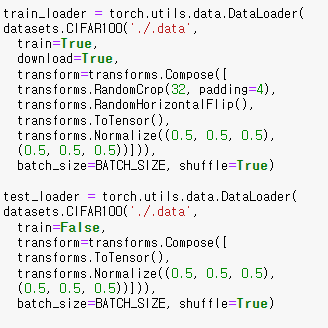
Load dataset

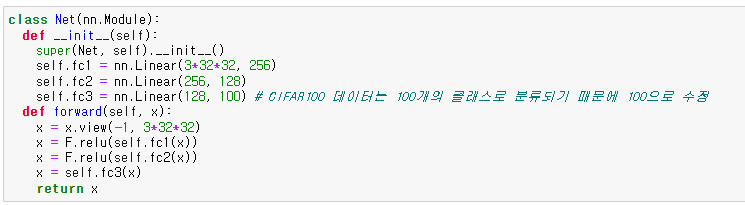


Data augmentation: RandomCrop, RandomHorizontalFlip

학습 시간을 우려하여 별도의 데이터 증강을 하진 않았다.

**\*알고리즘 설명**

1. DNN



Layer 1: (3 x 32 x 32, 256)

Layer 2: (256, 128)

Layer 3: (128, 100)

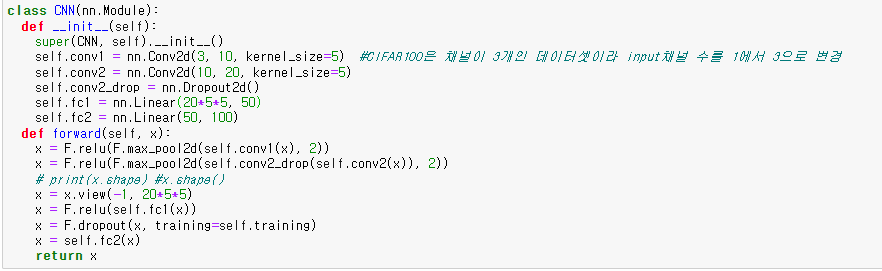
EPOCHS = 100

BATCH\_SIZE = 256

입력 데이터의 크기가 3 x 32 x 32이고 100개의 클래스로 분류해야하기 때문에

self.fc1에서 784 → 3\*3\*32, self.fc3에서 10 → 100으로 수정하였다.

2. CNN



conv1: (32 x 32 x 3, 28 x 28 x 10), max pooling: (28 x 28 x 10, 14 x 14 x 10)

conv2: (14 x 14 x 10, 10 x 10 x 20), max pooling: (10 x 10 x 20, 5 x 5 x 20)

fc1: (5 x 5 x 20, 50) fc2: (50, 100)

EPOCHS = 100

BATCH\_SIZE = 256

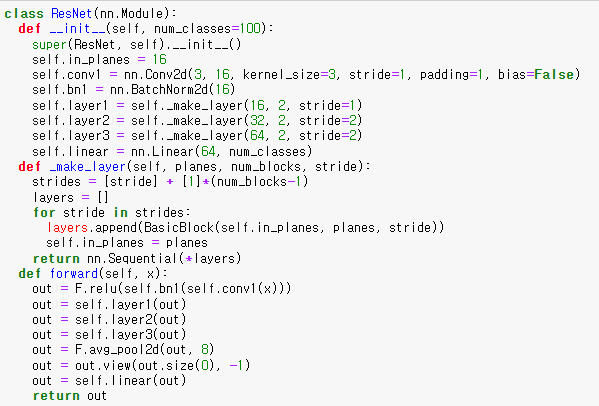
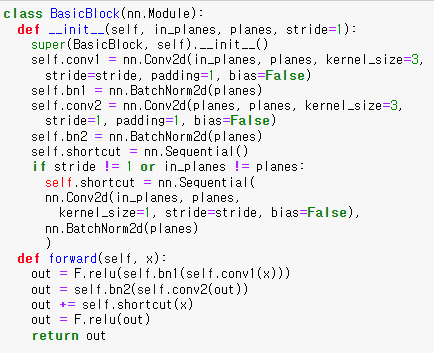
self.conv1에서 1 → 3 수정, 입력 이미지가 3개의 채널을 가지기 때문에

