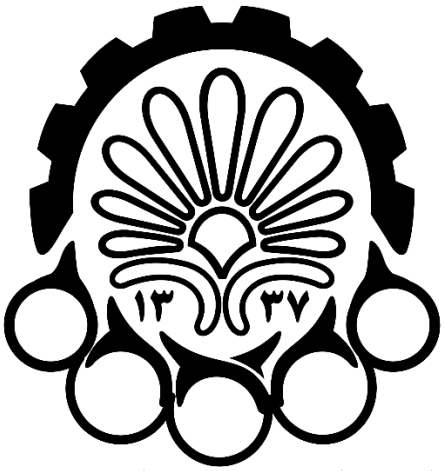


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمرین چهارم درس شبکه عصبی

دکتر صفابخش

غلامرضا دار ۴۰۰۱۳۱۰۱۸

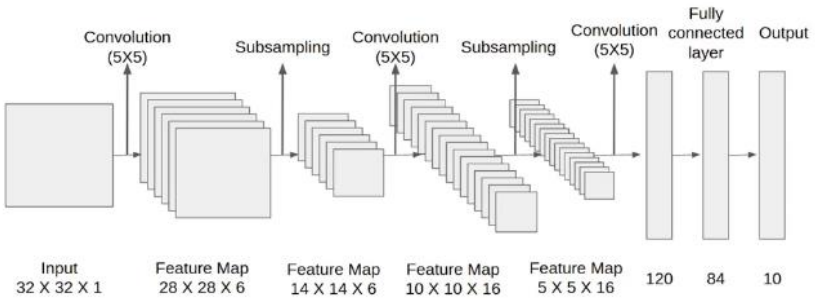
بهار ۱۴۰۱

فهرست مطالب

سوال ۱)

معماری LeNet: این معماری برای اولین بار در سال ۱۹۹۸ توسط Yann LeCun معرفی شد. کاربرد اصلی این معماری تشخیص ارقام دست نویس دیتاست mnist بود. سادگی و راحتی این معماری یکی از دلایل محبوبیت این مدل است. این معماری یک معماری بر پایه شبکه‌های کانولوشنی است. در ادامه بیشتر با جزییات این معماری آشنا می‌شویم.

معماری LeNet-5 همانطور که از اسمش مشخص است، شامل ۵ لایه اصلی است. سه لایه کانولوشنی و ۲ لایه کاملا متصل این معماری را می‌سازند.

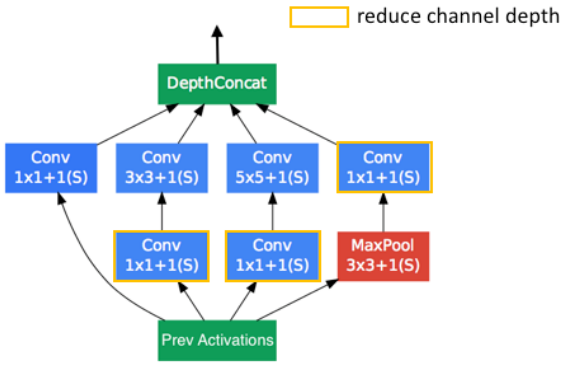


ورودی‌های این معماری تصاویر تک کاناله ۳۲ پیکسل در ۳۲ پیکسل هستند. یک لایه کانولوشنی با اندازه ۵ در ۵، این تصاویر را به فیچرمپ‌هایی با اندازه ۲۸ در ۲۸ تبدیل می‌کند(تعداد فیلترهای ۵ در ۵ در این لایه ۶ عدد در نظر گرفته شده اند). در ادامه یک لایه Average Pooling باعث می‌شود که سایز فیچرمپ‌ها نصف شود. پس از این مرحله یک لایه کانولوشنی با اندازه ۵ در ۵ دیگر و این‌بار با تعداد ۱۶ فیلتر به نتیجه Average Pooling اعمال می‌شود. این روند مانند شکل یک بار دیگر نیز اتفاق می‌افتد و پس از اعمال این سه مرحله کانولوشن، یک لایه کاملا متصل با تعداد ۸۴ نورون، اطلاعات مرحله قبل را میگیرد و در نهایت یک لایه کاملا متصل دیگر که دارای ۱۰ نورون است(هر نورون برای یک رقم) نتیجه دسته بندی نهایی را مشخص می‌کند. این مدل تعداد ۶۰ هزار پارامتر قابل تنظیم دارد که طبق استانداردهای امروزی عدد کوچکی به حساب می‌آید.

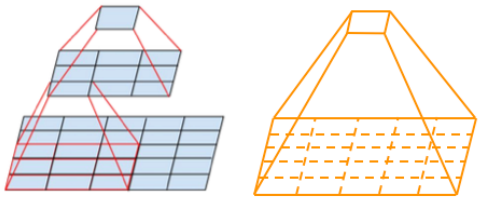
همانطور که دیدیم این مدل بسیار ساده و قابل فهم است و با اینکه بر روی تصاویر ارقام که اندازه کوچکی نیز دارند نتیجه قابل قبولی بدست می‌آورد، در کاربردهای دیگر با اندازه تصاویر بزرگتر و تعداد کلاس بیشتر ممکن است نتیجه نامطلوبی بدست آورد. در کارهای اخیری که بر روی دسته بندی تصاویر رخ داده، نشان داده شده که معماری‌هایی با عمق بیشتر و تعداد لایه های بیشتر عملکرد قابل قبول‌تری دارند. در ادامه یکی از این مدل ها به نام Inception-v3 را مورد بررسی قرار می‌دهیم.

