[머신러닝딥러닝 실습] MNIST 97.75% 만들기

진행 전략:

모델을 성능을 높이기 위해 하이퍼 파리미터를 바꾸면 좋겠다고 생각을 했습니다. 저는 Learning Rate, Batch size, Epochs, Optimizer, Activation Function, 모델 레이어 구조, BN, DO 위주로 하이퍼 파라미터를 바꾸며 모델의 성능을 비교 해보았습니다.

(1) Learning Rate (학습률)

학습률은 보폭을 결정합니다. 즉, 얼만큼 이동할지를 조정하는 하이퍼 파라미터로, 경사 하강법에 어느 정도로 기울기 값을 적용할지 결정합니다. 학습률이 크면 보폭이 크므로, 성큼 성큼 이동하고, 작으면 조금씩만 이동하는 개념입니다.

학습률이 너무 작거나 클 때는 다음과 같습니다. 왼쪽 그림과 같이 학습률이 작으면, 최적점에 이르기까지 매우 오래 걸립니다. 학습률이 너무 크면, 발산하면서 모델이 최적값을 찾을 수 없을 수 있습니다.

저 좋은 모델의 성능을 찾기위해 모델의 합습률을 0.001 -> 0.025 -> 0.0001 세 번을 바꿔가면서 비교를 해보았습니다.

(2) Batch size

Batch size란 전체 트레이닝 데이터 셋을 여러 작은 그룹을 나누었을 때 batch size는 하나의 소그룹에 속하는 데이터 수를 의미합니다. 전체 트레이닝 셋을 작게 나누는 이유는 트레이닝 데이터를 통째로 신경망에 넣으면 비효율적이 리소스 사용으로 학습 시간이 오래 걸리기 때문입니다.

Batch size가 더 작을수록 더 좋은 성능의 모델을 만들 수 있을것이라고 판단했습니다. Batch size를 128 -> 64 -> 32 로 줄여가며 비교를 해보았습니다.

(3) Epochs

딥러닝에서 epoch는 전체 트레이닝 셋이 신경망을 통과한 횟수 의미합니다. 예를 들어, 1-epoch는 전체 트레이닝 셋이 하나의 신경망에 적용되어 순전파와 역전파를 통해 신경망을 한 번 통과했다는 것을 의미합니다.

Epochs가 높으면 높을수록 더 성능 좋은 모델을 찾을수 있겠다고 생각을 했지만, Epochs를 늘리면 늘릴수록 모델의 학습 시간이 너무 오래 걸려서 적당히 20 -> 30 -> 100 까지만 늘려보았습니다. 어차피 Early Stopping 때문에 50이상의 숫자는 의미가 없습니다.

(4) 모델 레이어 구조

- Conv2d()를 이용하여 컨볼루션 레이어를 만들 수 있습니다. 이것은 특징을 뽑아주는 용도로 생각하면 됩니다.

kernel_size : 컨볼루션 연산을 할 크기(행,열)
filters : 필터 이미지 개수
padding : 경계처리 방법 (same은 출력사이즈가 입력사이즈와 동일하게 합니다)

- MaxPooling2D()를 이용하여 맥스풀링 레이어를 만듭니다. 이것은 이미지의 작은 부분을 무시하기 위해 사용하고 학습할 파라미터의 양도 줄일 수 있습니다.

pool_size : 맥스풀링할 크기(행,열)

- Flatten()을 이용하여 Fully Connected 레이어를 만듭니다. 이것은 다차원의 입력을 단순히 일차원으로 만들기 위해 사용했습니다.

- 그리고 기존에는 없던 Dropout() 레이어도 추가합니다. Overfitting을 방지하기 위해서 사용하였습니다.

rate: dropout할 비율(1이면 100%)

MNIST를 학습하는 모델을 컨볼루션으로 특징을 추출하고 이를 다시 일차원으로 변경시켜 딥 네트워크로 구성하여 학습하도록 진행하였습니다.

```
model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu", input_shape=(28,28,1)))
model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(512,activation="relu"))
```

model.add(Dense(10,activation="softmax"))
을 사용하여 레이어 구조를 조절해 보았습니다.

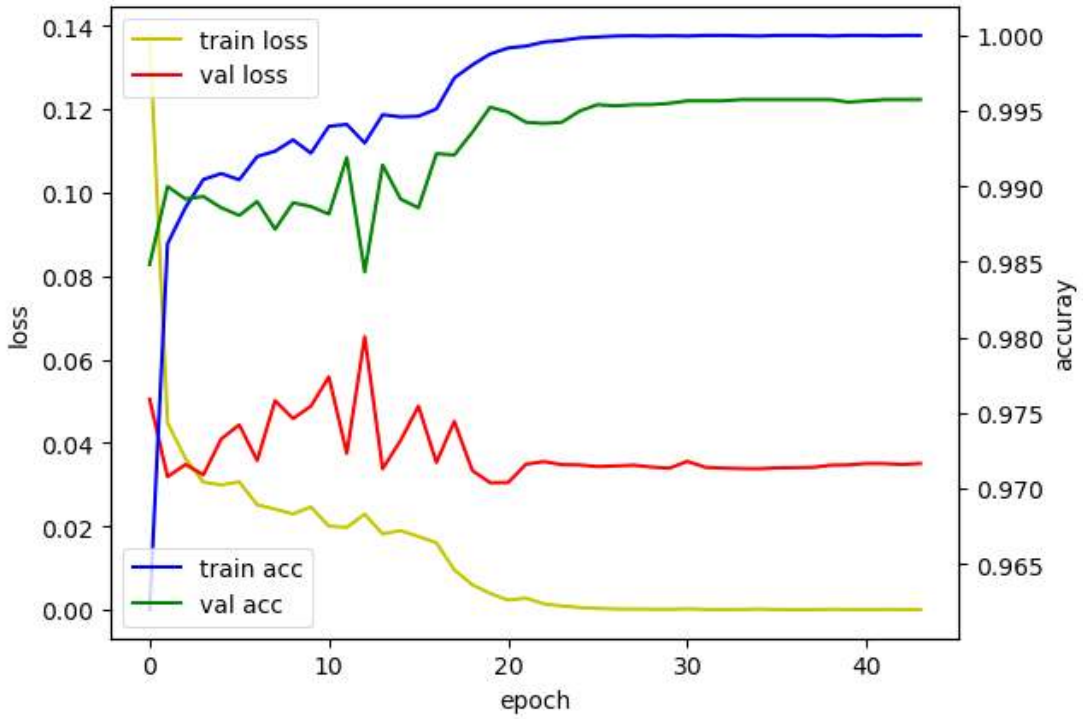
(5) ImageDataGenerator

keras에서는 이미지데이터 학습을 쉽게하도록 하기위해 다양한 패키지를 제공합니다. 그 중 하나가 ImageDataGenerator 클래스입니다.

ImageDataGenerator 클래스를 통해 객체를 생성할 때 파라미터를 전달해주는 것을 통해 데이터의 전처리를 쉽게할 수 있었고, 또 이 객체의 flow_from_directory 메소드를 활용하면 폴더 형태로된 데이터 구조를 바로 가져와서 사용할 수 있었습니다. 이 과정은 매우 직관적이고 코드도 ImageDataGenerator를 사용하지 않는 방법에 비해 상당히 짧아집니다. 환경은 keras tensorflow backend를 이용하였습니다.

학습 분석:

학습 분석을 위해서, History 확인을 해보았습니다. 먼저 History를 통해서 Epoch마다의 학습 추이를 확인할 수 있었습니다. 시각화를 해서 4개 지표의 변화 추이를 보았습니다.



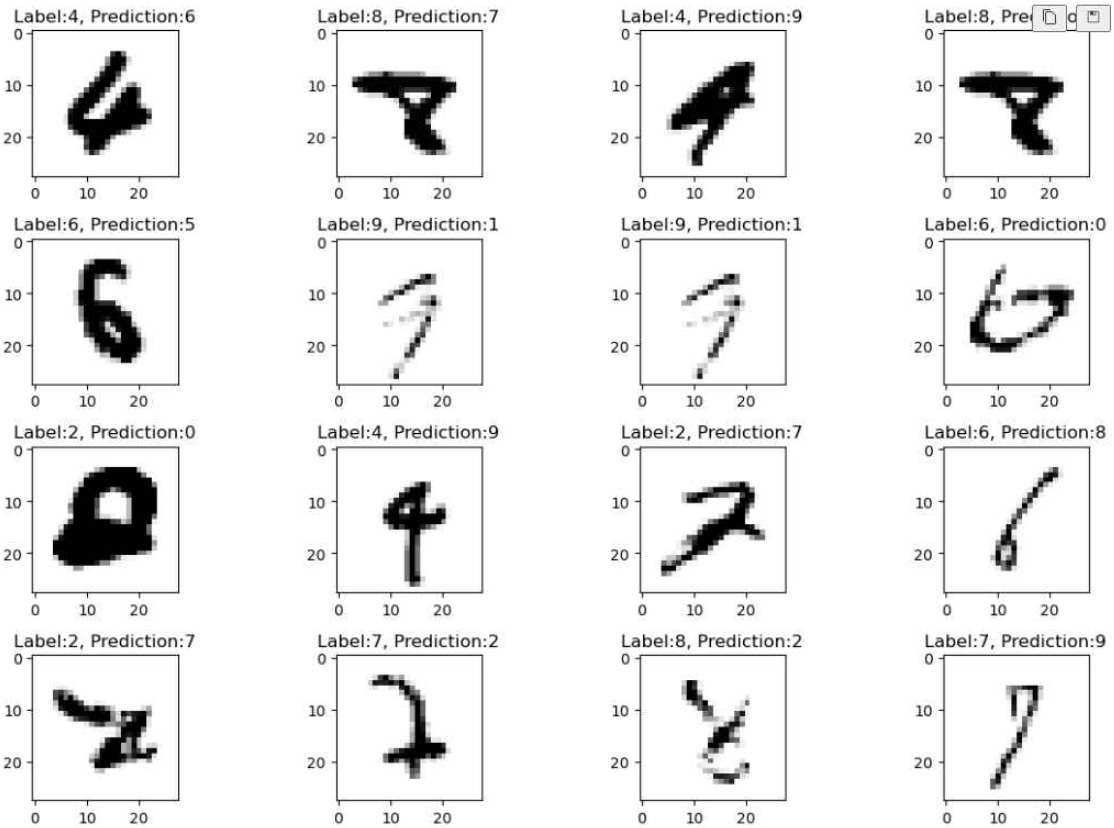
위 그래프의 결과를 해석해보면 다음과 같습니다.
1. 훈련 셋과 검증 셋의 그래프가 비슷한 형태를 보였습니다. 이는 과적합(Overfitting)이 되지 않았다는 것을 의미합니다.
2, 훈련 셋, 검증 셋의 Accuracy는 1을 향해 상승하였습니다. loss는 0을 향해 하강하는 형태를 보였습니다. 이는 안정적으로 학습을 하는 것을 의미합니다.

3. 훈련 셋, 검증 셋의 Accuracy와 loss는 epoch 20부터 수렴하기 시작했습니다. (수정)
4. 검증 셋의 loss는 epoch가 20을 넘는 순간부터 소폭 증가하는 형태를 보였습니다.(수정)
5. 검증 셋의 손실 값이 최저값을 약 epoch 20에 달성하였으므로, 모델이 완전히 수렴하였다고 할 수 있습니다.

기존에 epochs를 100으로 설정하였으나, loss와 accuracy가 수렴한 지점을 볼 때, 이는 과하게 큰 값임을 알 수 있었습니다. Epochs를 30으로 줄여서 다시 학습해보도록 하면 좋을 거 같습니다. epoch가 불필요하게 큰 경우, 리소스의 낭비가 발생하기도 하지만, 과적합이 될 위험이 있으므로, 개선사항으로 적합한 epoch에서 학습을 해주는 것이 좋을 것 같습니다. epoch가 커지면 커질수록 훈련 셋의 성능은 검증 셋의 성능보다 높게 나옵니다. 검증 셋의 손실(var_loss)의 감소가 아직 이루어지고 있는 상태라면, 모델이 아주 수렴되지 않은 상태라 할 수 있으므로, 검증 셋 손실의 감소가 더 이상 이루어지지 않을 때까진 학습을 해야 합니다.

성능 분석:

Wrong Sample과 confusion metrics를 통해서 잘못된 분류 케이스를 확인할 수 있었습니다.



```
array([[ 978,    0,    0,    0,    0,    0,    1,    1,    0,    0],
       [   0, 1135,    0,    0,    0,    0,    0,    0,    0,    0],
       [   1,    0, 1026,    0,    0,    0,    0,    5,    0,    0],
       [   0,    1,    0, 1006,    0,    3,    0,    0,    0,    0],
       [   0,    0,    1,    0,  976,    0,    1,    0,    0,    4],
       [   0,    0,    0,    4,    0,  887,    1,    0,    0,    0],
       [   6,    2,    0,    0,    1,    1,  947,    0,    1,    0],
       [   0,    3,    2,    1,    0,    0,    0, 1021,    0,    1],
       [   1,    0,    1,    0,    0,    1,    0,    1,  970,    0],
       [   0,    1,    1,    0,    5,    1,    0,    1,    0, 1000]])
```

위의 사진을 바탕으로 학습된 모델이 잘못 분류할 케이스를 분석해보았고, 모델의 Prediction 결과가 Ground Truth랑 다른 경우를 예측 확률 순위로 1위, 2위, 3위를 출력했습니다. (정답지가 4인 사진을 6으로 잘못 예측 했으며, 두 번째, 세 번째 후보는 8과 4입니다. 그리고 Wrong Sample과 confusion metrics를 통해서 모델이 잘못했는지, 데이터가 잘못되었는지를 판단할 수 있었습니다. 이를 통해 학습을 통한 개선이 가능한지 채크해보았습니다. 개선하기 위해서 더 많은 데이터가 필요하다고 생각했습니다. 학습 데이터 수가 많으면 더 좋은 성능을 가진 모델이 될 수 있기 때문입니다. 그로 인해 데이터 증강 기법을 사용하거나 MNIST에 Gan을 사용하면 더 좋은 결괏값을 얻을 수 있을 거 같습니다.