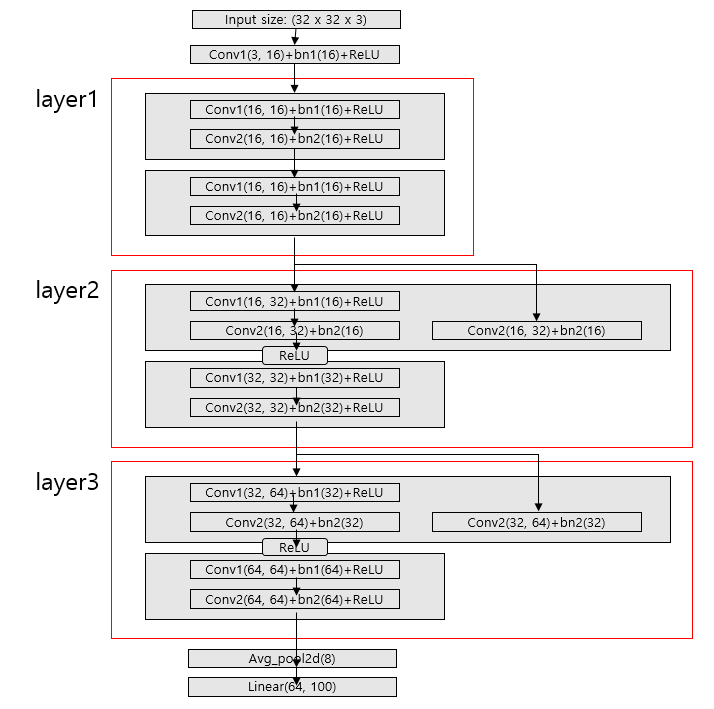
self.fc1에서 320 → 20\*5\*5 수정

conv1을 지나면 28 x 28 x 10, 여기서 max pooling(2)를 적용하면 14 x 14 x 10

conv2를 지나면 10 x 10 x 20, 여기서 max pooling(2)를 적용하면 5 x 5 x 20이기 때문에

3. ResNet



EPOCHS = 100

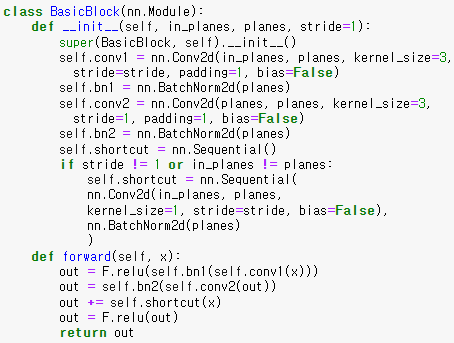
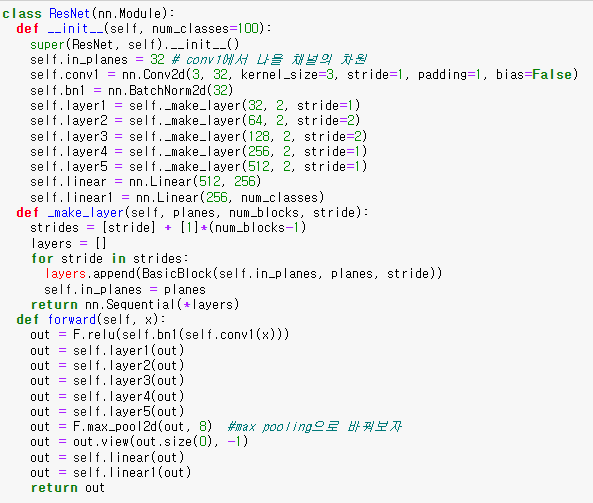
BATCH\_SIZE = 256

100개의 클래스로 분류하기 때문에 num\_classes=100으로 수정

scheduler를 적용한다. epoch 50 이후에 lr에 0.1을 곱한다. 이로 인해 학습할 수록 점점 더 정교하게 최적화할 수 있다.

