به نام خدا

درس مبانی بینایی کامپیوتر

تمرین سری یازدهم

مدرس درس:
جناب آقای دکتر محمدی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۱/۶

سوال ۱ :

به سوالات زیر پاسخ دهید : (۳۰ نمره) (زمان: ۱۵۰ دقیقه)

الف) چرا برای پردازش تصویر لایه‌های کانولوشنی که پارامترهای کمتری نسبت به لایه‌های fully connected دارند؛ عملکرد بهتری دارند؟(۳ نمره) (زمان: ۱۵ دقیقه)

ب) فرض کنید یک حجم ۱۶ در ۱۶ در ۵ وارد یک لایه کانوولوشنی که ۱۶ فیلتر ۵ در ۵ دارد، می‌شود. مقدار گسترش مرزها چقدر باید باشد تا طول و عرض خروجی این لایه تفاوتی نکند. تعداد پارامترهای این لایه را نیز به دست آورید. (۷ نمره) (زمان: ۱۵ دقیقه)

پ) اگر یک تصویر ۳ کاناله با ابعاد ۳۲ در ۳۲ وارد یک لایه کانوولوشنی با ۳ فیلتر ۵ در ۵ بدون صفرافزونه و با اندازه گام ۱ شود ابعاد خروجی چه خواهد شد؟ اگر همان تصویر را به دو لایه کانولوشنی که هر دو ۹ فیلتر ۳ در ۳ بدون صفر افزونه و اندازه گام ۱ دارند بدهیم، ابعاد خروجی چه خواهد شد؟ (۷ نمره) (زمان: ۱۰ دقیقه)

ت) هر کدام از max(min) pooling، average pooling و global avarage pooling در چه تسک‌هایی بهتر است استفاده شوند؟ چرا؟ (۵ نمره) (زمان: ۶۰ دقیقه)

ث) مقایسه‌ای بین Resnet و VGG-net انجام دهید. به جز عامل تعداد پارامترها چرا Resnet سریع‌تر از VGG است؟ ایده خاص هر کدام برای بهبود مدل‌های کانولوشنی چیست؟ (۸ نمره) (زمان: ۶۰ دقیقه)

پاسخ ۱ :

الف) convolutionها به طور کامل به هم متصل نیستند، همه nodeهای ورودی بر همه nodeهای خروجی تأثیر نمی‌گذارند. این به لایه‌های کانولوشن انعطاف بیشتری در یادگیری می‌دهد. علاوه بر این، تعداد وزن‌ها در هر لایه بسیار کمتر است، که به ورودی‌های با ابعاد بالا مانند داده‌های تصویر کمک زیادی می‌کند. همچنین اگرچه شبکه‌های کاملاً متصل هیچ فرضی در مورد ورودی ندارند، اما عملکرد ضعیف‌تری دارند و برای استخراج ویژگی خوب نیستند. به علاوه، آن‌ها تعداد وزن‌های بیشتری برای train دارند که منجر به زمان train زیاد می شود، در حالی که از طرف دیگر CNN ها برای شناسایی و استخراج بهترین ویژگی‌ها از تصاویر برای مشکل مورد نظر با پارامترهای نسبتاً کمتری آموزش دیده‌اند. برای استخراج local ویژگی‌ها نیز مکان مهم است و بنابراین باز هم نتیجه

می‌گیریم شبکه‌های کانولوشنی بهتر هستند

ب) با اعمال لایه کانولوشنی 16 * 5 * 5 به تصویر 5 * 16 * 16، می‌دانیم تصویر حاصل، ابعاد مکانی‌اش به اندازه ابعاد لایه کانولوشنی به علاوه ۱ کاهش پیدا می‌کند. بنابراین ابعاد تصویر حاصل 16 * 12 * 12 می‌شود که برای اینکه طول و عرض تغییر نکند، باید padding به اندازه 4 * 4 بدهیم که معادل ۴ سطر افقی و ۴ سطر عمودی می‌باشد. همچنین برای به دست آوردن تعداد پارامترهای این لایه، ابعاد فیلتر را در عمق تصویر ضرب، به علاوه bias و در تعداد فیلترها ضرب می‌کنیم. بنابراین داریم:

$$The\ number\ of\ parameters = 16 * (5 * 5 * 5 + 1) = 2016$$

پ) با وارد کردن تصویر با ابعاد 5 * 32 * 32 به لایه کانولوشنی با ابعاد 3 * 5 * 5، ابعاد تصویر حاصل، 3 * 28 * 28 می‌شود و چون stride یک است، تغییر دیگری در ابعاد نداریم.