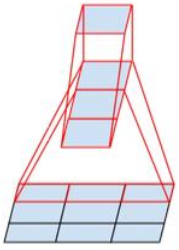
معماری Inception-V3: قبل از بحث در مورد معماری Inception-v3 بهتر است با نسخه ابتدایی تر آن یعنی GoogLeNet آشنا شویم. معماری Inception یا GoogLeNet در سال ۲۰۱۴ توانست در مسابقه تشخیص اشیا ImageNet مقام اول را کسب کند. این مدل از اولین مدل هایی بود که تصمیم گرفت با پهن کردن(نه عمیق کردن) مدل، قدرت مدل را افزایش دهد.

این معماری از تعدادی سلول Inception ساخته شده است. هر سلول Inception مانند تصویر زیر از تعدادی عمل کانولوشنی در سایز های مختلف ایجاد شده. نتیجه این عملیات کانولوشنی با هم ترکیب می‌شوند و خروجی سلول Inception تولید می‌شود.



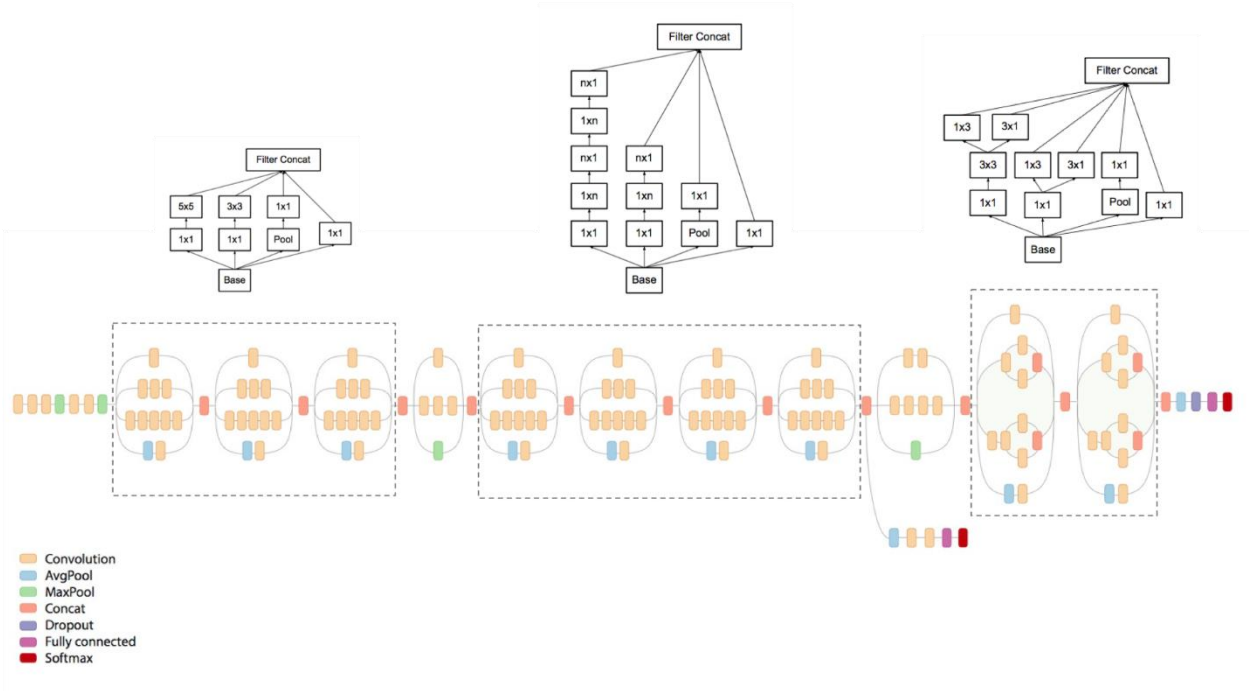
دو سال بعد نویسنده های مقاله Inception مقاله ای تحت عنوان Rethinking the Inception Architecture for Computer Vision چاپ کردند که تعدادی تکنیک برای بهبود معماری قبلی را ارائه می‌داد. یکی از این تکنیک‌ها جایگزین کردن لایه های کانولوشنی ۵ در ۵ با دو لایه کانولوشنی ۳ در ۳ بود. همان‌طور که در تصویر زیر می‌بینید، این دو فیلتر نتیجه یکسانی دارند اما اعمال کردن دو فیلتر ۳ در ۳ بسیار بهینه تر است. تکنیک دیگری که مورد استفاده قرار گرفت، جایگزینی فیلتر ۳ در ۳ با دو فیلتر ۳ در ۱ و ۱ در ۳ بود. همه این بهینه سازی ها باعث شد معماری Inception-v3 بتواند با میزان زمان پردازشی مشابه، تعداد پارامترهای بسیار بیشتری داشته باشد و به نتایج قابل قبول تری دست یابد.





در تصویر زیر می‌توان معماری کامل Inception-v3 را مشاهده کرد. این مدل شامل ۲۳ میلیون پارامتر قابل تنظیم است.

این معماری علاوه بر تکنیک های ذکر شده، از یک سری دسته بند کمکی نیز بهره می‌برد(Auxiliary Classifiers) که به نوعی به عنوان Regularizer عمل میکنند و تعمیم پذیری مدل را بالا می‌برند.



