딥러닝 기초 기말 Project

학번: 20205218

이름: 이다해

**# 서약**

아래 보고서는 **본인의 힘만으로 작성**해야 하며, 다른 학생에게 질문과 다른 학생의 코드를 참고 하는 행위는 모두 금지합니다

\* 수업에서 제공한 코드, 노트북은 모두 재활용 가능하며, 카피로 규정하지 않습니다

\* 수업 자료 이외에 참고자료가 있다면, 출처와 사용 부분에 모두 표시하는 경우는 모두 합당한 자료로 인정하겠습니다

\* 위에 대해서 모두 이해하고 동의했다면, 아래 `서약글`에 다음을 작성해주세요:

"본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 반칙을 할 경우 (제공자 포함) 본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다."

학번: 20205218

이름: 이다해

서약글: 본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 부정행위를 할 경우(제공자 포함)본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다.

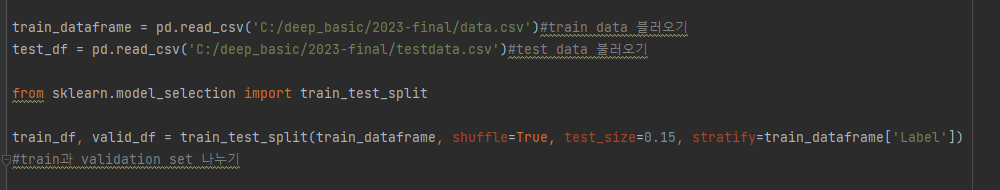
\*모든 코드에는 주석을 작성해 주세요

최종 제출시, 본 보고서와 .ipynb 노트북파일, test에 사용한 모델(.pt)파일을 압축해 제출해 주세요.

중요: 사용한 기법은 자신이 이해한 것 만을 사용하세요. 설명하지 않은 기법을 사용하면 그 부분을 제외하고 채점하겠습니다. 예를 들어서 자신의 힘으로 찾은 코드를 이용하려하는 경우 내용을 이해하고 보고서에 이해한 내용이 충분히 설명이 되어야만 사용을 허용합니다.

Step 1: Dataset 준비하기

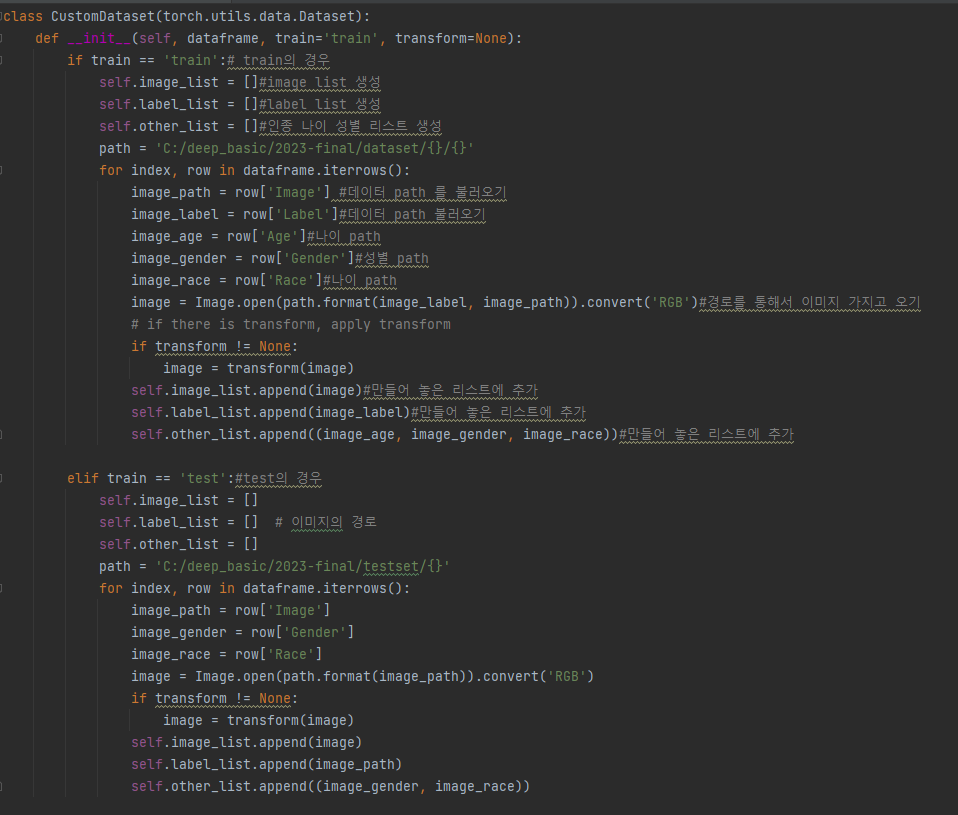
<코드 캡쳐 첨부>

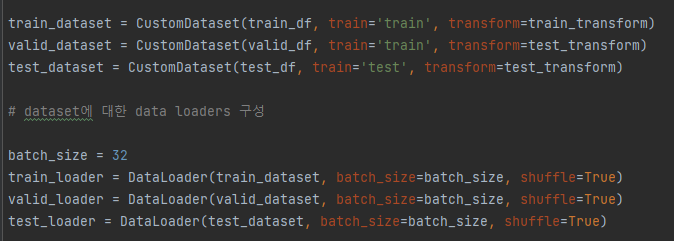


train\_dataframe으로 데이터를 불러오고 train에서 0.15프로를 test로 주었습니다. 보편적으로 하는 7:3 비율로도 나누어 보았지만 저는 저 경우가 가장 학습이 잘되었습니다.

Step 2: Dataset에 대한 Data Loaders 구성

<코드 캡쳐 첨부>

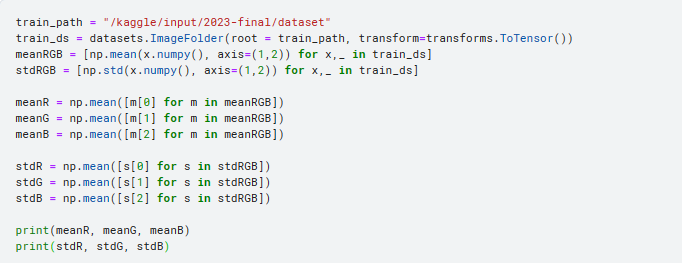




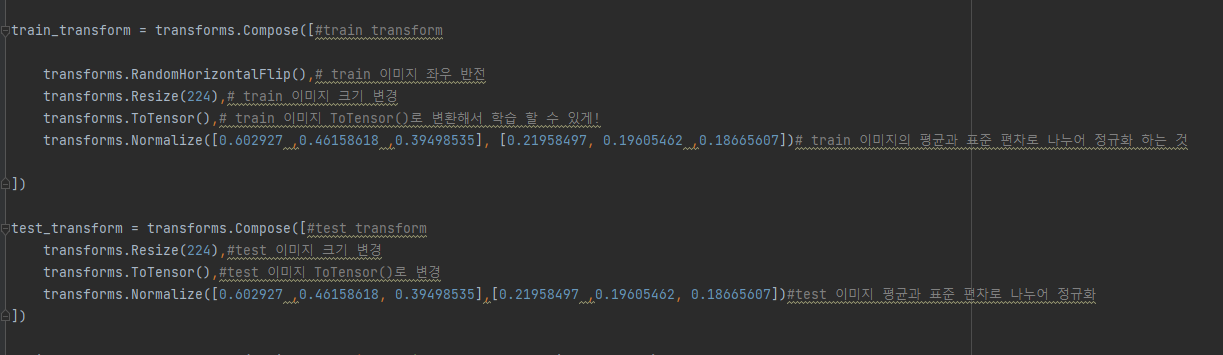
위에서 만든 CustomDataset을 통해서 train, validation, test를 불러오고 Dataloader를 통해서 미니 배치만큼의 data들 묶어서 데이터를 생성한다. 이 단계를 거친 데이터는 모델에 넣어 학습이 가능하다.

데이터 전처리에 대한 설명

데이터 증강

데이터 증강 기법인 data augmentation은 데이터의 양을 증가시켜서 과적합을 막아준다. 다양한 기법들이 있지만 이번 task는 사람 얼굴이기 때문에 수직 변환(Vertical Flip), 색깔 변환을 주면 안될 것이라고 생각을 했다. 사람의 얼굴은 좌우가 같기 때문에 Random HorizontalFlip()을 주었다. **Normalizatio**n은 데이터의 분포를 고르게 하여 gradient 업데이트 시 더욱 원활한 학습 환경을 만들어 주기 위해서 한다. 하지 않을 경우 느리고 성능이 좋지 못할 수 있어 학습 시 하는 것이 좋다. Normalization을 하게 되면 값의 범위를 0~1 사이로 옮겨준다. 코드는 다음과 같다.