در حالت بعد نیز چون باز هم stride یک است، تاثیری ندارد؛ اما با وارد کردن تصویر * 32 * 32 3 به لایه کانولوشنی اول با ابعاد 9 * 3 * 3، ابعاد تصویر 9 * 30 * 30 می‌شود و با ورود به لایه دوم کانولوشنی، ابعاد نهایی تصویر، 9 * 28 * 28 می‌شود.

ت) Max Pooling پیکسل‌های روشن‌تر را از تصویر انتخاب می کند. زمانی مفید است که پس‌زمینه تصویر تیره است و ما فقط به پیکسل‌های روشن‌تر تصویر اهمیت می‌دهیم. به عنوان مثال: در مجموعه داده MNIST، ارقام به رنگ سفید و پس‌زمینه سیاه نشان داده می‌شوند. بنابراین، Max Pooling استفاده می‌شود. به طور مشابه، Min Pooling به شکلی دیگر استفاده می‌شود و پیکسل‌های تیره را از تصویر انتخاب می‌کند. بنابراین، از Min Pooling نیز زمانی که پس‌زمینه سفید است، استفاده می‌کنیم. به طور کلی، Max(Min) Pooling، تصویر واضح‌تری را ارائه می‌دهد که بر روی حداکثر(حداقل) مقادیر متمرکز شده است. Max Pooling تنها برجسته ترین ویژگی های داده ها را استخراج می کند.

Average Pooling یک عملیات pooling است که مقدار میانگین patchهای یک نقشه ویژگی را محاسبه می‌کند و از آن برای ایجاد یک نقشه ویژگی downsampled (pooled) استفاده می‌کند. معمولاً بعد از لایه کانولوشن استفاده می‌شود. ویژگی‌ها را راحت‌تر از Pooling Max استخراج می‌کند، در حالی که pooling max ویژگی‌های برجسته‌تری مانند لبه‌ها را استخراج می‌کند. روش Average Pooling تصویر صاف تری ارائه می دهد که ماهیت ویژگی های تصویر را حفظ می کند

و از این رو ممکن است هنگام استفاده از این روش pooling، ویژگی های واضح شناسایی نشود.
Global Average Pooling، ایده میانگین‌گیری از هر یک از ویژگی‌ها برای کاهش ابعاد را مطرح می‌کند. بنابراین برخلاف لایه Flatten، تعداد پارامترها را خیلی کم می‌کند. اما نکته قابل توجه در این روش این است که دیگر مکان را از دست می‌دهیم. در واقع در این روش خیلی از اطلاعات را از دست می‌دهیم اما مزیت آن، کاهش حجم می‌باشد و از overfitting جلوگیری می‌کند. در شبکه‌های عصبی اخیر، بعد از GoogleNet، معمولا از GAP استفاده می‌شود. یعنی اگر تعدادی نقشه ویژگی داشته باشیم، از آن‌ها میانگین می‌گیریم و به یک عدد تبدیل می‌شود. بنابراین، پس از GAP، یک لایه کاملا متصل (Dense) داریم که می‌خواهد دسته‌بندی را انجام دهد. از GAP، می‌توان در نقشه‌های نمایش فعالیت استفاده کرد.

ث) ResNet تفاوت خاصی با VGG-net نمی‌کند؛ فقط اتصالات residual را وصل کرده و وزن‌های مدل‌های کم‌عمق را به لایه‌های نخست شبکه عمیق کپی می‌کنیم و لایه‌های اضافی را به گونه‌ای تنظیم کرده که نگاشت همانی را انجام دهد. در واقع ResNet، به جای آموختن نگاشت مطلوب، باقی مانده را یاد می‌گیرد. یعنی تفاوت در اتصالات residual و قرار دادن یک لایه کانولوشنی در ابتدای آن است (بر خلاف VGG که دو تا فیلتر 3 * 3 داشت، این یک فیلتر 7 * 7 دارد). همچنین بر خلاف VGG که از Flatten استفاده می‌کرد، اینجا از Global Average Pooling استفاده می‌شود که تعداد پارامترها را تا حد زیادی کاهش می‌دهد.