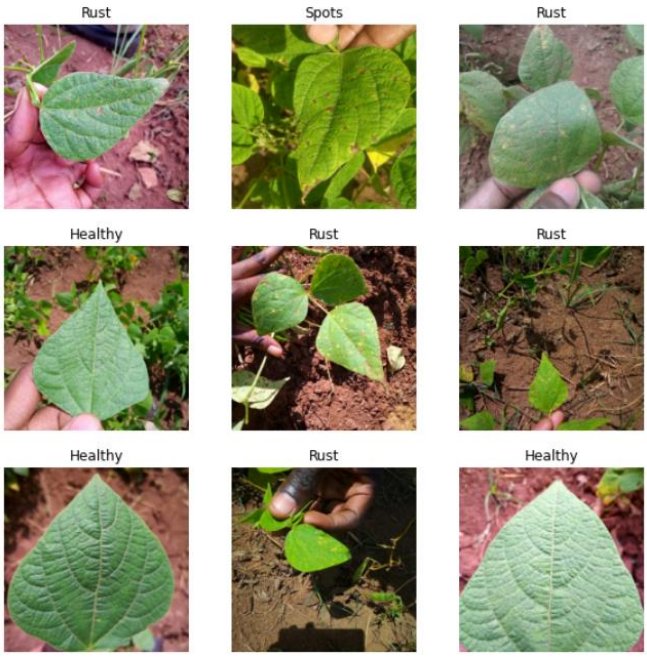
منابع:

https://www.jeremyjordan.me/convnet-architectures/

https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

سوال ۲)

ابتدا با کمک ابزار tfds دیتاست beans را فراخوانی میکنیم. برای هماهنگی و سادگی انجام مقایسه ها، تصاویر را هم برای معماری LeNet و هم برای معماری Inception به یک اندازه در می‌آوریم. اندازه انتخابی ۲۹۹ در ۲۹۹ است (وزن های مدلِ از قبل آموزش دیده Inception با این اندازه آموزش دیده اند).



سپس معماری LeNet را پیاده سازی میکنیم.

```
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 299, 299, 3)       0

 conv2d_6 (Conv2D)           (None, 295, 295, 6)       456

 max_pooling2d_4 (MaxPooling  (None, 147, 147, 6)      0
 2D)

 conv2d_7 (Conv2D)           (None, 143, 143, 16)      2416

 max_pooling2d_5 (MaxPooling  (None, 71, 71, 16)       0
 2D)

 conv2d_8 (Conv2D)           (None, 67, 67, 120)       48120

 flatten_2 (Flatten)         (None, 538680)            0

 dense_4 (Dense)             (None, 50)                26934050

 dense_5 (Dense)             (None, 3)                 153

=================================================================
Total params: 26,985,195
Trainable params: 26,985,195
Non-trainable params: 0
_________________________________________________________________
```

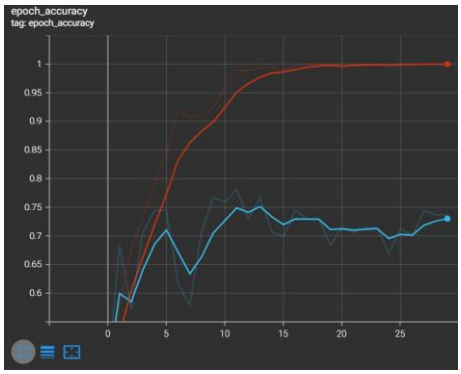
پس از آموزش دادن مدل برای ۲۰ epoch به صحت ۶۹ درصد تست میرسیم.

```
Epoch 14/20
17/17 [==============================] - 6s 361ms/step - loss: 0.0314 - accuracy: 0.9961 - val_loss: 0.7830 - val_accuracy: 0.7444
Epoch 15/20
17/17 [==============================] - 6s 366ms/step - loss: 0.0180 - accuracy: 0.9990 - val_loss: 1.0034 - val_accuracy: 0.7143
Epoch 16/20
17/17 [==============================] - 6s 361ms/step - loss: 0.0093 - accuracy: 0.9990 - val_loss: 0.9655 - val_accuracy: 0.7594
Epoch 17/20
17/17 [==============================] - 6s 364ms/step - loss: 0.0074 - accuracy: 0.9981 - val_loss: 0.8188 - val_accuracy: 0.7293
Epoch 18/20
17/17 [==============================] - 6s 365ms/step - loss: 0.0097 - accuracy: 0.9990 - val_loss: 1.1208 - val_accuracy: 0.6992
Epoch 19/20
17/17 [==============================] - 6s 360ms/step - loss: 0.0185 - accuracy: 0.9942 - val_loss: 0.7800 - val_accuracy: 0.7368
Epoch 20/20
17/17 [==============================] - 6s 363ms/step - loss: 0.0302 - accuracy: 0.9913 - val_loss: 0.8389 - val_accuracy: 0.7594
<keras.callbacks.History at 0x7f50c4344b90>

[ ] lenet_model.evaluate(test_ds)

2/2 [==============================] - 1s 164ms/step - loss: 1.5283 - accuracy: 0.6953
[1.5282679796218872, 0.6953125]
```

با بررسی نمودارهای accuracy, loss متوجه می‌شویم که این مدل به شدت Overfit می‌شود. پیچیده بودن مدل LeNet برای این حجم از داده (حدود ۲۴ میلیون پارامتر)یکی از دلایل این بیش برازش است. (نمودار قرمز train_acc و نمودار آبی valid_acc است)



۱. آزمایش های مربوط به ریگولاریزیشن:

اضافه کردن L1:

اضافه کردن ریگولارایزر L1 به لایه‌های کانولوشن باعث می‌شود مدل از حالت بیش‌برازشی که داشت خارج شود. (مقدار ۰,۰۰۱انتخاب شد)

```
Epoch 14/20
17/17 [==============================] - 6s 369ms/step - loss: 1.5921 - accuracy: 0.9062 - val_loss: 2.1515 - val_accuracy: 0.6165
Epoch 15/20
17/17 [==============================] - 6s 367ms/step - loss: 1.4778 - accuracy: 0.9246 - val_loss: 2.0127 - val_accuracy: 0.6917
Epoch 16/20
17/17 [==============================] - 6s 370ms/step - loss: 1.4002 - accuracy: 0.9342 - val_loss: 2.0307 - val_accuracy: 0.6391
Epoch 17/20
17/17 [==============================] - 6s 368ms/step - loss: 1.3235 - accuracy: 0.9265 - val_loss: 2.4393 - val_accuracy: 0.6015
Epoch 18/20
17/17 [==============================] - 6s 370ms/step - loss: 1.2879 - accuracy: 0.9294 - val_loss: 2.2187 - val_accuracy: 0.5865
Epoch 19/20
17/17 [==============================] - 6s 371ms/step - loss: 1.2343 - accuracy: 0.9255 - val_loss: 2.3589 - val_accuracy: 0.5940
Epoch 20/20
17/17 [==============================] - 6s 369ms/step - loss: 1.2480 - accuracy: 0.9004 - val_loss: 2.2387 - val_accuracy: 0.6165

[ ] lenet_model.evaluate(test_ds)

2/2 [==============================] - 0s 72ms/step - loss: 2.1251 - accuracy: 0.6406
[2.12514328956604, 0.640625]
```

اضافه کردن L2:

اضافه کردن ریگولارایزر L2 به لایه‌های کانولوشن باعث می‌شود مدل از حالت بیش‌برازشی که داشت خارج شود و همچنین نسبت به نتیجه L1 صحت بالاتری بدست می‌آید.

```
Epoch 14/20
17/17 [==============================] - 6s 365ms/step - loss: 0.4445 - accuracy: 0.8926 - val_loss: 0.8314 - val_accuracy: 0.7068
Epoch 15/20
17/17 [==============================] - 6s 364ms/step - loss: 0.4416 - accuracy: 0.8936 - val_loss: 1.8634 - val_accuracy: 0.6617
Epoch 16/20
17/17 [==============================] - 6s 367ms/step - loss: 0.5748 - accuracy: 0.8404 - val_loss: 0.9180 - val_accuracy: 0.7218
Epoch 17/20
17/17 [==============================] - 6s 370ms/step - loss: 0.4778 - accuracy: 0.8839 - val_loss: 1.0157 - val_accuracy: 0.7218
Epoch 18/20
17/17 [==============================] - 6s 365ms/step - loss: 0.3674 - accuracy: 0.9371 - val_loss: 1.5609 - val_accuracy: 0.6541
Epoch 19/20
17/17 [==============================] - 6s 362ms/step - loss: 0.3885 - accuracy: 0.9159 - val_loss: 1.3618 - val_accuracy: 0.6692
Epoch 20/20
17/17 [==============================] - 6s 366ms/step - loss: 0.3712 - accuracy: 0.9275 - val_loss: 0.9414 - val_accuracy: 0.7293

[ ] lenet_model.evaluate(test_ds)

2/2 [==============================] - 0s 72ms/step - loss: 1.0537 - accuracy: 0.7109
[1.0536556243896484, 0.7109375]
```

اضافه کردن Dropout:

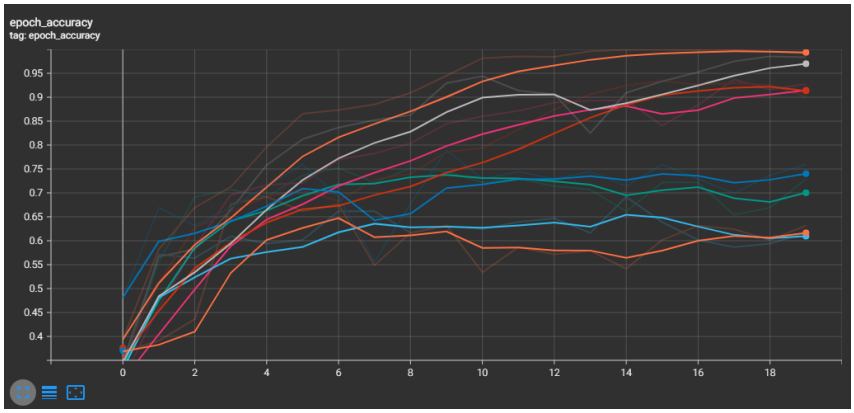
افزودن لایه Dropout نیز یکی دیگر از راه‌های کاهش بیش‌برازش و افزایش تعمیم‌پذیری مدل است که با توجه به آزمایش‌های انجام شده نسبت به مدل اصلی نتیجه بهتری می‌دهد. کار لایه Dropout غیرفعال کردن تعدادی از نورون‌ها به صورت رندوم است. این اتفاق باعث می‌شود یک سری ارتباطات بین نورونی که نباید وجود داشته باشند از بین برود و نورون ها به نورون های دیگر وابستگی کمتری داشته باشند. برای این سوال استفاده از Dropout ۰,۲ توانست از بیش برازش جلوگیری کند اما صحت دسته بندی را کاهش داد.

```
Epoch 14/20
17/17 [==============================] - 6s 383ms/step - loss: 0.4382 - accuracy: 0.8250 - val_loss: 0.9058 - val_accuracy: 0.5789
Epoch 15/20
17/17 [==============================] - 6s 382ms/step - loss: 0.2457 - accuracy: 0.9091 - val_loss: 1.5885 - val_accuracy: 0.5414
Epoch 16/20
17/17 [==============================] - 6s 385ms/step - loss: 0.2193 - accuracy: 0.9333 - val_loss: 1.2142 - val_accuracy: 0.6015
Epoch 17/20
17/17 [==============================] - 6s 386ms/step - loss: 0.1529 - accuracy: 0.9526 - val_loss: 1.1682 - val_accuracy: 0.6316
Epoch 18/20
17/17 [==============================] - 6s 384ms/step - loss: 0.0828 - accuracy: 0.9749 - val_loss: 1.1463 - val_accuracy: 0.6241
Epoch 19/20
17/17 [==============================] - 6s 384ms/step - loss: 0.0569 - accuracy: 0.9855 - val_loss: 1.5517 - val_accuracy: 0.6015
Epoch 20/20
17/17 [==============================] - 6s 387ms/step - loss: 0.0423 - accuracy: 0.9836 - val_loss: 1.3835 - val_accuracy: 0.6316

lenet_model.evaluate(test_ds)

2/2 [==============================] - 0s 72ms/step - loss: 1.2425 - accuracy: 0.6719
[1.24253511428833, 0.671875]
```

