بینایی کامپیوتری

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

تمرین سری پنجم

نام و نام‌خانوادگی: امین کشیری

شماره دانشجویی: ۹۷۱۰۱۰۲۶

توضیحات کلی

- فایل zip شده‌ی داده شده را در پوشه‌ای که کد قرار دارد extract کنید.
- دو پوشه به نام‌های outputs و models را هم رده‌ی فایل پایتون بسازید. در پوشه‌ی outputs نمودارها قرار می‌گیرند.
- کد من با نام HW۵ در فایل جواب قرار دارد.
- بعضی از خط‌ها در کد، دارای comment هستند، که با uncomment کردن آن‌ها معمولاً می‌توانید نتایج میانی ببینید (توضیحات بیشتر را در صورت لزوم، در توضیحات هر سوال داده‌ام).
- همراه فایل‌های یک فایل requirements.txt قرار دارد که محیط اجرای کدهای من است (در صورت نیاز).

سوال ۱

توضیحات کد

در ابتدای کد، تمام seed ها را می‌کاریم تا مقداری از تصادفی بودن کد را کاهش دهیم. از gc و empty_cache() برای کاهش فایل‌های اضافی روی مموری و کارت گرافیک استفاده می‌کنم. در چند خط بعدی، متغییر های اولیه مثل learning_rate و اندازه‌ی batch ها و تعداد ایپاک‌ها را ست می کنم. با استفاده از تابع get_data داده‌ها را load می‌کنم. روی داده‌های آموزش و تست تبدیل‌های متفاوتی اعمال می‌کنم که داده‌ها را augment کنم. البته روی داده‌های تست، کار خاصی انجام نمی‌شود (نیازی به زیاد کردن داده‌های تست نداریم). روی داده آموزش نیز تبدیل‌هایی مانند تغییر اندازه، Flip کردن افقی تصادفی، تبدیل آفین تصادفی و .. انجام می‌دهیم (توضیحات دقیق تر در قسمت نتایج). تبدیل نرمالایز کردن به این دلیل انجام شده است که تمام مدل‌های خود Pytorch با این نرمال کردن آموزش داده شده‌اند. ما نیز برای این که در مراحل بعدی از مدل آموزش داده شده استفاده می‌کنیم، و نیاز به نزدیک بودن ورودی به آن داده‌ها داریم از آن استفاده می‌کنیم.

از لاگر تنها برای استفاده‌ی خودم استفاده کرده‌ام. حال به ازای هر کدام از ۵ حالت خواسته شده مراحل زیر را انجام می‌دهم:

وظیفه‌ی آماده کردن اولیه هر کدام از ۵ مرحله را تابع all_models بر عهده دارد که یک generator است (در زبان python). این تابع هر بار ۳ متغییر برمی‌گرداند که متغییر اول شماره‌ی مرحله است. متغییر دوم خود مدل ما است (خود شبکه‌ی عصبی) و متغییر سوم پارامترهایی است که قرار است بهبود دهیم و train کنیم.

ساختار قسمت ۱ و ۲ و۳ را در ۳ کلاس AlexNet1 و AlexNet2 و AlexNetFull دقیقا مانند پیاده سازی خودش انجام داده‌ام. موردی که در اینجا وجود داشت این بود که ساختار Alexnet در Pytorch مقداری با همین ساختار در بقیه جاها متفاوت بود. اما من برای این که بتوانم نتایج از قبل train شده‌ی Pytorch را استفاده کنم و بتوانم نتیجه مراحل مختلفم را با هم مقایسه کنم، از همان ساختار استفاده کردم. برای حالت ۴، همانطوری که می‌بینید، مدل آماده‌شده را برداشتم، و لایه‌ی آخر آن را تغییر دادم (تنها ۱۵ خروجی) و وزن‌هایی که قرار است تغییر دهیم را تنها وزن‌های این لایه در نظر گرفتم (نه تمام وزن‌ها). برای حالت ۵ نیز دقیقا همین کار را کردم اما ایندفعه تمام وزن‌ها را تغییر می‌دهیم.

