs = inputs.to(device)
    labels = labels.to(device)
    # We need to zero the gradients, don't forget it
    features = []

    # Time to carry out the forward training poss
    # We only need to log the loss stats if we are in training phase
    # SOME CHANGES SHOULD BE APPLIED HERE
    with torch.set_grad_enabled(phase == 'train'):
        outputs = model(inputs)
        for f in range(outputs.cpu().shape[0]):
            features.append(np.array(outputs[f].cpu()))
        # I WILL USE SVM INSTEAD OF THE NORMAL OPTIMIZER
        if phase == 'train':
            clf.fit(features, labels.cpu())

    preds = clf.predict(features)
    for i in range(outputs.cpu().shape[0]):
        if preds[i] == labels[i].data:
            current_corrects += 1


epoch_acc = current_corrects/dataset_sizes[phase]
```

نتایج حاصل از این Classifier نیز اصلا مناسب نبود به دلیل Over Fit شدن بیش از اندازه این شبکه، با توجه به دقت بسیار بالا برای حالت Train مقدار Test Accuracy بسیار بسیار پایین بود. و به دلیل سرعت پایین، بیش از اندازه Run کردن این سوال (بالغ بر 200 بار) علاوه بر تمام شدن GPU در هر دو Account، تعداد Epoch را برای حل ادامه آن 10 گذاشتم :)

توجه شود که سوال 1 به طور جداگانه نوتبوک ندارد، اما توضیح کامل در Report هست. سوال دو نیز جداگانه ندارد، و برای سوال 4 که ادامه آن است یک نوتبوک داریم، همچنین برای 3 و5.

```
Epoch 0/9
----------
Iterating through data...
train Acc: 97.9779
Iterating through data...
val Acc: 0.5952
```

دلیل این مقدار Test Accuracy را هرچند بیش از 100 بار تنها این سوال Run شد، متوجه نشدم!(((((((((((((((((((((:

d. در این قسمت ادامه سوال 2 را داریم، که Model آن را Save کرده، و همان را Load نمودنم. پس از این کار، تعدادی از Parameterهایی که در سوال دو تماما Non-Trainable کرده بودیم را دوباره Trainable کرده، و تمامی این Trainableها را دوباره آموزش می‌دهیم. متاسفانه با اینکه تمام Model را Save کرده بودم اما در هنگام Load شدن مشکل برایش ایجاد شد و فایل را قبول نکرد. بنابراین با توجه به کمبود وقت، بار دیگر، اما با تعداد Epoch 5 مدل Train شد و مراحل ذکر شده در بالا انجام شد. اما صرفا برای تست کردن Fine Tuning این کد اجرا شد، چون با توجه به تعداد کم Epoch نتایج مناسب نیست.

```
Epoch 4/4
----------
Iterating through data...
train Loss: 223.7831 Acc: 42.7206
Iterating through data...
val Loss: 312.2455 Acc: 30.6548

Training complete in 20m 40s
Best val Acc: 32.105656
```

** نتیجه بهتر برای قسمت دو در قبل آورده شده، این تنها برای انجام قسمت 4 است**

حال تعدادی از Parameterهای نهایی در مدل ResNet را Trainable کرده و دوباره Train می‌کنیم. به شکل زیر:

```python
x=0
for p in res_mod.parameters():
    if x > 140:
        p.requires_grad = True
    x += 1

params_to_update = []
for name,param in model.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)
```

در این مرحله که دوباره برخی از لایه‌ها را Trainable کردیم، آن را در 5 مرحله آموزش دادیم و نتایج حاصل به شکل زیر است:

```
Epoch 9/9
----------
Iterating through data...
train Loss: 381.1152 Acc: 12.4510
Iterating through data...
val Loss: 420.5701 Acc: 8.5565

Training complete in 42m 0s
Best val Acc: 8.556548
```

به دلیل یک اشتباه در کد، قسمت Tuning درست Train نشد، پس از اجرای دوباره و رفع مشکل برای قسمت دوم، GPU برای Account دوم نیز تمام شد و امکان تست درست Fine Tuning را ندارم اما توضیحات به طور کامل آورده شده.

تغییر خاصی در نحوه آموزش پیش نیامد. البته منطقا به دلیل متناسب شدن بیشتر وزن‌ها با Input Data ما کمی پیشرفت قابل انتظار است، اما به دلیل تعداد کم Epoch و زمان زیادی که نیاز به دارد، نمی‌توان نتایج را آنگونه که صحیح است نمایش داد. اما به دلیل آنکه لایه‌های Feature Extraction در واقع متانسب با Loss خروجی بیشتر تنظیم می‌شوند، نتایج مطلوب‌تری خواهد داشت. در واقع تاثیر مثبت بر روی Hyper Parameters نیز است. نکته بسیار مهمی که در این سوال مطرح است،آن است که در سوال دوم از 25 Epochs استفاده شده بود و جواب نیز در قسمت‌های قبلی قابل مشاهده است. اما در اینجا چون Model قبلی Retrieve نشد، با 5 Epochs هم قسمت دو هم قسمت 4 آموزش داده شد. از طرفی قسمت 5 نیز در هر دوی این مراحل انجام شد. نمودار آن را در قسمت بعدی مشاهده خواهید کرد.

e. در این قسمت نیز باید نمودار میزان تعداد Feature و Output برابر 0 را نسبت به هر لایه بررسی میکردیم، اما باز به دلیل تمام شدن GPU موفق به دیدن نتیجه درست نشدم:)
روش کلی به این صورت است، که یک Input را Random انتخاب کرده، Featureهای آن را برای هر لایه بدست آورده، و مقادیر موجود در آن را بررسی میکنیم، سپس درصد میزان 0 آن را حساب میکنیم. اگرچه کد دیگر قابل اجرا نیست بدون GPU، اما کدی موجود (با وجود ایرادات وارد) نمای کلی کاری است که باید انجام داده شود، که در واقع انتهای هر لایه مقادیر موجود در آن را حساب و ذخیره میکند:

```
randInput = iter(train_data_loader)
x_sample, y_sample = next(randInput)
# GPU USAGE
x_sample = x_sample.to(device)

firstElement = x_sample[0]
sample = firstElement.reshape(1, 3, 224, 224)

sample = sample.to(device)
features = {}
def get_features(name):
    def hook(model, input, output):
        features[name] = output.detach()
    return hook
model.register_forward_hook(get_features('feats'))
preds = model(sample)
  # placeholders
PREDS = []
FEATS = []

    # add feats and preds to lists
PREDS.append(preds.detach().cpu().numpy())
FEATS.append(features['feats'].cpu().numpy())

result = []
for i in range(len(FEATS)):
    result.append(len(FEATS[i]) - np.count_nonzero(FEATS[i])/len(FEATS[i]))

# plotting the points

plt.plot(result)
plt.xlabel('layers')
plt.show()
```

2) منابع:

a. https://androidkt.com/feature-extraction-from-an-image-using-pre-trained-pytorch-model/

b. https://kozodoi.me/python/deep%20learning/pytorch/tutorial/2021/05/27/extracting-features.html