بخشی از آزمایش های انجام شده برای این سوال را مشاهده میکنید.



۲. آزمایش های مربوط به اندازه کرنل:

برای آزمایش این بخش، از سه مقدار ۳،۵،۷ برای اندازه کرنل استفاده کردیم که دقت تست را برای هر حالت در جدول زیر مشاهده میکنید.

اندازه کرنل	صحت تست
۳	۷۲,۶۶٪
۵	۶۹,۵٪
۷	۷۱,۸۸٪

نتایج این آزمایش به دلیل وجود مقداری نوسان بین آزمایش ها و نزدیک بودن نتایج به یکدیگر قابل نتیجه گیری نیست اما اگر بخواهیم بهترین مدل را انتخاب کنیم مدل با اندازه کرنل ۳ نتیجه بهتری داد.

۳. آزمایش های مربوط به تعداد کرنل:

برای آزمایش این بخش، از سه حالت [۳،۸،۶۰] و [۶،۱۶،۱۲۰] و [۱۲،۳۲،۲۴۰] استفاده کردیم که دقت تست را برای هر حالت در جدول زیر مشاهده میکنید.

تعداد کرنل	صحت تست
[۳،۸،۶۰]	۵۸,۵۹٪
[۶،۱۶،۱۲۰]	۶۹,۵٪
[۱۲،۳۲،۲۴۰]	آزمایش انجام نشد(مشکل رم)

در مورد تعداد کرنل، با افزایش تعداد کرنل ها نتیجه بهتر شد اما از یک جایی به بعد به دلیل زیاد شدن پارامتر ها به مشکل رم برخورد کردیم و آموزش به اتمام نرسید.

سوال ۳)

تکنیک یادگیری انتقالی یک روش برای کوتاه کردن زمان آموزش و بهره گیری از مدل های از پیش آموزش داده شده و جنرال برای حل مسئله های خاص تر(که ممکن است دیتای کمتری داشته باشند) است.

مرحله اول یادگیری انتقالی

نحوه کلی اجرای یادگیری انتقالی به این شکل است که یک مدل قبلا توسط افراد دیگر و صرف هزینه پردازشی زیاد آموزش داده شده است. معمولا این مدل ها مدل های جنرالی هستند که مسائل خیلی کلی مانند تشخیص اشیا(از بین ۱۰۰۰ نوع لیبل مختلف) را حل میکنند. این مدل ها، علاوه بر اینکه یاد گرفته اند مسئله مورد نظر خود را حل کنند، یاد گرفته اند فیچرهای بسیار قوی و مناسبی را تولید کنند. در یادگیری انتقالی عموما هدف این است که از ان مدل های از پیش آموزش دیده بهره بگیریم تا این فیچرهای ارزشمند را برای مسئله خود بدون صرف هزینه زیاد تولید کنیم. معمولا در انتهای بسیاری از مدل های دسته بند، یک لایه تماما متصل برای دسته بندی داده به کلاس های مدنظر مسئله اصلی وجود دارد. ما این لایه را حذف میکنیم و یک لایه مخصوص مسئله خودمان قرار می‌دهیم. همچنین لایه های مربوط به مدل اصلی را از حالت قابل آموزش در میاوریم تا وزن های از قبل آموخته شده تغییر نکنند و در واقع عملکرد لایه های ابتدایی(لایه های مربوط به استخراج به ویژگی) عوض نشود. سپس مدل جدید را بر روی داده آموزش خود(که ممکن است خیلی کوچک باشد) آموزش میدهیم و با اینکار وزن های مربوط به لایه آخر(دسته بند) را متناسب با مسئله خود آپدیت میکنیم. در انتهای این مراحل، مدلی داریم که قدرت استخراج ویژگی مدل اصلی را دارد و همچنین یاد گرفته است که ازین ویژگی ها به چه شکل استفاده کند تا ورودی مسئله ما را به خروجی مورد نظرمان نسبت بدهد.

