표정 인식 딥러닝 모델 개발

을 통한 딥러닝 분석

em5stion

이시명, 김규랑, 손예림, 박희진

LIST OF CONTENTS

01 목차

02 기본형

03 배치사이즈 조절

04 활성화함수

05 옵티마이저

06 레이어/퍼셉트론/드롭아웃

07 통합본

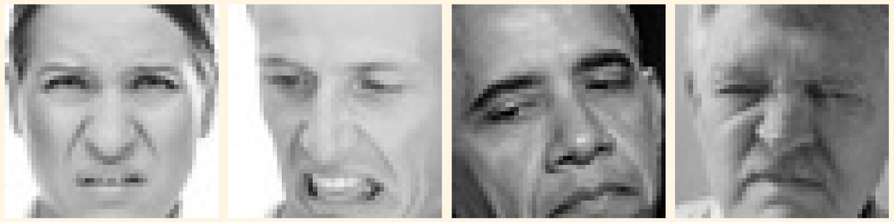
08 결론

역할분담

자료조사 / 주제 선정	김규량, 박희진, 손예림, 이시명
기본형 모델 제작	박희진
BATCH	김규량
ACTIVATION FUNCTION	박희진
OPTIMIZER	이시명
LAYER / PERCEPTRON	손예림
통합본 모델 제작	이시명
발표 자료 제작	김규량, 박희진, 손예림, 이시명

TRAIN DATA

ANGRY
3,996



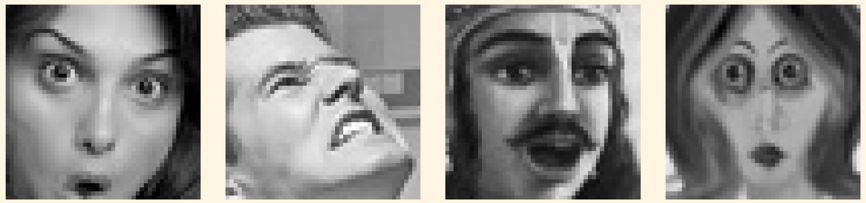
DISGUST
437

FEAR
4,098

HAPPY
7,216

NEUTRAL
4,966

SAD
4,831



SURPRISE
3,172

DATA

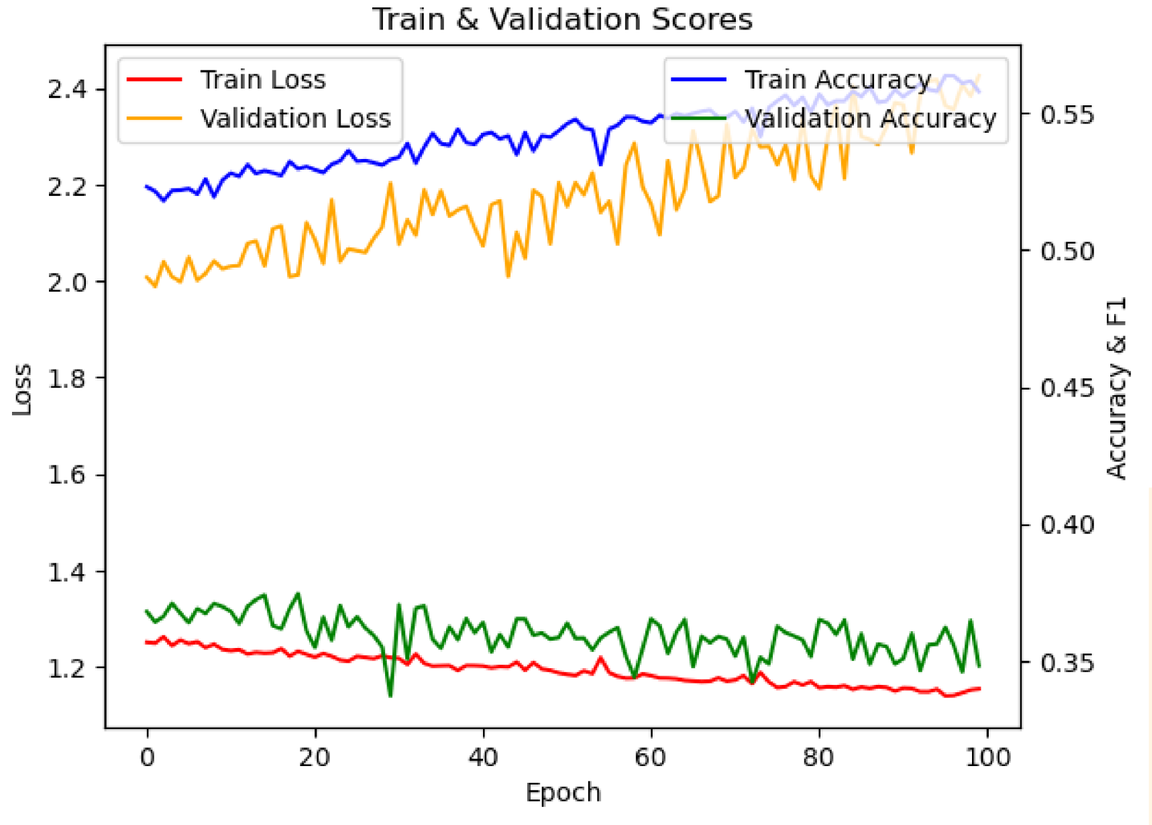
TRAIN : 28,717 => TRAIN 8 : VALID 2

TEST : 7,186

SIZE : 48*48 IMAGE

MAIN MODEL

Epoch : 100
배치 사이즈 : 32
손실함수 : CrossEntropy
활성화함수 : ReLU
옵티마이저 : Adam
스케쥴러 : patience 3
레이어 / 퍼셉트론 :
(IN-48, 128) ReLU (128, 32) ReLU (32, OUT-48)
평가지표 : Accuracy



Train Accuracy : 0.5578
Train Loss : 1.1550
Validation Accuracy :0. 3482
Validation Loss : 2.4252
Test Accuracy : 0.4507

김규량

배치사이즈와 에포크 횟수에 따른
모델 성능 차이

배치 사이즈 조절,
에포크 개수에 따른 결과

배치 사이즈: 전체 데이터를 작은
단위로 나눈 크기

에포크: 처음
부터 끝까지
학습하는 횟수

에포크 100, 배치사이즈:[16, 32, 64, 128, 256]

결과: Train-loss, Train_accuracy, Test-loss, Test_accuracy

배치사이즈: 256

```
[Train loss] ==> 1.4371222257614136    [Train Accuracy] ==> 0.44453126192092896
[Test loss] ==> 1.602426290512085    [Test Accuracy] ==> 0.3859630525112152
[6/100]
[Train loss] ==> 1.4360002279281616    [Train Accuracy] ==> 0.4448086619377136
[Test loss] ==> 1.6027051210403442    [Test Accuracy] ==> 0.3866424262523651
[7/100]
[Train loss] ==> 1.4351574182510376    [Train Accuracy] ==> 0.4450256824493408
[Test loss] ==> 1.6059582233428955    [Test Accuracy] ==> 0.3851138949394226
[8/100]
[Train loss] ==> 1.4335774183273315    [Train Accuracy] ==> 0.44631829857826233
[Test loss] ==> 1.6049386262893677    [Test Accuracy] ==> 0.38380661606788635
[9/100]
[Train loss] ==> 1.4311974048614502    [Train Accuracy] ==> 0.4472561776638031
[Test loss] ==> 1.6063905954360962    [Test Accuracy] ==> 0.3848848342895508
[10/100]
[Train loss] ==> 1.4299875497817993    [Train Accuracy] ==> 0.4476467967033386
[Test loss] ==> 1.6066800355911255    [Test Accuracy] ==> 0.3864133656024933
Early stopping at epoch 9
```

```python
# 조기 종료 기능 => 조건 : val_loss가 지정된 횟수 이상 개선이 안되면 학습 종료
if SCHEDULER.num_bad_epochs >= SCHEDULER.patience or cnt >= 5:
    print(f"Early stopping at epoch {eps}")
    break
```

Test loss: 1.606, Test Accuracy: 0.386

배치사이즈: 32

```
[8/100]
[Train loss] ==> 1.490909457206726    [Train Accuracy] ==> 0.4214542508125305
[Test loss] ==> 1.5789272785186768    [Test Accuracy] ==> 0.3782719075679779
[9/100]
[Train loss] ==> 1.4876972436904907    [Train Accuracy] ==> 0.4214397668838501
[Test loss] ==> 1.581390380859375    [Test Accuracy] ==> 0.37922006845474243
[10/100]
[Train loss] ==> 1.4852467775344849    [Train Accuracy] ==> 0.42226672172546387
[Test loss] ==> 1.586013674736023    [Test Accuracy] ==> 0.3755875825881958
[11/100]
[Train loss] ==> 1.483741044998169    [Train Accuracy] ==> 0.4227890074253082
[Test loss] ==> 1.5895851850509644    [Test Accuracy] ==> 0.3740250766277313
Early stopping at epoch 10
```

Test loss: 1.589, Test Accuracy: 0.374

에포크 2000, 배치사이즈:[16, 32, 64, 128, 256]

결과: Train-loss, Train_accuracy, Test-loss, Test_accuracy

배치사이즈: 256

```
[1/2000]
[Train loss] ==> 1.4792612791061401    [Train Accuracy] ==> 0.4275796115398407
[Test loss] ==> 1.5862482786178589    [Test Accuracy] ==> 0.38198980689048767
[2/2000]
[Train loss] ==> 1.4774409532546997    [Train Accuracy] ==> 0.426885187625885
[Test loss] ==> 1.5868510007858276    [Test Accuracy] ==> 0.38198980689048767
[3/2000]
[Train loss] ==> 1.4758890867233276    [Train Accuracy] ==> 0.42721545696258545
[Test loss] ==> 1.5870943069458008    [Test Accuracy] ==> 0.38215965032577515
[4/2000]
[Train loss] ==> 1.4746534824371338    [Train Accuracy] ==> 0.42893457412719727
[Test loss] ==> 1.5889939069747925    [Test Accuracy] ==> 0.38295742869377136
[5/2000]
[Train loss] ==> 1.474161148071289    [Train Accuracy] ==> 0.4289175570011139
[Test loss] ==> 1.5887049436569214    [Test Accuracy] ==> 0.38227027654647827
[6/2000]
[Train loss] ==> 1.47359037399292    [Train Accuracy] ==> 0.42954221367836
[Test loss] ==> 1.5903472900390625    [Test Accuracy] ==> 0.38198980689048767
Early stopping at epoch 5
```

Test loss: 1.590, Test Accuracy: 0.381

배치사이즈: 32

```
[1/2000]
[Train loss] ==> 1.5368480682373047    [Train Accuracy] ==> 0.4000116288661957
[Test loss] ==> 1.5808161497116089    [Test Accuracy] ==> 0.37835201621055603
[2/2000]
[Train loss] ==> 1.5344548225402832    [Train Accuracy] ==> 0.4012157618999481
[Test loss] ==> 1.5786603689193726    [Test Accuracy] ==> 0.3823450803756714
[3/2000]
[Train loss] ==> 1.532713770866394    [Train Accuracy] ==> 0.40112870931625366
[Test loss] ==> 1.5855921506881714    [Test Accuracy] ==> 0.37765756249427795
[4/2000]
[Train loss] ==> 1.5302766561508179    [Train Accuracy] ==> 0.40281161665916443
[Test loss] ==> 1.5836799144744873    [Test Accuracy] ==> 0.3833867311477661
[5/2000]
[Train loss] ==> 1.527946949005127    [Train Accuracy] ==> 0.4049443006515503
[Test loss] ==> 1.5838996171951294    [Test Accuracy] ==> 0.37765756249427795
[6/2000]
[Train loss] ==> 1.5235779285430908    [Train Accuracy] ==> 0.40878885984420776
[Test loss] ==> 1.5872365236282349    [Test Accuracy] ==> 0.379647433757782
[7/2000]
[Train loss] ==> 1.5209882259368896    [Train Accuracy] ==> 0.4090064764022827
[Test loss] ==> 1.5926799774169922    [Test Accuracy] ==> 0.38043534755706787
Early stopping at epoch 6
```

Test loss: 1.592, Test Accuracy: 0.380

에포크 20000, 배치사이즈:[16, 32, 64, 128, 256]

결과: Train-loss, Train_accuracy, Test-loss, Test_accuracy

배치사이즈: 256

```
[1/20000]
[Train loss] ==> 1.4954419136047363    [Train Accuracy] ==> 0.4178687334060669
[Test loss] ==> 1.5906397104263306    [Test Accuracy] ==> 0.3790355622768402
[2/20000]
[Train loss] ==> 1.4918726682662964    [Train Accuracy] ==> 0.4206635057926178
[Test loss] ==> 1.5939199924468994    [Test Accuracy] ==> 0.3793160319328308
[3/20000]
[Train loss] ==> 1.4909437894821167    [Train Accuracy] ==> 0.4204898774623871
[Test loss] ==> 1.5931991338729858    [Test Accuracy] ==> 0.380113810300827
[4/20000]
[Train loss] ==> 1.490212321281433    [Train Accuracy] ==> 0.4205936789512634
[Test loss] ==> 1.59394419193267 82    [Test Accuracy] ==> 0.37801656126976013
[5/20000]
[Train loss] ==> 1.4895333051681519    [Train Accuracy] ==> 0.42078426480293274
[Test loss] ==> 1.5920915603637695    [Test Accuracy] ==> 0.3808445632457733
[6/20000]
[Train loss] ==> 1.4887639284133911    [Train Accuracy] ==> 0.42116546630859375
[Test loss] ==> 1.594159841537 4756    [Test Accuracy] ==> 0.37897637486457825
Early stopping at epoch 5
```

Test loss: 1.594, Test Accuracy: 0.378

배치사이즈: 32

```
[1/20000]
[Train loss] ==> 1.5364391803741455    [Train Accuracy] ==> 0.39880746603012085
[Test loss] ==> 1.5724475383758545    [Test Accuracy] ==> 0.3854706249427795
[2/20000]
[Train loss] ==> 1.5350133180618286    [Train Accuracy] ==> 0.3987204134464264
[Test loss] ==> 1.5737500190734863    [Test Accuracy] ==> 0.3840811848640442
[3/20000]
[Train loss] ==> 1.534448266029358    [Train Accuracy] ==> 0.3974582254886627
[Test loss] ==> 1.5800448656082153    [Test Accuracy] ==> 0.38311967253685
[4/20000]
[Train loss] ==> 1.5338586568832397    [Train Accuracy] ==> 0.3987058997154236
[Test loss] ==> 1.577204704284668    [Test Accuracy] ==> 0.3848557770252228
[5/20000]
[Train loss] ==> 1.5312492847442627    [Train Accuracy] ==> 0.40212973952293396
[Test loss] ==> 1.57501220703125    [Test Accuracy] ==> 0.3839877247810364
[6/20000]
[Train loss] ==> 1.5302224159240723    [Train Accuracy] ==> 0.39848828315734863
[Test loss] ==> 1.5743855237960815    [Test Accuracy] ==> 0.3860710561275482
Early stopping at epoch 5
```

Test loss: 1.574, Test Accuracy: 0.386

배치사이즈별 정확도(에포크=100)

batch size	16	32	64	128	256
TRAIN ACC	0.4010	0.4227	0.4324	0.4420	0.4476
TEST ACC	0.3825	0.3740	0.3762	0.3851	0.3864

배치 사이즈 조절은 무조건 높다고 좋은 건 아님 => 어떤 모델을 선정할지 그 모델에 배치사이즈 설정은 본인 하기 나름

박희진

활성화 함수에 따른 모델 성능 차이

SIGMIOD FUNTION
SOFTMAX FUNTION
RELU FUNTION
LEAKY_RELU FUNTION
TANH FUNTION

모델 클래스 정의

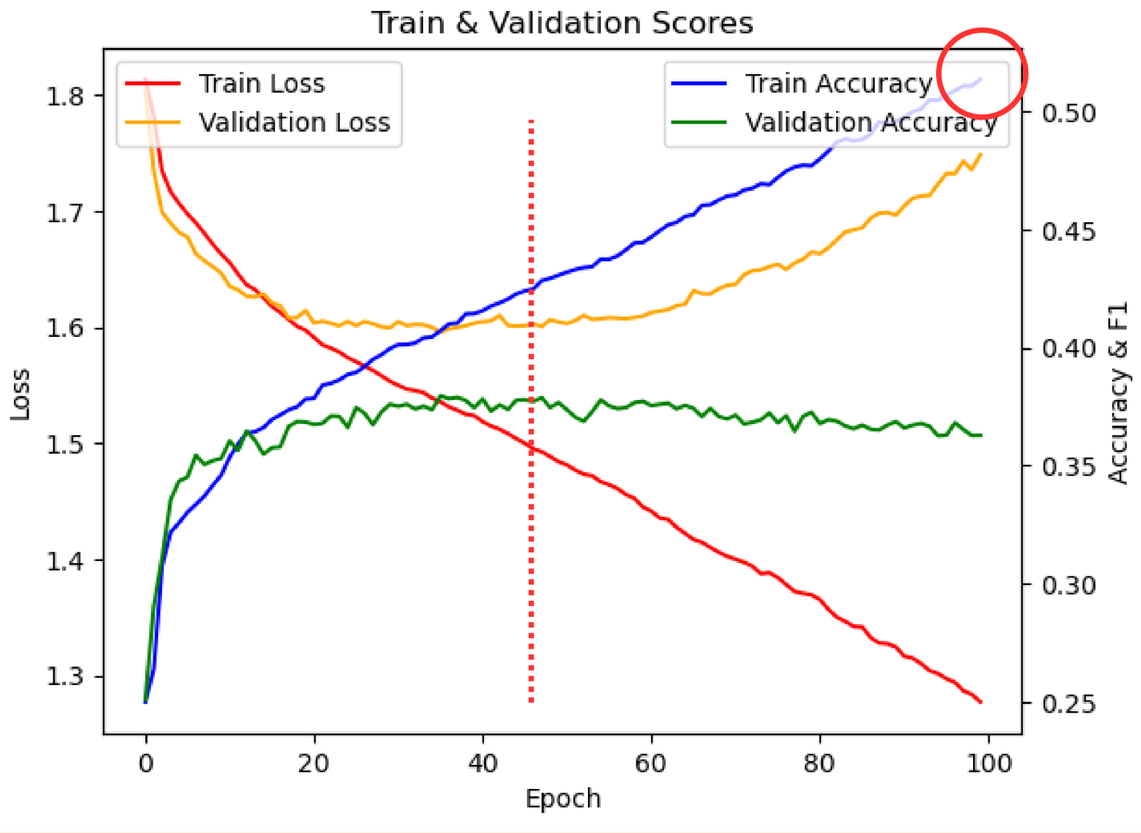
```python
# 모델 클래스 정의
class Model(nn.Module):

    def __init__(self, IN, OUT, AF):
        super().__init__()
        self.input = nn.Linear(IN, 128)
        self.af = AF()
        self.hidden = nn.Linear(128, 32)
        self.output = nn.Linear(32, OUT)

    def forward(self, x):
        y = self.input(x)
        y = self.af(y)
        y = self.hidden(y)
        y = self.af(y)
        y = self.output(y)

        return y
```

```python
# 모델 생성
sig_model = Model(IN, OUT, nn.Sigmoid).to(DEVICE)
soft_model = Model(IN, OUT, nn.Identity).to(DEVICE)
relu_model = Model(IN, OUT, nn.ReLU).to(DEVICE)
leaky_model = Model(IN, OUT, nn.LeakyReLU).to(DEVICE)
tanh_model = Model(IN, OUT, nn.Tanh).to(DEVICE)
```

SIGMOID FUNTION

Train Accuracy : 0.5136
Train Loss : 1.2271

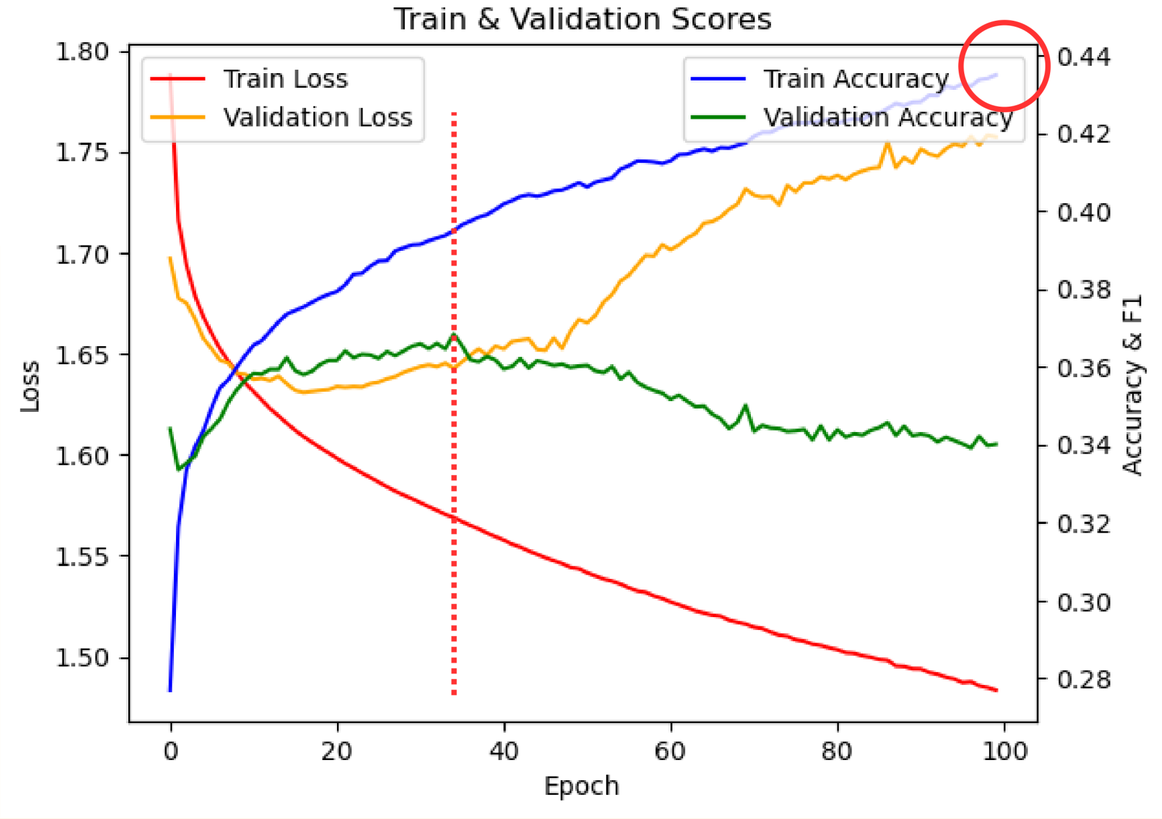
Validation Accuracy : 0.3628
Validation Loss : 1.7483



SOFTMAX FUNTION

Train Accuracy : 0.4349
Train Loss : 1.4834

Validation Accuracy : 0.3401
Validation Loss : 1.7573

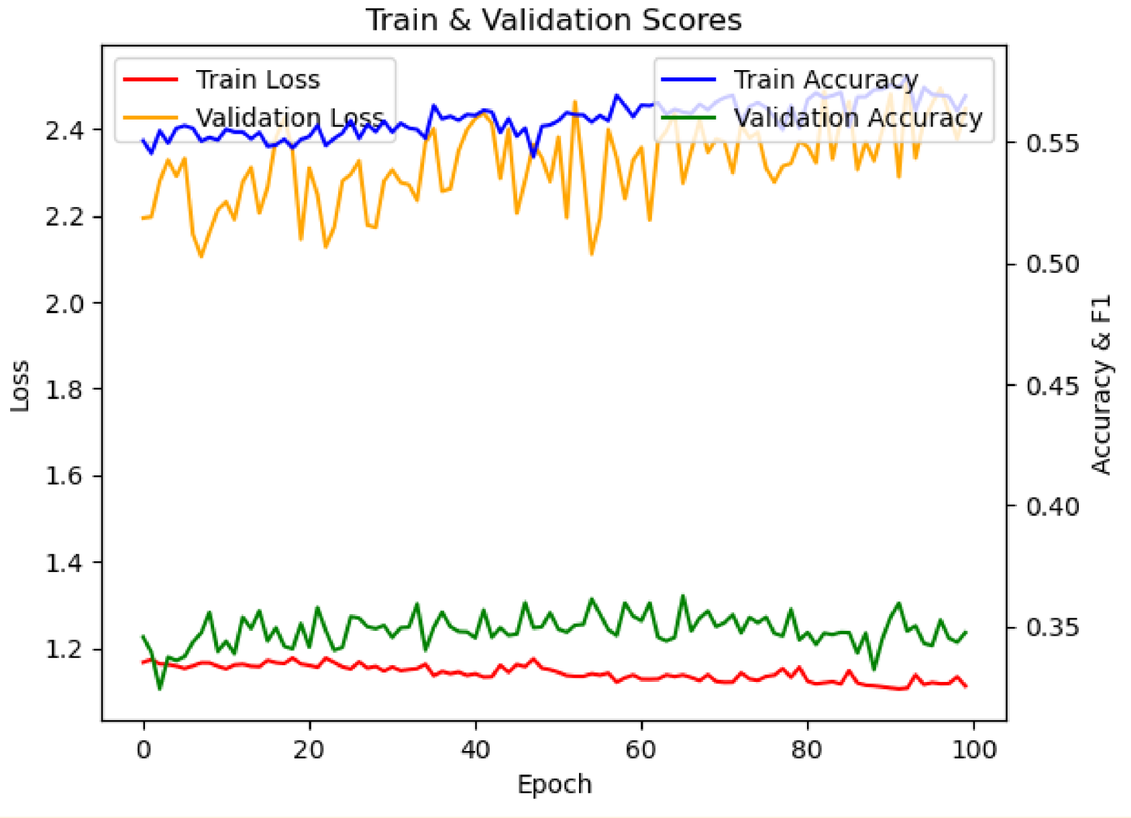


RELU FUNTION

Train Accuracy : 0.5687
Train Loss : 1.1123

Validation Accuracy : 0.3475
Validation Loss : 2.4468

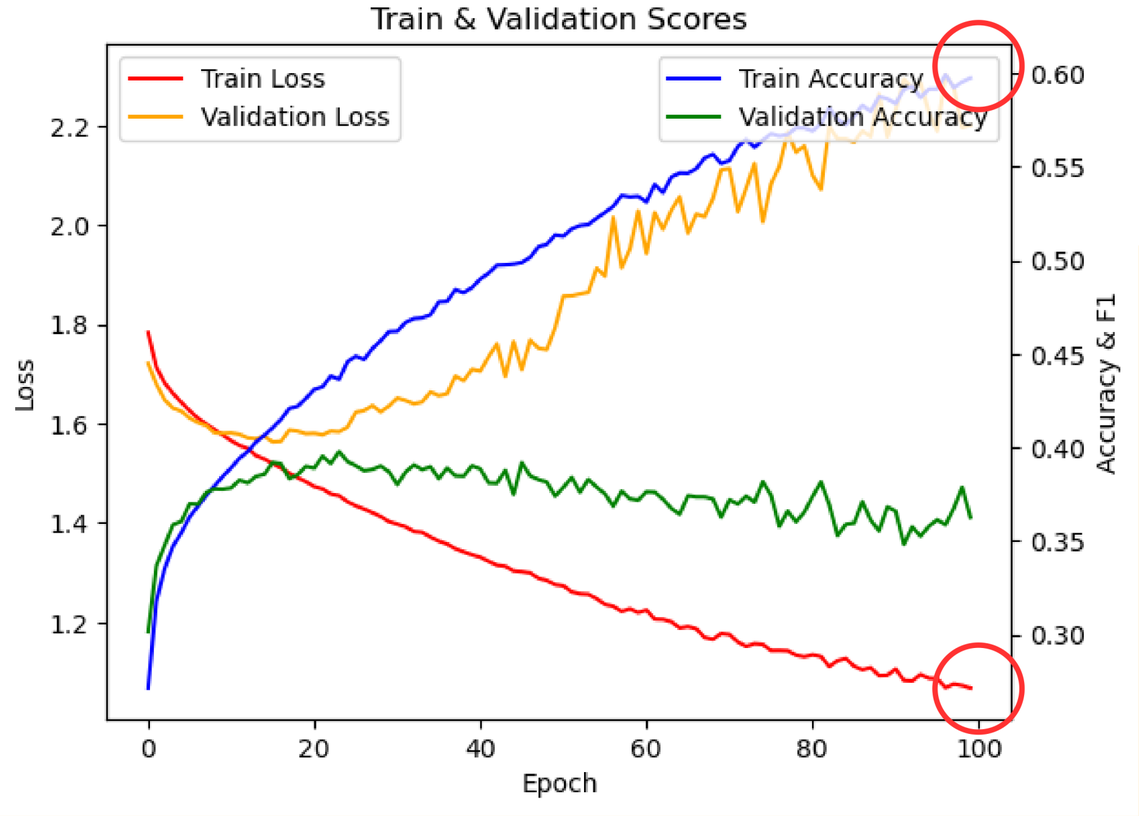
모델 구조의 문제



LEAKY_RELU FUNTION

Train Accuracy : 0.5973
Train Loss : 1.0674

Validation Accuracy : 0.3627
Validation Loss : 2.2004

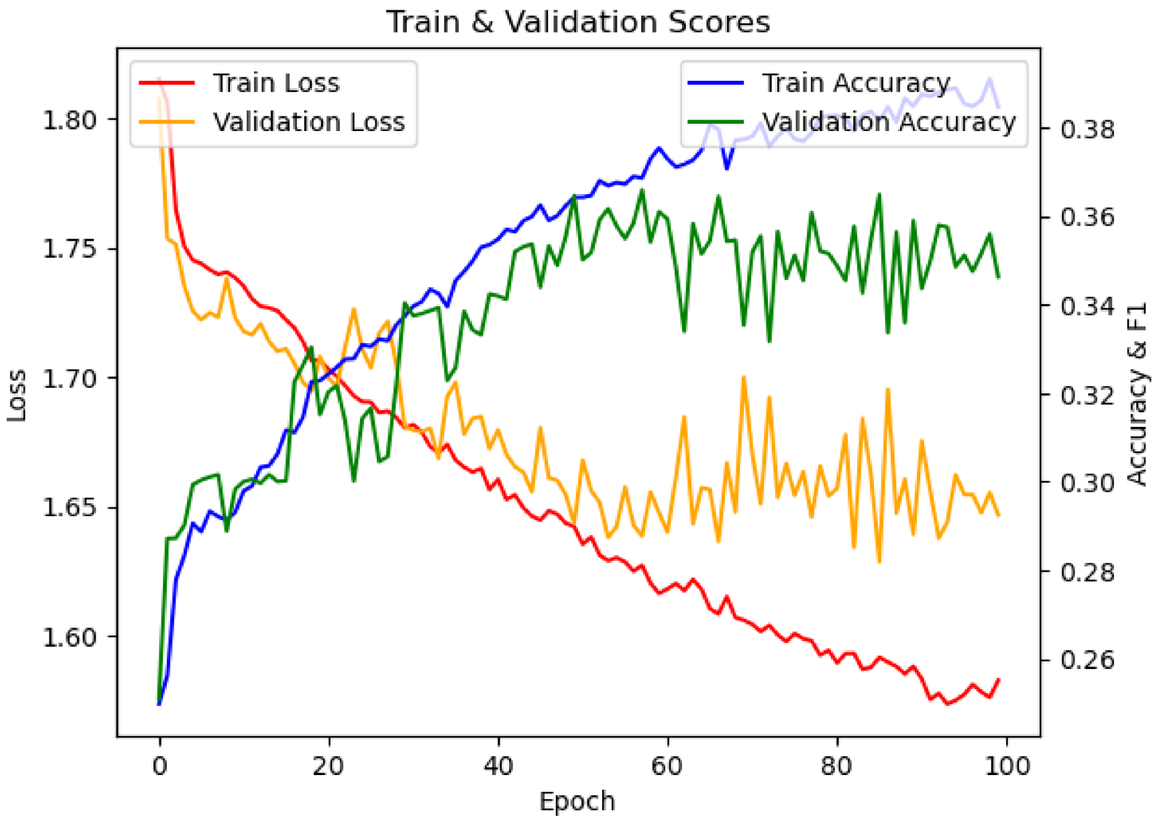


TANH FUNTION

‘해당 모델 구조에서는 !‘
Epoch를 늘인다면 가장 성능이 좋을 것으로 예상됨

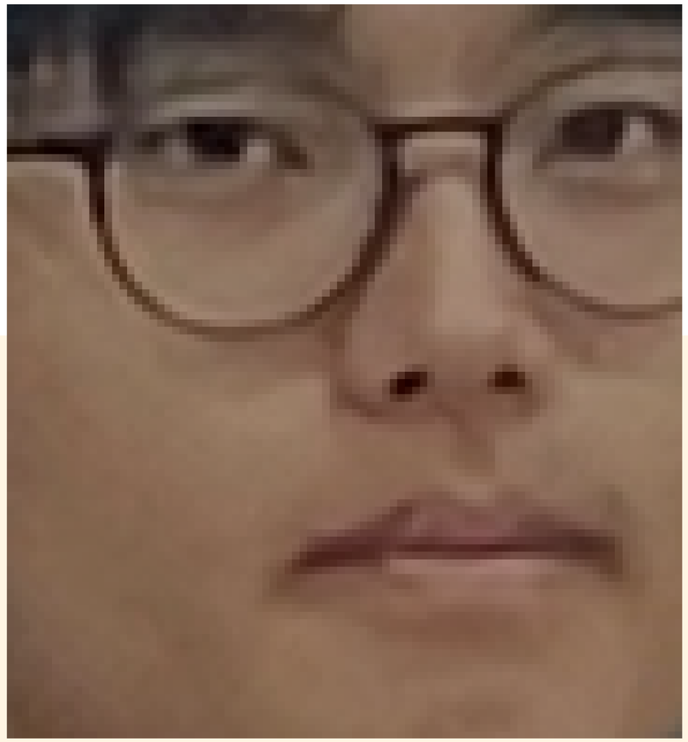
Train Accuracy : 0.3847
Train Loss : 1.5828

Validation Accuracy : 0.3463
Validation Loss : 1.6467

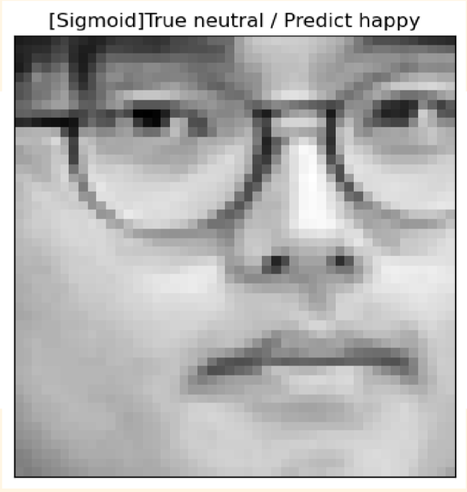


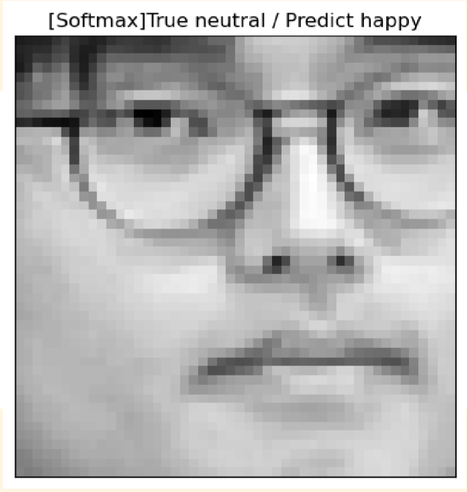
예측

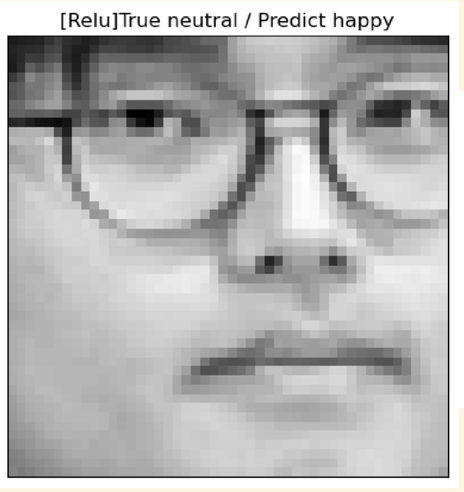
멤모마콘



예측 ⋯⋯→ HAPPY


[Sigmoid]True neutral / Predict happy


[Softmax]True neutral / Predict happy


[Relu]True neutral / Predict happy


[Leaky]True neutral / Predict happy


[Tanh]True neutral / Predict happy

예측

[2] True sad / Predict neutral

[6] True neutral / Predict neutral

[8] True surprise / Predict neutral

[9] True fear / Predict fear

[3] True happy / Predict neutral

[5] True angry / Predict surprise

테스트 결과 및 결론

	Sigmoid	Softmax	Relu	Leaky_Relu	Tanh
Train Accuracy	0.5136	0.4349	0.5687	0.5873	0.3847
Train Loss	1.2271	1.4834	1.1123	1.0674	1.5828
Valid Accuracy	0.3628	0.3401	0.3475	0.3627	0.3463
Valid Loss	1.7483	1.7573	2.4468	2.2004	1.6467
Test Acc	0.4006	0.3721	0.4468	0.4124	0.3882

Relu의 기울기 소실 문제를 보완한 **Leaky_Relu**가 검증에서 성능이 더 좋게 나왔지만 테스트를 진행해보니 **Relu**가 더 좋았다.
그러나 '내 모델 구조에서는' **Epoch**를 더 늘인다면 **Tanh**이 성능이 가장 좋을 것으로 예상된다.

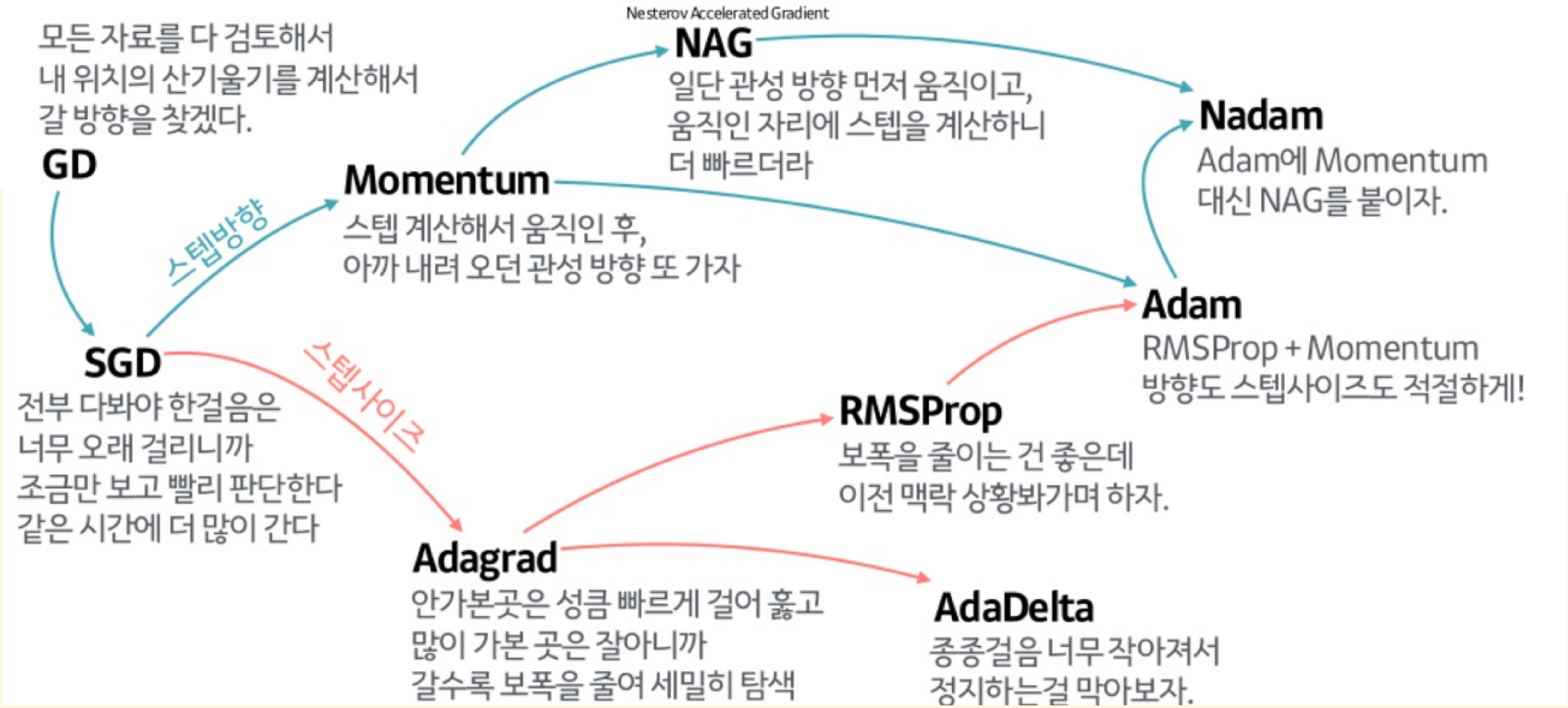
이시명

옵티마이저에 따른 모델 성능 차이

SGD
Adagrad
RMSProp
AdaDelta
Adam
NADAM

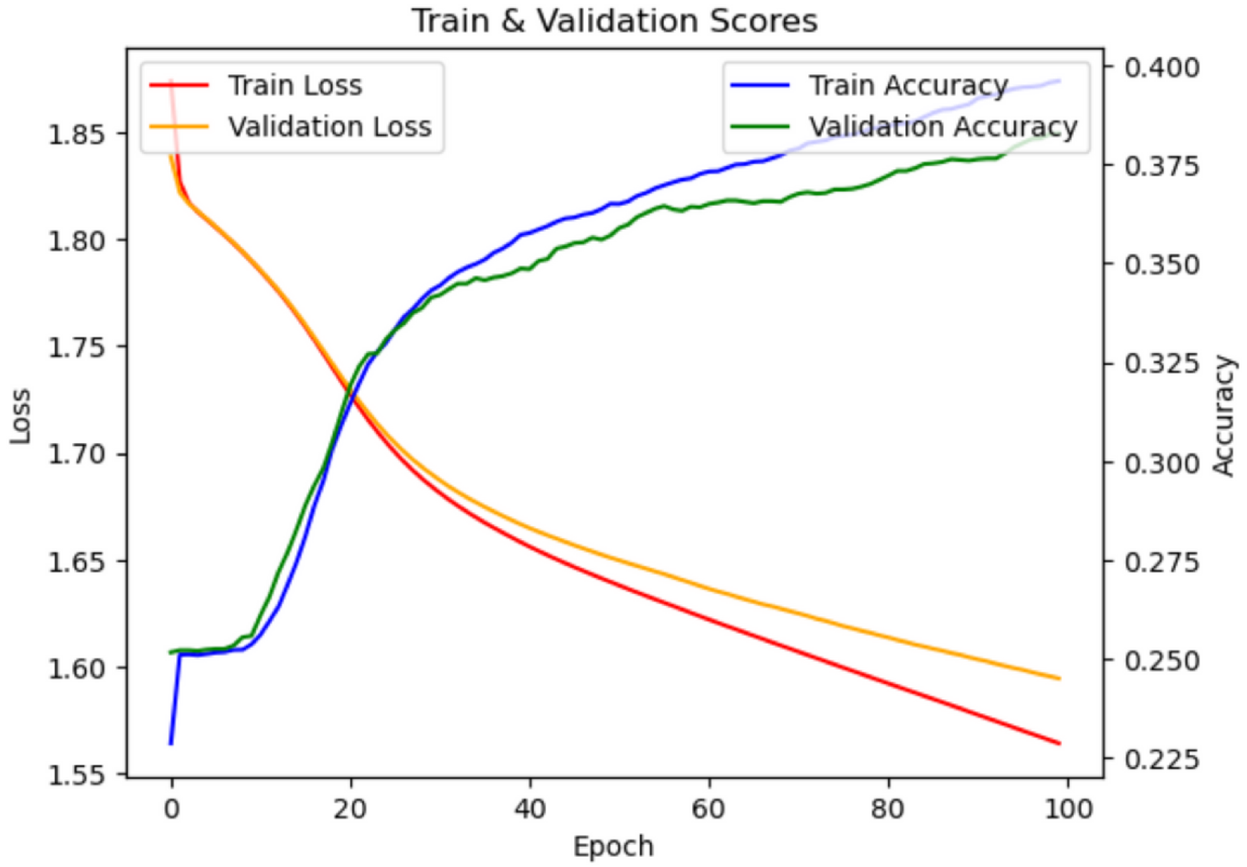
OPTIMIZER

산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



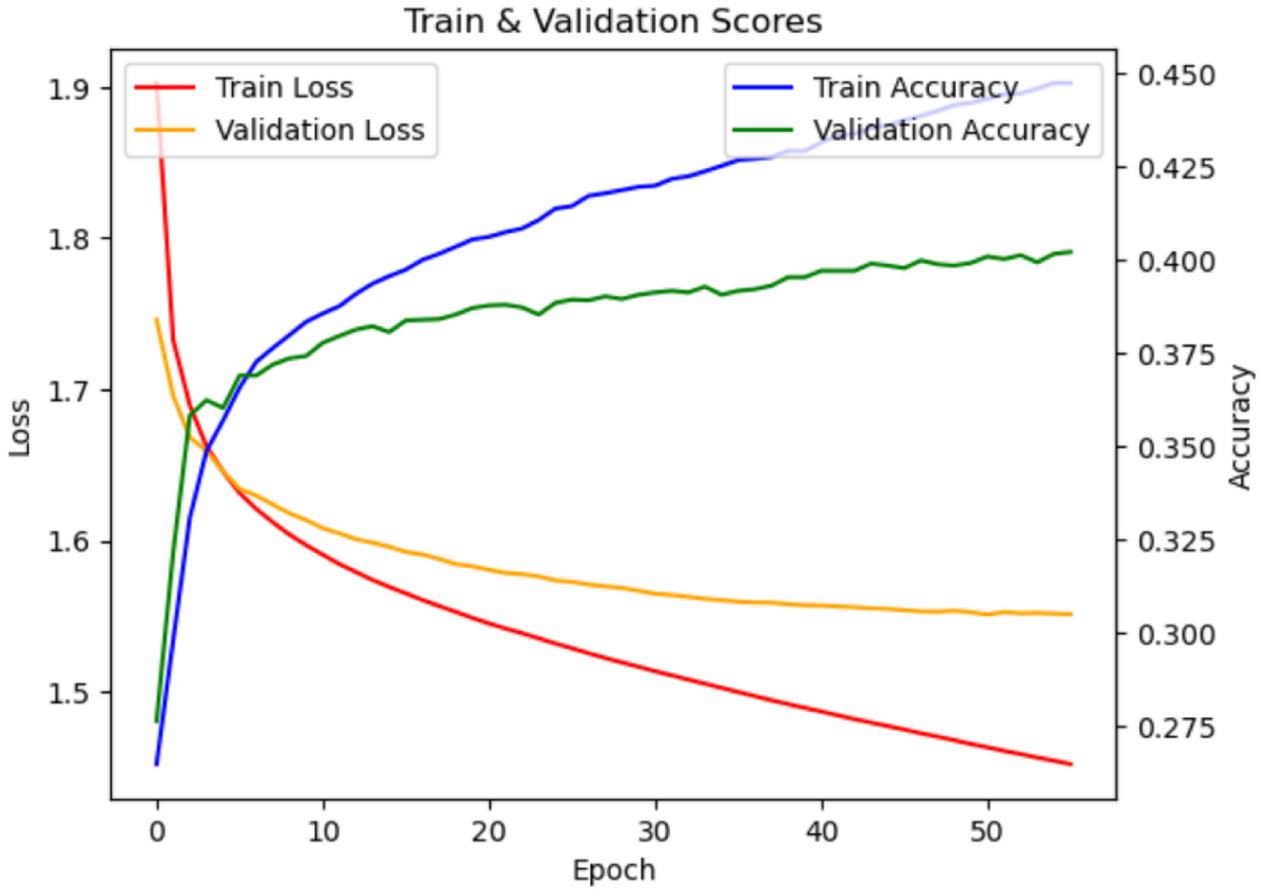
OPTIMIZER - SGD

EPOCH	100/100
Train Accuracy	1.5543
Train Loss	0.4023
Valid Accuracy	1.5875
Valid Loss	0.3829
Test Accuracy Total : 7178	0.25 Correct : 1784



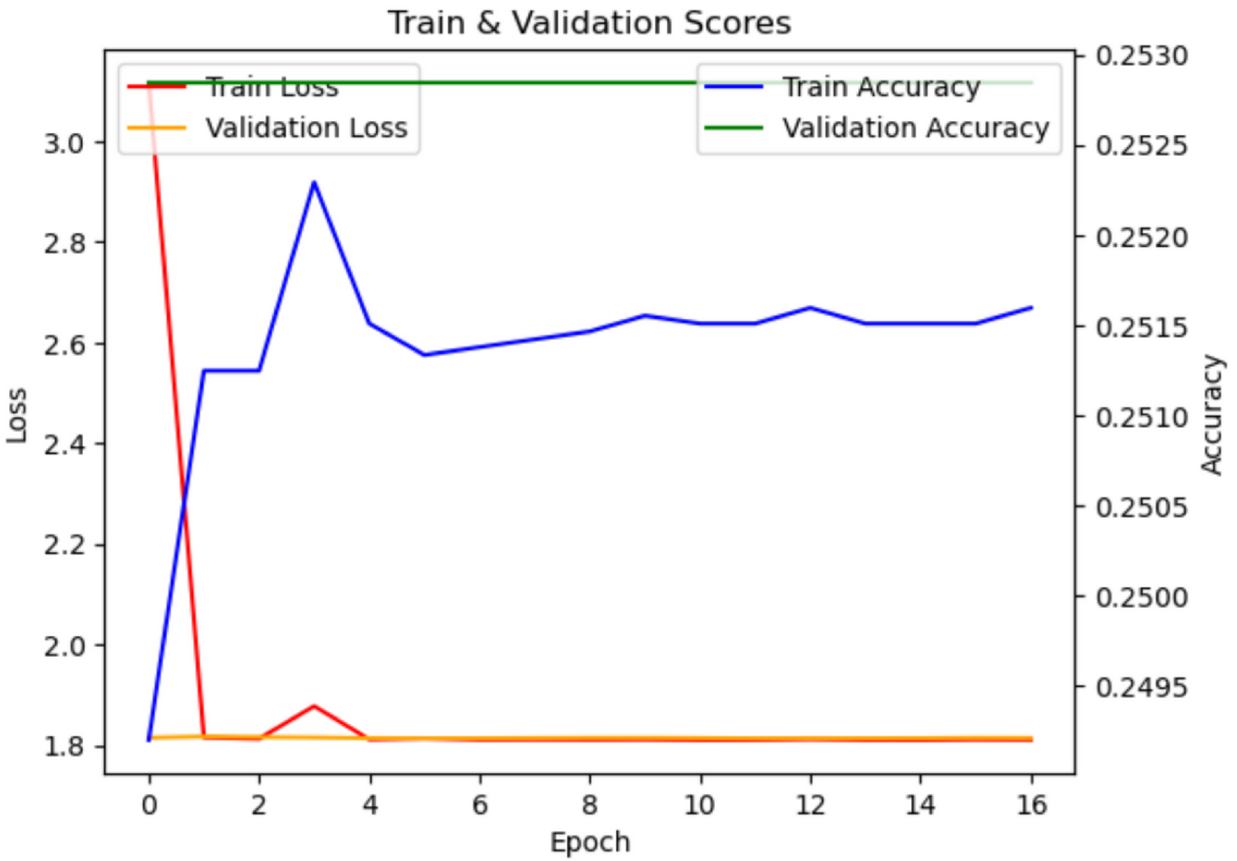
OPTIMIZER - ADAGRAD

EPOCH	5/100
Train Accuracy	1.4356
Train Loss	0.4523
Valid Accuracy	1.5502
Valid Loss	0.4034
Test Accuracy Total : 7178	0.25 Correct : 1805



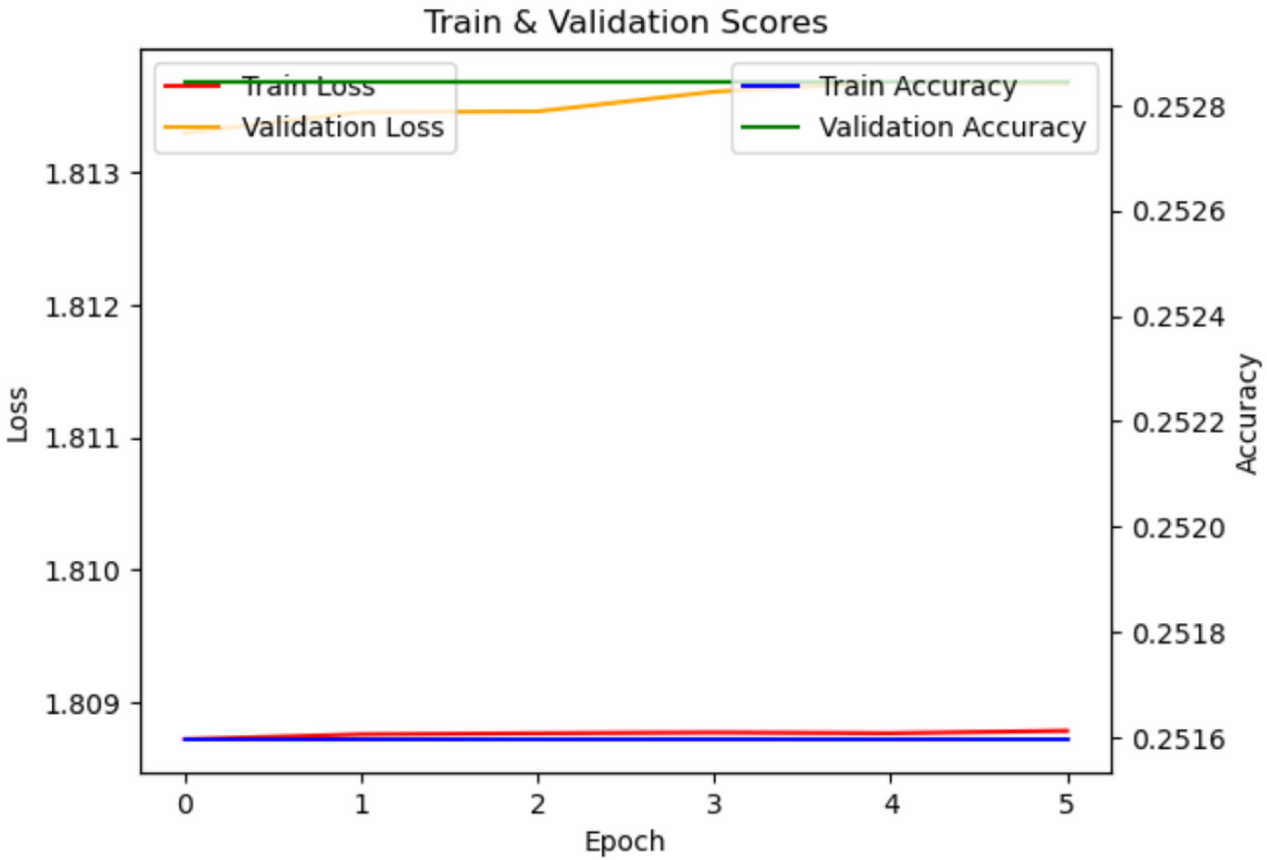
OPTIMIZER - RMSPROP

EPOCH	5/100
Train Accuracy	1.8107
Train Loss	0.2511
Valid Accuracy	1.8139
Valid Loss	0.2528
Test Accuracy Total : 7178	0.25 Correct : 1774



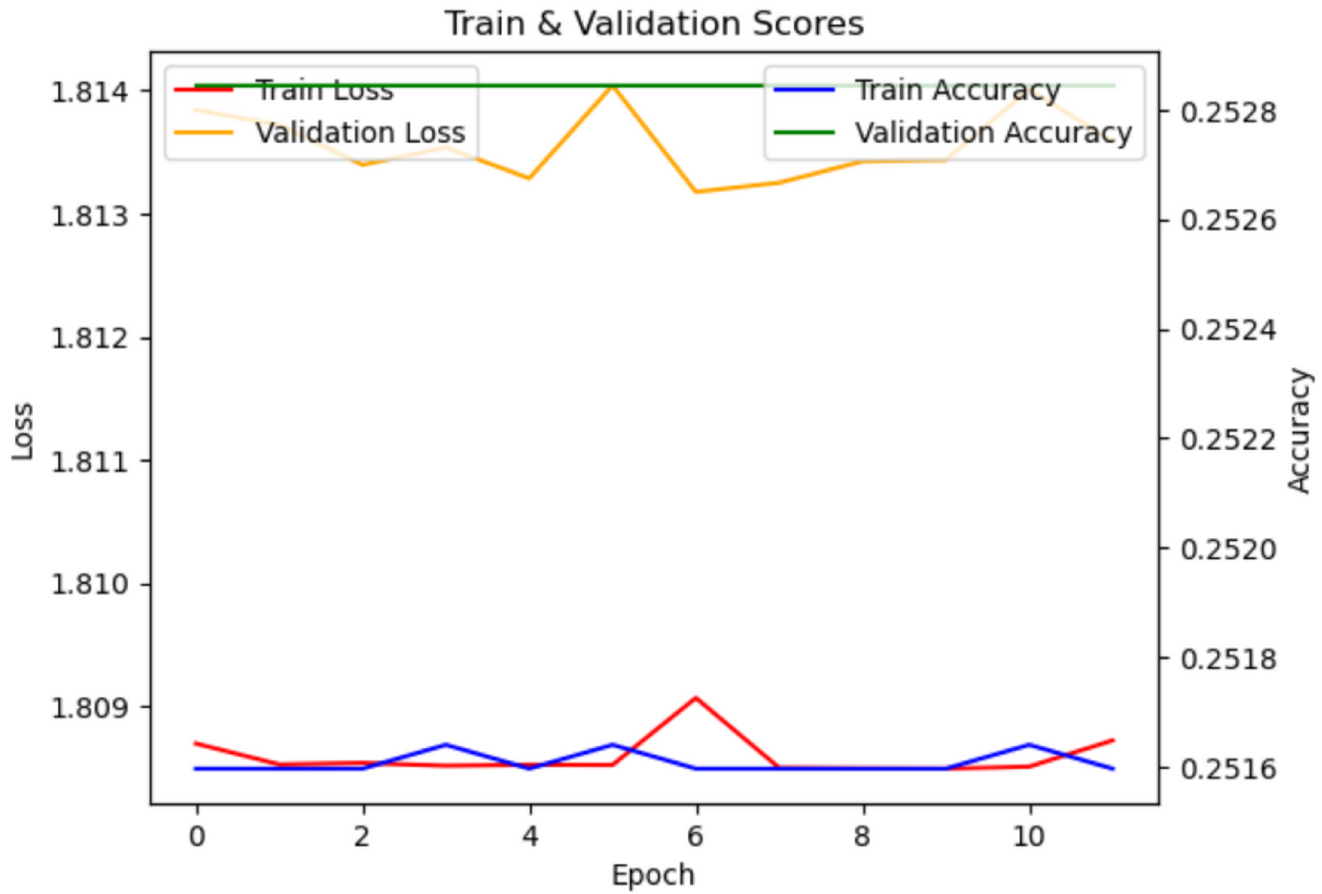
OPTIMIZER - ADADELTA

EPOCH	8/100
Train Accuracy	1.8093
Train Loss	0.2512
Valid Accuracy	1.8107
Valid Loss	0.2528
Test Accuracy Total : 7178	0.25 Correct : 1774



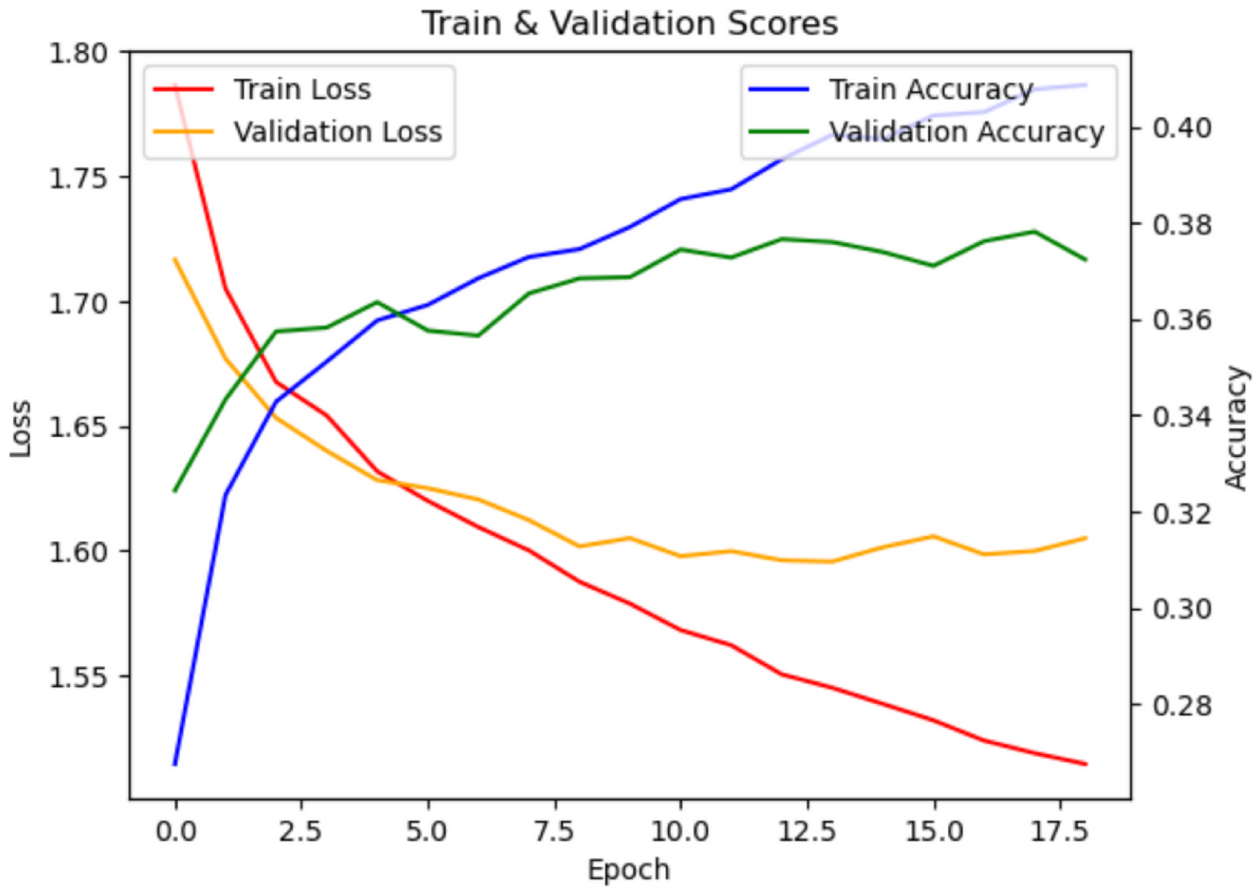
OPTIMIZER - ADAM

EPOCH	11/100
Train Accuracy	1.8087
Train Loss	0.2515
Valid Accuracy	1.8135
Valid Loss	0.2528
Test Accuracy Total : 7178	0.25 Correct : 1774



OPTIMIZER - NADAM

EPOCH	5/100
Train Accuracy	1.5145
Train Loss	0.4086
Valid Accuracy	1.6050
Valid Loss	0.3723
Test Accuracy Total : 7178	0.25 Correct : 1788



손예림

히든레이어 수에 따른 모델 성능 차이

MODEL01
[HIDDEN LAYER=[512,256]]

Train Loss	1.3190
Train Accuracy	0.5071
Test Accuracy	0.1774

MODEL02 [HIDDEN LAYER=[512,256,128]]

Train Loss	1.3586
Train Accuracy	0.4860
Test Accuracy	0.2468

MODEL03 [HIDDEN LAYER=[256,128]]

Train Loss	1.2847
Train Accuracy	0.4991
Test Accuracy	0.2400

MODEL04
[HIDDEN LAYER=[64,32,16]]

Train Loss	1.3132
Train Accuracy	0.4868
Test Accuracy	0.2120

MODEL05 [HIDDEN LAYER= [884,886,888]]

Train Loss	1.6075
Train Accuracy	0.4683
Test Accuracy	0.1725

MODEL06 [HIDDEN LAYER= [500,550,600,650,700]]

Train Loss	1.6648
Train Accuracy	0.4403
Test Accuracy	0.1475

문제점
:TEST ACCURACY가 다 낮게 나옴

MODEL	TEST ACCURACY
1	0.1774
2	0.2468
3	0.2400
4	0.2120
5	0.1725
6	0.1475

이유 예상

테스트 정확도(Test Accuracy)가 낮게 나오는 주요 원인

출처-Claude

1. 과적합(Overfitting)

- 모델이 훈련 데이터에 지나치게 최적화되어 일반화 성능이 떨어지는 경우입니다.
- 이는 모델이 훈련 데이터의 노이즈나 특이점까지 학습하여 새로운 데이터에 대한 예측 능력이 낮아지기 때문입니다.

2. 데이터 불균형(Data Imbalance)

- 데이터 클래스 간 분포가 불균형할 때 발생할 수 있습니다.
- 예를 들어 이진 분류 문제에서 클래스 0이 90%, 클래스 1이 10%라면 모델이 0을 많이 예측하게 됩니다.

3. 부적절한 모델 구조

- 모델의 구조(층의 수, 노드 수 등)가 문제의 복잡도에 비해 너무 단순하거나 복잡할 경우 발생할 수 있습니다.

4. 부적절한 하이퍼파라미터

- 학습률, 정규화 계수, 드롭아웃 비율 등의 하이퍼파라미터 설정이 적절하지 않으면 성능이 저하될 수 있습니다.

5. 데이터 전처리 미흡

- 데이터의 스케일링, 정규화, 인코딩 등이 제대로 이루어지지 않으면 모델 학습에 어려움이 있을 수 있습니다.

6. 훈련 데이터 부족

- 훈련 데이터의 양이 부족하면 모델이 제대로 일반화되기 어렵습니다.

7. 데이터 분포 차이

- 훈련 데이터와 테스트 데이터의 분포가 다른 경우, 예측 성능이 저하될 수 있습니다.

통합모델

MACHINE LEARNING

BATCH

ACTIVATION FUNCTION

OPTIMIZER

LAYER / PERCEPTRON
/ DROPOUT

MACHINE LEARNING

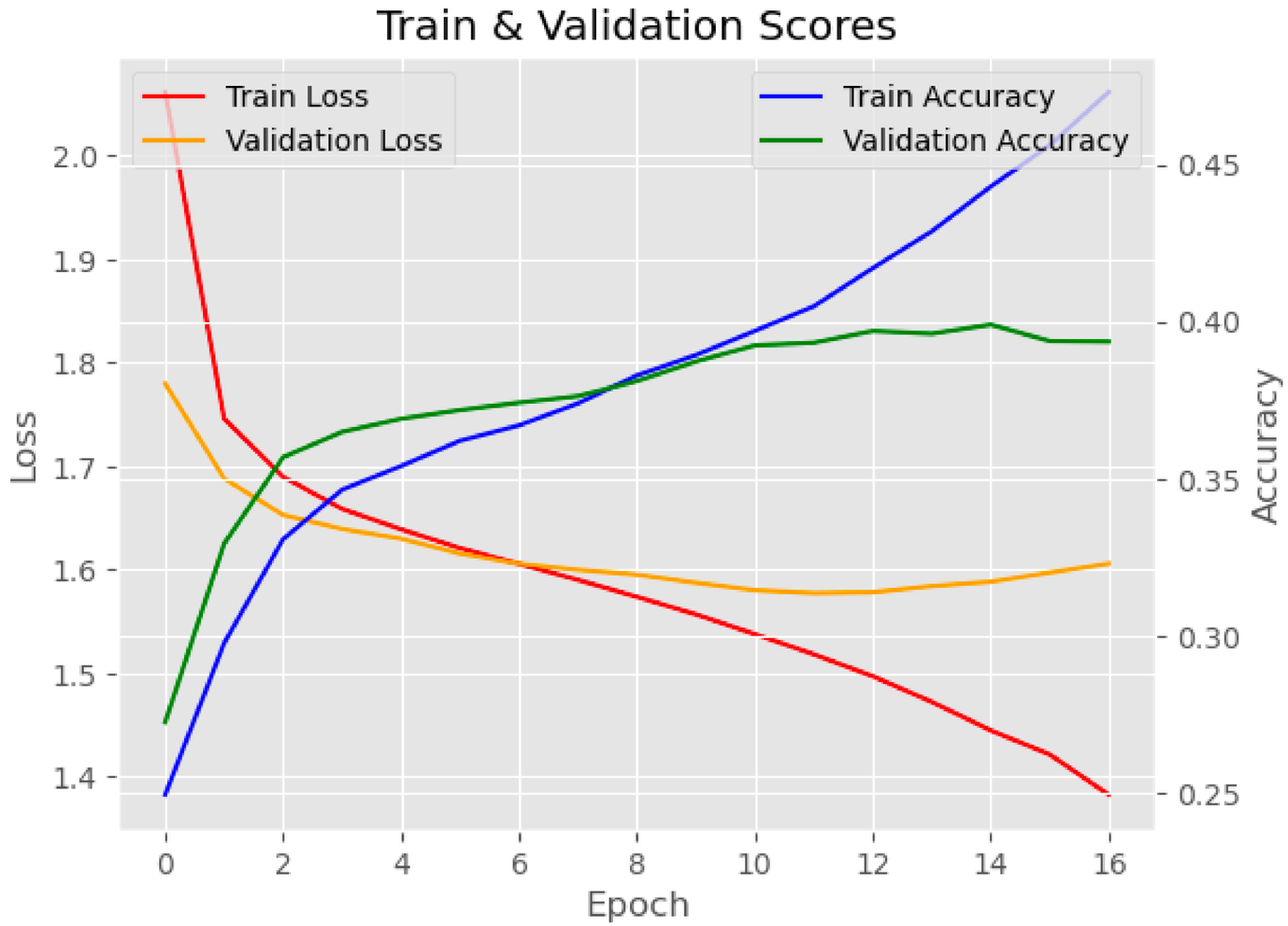
```
1  from sklearn.svm import SVC
2  from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
3
4  model = SVC(kernel='linear')
5  model.fit(scaled_X_train, target_data)
6  train_score = model.score(scaled_X_train, target_data)
7  test_score = model.score(scaled_x_data, target_test)
8
9  print(f'Train score : {train_score}, Test score : {test_score}')
10
11 y_pred = model.predict(scaled_x_test)
12 accuracy = accuracy_score(target_test, y_pred)
13
14 print(f'{accuracy}')
```

307m 11.6s

train_score 5시간 경과 후 에러
대량의 데이터에 적합하지 않음

FINAL MODEL

```
--- Training with Adagrad optimizer ---
[18/100] Adagrad count : 5
Train_loss : 1.4073805809020996, Train_accuracy : 0.4600743055343628
Val_loss : 1.6026445627212524, Val_accuracy : 0.3884214758872986
Early stopping at epoch 17
[18/100] Adagrad count : 5
Train_loss : 1.4073805809020996, Train_accuracy : 0.4600743055343628
Val_loss : 1.6026445627212524, Val_accuracy : 0.3884214758872986
```



FINAL MODEL

	MAIN	Batch	Activation Function	OPTIMIZER	Layer	FINAL
		32	ReLU	Adagrad	hidden_layaer =[512,256]	ALL
TRAIN ACC	0.5578	0.4316	0.5687	0.4523	0.5071	0.4600
TEST ACC	0. 3482	0.3969	0.4468	0.4034	0.1774	0.3888

THANKS FOR WATCHING