سوال عملی تمرین سری هفتم یادگیری ماشین

آتوسا مالمیر چگینی

۹۷۱۰۶۲۵۱

توضیحات کد:

سل اول: در ابتدا به Google.Drive متصل می‌شویم تا بتوانیم از فایل kaggle.json که در آنجا save کرده‌ایم استفاده کنیم.

سل دوم: در این سل مسیری که در آن فایل kaggle.json را ذخیره کردیم را مشخص می‌کنیم.

سل سوم: در این سل به مسیر مشخص شده می‌رویم.

سل چهارم: dataset را از kaggle دانلود می‌کنیم و فایل زیپ‌شده‌ی آن در دایرکتوری مشخص شده قرار می‌گیرد.

سل پنجم: کتاب‌خانه‌های مورد نیاز را import می‌کنیم.

سل ششم: در اینجا image_size را برابر با ۲۱۶ و batch_size را برابر با ۱۶ و تعداد epoch ها را برابر با ۴۰ و learning_rate اولیه را برابر با ۰,۰۰۱ قرار می‌دهیم.

سل هفتم: در این سل کلاسی برای dataset ایجاد می‌کنیم که مسیر داده‌ها و نوع داده‌ها (train, test, valid) و transform ها را می‌گیرد و روی دیتا مورد نظر اعمال می‌کند. در تابع __getitem__ هم داده‌ها را به اندازه‌ی image_size تبدیل می‌کنیم و لیبل داده‌های نرمال را برابر با ۰ و anomaly ها را برابر با ۱ قرار می‌دهیم.

سل هشتم: در اینجا سه تا ترنسفورم ایجاد می‌کنیم تا بر روی داده‌های train اعمال کنیم.

سل نهم: دیتاست‌های train و test و valid را با استفاده از کلاس MyDataset() ایجاد می‌کنیم.

سل دهم: dataloader سه دیتاست را ایجاد می‌کنیم.

سل یازدهم: اینجا شبکه نورونی ما است که ساختار آن طبق موارد گفته شده در سوال است. سه بلوک residual دارد که هر کدام دو لایه convolutional دارند. در ابتدا یک کلاس برای بلوک residual داریم که سایز ورودی و خروجی را می‌گیرد. به علاوه سایز کرنل و downsample را هم می‌گیرد. دو لایه convolutional را می‌سازد و dropout هم دارد. در forward هم صرفا ورودی را وارد این شبکه می‌کنیم. در اینجا skip-connection را هم پیاده سازی می‌کنیم. سپس یک کلاس داریم که شبکه اصلی خود را آنجا می‌سازیم. سه تا بلوک residual می‌سازیم که سایز خروجی آنها به ترتیب ۱۶و ۳۲و ۶۴ است. این کلاس یک لیست padding را هم ورودی می‌گیرد که با تست کردن به دست آوردم تا مشکل سایز نخورد. در آخر یک لایه fully connected قرار دادم که سایز ورودی آن را از سایز خروجی residual block آخر به دست آوردم. خروجی این لایه سایز ۲ دارد زیرا دو تا کلاس داریم. در تابع make_layer اگر سایز ورودی و خروجی یکسان نبود (طبق گفته‌ی سوال) downsampling را اعمال می‌کنیم. خورد downsample در واقع یک لایه convolutional است. در تابع forward هم ورودی را وارد شبکه می‌کنیم. فقط باید حواسمان باشد که قبل از پاس دادن به fc حتما با استفاده از view شکل ورودی fc را دو بعدی کنیم.

سل دوازدهم: در اینجا ابتدا مدل را با سایز padding ۰ و ۲و ۶ برای بلوک‌های مختلف residual ایجاد می‌کنیم. سپس مدل را به cuda می‌بریم تا عملیات سریع‌تر انجام شود. بعد برای بهینه‌ساز از Adam استفاده می‌کنیم. تابعی به نام update_lr هم تعریف کرده‌ایم که learning rate را تغییر می‌دهد. (از این تابع استفاده می‌کنیم تا learning rate را در زمان اجرا به تدریج کمتر کنیم.) در خط بعدی current_learning_rate را برابر با ۰,۰۰۱ قرار می‌دهیم.

توجه کنید که ما از داده‌های valid استفاده می‌کنیم تا از overfit جلوگیری کنیم. در طی اجرا در هر epoch مدل را روی داده‌های valid تست می‌کنیم و هر مدلی که کمترین loss را روی داده‌های valid داشت ذخیره می‌کنیم. به همین دلیل در آخرین خط این سل یک مقدار خیلی زیاد را به min_valid_loss می‌دهیم.