مرحله دوم یادگیری انتقالی(Finetune کردن مدل)

پس از انجام مرحله قبل، می‌توانیم برای بهبود نتایج یک کار دیگر نیز انجام دهیم. میتوان با آزادسازی وزن چند لایه آخر مدل اصلی(مدل ترین شده) وزن این لایه ها را نیز برای مسئله خود بهینه کنیم. چون معمولا در اکثر مدل ها، فیچر های استخراج شده در لایه های آخر، مخصوص تر هستند و ویژگی های استخراج شده در لایه های اول، کلی تر و عمومی تر هستند، این تصمیم را گرفتیم که فقط لایه های آخر را آزاد کنیم و برای مسئله خود بهینه کنیم. دلیل اینکه تمام لایه ها را نیز آزاد نمیکنیم این است که علاوه بر بار محاسباتی زیادی که برایمان دارد، ممکن است مدلمان، به دلیل داشتن پیچیدگی زیاد نسبت به داده ای که داریم، اورفیت شود.

یک نکته مهم این است که حتما باید قبل از فاین تون کردن مدل، یکبار لایه آخری که اضافه کرده ایم را آموزش دهیم. اگر این کار را نکنیم باعث میشود خطاهای انتشار یافته به سمت لایه های مدل اصلی بسیار زیاد باشند و وزن های آموخته شده را خراب کند.

سوال ۴)

همانطور که در سوال ۳ توضیح داده شد، در این بخش میخواهیم با استفاده از مدل آموزش دیده‌ی Inception-v3 عملیات دسته بندی انواع برگ لوبیا را انجام دهیم. مدل Inception-v3 برای تشخیص انواع برگ های لوبیا آموزش دیده نشده است. این مدل برای تشخیص اشیا مختلف(از ۱۰۰۰ کلاس) در دیتاست ImageNet طراحی شده.

ما می‌توانیم با استفاده از تکنیک یادگیری انتقالی، از فیچرهای آموخته شده در Inception-v3 و در حین یادگیری پترن‌های بین تصاویر متنوع مجموعه ImageNet، بهره بگیریم و ادامه کار را از این نقطه شروع کنیم. در واقع با این کار، ما فرض میکنیم نیازی به استخراج فیچرهای خام از تصاویر نیست(عموما این مرحله بسیار ضروری است زیرا تصاویر خام برای مدل قابل درک نیستند) اما در این مثال، ما به Inception-v3 آموزش دیده دسترسی داریم که به طور خود به خود ویژگی ها را از تصاویر برای ما استخراج میکند.

حال به پیاده سازی نکات گفته شده در بخش های قبل میپردازیم. ابتدا لازم است مدل Inception-v3 را از کتابخانه Keras فراخوانی کنیم. به هنگام فراخوانی، اعلام میکنیم که لایه آخر(دسته بند) این مدل را احتیاج نداریم و همچنین به وزن های از قبل تعیین شده این مدل نیازمندیم.

```
base_model = tf.keras.applications.inception_v3.InceptionV3(
  include_top=False,
  weights='imagenet',
  input_shape=INPUT_SHAPE,
  classes=1000,
  classifier_activation='softmax'
  )
```

پس از فراخوانی مدل، وزن های آن را فریز میکنیم و با افزودن لایه های تماما متصل ذکر شده، یکبار مدل را آموزش میدهیم. همچنین اگر به doc اینسپشن در کراس مراجعه کنید میبینید که اعداد ورودی به این مدل باید بین منفی ۱ تا ۱ باشند بنابراین با افزودن یک لایه Rescaling این امر را محقق می‌سازیم. تعداد نورون های لایه تماما متصل را ۱۰۲۴ در نظر میگیریم و یک لایه تماما متصل به اندازه ۳ نورون نیز برای مرحله آخر دسته بندی اضافه میکنیم.

```
# Freeze the base_model
base_model.trainable = False

# Create new model on top
inputs = tf.keras.Input(shape=INPUT_SHAPE)

# Map data from 0 - 255 to -1 - 1
scale_layer = tf.keras.layers.Rescaling(scale=1 / 127.5, offset=-1)
x = scale_layer(inputs)

# Training=False is for BatchNorm layers
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(1024)(x)
outputs = tf.keras.layers.Dense(NUM_CLASSES)(x)

new_model = tf.keras.Model(inputs, outputs)
```

پس از آموزش این مدل برای ۲۰ epoch به صحت ولیدیشن ۸۷٪ و صحت تست ۷۷٪ میرسیم.

```
Epoch 10/20
17/17 [==============================] - 9s 544ms/step - loss: 0.7888 - accuracy: 0.7718 - val_loss: 0.4117 - val_accuracy: 0.8271
Epoch 11/20
17/17 [==============================] - 9s 543ms/step - loss: 0.5504 - accuracy: 0.8259 - val_loss: 1.3001 - val_accuracy: 0.6617
Epoch 12/20
17/17 [==============================] - 9s 544ms/step - loss: 0.4785 - accuracy: 0.8230 - val_loss: 1.5035 - val_accuracy: 0.6090
Epoch 13/20
17/17 [==============================] - 10s 584ms/step - loss: 0.4291 - accuracy: 0.8366 - val_loss: 0.3259 - val_accuracy: 0.8647
Epoch 14/20
17/17 [==============================] - 9s 545ms/step - loss: 0.5392 - accuracy: 0.8153 - val_loss: 0.5505 - val_accuracy: 0.8271
Epoch 15/20
17/17 [==============================] - 9s 545ms/step - loss: 0.3275 - accuracy: 0.8636 - val_loss: 0.3125 - val_accuracy: 0.8647
Epoch 16/20
17/17 [==============================] - 9s 541ms/step - loss: 0.6409 - accuracy: 0.7892 - val_loss: 0.3043 - val_accuracy: 0.8571
Epoch 17/20
17/17 [==============================] - 10s 584ms/step - loss: 0.2861 - accuracy: 0.8849 - val_loss: 0.4967 - val_accuracy: 0.8195
Epoch 18/20
17/17 [==============================] - 9s 544ms/step - loss: 0.4442 - accuracy: 0.8472 - val_loss: 0.3164 - val_accuracy: 0.8647
Epoch 19/20
17/17 [==============================] - 9s 545ms/step - loss: 0.4513 - accuracy: 0.8491 - val_loss: 0.2910 - val_accuracy: 0.8722
Epoch 20/20
17/17 [==============================] - 10s 584ms/step - loss: 0.2821 - accuracy: 0.8723 - val_loss: 0.3000 - val_accuracy: 0.8797
<keras.callbacks.History at 0x7fa7662c1dd0>
```

```
[12] new_model.evaluate(test_ds)

     2/2 [==============================] - 1s 234ms/step - loss: 0.4465 - accuracy: 0.7969
     [0.4464957118034363, 0.796875]
```