3채널을 가지고 있기 때문에 meanR, meanG, meanB로 나누어 평균을 구해주표준 편차도 이와 마찬가지로 해주었다. 그 값을 transforsms.Normalization([RGB각 평균],[RGB 각 표준편차]) 안에 넣어 주었다.

<코드 캡쳐 첨부>

Test에는 변형을 많이 주면 영향이 있을 것이라고 생각하여 기본적인 resize(),ToTensor(), Normalize() 정도를 해주었습니다.

Step 3: Neural Network 생성

저는 모델 4개를 앙상블하여 최종 모델을 제출하여 4개의 모델 구조를 첨부하였습니다.

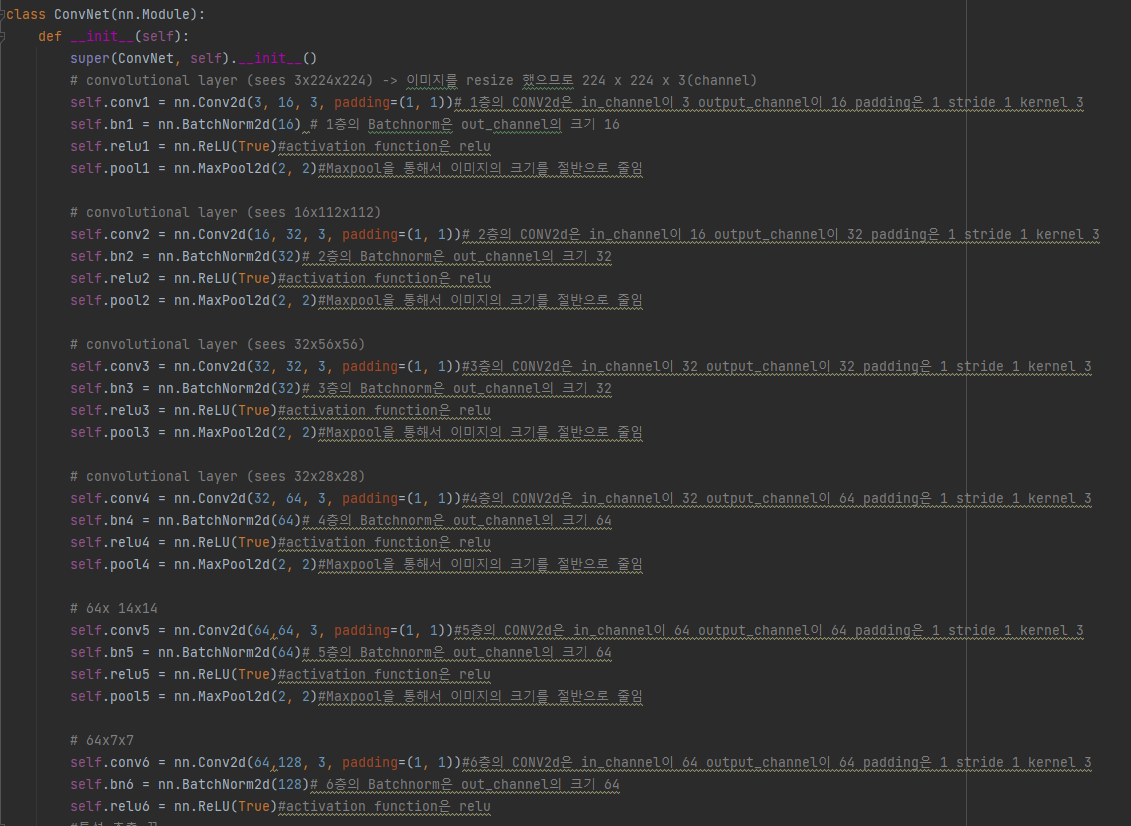
<코드 캡쳐 첨부>

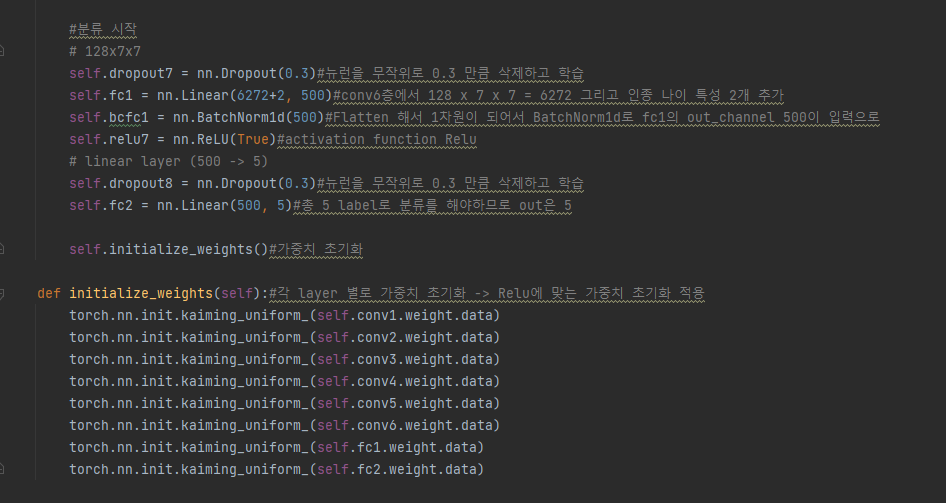
모델 5번과 모델 2번은 구조가 같습니다.

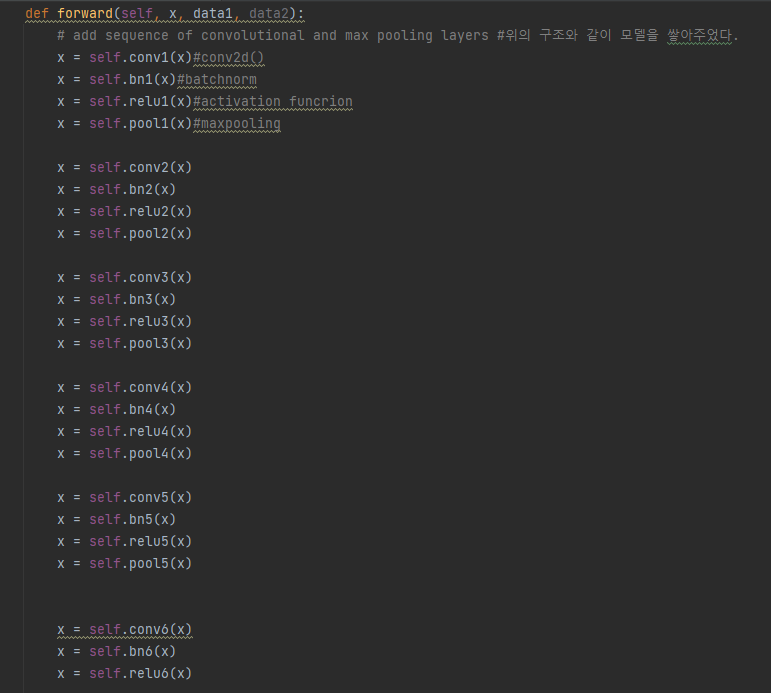
모델 5> public score -> 64.72/model5: convet\_best3\_6\_19\_11\_36\_1\_MODEL.pt

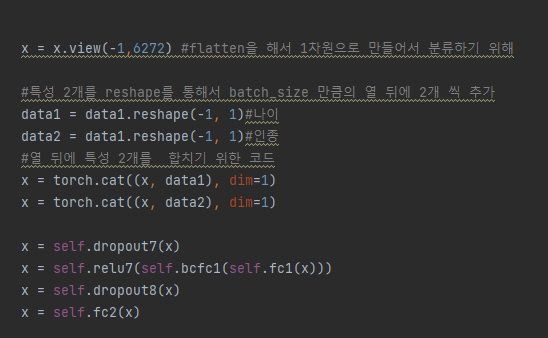
모델 2> public score -> 63.746/model2: convet\_best3\_6\_19\_8\_49\_1\_MODEL.pt

