۱. الف) استفاده از لایه‌های کانولوشنی به ما این قابلیت را می‌دهد که یک فرمول خاص را روی تمامی پیکسل‌های تصویر اعمال کنیم. برای مثال می‌دانیم در پردازش تصویر لبه‌یاب Sobel به‌صورت کانولوشن یک کرنل خاص پیاده‌سازی می‌شود. مشابه Sobel می‌توانیم انواع کرنل‌هایی داشته باشیم که حاصل اعمال کانولوشن آن‌ها روی تصاویر ویژگی‌های کاربردی و معنی‌داری از تصویر به ما خروجی بدهند با وجود اینکه تعداد پارامترهای آن نسبت به یک لایه‌ی FC [۱]بسیار ناچیز است اما با بدست آوردن یک فرمول مناسب برای اعمال روی تمامی پیکسل‌های تصویر با توجه به تعدادی از پیکسل‌های همسایگی آن می‌توان برای هر پیکسل ویژگی مناسبی استخراج کرد که در کاربردهای پردازش تصویر و بینایی ماشین بسیار کاربردی‌اند چون بین مقدارهای یک پیکسل و پیکسل‌های همسایگی آن معمولا یک Correlation یا رابطه معنایی خاص و سودمندی وجود دارد که شاید بطور کلی در همه‌ی کاربردهای یادگیری ماشین غیر از بینایی ماشین این رابطه معنایی بین تعدادی از دیتاهای موجود وجود نداشته باشد.

ب) بدون داشتن گسترش مرز ابعاد خروجی ۱۲x۱۲x۱۶ خواهد بود. اگر بخواهیم خروجی ۱۶x۱۶x۱۶ باشد باید از هر سمت ۲ سطر و ۲ ستون اضافه کنیم (مجموعا ۴ سطر و ۴ ستون اضافی) که بعد از کم‌شدن ۴ سطر و ستون حاصل از کانولوشن ابعاد طول و عرض خروجی دوباره ۱۶ شود. تعداد پارامترها برابر ۵x۵x۱۶ = ۴۰۰ است.

پ) با ۳فیلتر ۵x۵ خروجی ۲۸x۲۸x۳ خواهد شد.

با دو لایه هریک ۹فیلتر ۳x۳ خروجی لایه‌ی اول ۳۰x۳۰x۹ می‌شود و در نهایت خروجی لایه دوم ۲۸x۲۸x۹ خواهد شد.

ت) متداول‌ترین نوع pooling ماکسیمم و مینیمم است برای بدست‌آوردن ویژگی‌هایی که در آن‌ها دنبال اکسترمم بودن یک خاصیت هستیم کاربرد دارند چون نورون فعالتر را در نظر می‌گیرد ولی در average pooling همه‌ی نورون‌ها در نظر گرفته می‌شوند و برای تسک‌هایی که دنبال یک ویژگی اکسترمم و نورون فعال‌تر باشیم عملکرد خوبی ندارد. مثلا در object detection بهتر است از max/min pooling استفاده شود چون دنبال ویژگی‌های حداقل و حداکثر بودن هستیم. در برخی کاربردها که می‌خواهیم از یادگیری ویژگی‌های ساختاری تصویر مثل لبه‌ها و بافت‌ها جلوگیری کنیم استفاده از average pooling می‌تواند کارساز باشد مثلا در کاربرد یادگیری کنتراست تصویر استفاده از average pooling مزیت دارد.