این نشان می‌دهد که مدل ما توانسته از فیچرهای استخراج شده به خوبی استفاده کند و روی مجموعه داده تست به ۷۷ درصد تصاویر برچسب درست بزند. اما ما میتوانیم با استفاده از Finetuning عملکرد مدل را از این هم بالاتر ببریم.

برای اینکار ابتدا باید تصمیم بگیریم که چند لایه بالای مدل اینسپشن را آزاد کنیم. با بررسی های انجام شده و توضیحاتی که در اینترنت خواندیم، بهتر است لایه ها را به طور مستقل از هم آزاد و فریز نکنیم. بلکه لایه های مربوط به هم (یک بلاک) را همزمان یا فریز کنیم یا آزاد.

در این سوال سه حالت را بررسی میکنیم:

- آزادسازی ۲ بلاک کانولوشنی بالا (لایه ۲۴۹ به بعد)
- آزادسازی ۳ بلاک کانولوشنی بالا (لایه ۲۲۹ به بعد)
- آزادسازی ۶ بلاک کانولوشنی بالا (لایه ۱۳۳ به بعد)

نکته دیگری که باید به خاطر بسپاریم این است که در این مرحله بهتر است از نرخ یادگیری بسیار کمتر استفاده کنیم تا وزن‌های آموخته شده خراب نشوند و آرام آرام به سمت بهینه شدن حرکت کنند.

نکته دیگر تعداد epoch های لازم است که در این مرحله عموما بهتر است کمتر از مرحله قبل باشد زیرا فقط می‌خواهیم کمی وزن‌ها را به سمت بهینگی حرکت دهیم.

۱. آزادسازی ۲ بلاک کانولوشنی

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 299, 299, 3)]     0

 rescaling (Rescaling)       (None, 299, 299, 3)       0

 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 1024)              2098176

 dense_1 (Dense)             (None, 3)                 3075

=================================================================
Total params: 23,904,035
Trainable params: 13,216,131
Non-trainable params: 10,687,904
```

با آزادسازی ۲ بلاک کانولوشنی، تعداد پارامترهای قابل آموزش به عدد **۱۳** میلیون رسید. پس از آموزش این مدل برای تعداد epoch کم و نرخ یادگیری کم به صحت ولیدیشن و تست **۸۸**٪ رسیدیم. که نسبت به مدل فاین تون نشده بهبود بزرگی است.

```
Epoch 1/6
17/17 [==============================] - 20s 865ms/step - loss: 0.1757 - accuracy: 0.9371 - val_loss: 0.2491 - val_accuracy: 0.8872
Epoch 2/6
17/17 [==============================] - 11s 681ms/step - loss: 0.1409 - accuracy: 0.9458 - val_loss: 0.2483 - val_accuracy: 0.8872
Epoch 3/6
17/17 [==============================] - 11s 643ms/step - loss: 0.1170 - accuracy: 0.9507 - val_loss: 0.2535 - val_accuracy: 0.8797
Epoch 4/6
17/17 [==============================] - 10s 600ms/step - loss: 0.0987 - accuracy: 0.9632 - val_loss: 0.2490 - val_accuracy: 0.8872
Epoch 5/6
17/17 [==============================] - 10s 609ms/step - loss: 0.0837 - accuracy: 0.9739 - val_loss: 0.2436 - val_accuracy: 0.8872
Epoch 6/6
17/17 [==============================] - 10s 601ms/step - loss: 0.0709 - accuracy: 0.9816 - val_loss: 0.2405 - val_accuracy: 0.8872
<keras.callbacks.History at 0x7fa73ff11e90>

[15] new_model.evaluate(test_ds)

2/2 [==============================] - 1s 234ms/step - loss: 0.2682 - accuracy: 0.8828
[0.2682493329048157, 0.8828125]
```

۲. آزادسازی ۳ بلاک کانولوشنی

با آزادسازی ۳ بلاک کانولوشنی تعداد پارامترهای قابل آموزش به عدد **۱۵** میلیون میرسد.

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 299, 299, 3)]     0

 rescaling_1 (Rescaling)     (None, 299, 299, 3)       0

 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784

 global_average_pooling2d_1   (None, 2048)             0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 1024)              2098176

 dense_3 (Dense)             (None, 3)                 3075

=================================================================
Total params: 23,904,035
Trainable params: 14,913,155
Non-trainable params: 8,990,880
```