**특성을 추출하기 위한 conv 6층**



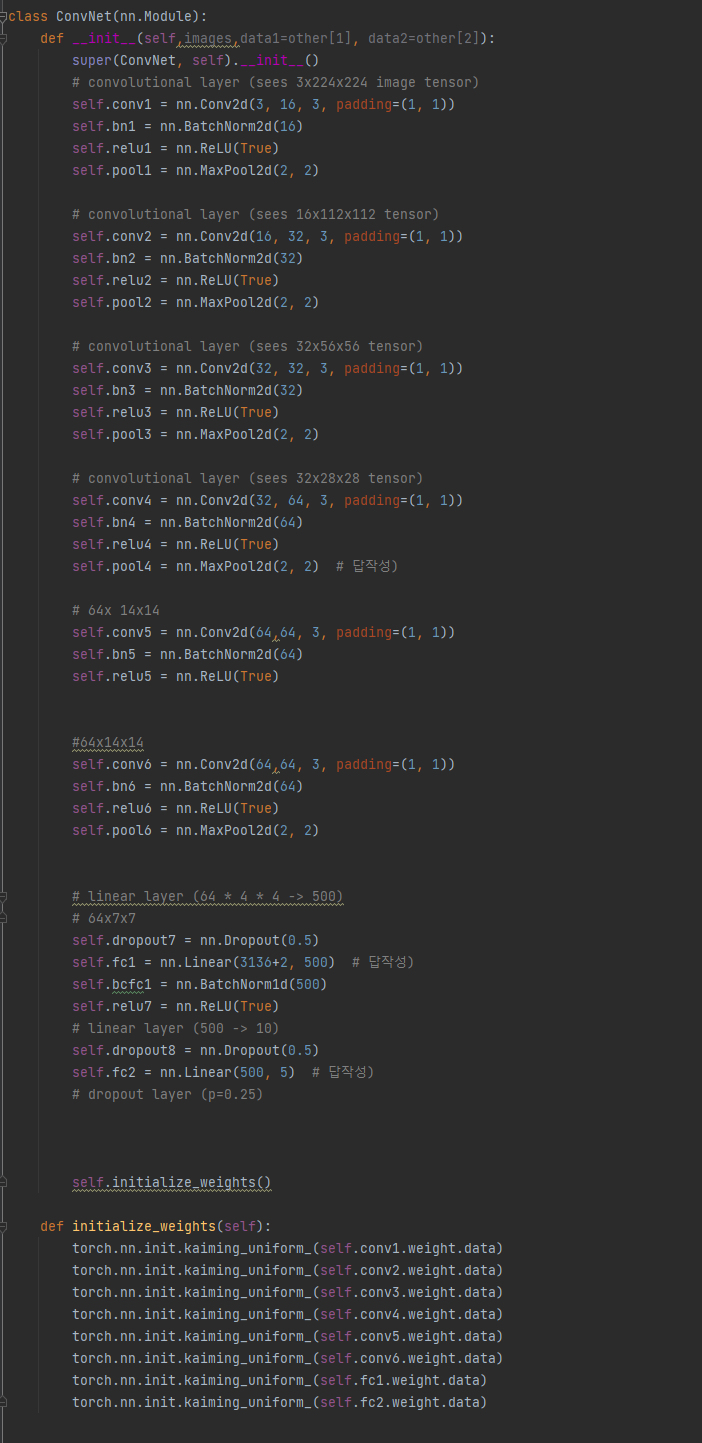
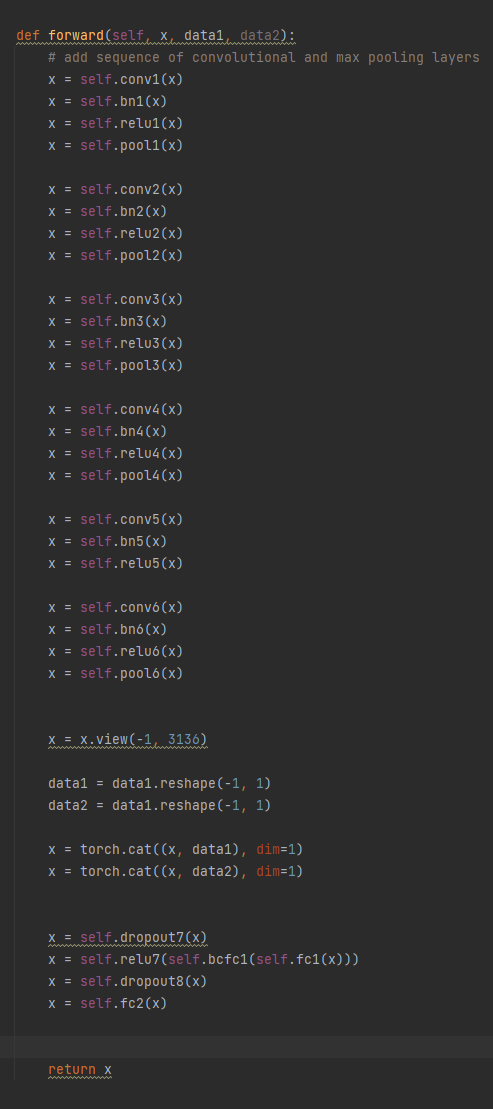
분류를 위한 fc1, fc2 과 가중치 초기화

위에 쌓아 놓았던 모델 구조를 forward 과정에서 차례대로 계산



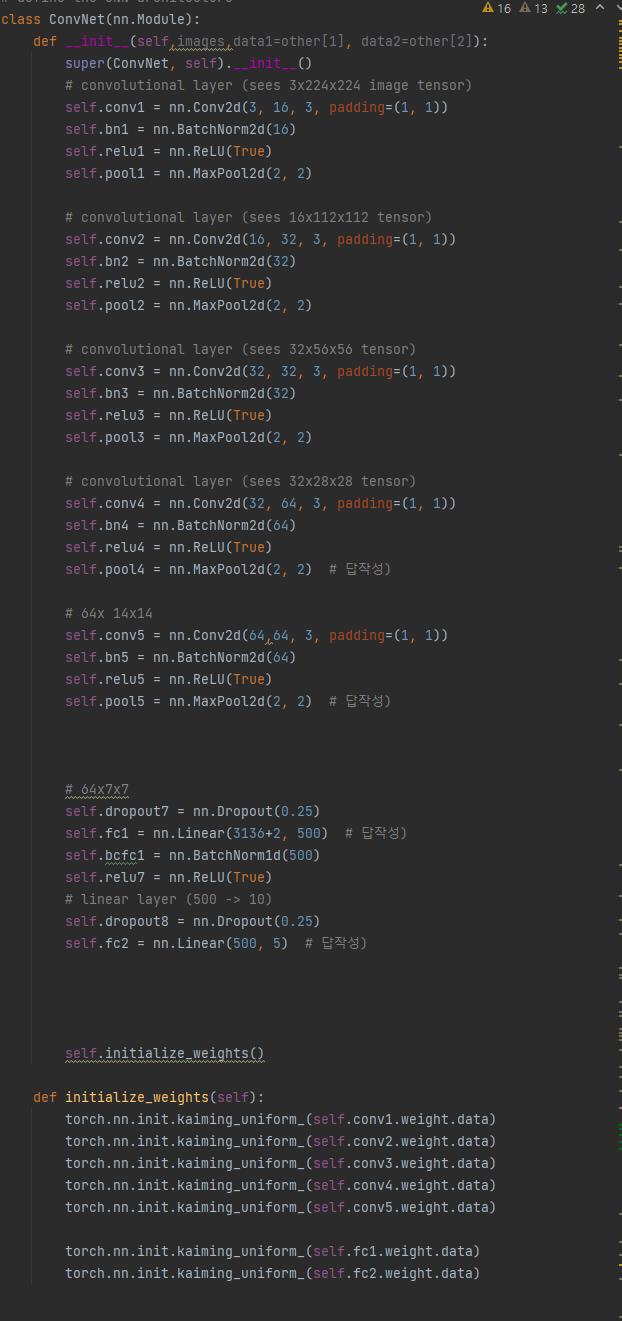
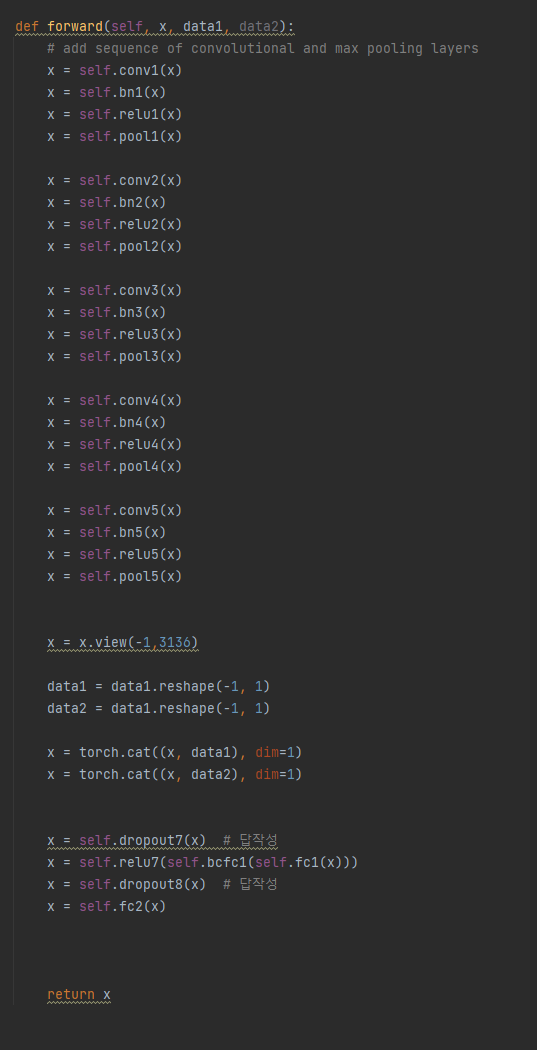
모델 3> public score -> 62.65 / Model3 : convet\_best3\_6\_2011512839\_MODEL.pt

