ECE30001 Deep Learning Application Final Test

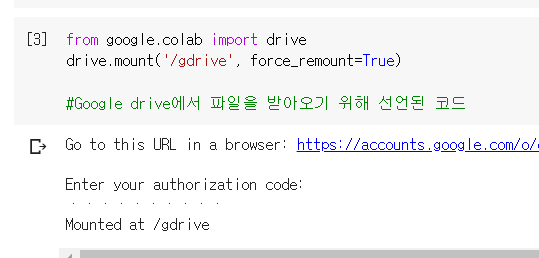
Short Report

1. Usage (사용법)

1) Trainer 파일

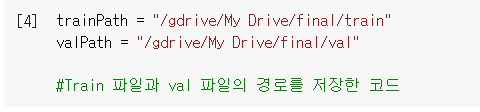
(1) 파일 불러오기

(구글 드라이브를 사용할 경우)



위 코드를 실행시켜 구글 드라이브와 연동을 시킨다

(파일 경로 설정)



TrainPath와 valPath를 ‘train파일’과 ‘val파일’ 이 저장된 위치로 설정해준다.

그 다음에는 코드를 위에서부터 순서대로 run하면 된다.

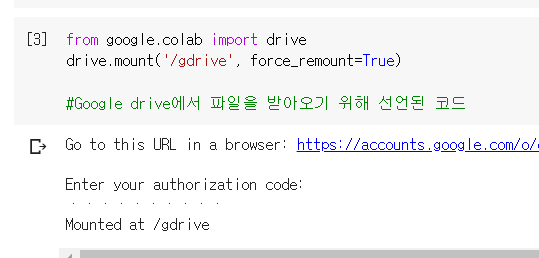


학습이 끝난 뒤에는 위 코드를 사용하여 parameter(weight)를 저장해준다.

2) Tester 파일

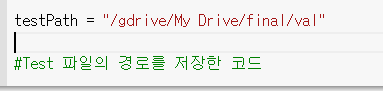
(1) 파일 불러오기

(구글 드라이브를 사용할 경우)

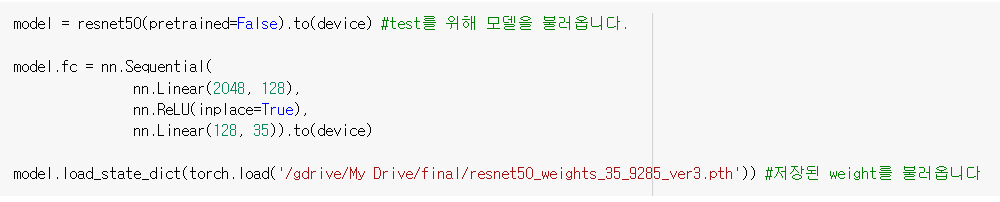


위 코드를 실행시켜 구글 드라이브와 연동을 시킨다.

(파일 경로 설정)

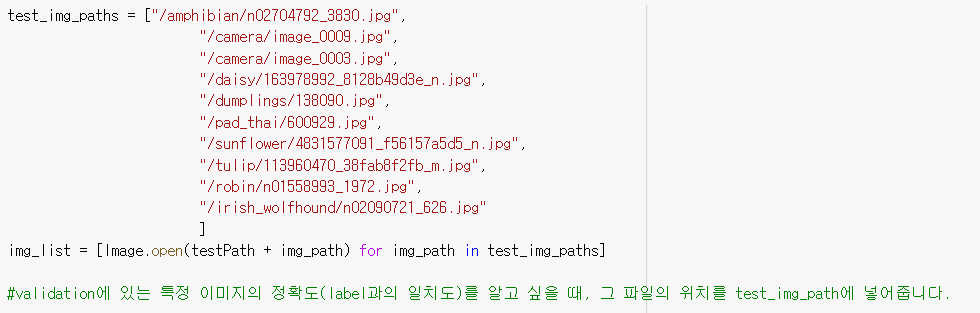


testPath를 ‘test파일’이 저장된 위치로 설정해준다.



위 코드에서 model.load\_state\_dict(torch.load()) 중 torch.load함수의 argument값이 저장된 weight의 위치이다.