4. Customized ResNet (성능을 향상시키기 위해 적용한 방법)

EPOCHS = 200, BATCH\_SIZE = 16

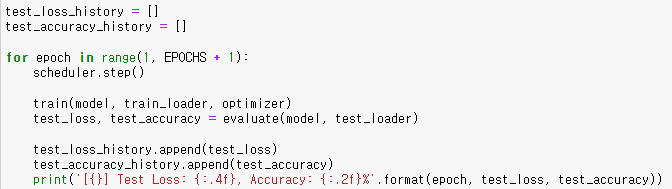


(1) self.conv1를 16에서 32로 늘렸다. 채널의 수를 늘려 연산할 파라미터 수를 늘렸다. 계산 비용이 더 높아진다는 단점이 있지만 정보가 많아짐으로써 더 다양한 특징을 인식하고 더 정확한 예측을 수행할 수 있을 것이라 기대했다. (2) self.layer4와 self.layer5, self.linear1를 추가했다. 일반적으로 신경망을 더 쌓으면 성능이 높아지기 때문이다. (3) self.layer4는 128채널을 받아 256채널을 출력하는 BasicBlock 1개와 256채널에서 256채널을 내보내는 BasicBlock 1개로 구성되어 있다. 128채널을 받아 256채널을 출력하는 BasicBlock에서 shortcut이 이어진다. (4) self.layer5는 256채널을 받아 512채널을 출력하는 BasicBlock 1개와 512채널에서 512채널을 내보내는 BasicBlock 1개로 구성되어 있다. 256채널을 받아 512채널을 출력하는 BasicBlock에서 shortcut이 이어진다.

(5) avg\_pool2d을 max\_pool2d로 바꿨다. max\_pooling은 가장 두드러지는 특징을 찾을 수 있기 때문에 성능이 더 올라갈 것이라 예상했다.



(6) Optimizer를 SGD에서 Adam으로 교체하였다. SGD는 현재 위치에서의 기울기만 고려하여 최적값을 찾기 때문에 local minima에 빠질 확률이 높다는 단점이 있다. Adam은 기울기의 방향과 stepsize를 적절하게 조절하기 때문에 성능이 좋다.



(7) batch size를 16으로 줄이고 학습률도 줄였다.

배치사이즈가 작으면 학습률도 작아야 성능이 좋고, 반대로 배치사이즈가 크면 학습률도 커야 성능이 좋다.

배치사이즈가 작고 학습률도 작으면 한 번에 적은 데이터로 조금씩 학습하기 때문에 그 만큼 step, 학습 수가 많아지므로 local minima에 빠져 optimal minima에 도달하긴 힘들지만, 초기 방향은 쉽게 잡히고 수렴은 잘 될 것이다.

배치사이즈가 크고 학습률도 크면 한 번에 많은 데이터로 많이 학습하기 때문에 초기 방향을 잡기 힘들어 lr을 잘 조절해야 할 것이다. 과적합의 문제도 존재한다. local minima에 걸릴 확률이 적고, optimal minima에 수렴할 수 있다.

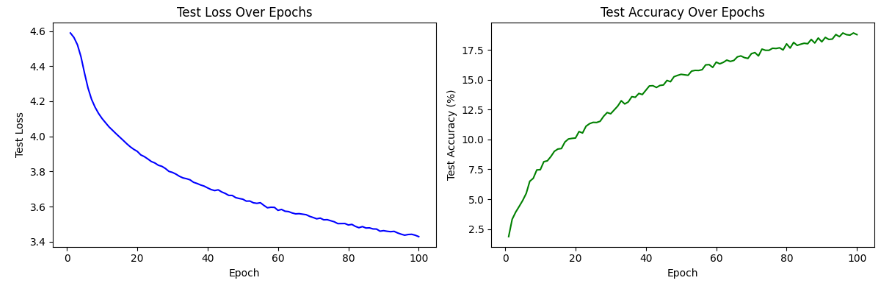
배치사이즈(256), lr(0.01)에서는 정확도가 70.65%였지만

배치사이즈(16), lr(0.0001)에서는 정확도가 73.77%로 성능이 올랐다.