از معایب VGG، می‌توان به vanishing graadient اشاره کرد که در هیچ مدل دیگری وجود ندارد و این مشکل با ResNet حل شد. همچنین VGG از ResNet سرعت کمتری دارد که ResNet با استفاده از مفهوم یادگیری residual این مشکل را حل کرد و یکی از دلایل سرعت بیشتر ResNet نسبت به VGG، اتصالات residual می‌باشد. معماری ResNet نیازی به فعال کردن همه نورون‌ها در هر دوره ندارد. این امر زمان تمرین را تا حد زیادی کاهش می‌دهد و دقت را بهبود می‌بخشد. وقتی یک ویژگی یاد گرفت، سعی نمی‌کند دوباره آن را یاد بگیرد، بلکه بر یادگیری ویژگی‌های جدیدتر تمرکز می‌کند. یک رویکرد بسیار هوشمند که عملکرد آموزش مدل را تا حد زیادی بهبود بخشید.

ایده ResNet آن است که لایه‌های شبکه بجای آموختن نگاشت مطلوب، باقی‌مانده آن را یاد بگیرند. ایده VGG نیز این است که از فیلترهای کوچک‌تر استفاده می‌کند، اما شبکه را عمیق‌تر می‌کند (تعداد لایه‌های بیشتر).

لینک‌های مورد استفاده برای حل این سوال:

https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5 https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9 https://paperswithcode.com/method/average-pooling#:~:text=Average%20Pooling%20is%20a%20pooling,used%20after%20a%20convolutional%20layer.

سوال ۲:

در فایل HW11.ipynb قطعه کدی آورده شده است. با اضافه کردن لایه‌ها و تابع فعال‌سازی مناسب برای لایه آخر با توجه به تابع ضرر تعریف شده در تابع compile ، مدل‌ها را طوری کامل کنید که مدل fully connected تقریبا دو برابر مدل کانولوشنی پارامتر داشته باشد. با اجرا گرفتن کامل از کد به موارد زیر پاسخ دهید. (۲۵ نمره) (زمان تخمینی: ۱۲۰ دقیقه)

الف) میزان خطا و دقت را روی داده‌های Test گزارش کنید. آیا خطای کمتر به معنای دقت بیشتر است؟

ب) مدت زمان اجرای هر ایپاک در دو مدل چقدر است؟

پ) آیا بین زمان و پارامترهای مدل‌ها ارتباط مستقیمی وجود دارد؟ چرا؟

پاسخ ۲:

الف) در تصویر زیر، میزان خطا و دقت روی داده‌های تست برای مدل fully connected مشاهده می‌کنید.

```
Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 1.6263 - accuracy: 0.4147
```

همچنین این مقادیر برای مدل convolutionی به شرح زیر است.

```
Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 0.9990 - accuracy: 0.6560
```

همانطور که در بخش الف سوال قبل توضیح دادیم، در اینجا دقت مدل کانولوشنی بیشتر و خطای آن کمتر می‌باشد.

در مورد خطا و دقت، هیچ رابطه‌ای بین این دو معیار وجود ندارد. loss را می‌توان به عنوان فاصله بین مقادیر واقعی مسئله و مقادیر پیش بینی شده توسط مدل در نظر گرفت. loss هر چه بیشتر باشد، خطاهایی که مدل در داده‌ها مرتکب شده است، بزرگ‌تر است. در واقع، loss function توسط مدل برای یادگیری استفاده می‌شود. دقت ساده‌تر است. با مقایسه پیش‌بینی‌های مدل با مقادیر واقعی بر حسب درصد، میزان پیش‌بینی مدل ما را اندازه‌گیری می‌کند. در واقع دقت یا accuracy، عملکرد مدل ما را بررسی می‌کند. در ادامه حالت‌های ممکن برای این دو مقدار را بررسی می‌کنیم.