از نظر نتایج این مدل با مدل قبلی فرق چندانی نداشت. صحت آموزش و تست حدود ۸۸٪.

```
Epoch 1/6
17/17 [==============================] - 17s 713ms/step - loss: 0.1772 - accuracy: 0.9371 - val_loss: 0.2395 - val_accuracy: 0.8722
Epoch 2/6
17/17 [==============================] - 11s 670ms/step - loss: 0.1344 - accuracy: 0.9478 - val_loss: 0.2353 - val_accuracy: 0.8872
Epoch 3/6
17/17 [==============================] - 11s 669ms/step - loss: 0.1048 - accuracy: 0.9603 - val_loss: 0.2450 - val_accuracy: 0.8797
Epoch 4/6
17/17 [==============================] - 11s 667ms/step - loss: 0.0830 - accuracy: 0.9710 - val_loss: 0.2571 - val_accuracy: 0.8797
Epoch 5/6
17/17 [==============================] - 11s 627ms/step - loss: 0.0674 - accuracy: 0.9826 - val_loss: 0.2639 - val_accuracy: 0.8722
Epoch 6/6
17/17 [==============================] - 11s 626ms/step - loss: 0.0561 - accuracy: 0.9874 - val_loss: 0.2680 - val_accuracy: 0.8797
2/2 [==============================] - 1s 228ms/step - loss: 0.2519 - accuracy: 0.8828
[0.2519141435623169, 0.8828125]
```

۳. آزادسازی ۶ بلاک کانولوشنی

با آزادسازی ۶ بلاک کانولوشنی تعداد پارامترهای قابل آموزش به عدد **۲۰** میلیون میرسد.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 299, 299, 3)]     0

 rescaling_2 (Rescaling)     (None, 299, 299, 3)       0

 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784

 global_average_pooling2d_2   (None, 2048)             0
 (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1024)              2098176

 dense_5 (Dense)             (None, 3)                 3075

=================================================================
Total params: 23,904,035
Trainable params: 20,431,747
Non-trainable params: 3,472,288
```

همانطور که در نتایج می‌بینید این مدل بهترین نتیجه را داد و توانست به صحت تست ۹۰٪ برسد. اما باید دقت کنیم که مدل در حال اورفیت شدن نیز هست. در بخش جمع بندی به این قسمت اشاره ای میکنیم.

```
Epoch 1/6
17/17 [==============================] - 19s 866ms/step - loss: 0.2258 - accuracy: 0.9062 - val_loss: 0.2299 - val_accuracy: 0.8872
Epoch 2/6
17/17 [==============================] - 13s 790ms/step - loss: 0.1238 - accuracy: 0.9594 - val_loss: 0.2166 - val_accuracy: 0.9023
Epoch 3/6
17/17 [==============================] - 13s 790ms/step - loss: 0.0898 - accuracy: 0.9691 - val_loss: 0.2153 - val_accuracy: 0.9098
Epoch 4/6
17/17 [==============================] - 14s 821ms/step - loss: 0.0644 - accuracy: 0.9855 - val_loss: 0.2350 - val_accuracy: 0.8797
Epoch 5/6
17/17 [==============================] - 13s 785ms/step - loss: 0.0442 - accuracy: 0.9952 - val_loss: 0.2461 - val_accuracy: 0.8722
Epoch 6/6
17/17 [==============================] - 13s 787ms/step - loss: 0.0335 - accuracy: 0.9981 - val_loss: 0.2489 - val_accuracy: 0.8797
2/2 [==============================] - 1s 238ms/step - loss: 0.2315 - accuracy: 0.9062
[0.2314682453707733, 0.90625]
```

نتیجه گیری در مورد تعداد لایه های فریز شده:

افزایش تعداد لایه های قابل آموزش، می‌تواند مدل را برای مسئله ما شخصی‌سازی کند و عملکرد مدلرا در مورد مسئله ما را افزایش دهد. اما افزایش لایه ها به معنی افزایش تعداد پارامترهای قابل یادگیری و در نتیجه پیچیده شدن مدل است. میدانیم مدل پیچیده و تعداد داده آموزش کم معمولا به Overfitting می‌انجامد بنابراین باید مراقب باشیم و با آزمون و خطا بهترین میزان لایه قابل آموزش را پیدا کنیم.

مقایسه LeNet و Inception-V3 :

همانطور که ذکر شد، معماری LeNet برای یک مسئله بسیار ساده تر از مسئله ما طراحی شده بود. با این وجود توانست دقت نه چندان بدی بدست بیاورد و البته بخاطر اینکه داده های ما هم از نظر تعداد و هم از نظر ابعاد برای این مدل بسیار زیاد بود دچار Overfitting شد. بنابراین با توجه به آزمایش هایی که دیدیم از نظر دقت دسته بندی، معماری Inception قطعا برنده است.

ازنظر سرعت همگرایی، مدل اینسپشن بهتر عمل کرد. البته باید توجه داشت که بخش عظیمی از آموزش Inception در مرحله قبل و توسط نویسنده های اصلی این مدل انجام شده و به این علت ما با تعداد epoch کم نیز به صحت بالایی رسیدیم.

از نظر تعمیم پذیری نیز معماری LeNet بیشتر مستعد بیش برازش بود و نیاز بود با استفاده از ریگولاریزیشن این بیش برازش را کاهش دهیم.

منبع مفید برای یادگیری Transfer Learning : https://keras.io/guides/transfer_learning/

پایان