**휴먼지능정보공학과 201710758 김진성**

**심층학습 중간 과제**

**최종 정확도 : 79.88%**

**try 1**

data augmentation : Original

activation function : Leakey Relu, 다잉 렐루를 예방하기 위해 변경

drop\_out : 각 layer마다 0.1씩 모두 추가

BatchNorm : 각 layer마다 모두 추가

fully-connected layer : 64\*8\*8 -> 100 -> 10

optimizer : SGD

scheduler : StepLR(step\_size = 10, gamma = 0.5)

batch\_size : 64, 빠르게 학습하기 위해 64로 설정

epoch : 100

learning rate : 0.1

정확도 : 61

**코멘트 : compute acc를 train, test 둘다 10 epoch마다 찍어보았는데 train\_acc은 4~50에폭에 벌써 99%를 찍었고, test\_acc또한 60에서 맴도는 over fitting현상 발견**

(변화를 준 부분만 기입 했습니다.)

**try 2**

activation function : Leakey Relu -> ELU,

threshold를 -1로 낮춘다음 e^x를 이용하여 근사화하는 기법

batch\_size : 64 -> 32

learning rate : 0.1 -> 0.01

정확도 : 61 -> 62

**코멘트 : batch\_size와 learining\_rate조절로 오버피팅이 완화돼, acc가 향상된 것 같다.**

**try 3**

activation function : ELU -> Mish

, pytroch에서 제공해주지 않는 최신 기법 (https://github.com/digantamisra98/Mish)

무한에 의한 캡핑으로 인한 포화 회피, 약간의 음수를 허용해 gradient가 더 잘 흐름, 이러한 이유로 정확도 향상에 도움을 준다 해서 사용해 보았다.

정확도 : 62 -> 62

**코멘트 : try2에서 코멘트로 예측한 내용이 맞았다. activation function은 acc향상에 큰 영향을 주지 않는 것 같다. 솔직히 계산이 빨라진 것도 잘 못 느꼈다.**

**try 4**

data augmentation : Normalize(mean, std) + train데이터에 변형을 가해줌(random\_horiziontalflip, random\_rotation, random\_affine, color\_jitter(brightness=0.2, contrast=0.2, saturation=0.2))  
(무작위 수평 뒤집기, 무작위 회전, 램덤 ,컬러 특성 변화, 포지션 변화)

정확도 : 62 -> 67

**코멘트 : data augmentation이 정말 중요하단 것을 알게 됨.**

**try 5**

fully-connected layer : (64\*8\*8 , 100, 10) -> (64\*8\*8, 1024, 512, 256, 128, 64, 32, 10)

정확도 : 67 -> 67

**코멘트 : 각 fc layer에도 activation\_fuction을 추가해줬지만 향상하는 기척이 안보였다. 이유를 찾아보니 위에서 썩은 물이 내려오는데 아래에서 아무리 거른다 한들, 발전은 미미할 것 이라고 했다.**

**try 6**

optimizer : SGD -> Adagrad(weight\_decay == 1e-2)

SGD능력 + 학습해서 큰 변동이 있던 부분 가중치는 학습률을 감소 시키고, 아직 별 변동이 없던 가중치는 학습률을 증가시키는 기법

+ 가중치를 감소해서 가중치 정형화를 시키면 over fitting을 막을 수 있다해서,

weight\_decay를 사용해 l2 정형화를 시켜보았다.

정확도 : 67 -> 67

**코멘트 : 큰 변화가 없었다.**

**try 7**

optimizer : Adagrad(weight\_decay == 0.01 -> learning\_rate / num\_epoch)

정확도 : 67 -> 69

**코멘트 : weight\_decay를 크게 주면 overfitting이 발 생할 수 있다해서, 내 hyper parameterd에 맞게 주면 좋다해서 해당 식으로 weight decay를 줘 보았다.**

**learning\_rate == 0.01, num\_epoch == 100 -> 현재 weight\_decay = 0.0001**

**try 8**

activation\_function -> drop\_out ->batch\_norm순서를

batch\_norm -> activation\_function -> drop\_out순서로 변경, 구글링 중 이렇게 순서를 변경하면 학습이 더 잘 될 수 있다함.

drop\_out : layer별 모두 0.1씩 주는게 아닌, 0.1, 0.2, 0.3 다양성을 주었고, 마지막 fc\_layer에는 drop\_out 0.5를 주었다.

정확도 : 69 -> 71

**코멘트 : 처음으로 70을 넘겼다!**

**try 9**

fc\_layer마지막 부분에 nn.LogSoftmax추가

batch\_size : 32 -> 16

learning rate : 0.01 -> 0.005

num\_epoch : 100 -> 200

정확도 : 71 -> 79.88%

**최종 코멘트 : 가장 중요한 건 내 hyper\_parameter에 맞는 batch\_size와 거기에 맞는 learning\_rate, epoch을 찾는게 정확도를 높이는 가장 간단하면서 중요한 방법 이였던 것 같다. 왜냐하면 조금만 틀어져도 overfitting이나, underfitting이 발생해버리기 때문이다. 즉 아무리 좋은 기능과 layer를 무한정 늘려도 근본적인 틀이 안되어 있으면 효과를 발휘하지 못한다는 것을 깨닫게 된 정말 보람찬 실습 이였다.**