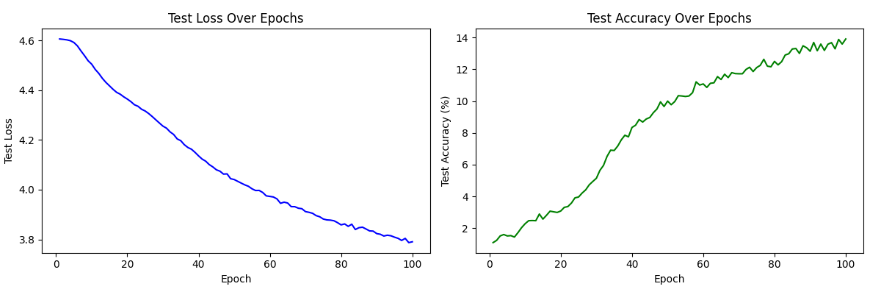
과적합을 우려하여 각 BasicBlock에 대해 Dropout을 적용해보았지만 성능이 오히려 더 떨어졌다.

**\*알고리즘 비교**

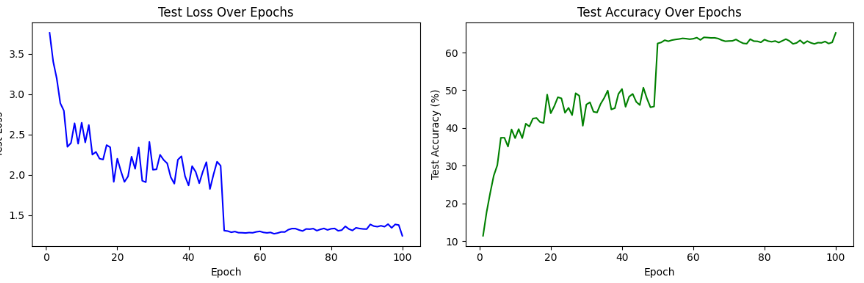
1. DNN

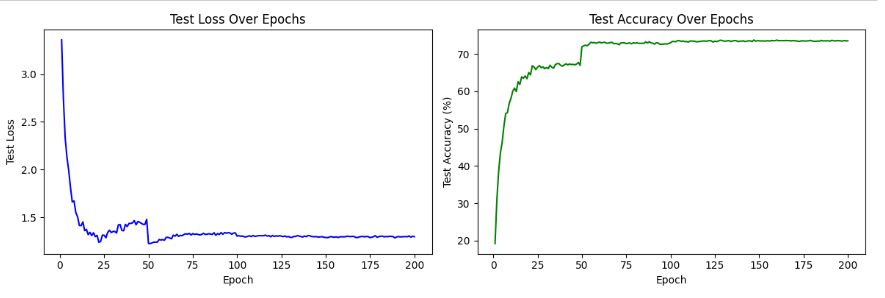
2. CNN

3. ResNet

4. Customized ResNet

ResNet은 DNN, CNN보다 성능이 2~3배 좋아졌다. ResNet과 Customized ResNet은 스케줄러를 이용하여 epoch 50마다 학습률을 줄였기 때문에 50의 배수의 epoch에서 성능이 올라간 것을 확인할 수 있다. H(x) – x에 대해 학습을 진행하기 때문에 이전의 특징을 반영할 수 있다.

ResNet보다 Customized ResNet이 더 성능이 좋아진 이유로 (1) self.in\_place의 초기값을 늘려줌으로써 채널의 수가 증가하여 정보가 많아짐으로써 더 다양한 특징을 인식하고 더 정확한 예측을 수행한 것으로 보인다. (2) 배치사이즈와 학습률을 같이 줄임으로 인해 좀 더 정교한 학습이 가능해진 것으로 보인다.

예상 외로 DNN보다 CNN이 성능이 더 안좋은 결과가 나왔다. 그 이유로 (1) CNN이 DNN에 비해 연산량이 많고, (2) Pooling층을 거치면서 정보의 손실이 일어난 것으로 추측된다.

**시도하지 못했던 성능을 높이기 위한 방법:** (1) 데이터 증강을 통해 학습할 데이터의 양을 늘리면 성능이 더 올라갈 것으로 추측한다. (2) ReLU대신 tanh 활성화 함수를 이용한다. tanh는 곡선 형태이기 때문에 입력 데이터의 미묘한 변화를 감지하는 데 유용하다. 실제로 epoch을 10으로 간단하게 학습해보니 ReLU보다 2~3% 성능이 높았다. 하지만 기울기 소실에 빠질 위험이 있고 ReLU보다 속도가 느리기 때문에 epoch을 크게하여 시도해보진 못했다.