داشتن دقت کم loss زیاد به این معنی است که مدل در بیشتر داده ها خطاهای بزرگی دارد.

اگر loss و هم دقت پایین باشد، به این معنی است که مدل در بیشتر داده‌ها خطاهای کوچکی دارد.

اگر هر دو بالا باشند، خطاهای بزرگی در برخی از داده‌ها ایجاد می‌کند.

اگر دقت بالا و loss کم باشد، مدل فقط در برخی از داده‌ها خطاهای کوچکی را مرتکب می‌شود که حالت ایده‌آل خواهد بود.

ب) در مدل fully connected مدت اجرای هر epoch، ۵ ثانیه و در مدل کانولوشنی، تقریبا ۸ ثانیه است.

پ) خیر؛ همانطور که در بخش قبل دیدیم، مدل fully connected با دارا بودن تقریبا دو برابر تعداد پارامترهای لایه کانولوشنی، زمان کمتری را برای هر epoch صرف می‌کرد که این موضوع نشان می‌دهد ارتباط مستقیمی بین آن‌ها نیست.

لینک‌های مورد استفاده در این سوال:

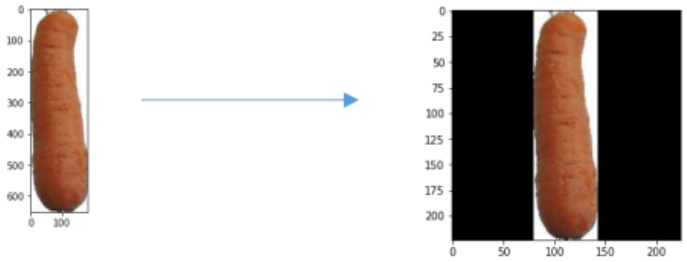
`https://www.baeldung.com/cs/ml-loss-accuracy`

سوال ۳:

در بخش Q3 مدل از قبل آموزش داده شده Resnet50 به روی دیتاست imagenet را به روی دیتاست میوه‌ها fine-tune کنید. (۴۵ نمره) (زمان: ۱۸۰ دقیقه)

الف) تصاویر این دیتاست ابعاد و طول و عرض برابر ندارند. برای این که بتوان تصاویر را به ورودی مدل داد باید تصاویر را resize کرد به طوری که ساختار تصویر میوه عوض نشود. در تابع

resize_img شما باید تصویر ورودی را به سایز مشخصی تبدیل کنید.



ب) مدل Resnet را یک بار بدون فریز کردن لایه‌ها و وزن‌های رندوم به روی دیتاست میوه‌ها آموزش دهید.

پ) این‌بار مدل Resnet آموزش دیده به روی Imagenet را load کنید، وزن‌های لایه‌ها را فریز کنید و با اضافه کردن لایه‌های مناسب، مدل را کامل کرده و به روی دیتاست مذکور آموزش دهید.

ت) نتایج به دست آمده را گزارش و مقایسه کنید.

پاسخ ۳:

الف) با استفاده از قطعه کد زیر، با استفاده از ابعاد فعلی تصویر، و ابعادی که می‌خواهیم شکل را به آن سایز برسانیم، نسبتی که لازم است تا طول و عرض با آن بزرگ شوند (r) را به دست می‌آوریم. سپس تصویر را به ابعاد جدید در آورده و مقدار padding که باید به آن داده شود تا به ابعاد دلخواهمان برسد را محاسبه می‌کنیم. در انتها، با استفاده از تابع copyMakeBorder، جاهایی که نیاز به padding داشت را یک border به رنگ سیاه می‌کشیم که معادل همان zero padding است. نکته‌ای که در اینجا باید توجه شود این است که مقدار padding از تمامی جهات (چپ، راست، بالا و پایین) باید داده شود تا تصویر در مرکز قرار بگیرد. خروجی این بخش، مطابق شکل داده شده می‌شود.

```
[51] def resize_img(img, desired_size = 224):
        # write your code here
        w, h = img.shape[:2]
        r = float(desired_size)/max(w, h)
        h2, w2 = tuple([int(x * r) for x in (w, h)])
        img = cv2.resize(img, (w2, h2))
        dw = desired_size - w2
        left, right = dw//2, dw - (dw//2)
        dh = desired_size - h2
        top, bottom = dh//2, dh - (dh//2)
        new_img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT, value=(0, 0, 0))
        return new_img
```