**conv 6층 & FC 2층 & forward & 가중치 초기화**

**** ****

모델 4> public score -> 62 / model4 : convet\_best3\_6\_19\_MODEL.pt

**conv 5층 & FC 2층 & forward & 가중치 초기화**

설계한 모델을 출력 후 네트워크를 구성한 방법과 이유를 각 단계별로 설명

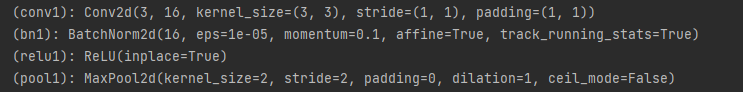
<서술형>

모델의 구조는 **conv – bn – relu – dropout – pool 순이 가장 일반적으로 사용되는 CNN 모델의 구조라고 블로그에서 보고 적용하였습니다. conv에서 특징을 추출하고 bastchnorm에서 정규화,relu에서 비선형을 추가하여 모델의 학습을 잘되게 하고 dropout을 통해 과적합을 막기 위해 뉴런을 무작위로 비활성화 시키고 마지막을 Pooling layer를 통해서 이미지의 특징 맵의 크기를 줄여 모델의 parameter를 줄여 학습이 빨리 되게 해준다.**

1. **차원 분석, 채널 수, 커널 크기, linear layer neuron 수**

**모델2번, 모델 5번>**

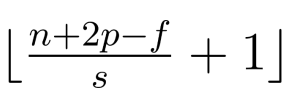
**Conv1층**

****

**Input\_channel은 3, output\_channel 은 16,kernel\_size는 3,stride와 padding은 1이다.**

**입력 차원은 (224,224,3)**

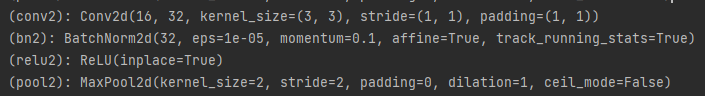
**Padding: p / Stride: s / Kernel: f / image (n x n)**

****

위의 공식을 대입하면 (224 + 2 – 3)/1 + 1 = 224 -> Maxpool(이미지 크기가 주어진 만큼 줄어든다) -> 112

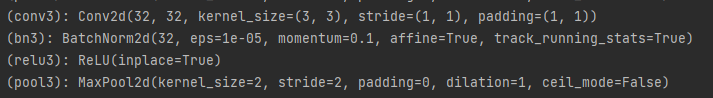
**출력 차원 (112x112x16)**

**Conv2층 -> Input\_channel=3, output\_channel=16, kernel\_size=3, stride,padding=1**

****

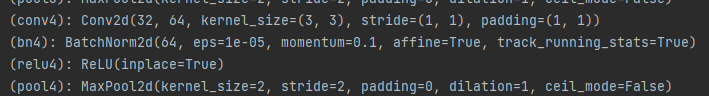
**입력: (112,112,16) -> 출력: (32x56x56)**

**Conv3층 -> Input\_channel=16, output\_channel=32, kernel\_size=3, stride,padding=1**

****

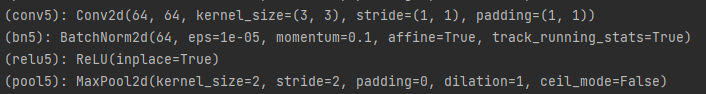
**입력: (32x56x56) -> 출력: (32x28x28)**

**Conv4층 -> Input\_channel=32, output\_channel=32, kernel\_size=3, stride,padding=1**

****

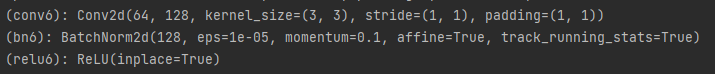
**입력: (32x28x28) -> 출력:(64 x14x14)**

**Conv5층 -> Input\_channel=64, output\_channel=64, kernel\_size=3, stride,padding=1**

****

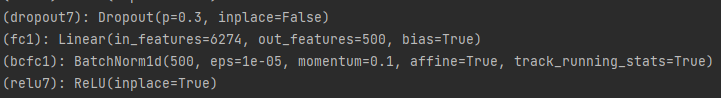
**입력: (64x14x14) ->출려:(64x7x7 )**

**Conv6층 -> Input\_channel=64, output\_channel=128, kernel\_size=3, stride,padding=1**

****

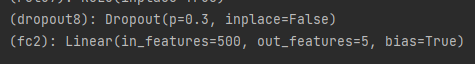
**입력: (64x7x7 ) ->출력: (128x7x7)**

**FC1 -> Input\_channel=6274, output\_channel=500**

****

**입력: (128x7x7) ->출력(500)**

**FC2 -> Input\_channel=500, output\_channel=5**

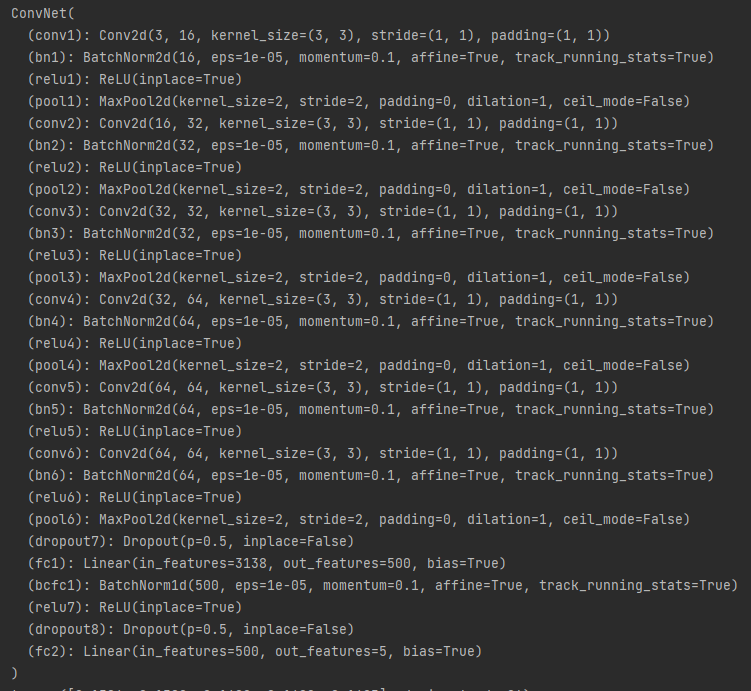
****

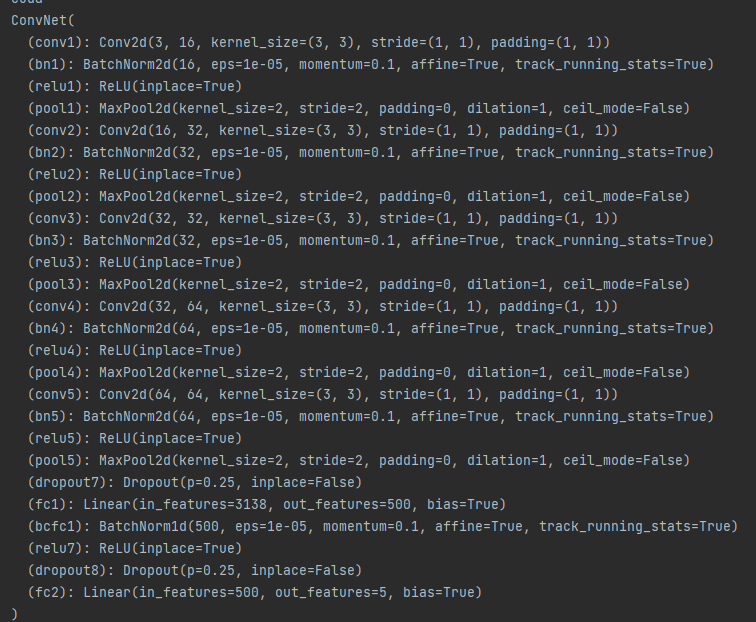
**입력: (500)->출력:(5)**

CNN의 경우 VGG16을 따라서 16층을 쌓아보았지만 이미지에 비해 모델이 너무 복잡하여 오버피팅이 나는 모습을 보였다. 그래서 layer층을 10층을 쌓아보았지만 10층에서도 loss가 발산하는 모습을 보여 5~7층 정도로 줄여서 실험을 해보았을 때 학습이 되는 모습을 보였다. 그래서 모델을 5층과 6층으로 주로 실험을 하여 학습하였다. 모델 2번과 모델 5번은 6층 CNN layer층을 쌓았다CNN에서 대부분의 층들은 input\_channel 과 output\_channel 이 다르지만 2층은 같다. Input\_channel과 output\_channel을 그대로 하면 이미지의 채널의 크기가 커지지 않고 Maxpooling이 있는 경우 이미지의 사이즈만 작아진다. FC층에서 입력이 너무 커지는 것을 막기 위해서 Input\_channel과 output\_channel이 같은 층을 쌓았다.

**모델3, 모델 4도 위와 같은 원리로 모델 구조를 설계하여 모델 구조 그림만 넣겠습니다.**

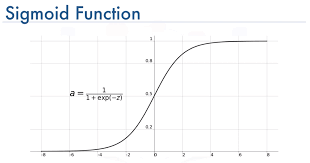
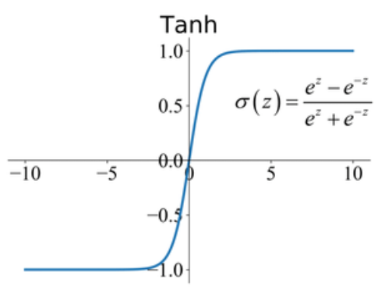
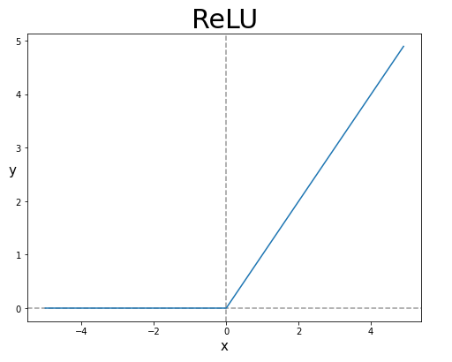
모델3>



****모델 4

1. **최종 layer에서 사용한 activation function이 무엇인지 왜 사용하였는지**

**추가적으로 Conv에 쓴 activation function이 이유**

Conv의 layer에 사용한 activation은 ReLU이다. ReLU를 사용한 이유는 아래 그림과 같이 Activation 함수의 일종인 sigmoid와 tanh는 음수 값일수록 미분 시 0을 내놓는다,양수 값으로 갈수록 미분 값이 0이 된다. 즉 backpropagation시 학습이 원활하게 되지 않을 수 있다. 그러나 ReLU는 그래프 모양을 보면 양수로 가도 자기 자신을 반환하므로 기울기가 소실될 수가 없어 학습이 원활 하게 되어 다른 활성화 함수들보다 속도가 빠를것이라고 생각하여 ReLU를 사용하였다.

최종 layer에는 activation function을 쓰지 않았다. 그 이유는 마지막 값을 softmax 넣어 클래스별 확률 값을 얻어야 하기 때문입니다.

1. **CNN 모델이 무엇을 입력 받고 출력하나요?**

CNN에는 사람 얼굴이 입력으로 들어오면 Conv을 거치면서 filter에 따라서 특징이 추출된다. Filter가 엣지를 검출하는 filter 일수도 있고, 빨간색을 검출하는 filter일수도 있다. 이 filter에 따라 feature map 추출된다.

30대 특징

Conv2

Conv1

Step 4: Cost (Loss) Function 과 Optimizer 선택

<코드 캡쳐 첨부>

모델 2> ->

optimizer = optim.Adam(model.parameters(), lr=0.001,weight\_decay=0.05)

모델 3>

optimizer = optim.Adam(model.parameters(), lr=0.001,weight\_decay=0.05)

모델 4>

optimizer = optim.Adam(model.parameters(), lr=0.001,weight\_decay=0.01)

모델 5> -.

optimizer = optim.Adam(model.parameters(), lr=0.001,weight\_decay=0.05)

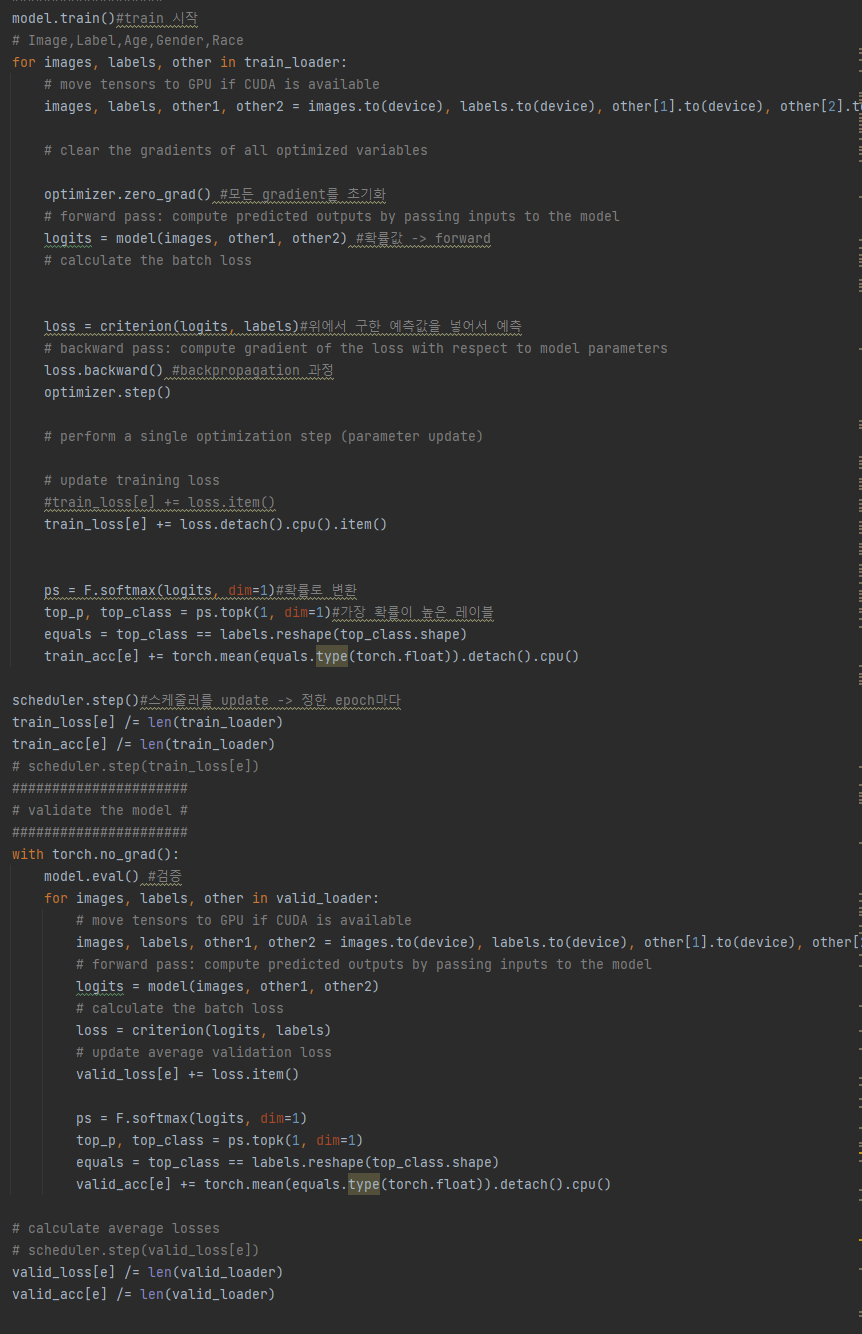
Optimizer와 Cost 함수를 선정한 이유와 선정하는데 중요하다고 생각하는 내용을 모두 작성합니다.

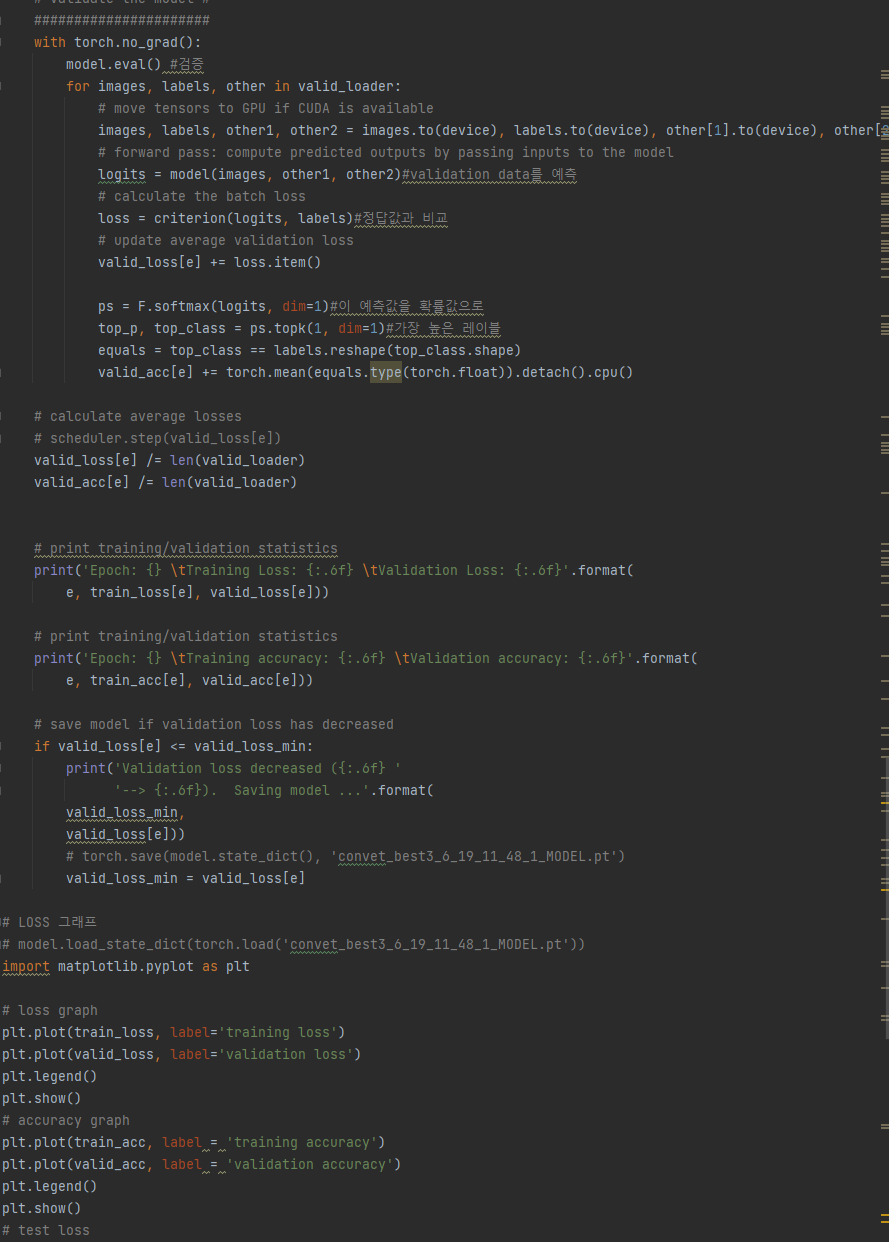
<서술형>

Adam을 선정한 이유는 SGD를 Cost 함수로 쓰면 학습이 불안정 해질 것이라고 생각하여 Adam을 사용하였습니다. SGD는 학습을 하다가 이리 저리 튈 가능성이 높다고 생각하여 momentum과

RMSProp의 장점을 모두 가지고 있는 Adam을 사용하였습니다.

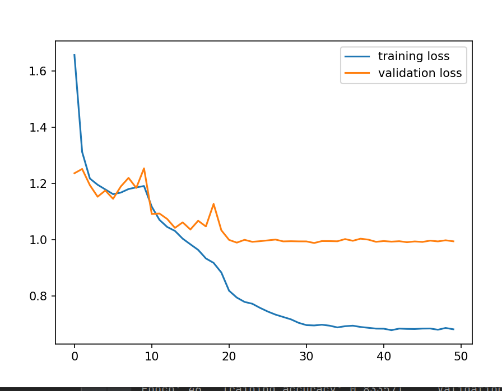
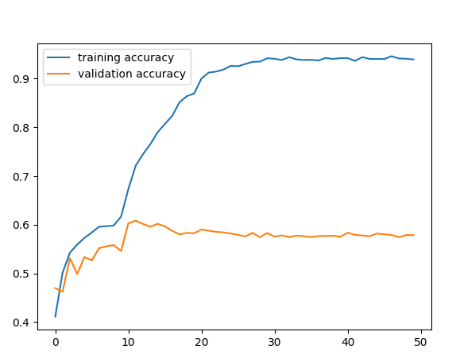
Step 5: 구성한 모델에 대한 Train and Validate 진행

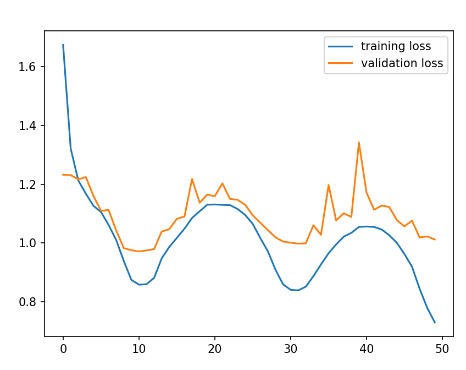
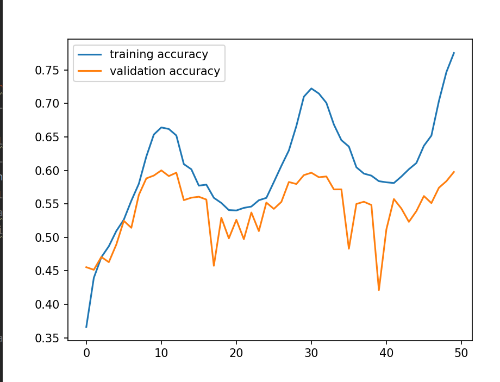




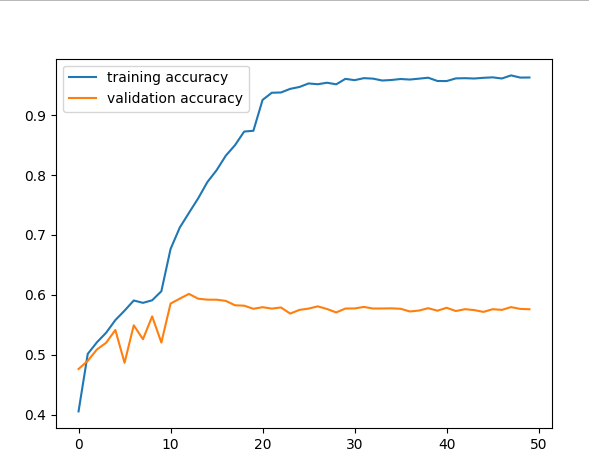
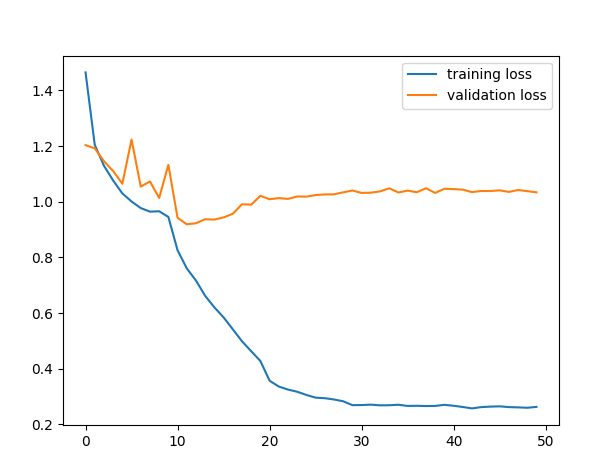
Step 6: CNN model training/validation 분석

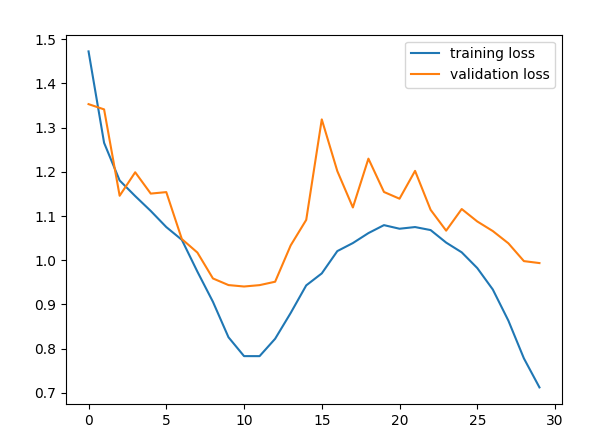
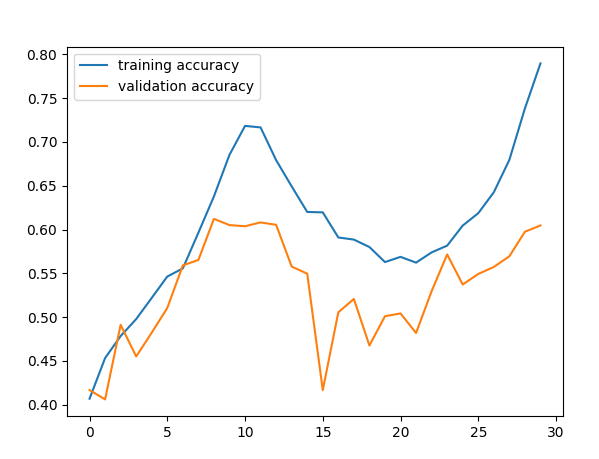
<코드 캡쳐 첨부>

모델 2>

모델 3>

모델 4>



모델 5>

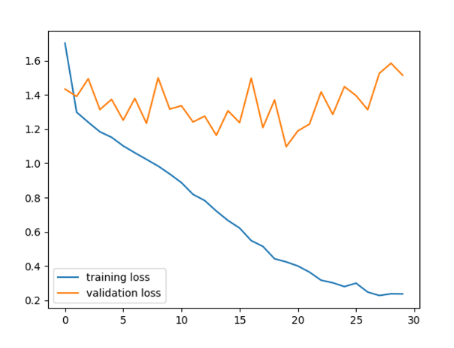
위에서 수행한 training + validation 과정을 설명하세요

training loss와 validation loss 그래프를 통해서 분석

Ex) hyper-parameter, model을 변경하면서 성능 개선한 과정을 최대한 설명하세요

overfitting, underfitting 분석 등..