در هر مرحله از روی Dataset یک Dataloader برای داده‌های آموزش و تست می‌سازیم. سپس مدل را به gpu منتقل می‌کنیم (اگر موجود باشد) و تابع Loss و Optimization را نیز انتخاب می‌کنیم که در اینجا Cross Entropy و روش SGD هستند. ۴ تنسور بعدی نیز برای کشیدن نمودارها استفاده می شود. حال در هر ایپاک، به صورت زیر عمل می کنیم:

به ازای تمام داده های (به صورت batch) ابتدا حدس مدل خود را به دست می آوریم، سپس گرادیان ها را back propagate می کنیم، و به کمک Optimizer وزن ها را آپدیت می کنیم. سپس Loss را برای داده‌ی آموزش به دست می آوریم (با تقسیم کردن مجموع Loss ها بر تعداد ورودی‌ها). در حلقه‌ی بعدی کاری مشابه با این اما روی داده‌ی تست انجام می‌دهیم. بعد از پیش بینی کردن توسط مدل، یک بار تعداد پیش بینی‌های کاملا صحیح را به دست می آوریم و یک بار چک می کنیم که آیا پیش بینی در بین ۵ حدس اول وجود داشته است یا خیر (و برای این کار، از ۵ تا پیش بینی برای هر ورودی، Label واقعی را کم می کنم و اگر حتی ۱ درایه ۰ شود یعنی جواب اصلی در ۵ حدس اول ما وجود داشته است). در نهایت دقت های خواسته شده را به دست می آورم و نمودار ها را نیز می کشم که در پوشه ی outputs قرار می گیرند.

نتایج

۳ دسته از نتایج خودم را به صورت کلی یادداشت می‌کنم، و پس ازین ۳ بهترین نتایج خود را ذکر می‌کنم.

پارامتر learning rate را در محدوده‌ی 0.001 تغییر می‌دادم. monementum را نیز در حوالی 0.9 تغییر می‌دادم و بهترین نتیجه‌ها را با همین دو مقدار گرفتم (البته این که بیشتر اجراهای من با این پارامترها بوده است نیز بی‌تاثیر نیست :)). بناراین برای تمام حالات زیر هایپرپارامترهای بالا همین مقدار را دارند.

به دلیل این که مقداری augmentation تصادفی روی داده انجام می‌شود و به این دلیل که در محاسبه‌ی loss اندازه‌ی batch ها دخیل می‌شود، در کد من اندازه‌ی batch ها نیز تاثیر داشت (متناسب با تعداد epoch ها، یعنی برای اندازه‌های مختلف batch تعداد epoch های متفاوتی نتیجه‌های مشابه می‌دادند).

گزارش دقیق تر هر کدام از سری نتایج زیر در فایل log.log همراه با پوشه‌ی آن وجود دارد (مقدار دقیق loss و accuracy در هر مرحله ...)

نتایج ۱

یکی از روش‌هایی که داده‌ها را augment کردم حالتی بود که ابتدا عکس‌ها را به اندازه‌ی کمی بزرگتر از ۲۲۴ ریسایز می‌کردم، سپس یک تبدیل آفین و یک فلیپ به تصادف روی‌آن‌ها اعمال می‌کردم و در نهایت یک مربع با ضلع ۲۲۴ از وسط آن می‌بردیم (و البته در این حالت باید ریسایز کردن به همان سایز و کراپ کردن نهایی را روی داده‌ی تست نیز انجام دهیم تا نتیجه معقول باشد). در این حالت ، بقیه متغییرها به صورت:

$$epochs = 30$$
$$batch\ size = 50, 50, 20, 40$$

نتایج این حالت را می‌توانید در پوشه‌ای به نام res۱ ببینید. در این حالت، بهترین نتیجه‌ای که رسیدم (به ترتیب top۱ و top۵):

$$part1 \Rightarrow 62\% - 94\%$$
$$part2 \Rightarrow 57\% - 94\%$$
$$part3 \Rightarrow 64\% - 96\%$$
$$part4 \Rightarrow 86\% - 99.6\%$$
$$part5 \Rightarrow 88.5\% - 99.5\%$$

البته همان‌طور که در نمودارها می‌بینید، قسمت ۴ و ۵ خیلی سریع overfit شده‌اند و به بیشینه‌ی دقت خود رسیده‌اند.