ب) با استفاده از کد زیر، ابتدا ResNet را به مدل خود add کرده و وزن‌ها را هم None که معادل مقداردهی اولیه تصادفی است، قرار می‌دهیم. سپس یک Flatten زده و در انتها، به علت ۲۴ کلاسه بودن مسئله، یک لایه Dense با ۲۴ نورون و تابع فعال‌سازی softmax می‌زنیم.

```
resnet = tf.keras.models.Sequential()
# Write your code here
added_model = ResNet50(
    input_shape=(224, 224, 3),
    include_top=False,
    weights=None,
    classes=24,
)
resnet.add(added_model)
resnet.add(Flatten())
resnet.add(Dense(24, activation='softmax'))
resnet.summary()
```

همانطور که در شکل زیر نیز مشخص است، دقت در این‌جا مقدار کمی دارد و loss آن تقریبا زیاد است. این موضوع می‌تواند به دلیل مقداردهی تصادفی وزن‌ها، عدم استفاده از لایه‌های بیشتر (کم عمق بودن شبکه) و انجام ندادن عملیات fine tuning باشد.

```
resnet.fit(train_generator, epochs=1)
```

```
65/65 [==============================] - 75s 1s/step - loss: 10.0237 - acc: 0.4258
<keras.callbacks.History at 0x7f124470b580>
```

پ) تفاوت این مرحله با مرحله قبل، این است که وزن‌ها را در resnet دیگر تصادفی نگذاشتیم و از وزن‌های imagenet استفاده کردیم. همچنین تعدادی لایه برای train بهتر اضافه کرده و وزن‌های شبکه از پیش آموخته شده را فریز کردیم.

```
fine_tune_resnet = tf.keras.models.Sequential()
# write your code here
added_model = ResNet50(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    classes=24,
)
fine_tune_resnet.add(added_model)
fine_tune_resnet.add(Flatten())
fine_tune_resnet.add(Dense(256, activation='relu'))
fine_tune_resnet.add(Dense(128, activation='relu'))
fine_tune_resnet.add(Dense(24, activation='softmax'))
added_model.trainable = False
fine_tune_resnet.summary()
```

نتیجه کد بالا، در تصویر زیر نمایان است که نشان می‌دهد دقت تا حد خیلی خوبی افزایش، و loss به مقدار قابل توجهی کاهش یافته است. علت این موضوع، استفاده از عملیات fine tuning، استفاده از وزن‌های شبکه imagenet برای وزن‌های شبکه از پیش آموخته شده‌مان، و استفاده از لایه‌های بیشتر در ادامه می‌باشد.

```
fine_tune_resnet.fit(train_generator, epochs=1)

65/65 [==============================] - 35s 535ms/step - loss: 0.8006 - acc: 0.8063
<keras.callbacks.History at 0x7f0fd1f80be0>
```

ت) نتایج حاصل از دو مدل تعریف شده در بخش ب و پ روی داده‌های test در شکل زیر نمایان است. همانطور که می‌بینید، دقت در مدل دوم بیشتر، و loss آن کمتر می‌باشد که نتایج بهتری به ما می‌دهد. علل این موضوع نیز در بخش قبل توضیح داده شد.

```
[96] resnet.evaluate(test_generator)

33/33 [==============================] - 19s 538ms/step - loss: 4.1548 - acc: 0.1617
[4.154778480529785, 0.16173633933067322]

fine_tune_resnet.evaluate(test_generator)

33/33 [==============================] - 18s 517ms/step - loss: 0.5990 - acc: 0.8309
[0.5989599227905273, 0.8308681845664978]
```

لینک‌های مورد استفاده در حل این سوال:

https://stackoverflow.com/questions/43391205/add-padding-to-images-to-get-them-into-the-same-shape