سل سیزدهم: این سل در واقع مدل را train می‌کند. در هر epoch ابتدا داده‌های train را batch batch می‌گیریم. به مدل داده و loss را محاسبه می‌کنیم و با loss.backward() به ابتدای شبکه برمی‌گردیم و optimizer را آپدیت می‌کنیم. در هر ۵۰ تا استپ هم loss را روی داده‌های train چاپ می‌کنیم تا بتوانیم یادگیری شبکه را بررسی کنیم. بعد از هر epoch هم لاس را پرینت می‌کنیم. سپس مدل را روی داده‌های valid آزمایش می‌کنیم و لاس این داده‌ها را هم محاسبه می‌کنیم. در هر ۵ epoch مقدار loss روی داده‌های valid را نمایش می‌دهیم و نمودار تغییرات train_loss را می‌کشیم. در هر ۴ epoch هم با استفاده از تابع update_lr مقدار learning_rate را تقسیم بر ۳ می‌کنیم. و بعد میبینیم اگر مقدار loss روی داده‌های valid کمتر شده بود min_valid_loss را آپدیت می‌کنیم و مدل را save می‌کنیم.

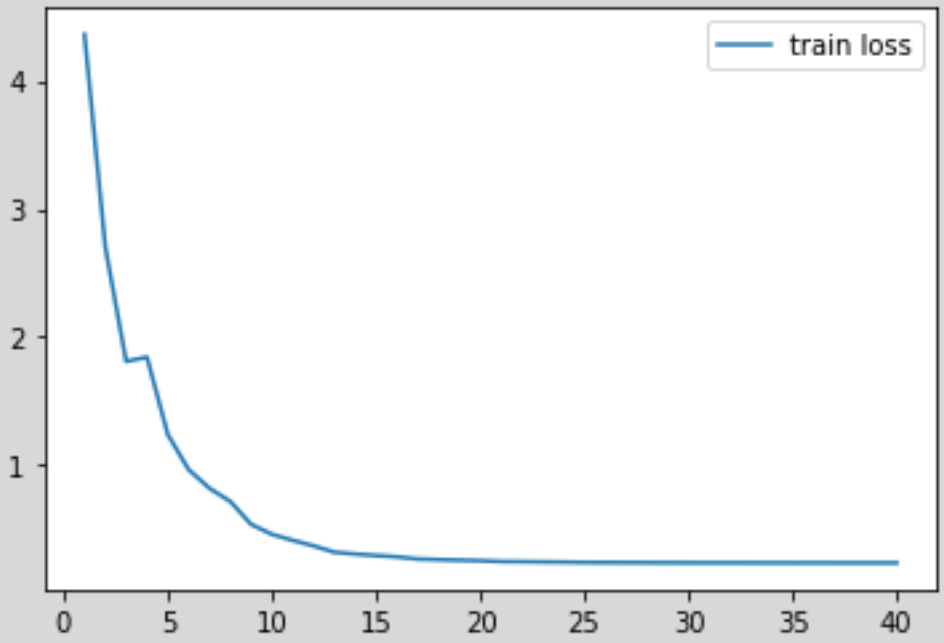
سل چهاردهم: دراینجا مدل save شده را load می‌کنیم و روی داده‌های تست امتحان می‌کنیم و مقدار loss و accuracy و confusion_matrix را چاپ می‌کنیم.

*توجه کنید که سل ۱۱ و ۱۲ و ۱۳ و ۱۴ برای مدل با dropout است.

سل ۱۵ و ۱۶ و ۱۷ و ۱۸ هم برای مدل بدون dropout است. (که دقیقا مشابه همین چهار تا سل بالا هستند با این تفاوت که dropout را در شبکه نداریم.)

مدل با dropout

همان طور که در صورت سوال گفته شد احتمال غیر فعال شدن هر نورون را برابر با ۰,۴ قرار می‌دهیم. آنگاه نتایج بعد از اتمام epoch ها به این صورت می‌شوند:




```
Epoch 40                    Train Loss: 0.2382928421020942
Epoch 40                    Validation Loss: 85.3968734741211
```