نتایج ۲

در این حالت از تبدیل‌ها آفین و کراپ کردن استفاده نکردم بلکه تنها از فلیپ کردن تصادفی و equalize کردن تصادفی تصاویر استفاده کردم (equalize کردن هیستوگرام تصویر به صورت احتمالی). در این حالت ، بقیه متغییرها به صورت:

$$epochs = 40$$
$$batch\ size = 10, 10, 10, 10$$

نتایج این حالت را می‌توانید در پوشه‌ای به نام res۲ ببینید. در این حالت، بهترین نتیجه‌ای که رسیدم (به ترتیب top۱ و top۵):

$$part1 \Rightarrow 67\% - 94.5\%$$
$$part2 \Rightarrow 72\% - 97.3\%$$

$part3 \Rightarrow 73\% - 97.9\%$
$part4 \Rightarrow 85.5\% - 99.6\%$
$part5 \Rightarrow 87.4\% - 99.5\%$

در این حالت دوم همانطور که می‌بینید، نتایج قسمت‌های ۱ تا ۳ به صورت قابل توجهی بهتر شده است اما دو قسمت آخر تغییر محسوس نکرده‌اند و حتی بدتر شده‌اند. همانطور که انتظار می‌رفت مدل ساده‌ای مثل حالت ۱ خیلی سریع overfit شده‌است، بعد از ۱۰ ۱۵ ایپاک، مدل ۲ کمی دیرتر از آن و مدل ۳ نیز همانطوری که مشخص است بعد از ۴۰ ایپاک هنوز overfit نشده بود و اگر فرصت داشتم تا آن را بیشتر اجرا کنم احتمالا نتایج بهتری نیز نمی‌توانستم بگیرم. اما مدل ۴ و ۵ نیز مثل حالت قبل سریع به حد بیشینه‌ی خود می‌رسند).

نتایج ۳

در این حالت transform ها دقیقا مانند حالت ۲ است. تنها تعداد ایپاک‌ها را تا ۶۰ بالا بردم و batch_size ها را روی ۱۶ قرار دادم.

$epochs = 60$
$batch\ size = 16, 16, 16, 16$

نتایج در پوشه‌ی res۳ و به صورت زیر:

$part1 \Rightarrow 66.5\% - 94.1\%$
$part2 \Rightarrow 73\% - 98\%$
$part3 \Rightarrow 75\% - 98\%$
$part4 \Rightarrow 86.5\% - 99.6\%$
$part5 \Rightarrow 89\% - 99.7\%$

نتایج این سری نیز مقداری شبیه به حالت قبلی بودند. باز هم حالت ۱ سریع تر overfit شد. حالت دوم کمی دیرتر (نزدیک ایپاک ۳۰) و حالت ۳ نیز تقریبا در آخرین ایپاک‌ها به بیشینه میزان خود رسید. باز هم نتایج حالت ۴ و ۵ مشابه حالت‌های گذشته بودند.

بقیه تلاش‌ها

در نهایت، این سه سری نتایجی بودند که من به صورت کامل رکورد کردم و یادداشت کردم. در تلاش‌های جداگانه برای حالت‌های مختلف، راهکارهای ابتکاری دیگری مثل موارد بالا نیز به کار برده بودم، مثل اضافه کردن لایه‌هایی دیگر با توابع activation متفاوت و یا اضافه کردن dropout بیشتر، که گاهی توانستند در بیشترین حالت مقدار کمی از نتایج ذکر شده بهتر شوند. اما به صورت کلی من نخواستم مرز بین حالت‌ها را از بین ببرم (مثلا تعداد زیادی لایه به حالت ۱ و ۲ اضافه کردم و دقت آن تا ۷۰ و ۷۵ بالا رفت)، اما بعد حس کردم که پیچیدگی آن‌ها دارد به حالت بعدی خود نزدیک می‌شود و در نهایت تصمیم گرفتم در چهارچوب شبکه‌هایی که در ۵ حالت سوال ذکر شده با متغییرها و تعریف شبکه‌ها بازی کنم (مثلا برای قسمت ۲م و ۳م تعداد ایپاک‌ها را بسیار بالاتر بردم). در نهایت، بهترین نتایجی که توانستم کسب کنم حدود زیر را داشت:

$part1 \Rightarrow 68/69\% - 94/95\%$
$part2 \Rightarrow 75/76\% - 98/99\%$
$part3 \Rightarrow 78/79\% - 98/99\%$
$part4 \Rightarrow 87/88\% - 99.7\%$
$part5 \Rightarrow 89/90\% - 99.9\%$