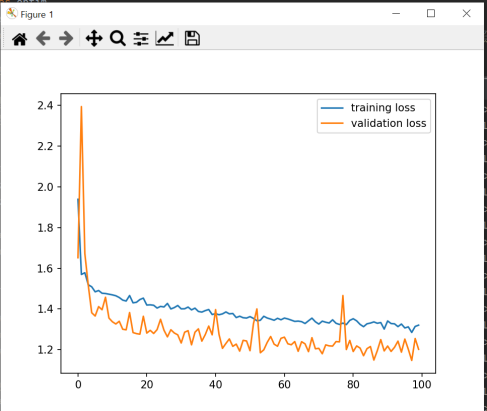
<서술형>

1. weight\_decay 없이 학습 돌리기를 할 때, batch\_size를 10으로 하고 lr은 1e-2 , weight\_decay없이 실행

저 당시에는 저 그래프를 보고 train loss는 내려 갔고 val loss는 떨어지지도 않고 발산한다고 생각하여 overfitting이라고 생각하였다.

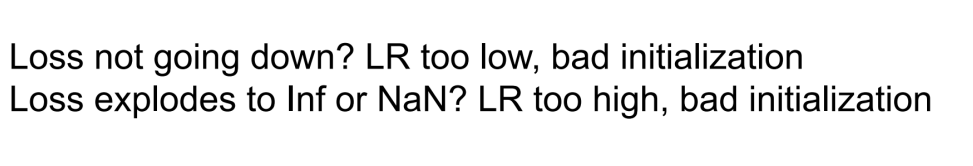
1. batch를 작게 주고 높은 accuracy를 시도

이때 loss가 줄어드는 속도가 매우 느렸었다.100epoch을 돌려도 그래프와 같이 나왔었다.

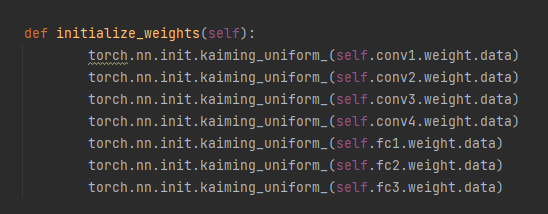


성능 또한 48%였다. 속도가 너무 느릴 때 lr을 높이면 효과가 있을 것이라고 생각했으나 속도는

빨라졌으나 loss가 크게 떨어지지 않았다. 그래서 weight 초기화가 처음에 잘못되었음을 알게 되었다.



그래서 relu에 맞는 **He weight initialization**을 적용하였다.



적용을 하고 53%로 성능이 향상되는 모습을 보였다. 딥러닝 학습이 초기 가중치 설정이 매우 중요한 역할을 한다는 것을 알게 되었다.

성능을 올리기 위하여 hyperparameter를 수정을 해보았다.

하지만 loss를 낮추는 최적의 조합을 찾지 못했다.

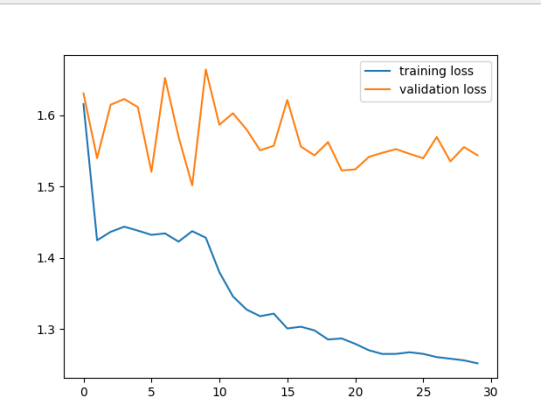
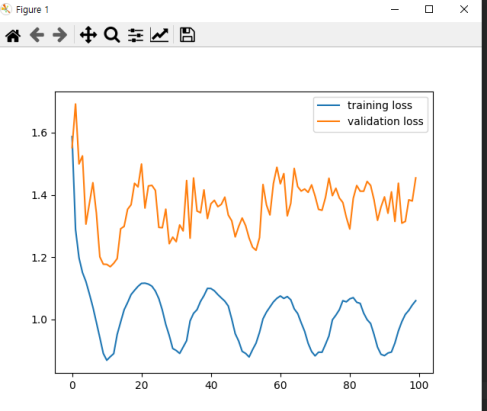
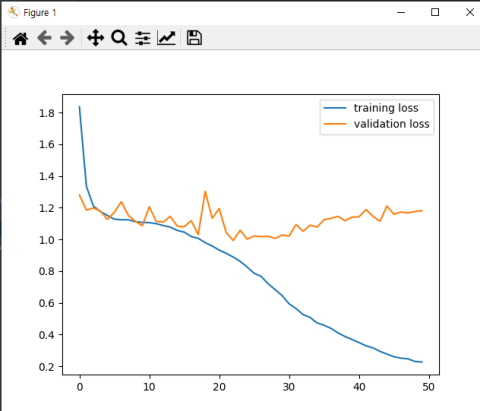
과적합을 해결하기 위해 모델의 사이즈를 줄여보았지만 해결을 하지 못하였다.

FC을 2층으로 변경 후 학습을 시켜도 크게 효과가 없었습니다.

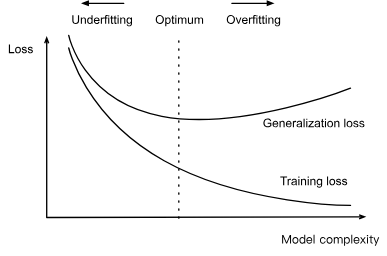
Weight\_decay와 regularization도 강하게 주었지만 더욱 발산하는 모양을 보였다.

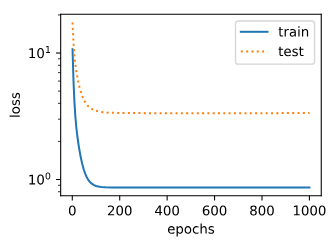
Dropout을 0.7 과 같이 크게도 주었지만 그대로였다.

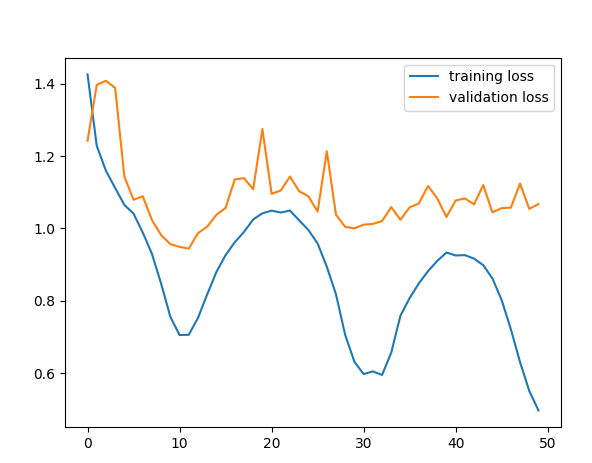
그래서 다시 원래 모델로 돌아가 3가지 스케줄러를 적용해 보았다.

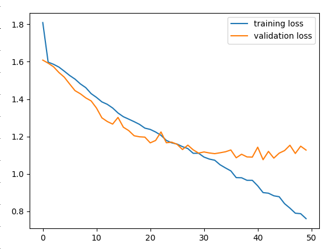
1. StepLR()
2. CosineAnnealingLR
3. ExponentialLR

사실 많은 시도를 했었지만 성능이 오르지는 못했었습니다 저 당시에는 저 문제를 오버피팅이랏고 생각을 했었다. 그래서 규제도 강하게 주고 transforms들도 다양하게 주는 등 문제를 해결하기 위해 다양한 기법을 이용하였다. 그때는 underfitting이라면 아래 그림과 같이 천천히 줄어드는 것이라고 개념을 잘못 알았다. 이것을 알기까지 정말 많은 학습을 시켜 봤었던 것 같다.



