به نام خدا

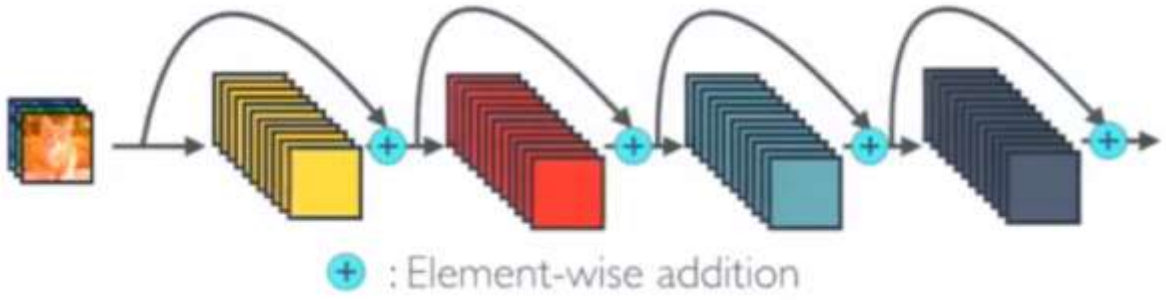
تمرین سری چهارم درس شبکه عصبی

زهرا دهقانیان

۹۸۱۳۱۰۵۹

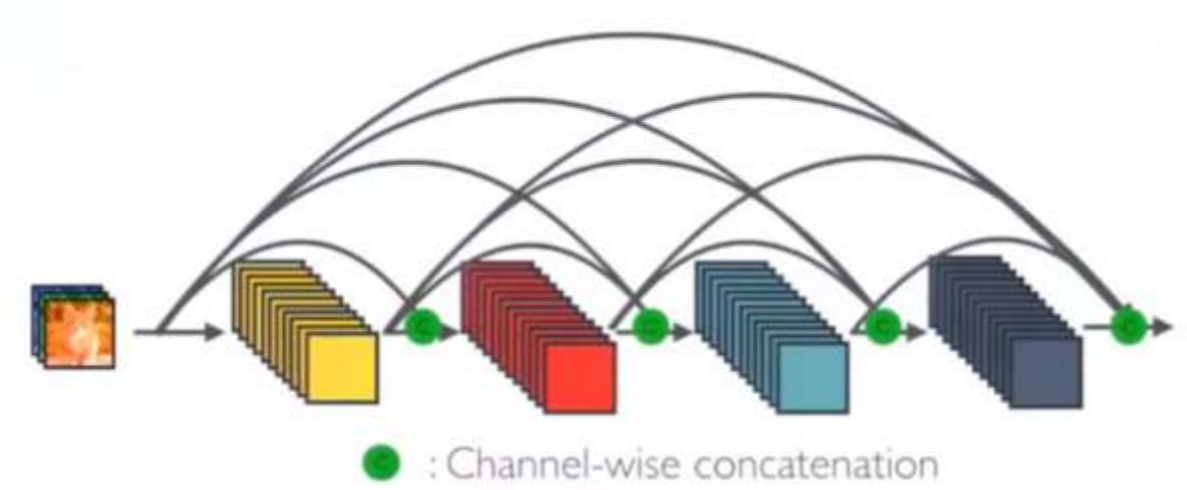
سوال اول)

شبکه رزنت در سال ۲۰۱۵ معرفی شد. معماری این شبکه از تعدادی بلوک رزنت پشت سر هم تشکیل شده است. ویژگی خاص این شبکه ، این است که در داخل هر بلوک،ورودی تابع activation علاوه بر خروجی F ، خود Input این لایه نیز می باشد. یعنی به طور کلی معماری به صورت زیر دارد:



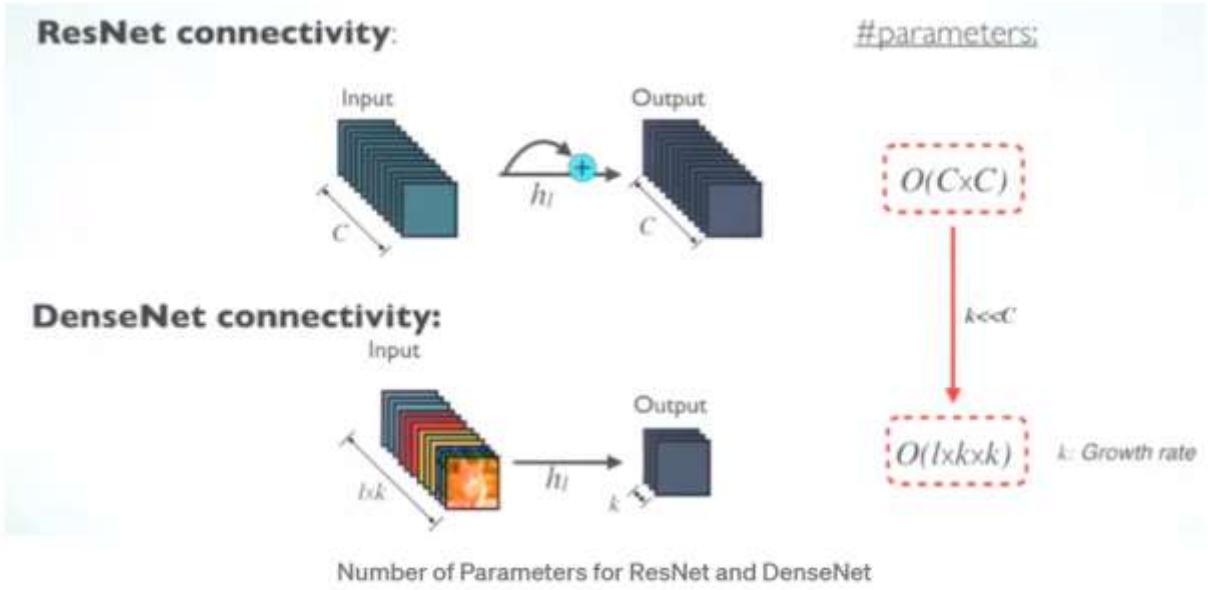
این شبکه (با معماری ۱۵۲ لایه) توانست نتایج خیلی خوبی (خطا ۳,۶٪) در مسابقات دسته بندی ImageNet بدست بیاورد.

در سال ۲۰۱۷ شبکه دنس نت با هدف جلوگیری از مشکل ناپدید شدن/ انفجاری شدن گرادیان پیشنهاد شد. در این شبکه علاوه بر این که ورودی به صورت مستقیم (در کنار خروجی F) به تابع فعالسازی همین لایه داده میشود، این ورودی به تمامی لایه های بعد از این نیز داده میشود.
معماری کلی آن همانند شکل زیر خواهد بود:



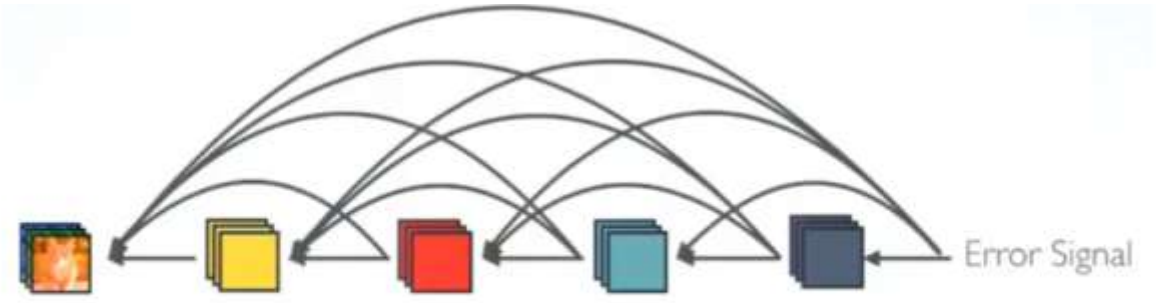
به طور کلی می توان گفت ورژن bc دنس نت در مقایسه با شبکه رزنت این برتری ها را دارد:

۱) تعداد پارامترهای رزنت در هر لایه از O(c*c) است اما برای دنس نت bc از مرتبه O(l*k*k) است و چون k<<c است پس تعداد پارامترهای به مراتب کمتری دارد



Number of Parameters for ResNet and DenseNet

۲) جریان انتشار گرادیان و سیگنال خطا خیلی راحت تر می تواند به لایه های اولیه مدل انشار یابد و لایه ابتدای شبکه میتوانند ارتباط و نظارت مستقیم از لایه های پایانی (classification layers) ها اشته باشند.



Implicit "Deep Supervision"

۳) چون لایه های دنس نت ورودی تمام لایه های قبل را می گیرد ، میتواند ویژگی های مستقل تر و الگوهای قوی تری پیدا کند.

** منبع https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

سوال دوم)

در این بخش با توجه به این برای validation در تابع fit حین آموزش مشخص می کنیم. داده ها را به دو بخش اموزش و تست تقسیم کرده ایم. نسبت استفاده شده در اینجا همانند تمرین قبل ۰,۱ برای تست و ۰,۲ برای validation می باشد.

سوال سوم)

از بین سه مدل توضیح داده شده برای انتقال یادگیری، به نظر می رسد که برای این شبکه باید از مدل سوم استفاده کنیم. به این صورت که ابتدا با ضرایب ثابت بخش feature extractor دسته بند خود را آموزش دهیم. سپس ضریب یادگیری را کم کنیم و چند لایه آخر شبکه را از حالت freeze خارج کنیم و کل شبکه را با هم آموزش دهیم تا بخش feature extractor تنظیم دقیق بشود و دقتمان افزایش یابد.

سوال چهارم)

در این تمرین همانند ویدیو تدریس یار ، از call back ، early stopping استفاده کردیم. داده ها را به کمک کتابخانه tensorflow.keras.preprocessing.image لود و تغییر سایز دادیم. سپس شبکه ResNet را ساختیم. به این صورت که ابتدا یک Instance از این شبکه بدون بخش دسته بند ساختیم و در ادامه یک شبکه با معماری (1500,500,150,50,10,2) قرار دادیم. خروجی بخش اول با feature extractor بدون tuning به صورت زیر است:

```
Epoch 1/1000
18/18 [==============================] - 444s 23s/step - loss: 2.2547 -
accuracy: 0.6434 - val_loss: 0.6739 - val_accuracy: 0.5417
Epoch 2/1000
18/18 [==============================] - 401s 23s/step - loss: 0.6405 -
accuracy: 0.6770 - val_loss: 1.4326 - val_accuracy: 0.9236

wait mode, step: 1
Epoch 3/1000
18/18 [==============================] - 406s 23s/step - loss: 0.7101 -
accuracy: 0.9520 - val_loss: 0.4788 - val_accuracy: 0.9167
Epoch 4/1000
18/18 [==============================] - 399s 22s/step - loss: 0.2471 -
accuracy: 0.9598 - val_loss: 0.8394 - val_accuracy: 0.8194

wait mode, step: 1
Epoch 5/1000
18/18 [==============================] - 401s 23s/step - loss: 0.2409 -
accuracy: 0.8810 - val_loss: 0.3855 - val_accuracy: 0.8889
Epoch 6/1000
18/18 [==============================] - 398s 22s/step - loss: 0.2022 -
accuracy: 0.9474 - val_loss: 2.2093 - val_accuracy: 0.9097

wait mode, step: 1
Epoch 7/1000
18/18 [==============================] - 394s 22s/step - loss: 0.0887 -
accuracy: 0.9870 - val_loss: 0.2763 - val_accuracy: 0.8958
Epoch 8/1000
18/18 [==============================] - 402s 23s/step - loss: 0.1833 -
accuracy: 0.9255 - val_loss: 1.9778 - val_accuracy: 0.9097

wait mode, step: 1
Epoch 9/1000
18/18 [==============================] - 395s 22s/step - loss: 0.1853 -
accuracy: 0.9796 - val_loss: 0.6731 - val_accuracy: 0.8611

wait mode, step: 2
Epoch 10/1000
18/18 [==============================] - 401s 23s/step - loss: 0.1545 -
accuracy: 0.9161 - val_loss: 1.8427 - val_accuracy: 0.9306

wait mode, step: 3
Epoch 11/1000
18/18 [==============================] - 400s 23s/step - loss: 0.1739 -
accuracy: 0.9762 - val_loss: 1.9225 - val_accuracy: 0.8472

wait mode, step: 4
Epoch 12/1000
18/18 [==============================] - 394s 22s/step - loss: 0.2854 -
accuracy: 0.9608 - val_loss: 1.3099 - val_accuracy: 0.9097
```

```
wait mode, step: 5
epoch: 11 : early stopping.
```

پس از این مرحله، ۵۰ وزن های لایه آخر را قابل آموزش می کنیم (۳ مقدار ۱۰ -۳۰-۵۰ لایه را آزمایش کردیم و بهترین حالت آموزش با ۵۰ لایه قابل یادگیری بود). در اینجا آموزش را با نرخ یادگیری کمتر ۰,۰۰۰۱ ادامه میدهیم . خروجی این مرحله به صورت زیر است:

```
Epoch 1/1000
18/18 [==============================] - 416s 23s/step - loss: 0.1404 -
accuracy: 0.9688 - val_loss: 0.8096 - val_accuracy: 0.9097

wait mode, step: 1
Epoch 2/1000
18/18 [==============================] - 405s 23s/step - loss: 0.1024 -
accuracy: 0.9809 - val_loss: 0.4889 - val_accuracy: 0.9097

wait mode, step: 2
Epoch 3/1000
18/18 [==============================] - 406s 23s/step - loss: 0.0692 -
accuracy: 0.9844 - val_loss: 0.8496 - val_accuracy: 0.9236

wait mode, step: 3
Epoch 4/1000
18/18 [==============================] - 406s 23s/step - loss: 0.1083 -
accuracy: 0.9792 - val_loss: 2.0366 - val_accuracy: 0.9236

wait mode, step: 4
Epoch 5/1000
18/18 [==============================] - 401s 23s/step - loss: 0.0318 -
accuracy: 0.9913 - val_loss: 1.0199 - val_accuracy: 0.9167

wait mode, step: 5
epoch: 4 : early stopping.
```

حاصل مقایسه دقت حالت پیش و پس از fine tuning :

```
ResNet Result ==============================>>
accuracy without fine tuning = 0.875
confusion matrix without fine tuning[[31 10]
 [ 0 39]]
accuracy with fine tuning = 0.875
confusion matrix with fine tuning[[31 10]
 [ 0 39]]
```

به نظر می آید ، عملیات تنظیم پارامتر تاثیر چندانی در نتیجه بر روی داده تست نداشته است.

برای مدل DenseNet دقیقا همین مراحل را طی می کنیم با این تفاوت که به عنوان base_model از مدل DenseNet201 استفاده می کنیم. خروجی مرحله ابتدایی به صورت زیر است :

```
Epoch 1/1000
18/18 [==============================] - 252s 12s/step - loss: 97.1027
- accuracy: 0.5136 - val_loss: 2.5230 - val_accuracy: 0.8542
Epoch 2/1000
18/18 [==============================] - 210s 12s/step - loss: 6.8943 -
accuracy: 0.7604 - val_loss: 5.1524 - val_accuracy: 0.8264

wait mode, step: 1
Epoch 3/1000
18/18 [==============================] - 213s 12s/step - loss: 2.7718 -
accuracy: 0.8655 - val_loss: 3.5038 - val_accuracy: 0.8264

wait mode, step: 2
Epoch 4/1000
18/18 [==============================] - 215s 12s/step - loss: 2.7458 -
accuracy: 0.8438 - val_loss: 2.6201 - val_accuracy: 0.8681

wait mode, step: 3
Epoch 5/1000
18/18 [==============================] - 212s 12s/step - loss: 0.9185 -
accuracy: 0.9297 - val_loss: 1.8983 - val_accuracy: 0.8750
Epoch 6/1000
18/18 [==============================] - 212s 12s/step - loss: 0.4232 -
accuracy: 0.9448 - val_loss: 1.9591 - val_accuracy: 0.8819

wait mode, step: 1
Epoch 7/1000
18/18 [==============================] - 212s 12s/step - loss: 0.1486 -
accuracy: 0.9735 - val_loss: 1.4877 - val_accuracy: 0.8889
Epoch 8/1000
18/18 [==============================] - 214s 12s/step - loss: 0.1231 -
accuracy: 0.9817 - val_loss: 1.4847 - val_accuracy: 0.8889
Epoch 9/1000
18/18 [==============================] - 215s 12s/step - loss: 0.0366 -
accuracy: 0.9901 - val_loss: 1.6193 - val_accuracy: 0.8958

wait mode, step: 1
Epoch 10/1000
18/18 [==============================] - 214s 12s/step - loss: 0.0715 -
accuracy: 0.9879 - val_loss: 1.5650 - val_accuracy: 0.8750

wait mode, step: 2
Epoch 11/1000
```

```
18/18 [==============================] - 213s 12s/step - loss: 0.0652 -
accuracy: 0.9846 - val_loss: 1.5377 - val_accuracy: 0.8958

wait mode, step: 3
Epoch 12/1000
18/18 [==============================] - 213s 12s/step - loss: 0.0554 -
accuracy: 0.9870 - val_loss: 1.8531 - val_accuracy: 0.8958

wait mode, step: 4
Epoch 13/1000
18/18 [==============================] - 213s 12s/step - loss: 0.0164 -
accuracy: 0.9964 - val_loss: 2.1877 - val_accuracy: 0.9028

wait mode, step: 5
epoch: 12 : early stopping.
```