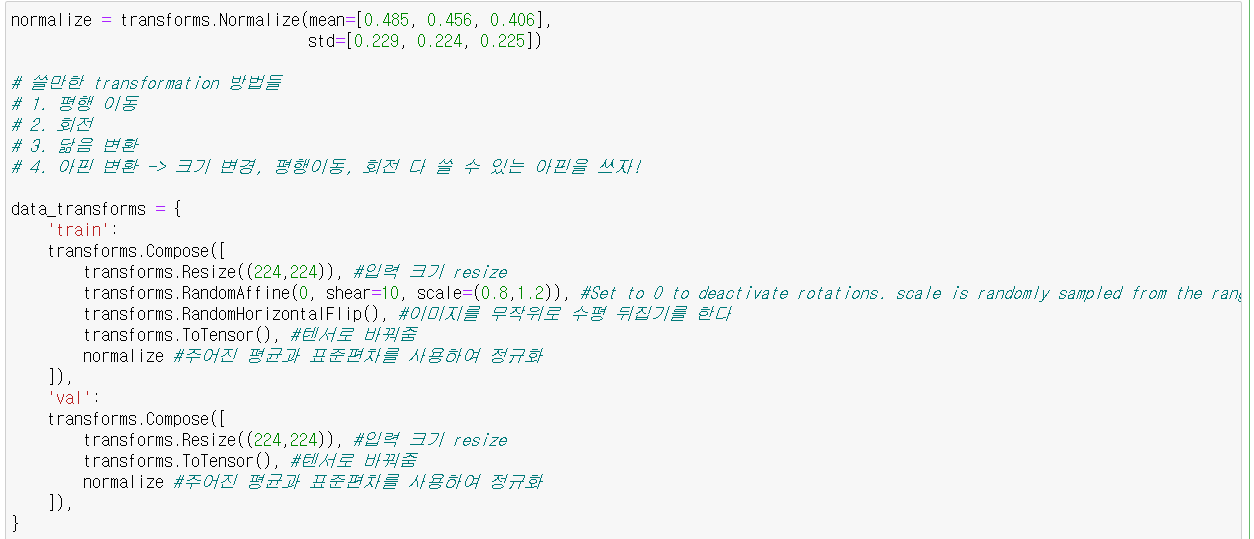
(2) 이미지 파일 하나의 정확도 알기



위 코드에서 test\_img\_paths의 값으로 원하는 이미지의 위치를 넣어준다.

2. Model design

첫번째 레이어에서는 7x7 convolution을 사용하였다. 그 외 모든 convolution 연산에서는 1x1이나 3x3 크기의 커널이 사용되었다. Feature map의 크기가 같은 레이어는 출력 feature map의 개수가 동일하다. 하지만 feature map의 크기가 반으로 작아지는 경우 출력 feature map의 개수가 2배가 된다. 복잡성을 줄이기 위해 pooling은 거의 사용되지 않고, convolution 연산의 stride를 2로 하여 feature map의 크기를 줄였다. 이미지가 반으로 작아진 경우, identity block이 사용되며, 입력 값을 바로 더하지 않고, 1x1 convolution 연산을 stride 2로 설정하여 feature map의 크기와 개수를 맞추어 준 다음 더해준다. Bottleneck layer는 연산 량의 줄이기 위해 Residual Block 내에, 1x1, 3x3, 1x1 convolution 연산을 쌓았다. 1x1 convolution 연산으로 feature map의 개수를 줄였다가 3x3을 거친 후, 1x1 convolution 연산으로 차원을 늘려준다. 이 모델의 경우 학습 속도도 빠르고(약 2시간 30분), 정확도 역시 좋다 (validation data에 대해 accuracy 92.85%).

3. Training technique

입력 크기를 (244, 244)로 resize를 해주었고, Affine 변환을 사용하였다. 데이터 양을 임의로 복사하여 증가시키고 이 복사한 데이터에 노이즈를 추가할까 고민도 하였지만, 그렇게 하지 않아도 학습 데이터 양이 충분하다고 생각하여 이 방법은 사용하지 않았다. 이미지를 불러올 때 무작위로 수평 뒤집기를 하도록 하였다. 그 뒤 tensor로 바꿔주고 정규화를 시켰다.

4. Experiments

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Renet18 | Resnet34 | Resnet50  (batch size  = 32) | Resnet50  (batch size  = 64) | Renet101 | VGG19 |
| Epoch 1 | 0.8817 | 0.8757 | 0.9003 | 0.9002 | 0.8960 | 0.8501 |
| Epoch 2 | 0.8878 | 0.9014 | 0.9064 | 0.9103 | 0.9153 | 0.8613 |
| Epoch 3 | 0.8946 | 0.9021 | 0.9110 | 0.9150 | 0.9171 | 0.8709 |
| Epoch 4 | 0.8907 | 0.9025 | 0.9143 | 0.9210 | 0.9235 | 0.8871 |
| Epoch 5 | 0.8892 | 0.8939 | 0.9146 | 0.9211 | 0.9200 | 0.8803 |
| Epoch 6 | 0.8903 | 0.8992 | 0.9214 | 0.9214 |  |  |
| Epoch 7 | 0.8892 | 0.9000 | 0.9146 | 0.9271 |  |  |
| Epoch 8 | 0.8910 | 0.9068 | 0.9160 | 0.9189 |  |  |
| Epoch 9 | 0.8882 | 0.9053 | 0.9218 | 0.9210 |  |  |
| Epoch 10 | 0.8910 | 0.9110 | 0.9280 | 0.9171 |  |  |

<표1. Epoch 마다 각 모델 별 validation data에 대한 accuracy>

Resnet18은 epoch 5까지 35분 밖에 안 걸렸을 정도로 학습 속도는 빨랐지만, validation accuracy가 0.9를 넘지 못 한다.

Resnet34는 epoch를 20까지 진행해 봤지만 validation accuracy가 9.2를 넘지 못 했다.

Resnet101은 Colab 메모리가 부족하여 epoch 6 이상은 학습하지 못했다. 특히 resnet101의 경우 특정 label 파일의 validation accuracy가 현저히 낮았다. 대부분 label의 validation accuracy는 90%를 넘은 반면, 한 label의 validation accuracy는 70%대가 나왔다.

VGG19는 성능도 잘 안 나왔고(validation에 대한 accuracy 88%), Epoch 5까지 3시간이 걸릴 정도로 training 시간이 너무 오래 걸려서 epoch 5까지만 학습했다.

여기서 눈에 띄었던 것이 resnet50과 resnet101의 학습 시간 차이는 많이 났지만, 성능의 차이는 적었다는 것이다. 이를 Resent50과 101이 128만장을 학습할 때 0.6의 error 차이가 나는데 이번 기말고사에서는 19000장 정도만 학습에 사용될 정도로 dataset이 적었기 때문에 이런 결과가 나온 것으로 해석하였다. 따라서 학습 속도도 빠르고, 가장 성능이 좋았던 resnet50을 사용하기로 하였다.

Batch size를 32와 64 둘 다 실험해보았다. 실험에 앞서 세운 가설은 batch size가 클수록 gradient는 정확해지고, 작은 batch size는 parameter space 상에서 상대적으로 최신 상태의 gradient를 사용할 수 있기 때문에 정확도에 차이가 있을 것이라고 세웠다. 실제로 실험을 해보니 둘의 차이가 크게 없었기 때문에 epoch 10까지 학습하였을 때 validation accuracy가 더 높고 학습 안정성이 더 좋은 32 batch size를 사용하기로 하였다.

Resnet50의 training epoch는 45까지 진행하였다. Epoch별 training loss와 training과 validation의 accuracy는 다음 표와 같다.

|  |  |
| --- | --- |
|  | Training loss / Training accuracy / validation accuracy |
| Epoch 1 | 0.6254 / 0.8330 / 0.9003 |
| Epoch 2 | 0.3311 / 0.8966 / 0.9064 |
| Epoch 3 | 0.3001 / 0.9038 / 0.9110 |
| Epoch 5 | 0.2606 / 0.9157 / 0.9143 |
| Epoch 10 | 0.2312 / 0.9258 / 0.9225 |
| Epoch 15 | 0.1996 / 0.9333 / 0.9214 |
| Epoch 20 | 0.1864 / 0.9382 / 0.9264 |
| Epoch 25 | 0.1670 / 0.9237 / 0.9221 |
| Epoch 35 | 0.1689 / 0.9476 / 0.9285 |
| Epoch 40 | 0.1593 / 0.9462 / 0.9293 |
| Epoch 45 | 0.1633 / 0.9456 / 0.9296 |

<표2. Batch size 32인 resnet50의 training loss, training accuracy, validation accuracy>

Epoch 25부터는 training loss가 정체되어 있는 것을 알 수 있었다. Epoch 20부터 Epoch 30까지는 validation loss가 0.2486에서 0.2717로 증가하고 validation accuracy도 0.9264에서 0.9201으로 감소하여 early stopping을 해야할지 고민하였지만, Epoch 31부터 Epoch 35까지는 다시 validation loss도 0.2455로 감소하였고 validation accuracy 역시 0.9285로 증가하였기 때문에 최종적으로 Epoch 35의 weight 값을 저장하였다.

5. Conclusion

Stride가 2인 1x1 convolution을 사용하면 convolution과 down sampling이 동시에 되어 operation이 cheaper computationally하게 된다. 따라서 pooing을 써서 dimensionality reduction을 하지 않아도 된다.

Resnet을 사용할 때, 기말고사 데이터 셋에 대해 90% 이상인 좋은 인식 결과를 얻기 위해서는 최소 50개 이상의 layer를 쓰는 것이 좋다.

Resnet50과 Resnet101은 Epoch 5까지 training 하는데 약 70분 정도 차이가 났다 (Resnet50은 110분, resnet101은 180분). 하지만 성능은 크게 차이가 없었기 때문에 이와 같은 dataset에서는 Resnet50을 쓰는 것이 효율적이다. 또한 메모리가 충분하지 않을 때는 depth가 깊지 않은 모델을 사용하는 것이 좋다.

추가적으로, pretrained 된 resnet하고 pretrained되지 않고 바닥부터 학습하는 resnet하고는 똑같은 성능(accuracy)을 내기 위해 epoch가 55만큼 차이 났다. 따라서 학습 시간을 줄이기 위해서는 pretrained된 것을 쓰는 것이 좋다.