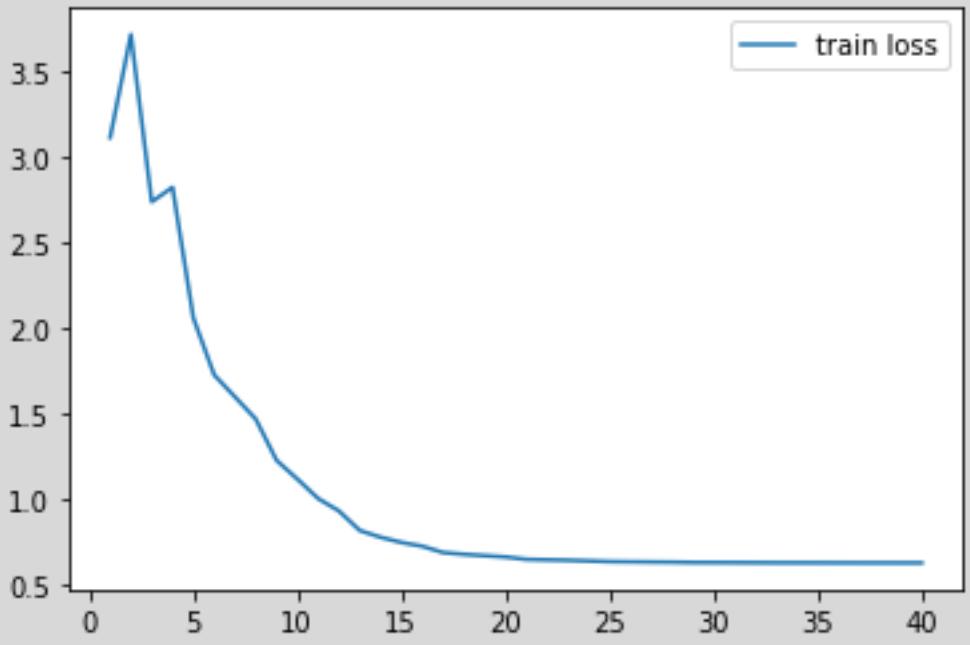
در مورد داده‌های تست هم داریم که:

```
Test Loss: 13.848788661798702
Accuracy on test dataset: 0.8974358974358974
Confusion matrix of test dataset is: [[ 172  62]
[   2 388]]
```

پس در این حالت دقت روی داده‌های تست ۰,۸۹ است.

مدل بدون dropout

در این حالت dropout را برمی‌داریم و مدل را بدون آن آموزش می‌دهیم. بعد از پایان فرآیند آموزش داریم:




```
Epoch 40                    Train Loss: 0.6281302218936795
Epoch 40                    Validation Loss: 46.978309631347656
```

در مورد داده‌های تست هم داریم که:

```
Test Loss: 18.85090729288566
Accuracy on test dataset: 0.8525641025641025
Confusion matrix of test dataset is: [[ 151  83]
 [   9 381]]
```

** همانطور که مشخص است با dropout دقت بیشتر می‌شود. چون dropout نوعی regularization است و از overfit شدن جلوگیری می‌کند.

** در ابتدا از binary_cross_entropy استفاده کردم و مشکل overfit اتفاق افتاد. به همین دلیل همانطور که سوال گفت از binary_cross_entropy_with_logits استفاده کردم که حالت وزن دار همان binary_cross_entropy است.

مکان قرارگیری dropout در مدل‌ها

با توجه به مقاله‌ی اصلی https://arxiv.org/pdf/1207.0580.pdf باید dropout را بعد از هر لایه‌ی fully connected قرار داد. که البته منطقی هم است چون این لایه‌ها نورون‌های زیادی دارند و ممکن است train کردن همه‌ی آنها باعث overfitting شود.

دلیل استفاده از transformation های موجود در notebook

به طور کلی دلیل استفاده از transformation های notebook مطالعه‌ی مقاله‌ی https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 در مورد بهترین transformation ها برای image در علم پزشکی بود.

در مورد تصاویر پزشکی از آنجایی که راستای قرارگیری تصویر نشان دهنده‌ی اطلاعات خاصی نیست، (بر عکس نثلا mnist که راستای قرارگیری تصویر درواقع نشان‌دهنده‌ی label است) میتوان برای افزایش قدرت مدل این تغییر جهت ها در راستای محور افقی و عمودی را انجام داد. به علاوه در مقاله‌ی بالا نشان داده شده است که این دو transformation با استفاده از مدل resnet که مشابه مدل ما هم است تاثیر خیلی خوبی روی accuracy روی داده‌های تست می‌گذارد (جدول نتایج این دو transformation را در https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0/tables/2 می‌توانید ببینید.)

در مورد random perspective هم بار هم به همین دلیل بالا قدرت مدل افزایش میابد: