|  |  |
| --- | --- |
| **Computer Vision Assignment #3** | |
| **학번: 202020790** | **이름: 민경현** |
| **최종 Test 성능: 78.31%** | |
| **제안 방식 요약 설명 (Contribution)** | |
| - Ajou BB 제출시 ipynb 파일, 본 결과 보고서 (PDF 변환 필수) 별도 제출  **\* 결과 보고서**  - 폰트 크기는 10  - 결과 보고서는 최대 1장  - 결과 보고서는 itemize 하여 작성 (contribution 위주 작성)  - 결과 보고서에 꼭 loss graph 포함시킬 것 (미 포함시 감점)  **\* 딥러닝 코드 관련 Constraints** (하기 제약 사항을 지키지 못할 경우 0점 처리)  - ~~최대~~ Conv 8개, FC 3개 (마지막 FC포함) 포함 총 최대 11개 사용 가능  - 최대 ~~20 epoch만~~ 돌림 (0 epoch부터 시작 19 epoch으로 끝남) -> 50 epoch  - 추가 DB 활용 금지 (Cifar-10만 활용할 것)  - Pretrain model 사용 금지 (ImageNet 등등) -> from scratch로 학습할 것  **\* 평가 방법**  - Test 성능 기반 그룹을 나누어서 차등 점수 부여 | |

<https://github.com/enjoeyland/CV_CIFAR10>

|  |  |
| --- | --- |
| 뼈대코드 | 78.31% (20 epoch) |
| RandomCrop, RandomHorizontalFlip 8 Conv, 1 FC epoch 20 | |

|  |  |
| --- | --- |
| 뼈대코드 | 85.44% |
| epoch만 50으로 늘렸다.  38 epoch부터 train accuracy는 91%, test accuracy는 85%에 머물러 있었다. | |

|  |  |
| --- | --- |
| ResNet BasicBlock [1,1,1,1] | 81.31% |
| 각 block은 2개의 conv  (64,8,8) 1 block, (128,4,4) 1block, (256,2,2) 1block, (512,1,1) 1block  약 20 epoch 정도부터 accuracy는 79.80%에 머물러 있게 된다.  VGG보다 빠른 수렵이 강점인 것 같다.  아직 cifar10에 맞게 customize가 필요할 것 같다.  (512 , 1, 1)은 의미가 있을지 의문이다. | |

|  |  |
| --- | --- |
| ResNet BasicBlock [1,2,1] | 82.09% |
| (64,8,8) 1 block, (128,4,4) 2block, (256,2,2) 1block  13 epoch부터 accuracy는 78.59%에 머물러 있게 된다.  파라미터 수를 150만개로 대폭 줄였다. 하지만 비슷한 결과가 나오고 있다.  표현력이 부족한 것 같다. | |

|  |  |
| --- | --- |
| ResNet BasicBlock [1,2,1] with bigger channel | 82.15% |
| (128,8,8) 1 block, (256,4,4) 2block, (512,2,2) 1block  파라미터 수가 601만개가 됐다.  6 epoch에 accuracy는 77.57% 학습이 늦어졌다.  비슷한 결과이다. Channel수의 문제가 아니다 | |

|  |  |
| --- | --- |
| ResNet BasicBlock [2,1,1] | 80.82% (18 epoch) |
| (64,8,8) 2 block, (128,4,4) 1block, (256,2,2) 1block  7 epoch부터 accuracy는 76.44%에 머물러 있게 된다.  더 안 좋아졌다.  ResNet의 한계가 있는 것 같다.  Shorcut을 만들때에도 Conv layer를 사용하게 된다. 11 layer로 제한되고 적은 layer를 사용하니 굳이 shorcut이 있는 ResNet을 사용하지 않는 것이 좋은 것 같다.  그러나 참조한 ResNet code에 다양한 기법이 있는 것 같으니 VGG에 적용하여 비교해보겠다. | |

|  |  |
| --- | --- |
| VGG AdamW | 87.30% |
| Optimizer를 SGD대신 AdamW를 사용하고  AdamW의 default learning rate인 0.01로 학습  SGD처럼 단순 gradient만 사용하는 것이 아닌  AdamW는 gradient 뿐만 아니라 square gradient(adaptive)를 사용한다. | |

|  |  |
| --- | --- |
| VGG에 shortcut 1개 추가 | 88.93% |
| ~~ResNet이 빠르게 학습되는 것이 shortcut 때문일 것이라고 생각이 되었다.~~  ~~80%를 넘는데 기본VGG는 9epoch가 걸렸다. 하지만 shorcut이 있을 때 11epoch가 걸렸다.~~  Shortcut은 초기 layer의 학습을 더 원활하게 할 것이다.  그러나 위의 ResNet처럼 많은 shortcut은 성능을 떨어트리는 것 같아서,  한 개의 shortcut만 추가하였다.  25 epoch 쯤부터 train accuracy 91%쯤에서 변동이 아주 적어진다. 이는 learning rate가 높아서 그런 것 같다. | |

|  |  |
| --- | --- |
| VGG\_shortcut\_1 with fine tuning | - |
| 25 epoch 쯤부터 train accuracy 91%쯤에서 변동이 아주 적어진다(최대 94%). 또한 accuracy가 낮아지는 경우도 있다. 이는 learning rate가 높아서 그런 것 같다.  25 epoch부터 learning rate를 낮추어 fine tunning을 한다.  의미 없음…. | |

|  |  |
| --- | --- |
| VGG\_shortcut\_1 with AdaptiveAvgPool2d(1,1) | 89.52% |
| VGG에서는 maxpooling을 통해 (512,2,2) shape으로 나와 2048x10 FC을 해야했다.  그런데 AdaptiveAvgPooling을 통해 (512,1,1) shape으로 바꾸닌 FC가 512로 줄었다.  아마 더 핵심 feature로 extract되어 성능이 증가된 것 같다.  8epoch만에 accuracy 80% 돌파 학습속도 향상  Train accuracy 96.57%로 증가  Epoch 27부터 추가학습 X | |

|  |  |
| --- | --- |
| VGG\_shortcut\_1 layer추가 및 구족 변경 | 88.64% |
| Output size가 작으므로 마지막 layer근처에서 feature extraction을 더 하여 작게 해야 backpropagation에서 잘 학습이 될것 같다.  (64, 32, 32)x2, (128, 16, 16)x2, (256, 8, 8)x2, (512, 4, 4)x1, (256, 4, 4)x1, (256,2,2)x1, (128,2,2)x1  4 epoch만에 accuracy 80% 돌파  Epoch 36부터 Train accuary를 96%로 유지 | |

|  |  |
| --- | --- |
| 데이터 augmentation 더 많이 적용 | 79.65% |
| Train accuracy와 test accuracy의 차이가 많이 나고 있다. 즉 overfiting 되고 있는 것 같다.  데이터의 일반화가 필요한 것 같다.  RandomVerticalFlip, ColorJitter, RandomAffine을 사용하여 augment를 하였다.  Train accuracy는 크게 낮아졌고(75.75%) test accuracy가 train accuracy보다 높게 나오고 있다. | |

|  |  |
| --- | --- |
| 데이터 augmentation 비율 조정 | 84.50% |
| Train accuracy는 크게 낮아졌고(76.93%) test accuracy가 train accuracy보다 높게 나오고 있다. | |