پس از fine tuning خروجی به صورت زیر است :

```
Epoch 1/1000
18/18 [==============================] - 444s 23s/step - loss: 2.2547 -
accuracy: 0.6434 - val_loss: 0.6739 - val_accuracy: 0.5417
Epoch 2/1000
18/18 [==============================] - 401s 23s/step - loss: 0.6405 -
accuracy: 0.6770 - val_loss: 1.4326 - val_accuracy: 0.9236

wait mode, step: 1
Epoch 3/1000
18/18 [==============================] - 406s 23s/step - loss: 0.7101 -
accuracy: 0.9520 - val_loss: 0.4788 - val_accuracy: 0.9167
Epoch 4/1000
18/18 [==============================] - 399s 22s/step - loss: 0.2471 -
accuracy: 0.9598 - val_loss: 0.8394 - val_accuracy: 0.8194

wait mode, step: 1
Epoch 5/1000
18/18 [==============================] - 401s 23s/step - loss: 0.2409 -
accuracy: 0.8810 - val_loss: 0.3855 - val_accuracy: 0.8889
Epoch 6/1000
18/18 [==============================] - 398s 22s/step - loss: 0.2022 -
accuracy: 0.9474 - val_loss: 2.2093 - val_accuracy: 0.9097

wait mode, step: 1
Epoch 7/1000
18/18 [==============================] - 394s 22s/step - loss: 0.0887 -
accuracy: 0.9870 - val_loss: 0.2763 - val_accuracy: 0.8958
Epoch 8/1000
18/18 [==============================] - 402s 23s/step - loss: 0.1833 -
accuracy: 0.9255 - val_loss: 1.9778 - val_accuracy: 0.9097

wait mode, step: 1
```

```
Epoch 9/1000
18/18 [==============================] - 395s 22s/step - loss: 0.1853 -
accuracy: 0.9796 - val_loss: 0.6731 - val_accuracy: 0.8611

wait mode, step: 2
Epoch 10/1000
18/18 [==============================] - 401s 23s/step - loss: 0.1545 -
accuracy: 0.9161 - val_loss: 1.8427 - val_accuracy: 0.9306

wait mode, step: 3
Epoch 11/1000
18/18 [==============================] - 400s 23s/step - loss: 0.1739 -
accuracy: 0.9762 - val_loss: 1.9225 - val_accuracy: 0.8472

wait mode, step: 4
Epoch 12/1000
18/18 [==============================] - 394s 22s/step - loss: 0.2854 -
accuracy: 0.9608 - val_loss: 1.3099 - val_accuracy: 0.9097

wait mode, step: 5
epoch: 11 : early stopping.
```

حاصل مقایسه دقت مدل پیش و پس از fine tuning :

```
DenseNet Result ====================>>>>
accuracy without fine tuning = 0.925
confusion matrix without fine tuning[[39  2]
 [ 4 35]]
accuracy with fine tuning = 0.9375
confusion matrix with fine tuning[[37  4]
 [ 1 38]]
```

در نهایت حاصل مقایسه دو مدل (با متریک دقت) به صورت زیر است:

```
DenseNet Result is better
==========================
accuracy DenseNet = 0.9375
accuracy ResNet = 0.875
```

به نظر می رسد ، طبق خروجی محاسبه شده، شبکه Dense بهتر از رزنت عمل کرده و توانسته دقت بالاتری در حدود ۶٪ بدست بیاورد.