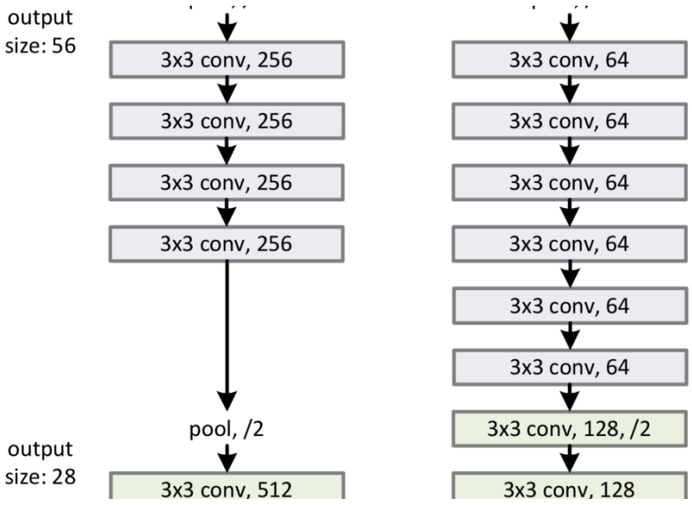
Global average pooling وقتی استفاده می‌شود که اطلاعات مکانی برایمان اهمیت ندارد و می‌خواهیم تعداد پارامترها را بسیار کاهش دهیم. در این صورت با Global average pooling می‌توانیم از هر فیلتر/کانال فقط یک مقدار عددی را نگه داریم چون دیگر برایمان مهم نیست این ویژگی در چه پیکسل‌هایی قوی‌تر است، برایمان مهم است در کل تصویر این ویژگی هست یا نه. برای مثال در ویژگی وجود چشم در تصویر برایمان مهم است تصویر حاوی چشم هست یا نه، برایمان مهم نیست کدام پیکسل‌ها، پس با Global average pooling هم به هدفمان می‌رسیم هم چندین برابر پارامترها را کاهش می‌دهیم. استفاده‌ی اصلی آن در لایه‌ی آخر و قبل از لایه‌ی softmax است که بجای لایه‌ی FC قرار گیرد و بردار خروجی از آن به softmax برود. با این کار مشکل overfitting در لایه‌ی FC آخر از بین می‌رود چون پارامتری وجود ندارد که بخواهد ویژگی‌های زائد Dataset را یاد بگیرد.

---

[۱] Fully Connected

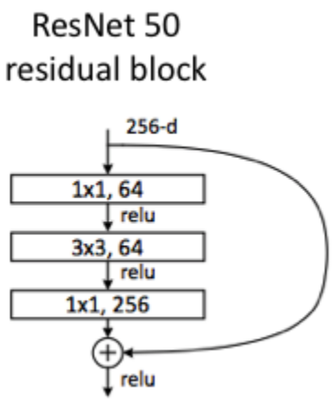
ث) VGG برای بهبود مدل‌های کانولوشنی ابعاد فیلترهای کانولوشنی را کاهش داد و از چندین فیلتر ۳x۳ پشت سر هم بجای فیلترهای بزرگتر استفاده کرد بطوری که عمق شبکه افزایش یابد. ایده‌ی Resnet بیشترکردن عمق لایه‌ها است و برای جلوگیری از مشکل در بهینه‌سازی، یک اتصال مستقیم بین ورودی و خروجی بلوک لایه‌های کانولوشنی اضافه می‌کند بطوری که x را بطور ثابت داشته باشیم و برای رسیدن به خروجی f(x)+x شبکه فقط f(x) را یاد بگیرد. در شبکه‌ی VGG در انتها از flatten استفاده می‌شود در حالی که در Resnet بعد از آخرین بلوک باقی‌مانده و قبل از FC از Global Average Pooling استفاده می‌شود.

دو دلیل اصلی برای بهتربودن سرعت Resnet نسبت به VGG :



۱. هزینه‌ی محاسباتی کانولوشن‌ها در Resnet کمتر است چون تعداد فیلترهای کانولوشنی را کم کم زیاد می‌کند و وقتی که تعداد فیلترها زیاد می‌شوند پارامترها خیلی کمتر شده‌اند چون اگر بدون padding کانولوشن بزنیم همینطور ابعاد کمتر و کمتر می‌شود ولی در VGG با وجود این که تعداد لایه‌ها کمتر است اما لایه‌ها هزینه‌ی محاسباتی بیشتری برای عمل کانولوشن دارند چون تعداد فیلترها از همان ابتدا بیشتر است و هزینه‌ی محاسباتی بیشتری به شبکه تحمیل می‌شود.

۲. ایده‌ی دیگری که به افزایش سرعت در Resnet نسبت به VGG می‌انجامد استفاده از کانولوشن‌های 1x1 است که به کاهش تعداد کانال‌ها می‌انجامد و در نتیجه در فیلترهای کانولوشنی بعدی هزینه‌ی انجام کانولوشن کمتر می‌شود چون کم شدن حجم ورودی به آن‌ها باعث کم شدن تعداد ضرب و تقسیم‌ها در هر کانولوشن و در نتیجه کاهش هزینه‌های محاسباتی و افزایش سرعت می‌شود.



۲. با توجه به تابع ضرر تعریف شده از تابع فعال‌سازی softmax در لایه آخر دو مدل استفاده کردم که معمولا برای تسک‌های دسته‌بندی بکار می‌رود.

الف) خطای مدل FC ≈ ۲.۳ دقت مدل FC ≈ ۱۰٪ خطای مدل کانولوشنی ≈ ۱.۰۶ دقت مدل کانولوشنی ≈ ۶۳٪

```
Convolutional Model
Epoch 1/5
1563/1563 [==============================] - 12s 3ms/step - loss: 1.4337 - accuracy: 0.4933
Epoch 2/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.1797 - accuracy: 0.5885
Epoch 3/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.1204 - accuracy: 0.6100
Epoch 4/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.0865 - accuracy: 0.6230
Epoch 5/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.0531 - accuracy: 0.6331

Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 1.0642 - accuracy: 0.6290
```

```
Fully Connected Model
Epoch 1/5
1563/1563 [==============================] - 5s 3ms/step - loss: 2.3031 - accuracy: 0.0992
Epoch 2/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0987
Epoch 3/5
1563/1563 [==============================] - 5s 3ms/step - loss: 2.3029 - accuracy: 0.0989
Epoch 4/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.1003
Epoch 5/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0988

Loss and Accuracy on Test set :
313/313 [==============================] - 1s 2ms/step - loss: 2.3027 - accuracy: 0.1000
```

خطای کمتر لزوما به معنای دقت بیشتر نیست. ممکن است دقت کمی داشته باشیم اما مقدار خطا در هر یک از تشخیص‌های اشتباه کم باشد و یا ممکن است دقت بالایی داشته باشیم ولی در تشخیص‌های اشتباه مقدار خطا بسیار بزرگ بوده باشد.

	Low Loss	**High Loss**
Low Accuracy	A lot of small errors	A lot of big errors
High Accuracy	A few small errors	A few big errors

ب) مدل کانولوشنی کمی کندتر از مدل FC آموزش می‌بیند. مخصوصا در Epoch اول مدل کانولوشنی بطور محسوس کندتر است.

اجرای اول:

```
Convolutional Model
Epoch 1/5
1563/1563 [==============================] - 12s 3ms/step - loss: 1.4337 - accuracy: 0.4933
Epoch 2/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.1797 - accuracy: 0.5885
Epoch 3/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.1204 - accuracy: 0.6100
Epoch 4/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.0865 - accuracy: 0.6230
Epoch 5/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.0531 - accuracy: 0.6331


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 1.0642 - accuracy: 0.6290
```

```
Fully Connected Model
Epoch 1/5
1563/1563 [==============================] - 5s 3ms/step - loss: 2.3031 - accuracy: 0.0992
Epoch 2/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0987
Epoch 3/5
1563/1563 [==============================] - 5s 3ms/step - loss: 2.3029 - accuracy: 0.0989
Epoch 4/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.1003
Epoch 5/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0988


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 2ms/step - loss: 2.3027 - accuracy: 0.1000
```

اجرای دوم:

```
Convolutional Model
Epoch 1/5
1563/1563 [==============================] - 12s 3ms/step - loss: 1.5051 - accuracy: 0.4671
Epoch 2/5
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2432 - accuracy: 0.5663
Epoch 3/5
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1619 - accuracy: 0.5959
Epoch 4/5
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1051 - accuracy: 0.6165
Epoch 5/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.0618 - accuracy: 0.6331


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 1.0966 - accuracy: 0.6238
```

```
Fully Connected Model
Epoch 1/5
1563/1563 [==============================] - 7s 3ms/step - loss: 2.3029 - accuracy: 0.1003
Epoch 2/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0997
Epoch 3/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0987
Epoch 4/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0962
Epoch 5/5
1563/1563 [==============================] - 4s 3ms/step - loss: 2.3029 - accuracy: 0.0978


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 2ms/step - loss: 2.3028 - accuracy: 0.1000
```

پ) پارامترهای کمتر همیشه به معنای سرعت بیشتر آموزش مدل نیست. تعداد پارامترهای کمتر به معنای این است که مقدار فضای لازم برای ذخیره‌سازی شبکه کمتر است اما سرعت آموزش دیدن شبکه به نوع و حجم محاسبات لایه‌ها بستگی دارد. برای مثال در همین دو شبکه‌ی پیاده‌سازی شده برای سوال۲، شبکه‌ی کانولوشنی با اینکه نصف شبکه‌ی FC پارامتر دارد اما کمی از آن کندتر است.

```
[65] resnet = tf.keras.models.Sequential()

     from tensorflow.keras.applications.resnet50 import ResNet50
     resnet.add(ResNet50(weights=None, classes=24))

     resnet.summary()
```

۳.ب) مدل ResNet50 را بدون مشخص کردن وزن‌ها و با مشخص کردن ۲۴ کلاس خروجی به عنوان یک لایه‌ی functional اضافه کردم.

```
fine_tune_resnet = tf.keras.models.Sequential()

original_resnet = ResNet50(weights="imagenet", classes=1000, include_top=False)

for layer in original_resnet.layers[:]:
    layer.trainable = False

fine_tune_resnet.add(original_resnet)
fine_tune_resnet.add(keras.layers.GlobalAveragePooling2D())
fine_tune_resnet.add(keras.layers.Dense(150, activation="relu"))
fine_tune_resnet.add(keras.layers.Dense(24, activation="softmax"))


fine_tune_resnet.summary()
```

۳.پ) این‌بار ResNet را با همان ۱۰۰۰کلاس و همان دیتاهای imagenet لود کردم ولی لایه FC سر آن را حذف کردم. در عوض در انتها با Global Average Pooling دیتاها را تک بعدی کردم و با دو لایه‌ی FC قابل آموزش آن را تکمیل کردم.

۳.ت) سعی کردم پارامترها را طوری تنظیم کنم مدل Fine Tune به اندازه‌ی مدل با وزن‌های رندوم خوب یا حتی بهتر از آن عمل کند اما در نهایت دقت مدل به ۸۹درصد رسید در حالی که مدل با وزن‌های رندوم خیلی زود به دقت حدود ۱۰۰درصدی می‌رسید.

```
✓ 18s [74] resnet.evaluate(test_generator)

     33/33 [==============================] - 17s 487ms/step - loss: 3.6954e-05 - acc: 1.0000
     [3.695361010613851e-05, 1.0]

✓ 17s [75] fine_tune_resnet.evaluate(test_generator)

     33/33 [==============================] - 18s 508ms/step - loss: 0.4422 - acc: 0.8907
     [0.44219139218330383, 0.8906752467155457]
```

منابع

سوال ۱.ت)

https://www.quora.com/What-is-the-benefit-of-using-average-pooling-rather-than-max-pooling

https://stats.stackexchange.com/questions/257321/what-is-global-max-pooling-layer-and-what-is-its-advantage-over-maxpooling-layer

سوال ۱.ث)

https://stats.stackexchange.com/questions/280179/why-is-resnet-faster-than-vgg

سوال ۲)

https://keras.io/guides/sequential_model/

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

سوال ۳)

https://stackoverflow.com/questions/44720580/resize-image-to-maintain-aspect-ratio-in-python-opencv

https://learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/