블로그를 찾다가 아래 그림과 같이 초기 epoch을 수행한 후 오류가 감소하지 않는 경우도 언더 피팅이라는 것을 알게 되었다,

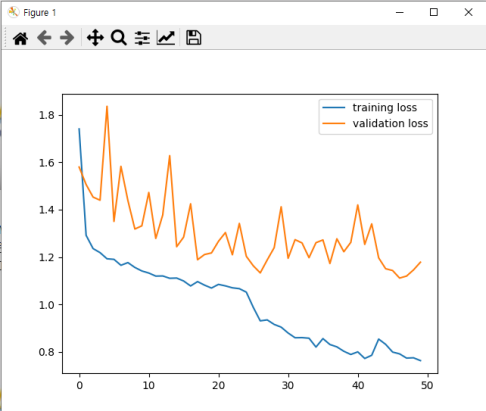
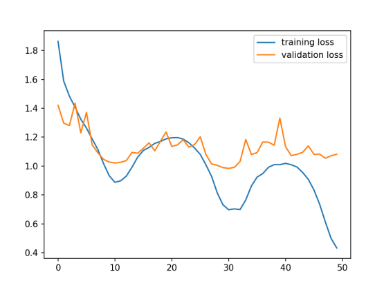
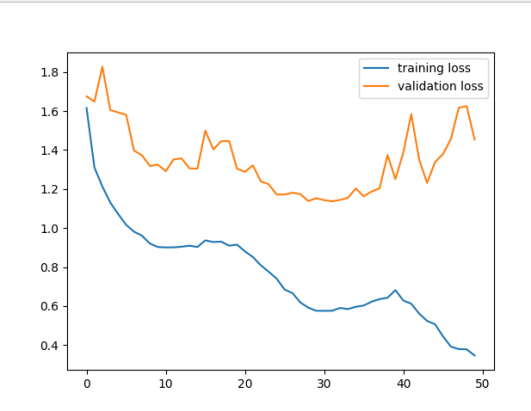
그래서 언더피팅을 해결하기 위해서 Conv 층에 주었던 모든 dropout을 빼고 FC layer에만 남겨 두었다. 이때 accuracy가 62%가장 크게 올라갔다. Validation loss도 매번 1.xx에서0.99정도로 줄어들었다. 아래 그래프와 같이 언더피팅을 해결하는 loss도 잘 떨어졌다.



이 실험 과정을 통해서 과적합이 났을 때 무엇을 해야 하는지 정말 많은 것을 해보며 알게 되었고 또한 loss 그래프 보는 방법도 배웠다.

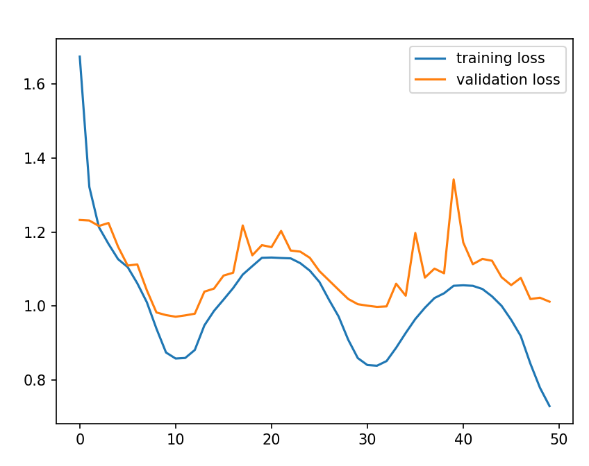
Validation loss를 더 내려 성능을 올리기 위해서 다양한 시도를 해보았다. 최적의 조합을 찾기 위하여 learning rate, weight\_decay, 스케줄러를 조절을 하며 실험을 해보았다. 실험하면서 다시 과적합이 났지만 끝으로 가서 해결을 하며 그래프가 거의 train과 validation이 같은 경향을 보이는 것이 많았다

실험

최종

모델을 낮추고 weight\_decay를 낮추고 스케줄을 cosine으로 주었을 때 결과가 좋았다.

Step 7: Predict with Test Data

<코드 캡쳐 첨부>



Step 8: Training Techniques

성능 개선을 위해서 사용한 기법 중에서 특별히 효과적이었던 부분이나 강조하고자 하는 내용을 작성해주세요.

과적합을 해결하기 위해서 **dropout**과 **batch normalization**을 추가하여 모델을 구성하였고 optimizer에 **regularization**을 주기 위하여 weight decay를 추가하였습니다.

1. **dropout** – 뉴런을 무작위로 제거를 하여 정규화의 효과를 주는 것이다

dropout을 강하게 줄수록 규제가 강하게 dropout을 약하게 줄수록 규제가 약하게 적용이 되고 0.5를 보편적으로 사용한다. 성능을 올리기 위해 과적합이 나면 올리고 언더피팅의 경우 없애거나 줄였다. 처음에 모든 Conv 층에 dropout을 주었는데 이것을 제거하고 성능이 올랐다

1. **batch normalization – 정규화**

conv안에 dropout과 batch normalization을 적용시 **conv – bn – relu – dropout – pool** 이런 순서로 주는게 가장 효과적이라고 블로그에서 봐서 이렇게 적용을 했다. Layer를 통해 들어오는 입력을 Batchnorm을 통해 정규화를 하여 학습을 안정화시키는 효과가 있다. 처음에는 Conv층을 conv- relu-pool만 주어서 성능이 50프로였지만 batchnorm을 주고 성능이 50프로 정도 올라갔다.

1. **regularization** – weight\_decay

과적합을 막는데 가장 좋았던 방법은 regularization이였다. 손실함수에 규제를 적용하여 오버피팅을 막는 기법이다. Weight\_decay 값이 커질수록 규제가 커지는 것이고 작을수록 작아지는 것이다, 그러나 너무 강하게 규제를 주면 모델이 과하게 규제되어 언더피팅이 나게 된다. 실제로 실험할 때 weight\_decay의 적절한 값을 찾는데 어려움이 많았다.

1. Ensemble

성능을 올리는데 가장 효과를 보았던 것은 앙상블이 였다. 앙상블은 여러 모델 결과를 모아서 최고의 성능을 내는 것이다. 앙상블 기법이 다양하지만 그 중에서 Soft voting을 사용하였다. Soft voting은 다수의 모델에서 예측한 확률 값을 한 곳으로 모아 더해 확률 기반으로 예측을 하는 것이다. 코드는 다음과 같다. Model.eval()을 통해서 모델의 각각의 확률 값을 도출해 내고 그 모든 것을 더하여 softmax에 넣어주었다. 이렇게 하면 모델 중 하나가 잘못 예측을 해도 나머지 학습이 잘 된 3개가 상쇄를 시켜준다.앙상블을 하여 5프로 정도 올랐다. 다음은 앙상블 코드다.

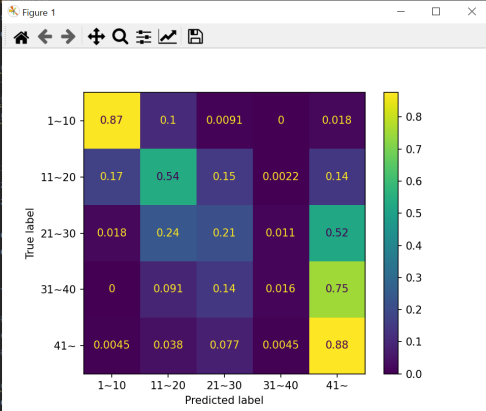
각각의 모델을 가지고 와서

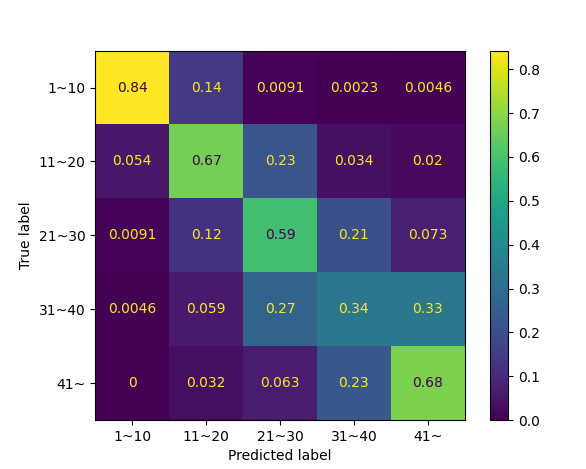


가중치 앙상블도 해보았다. 각각 모델에 가중치를 주어 어떤 모델은 영향을 많이 미치게 하고 어떤 모델은 낮추는 것이다. 이것은 실제로 효과를 보지 못하였다.

1. Label\_smoothing

-confusion matrix 첨부

실험을 하면서 항상 confusion matrix를 뽑아서 보면 31~40대를 구분을 잘 하지 못하는 결과가 나왔었다. 아래의 그림을 보면 유독 30~41대가 0.016으로 낮은 확률인 것을 볼 수 있다.

처음에는 label smoothing을 주어서 저 label의 정확도가 올라갔다고 생각을 했다. 하지만 label smoothing은 모델이 너무 정답을 확신하지 않고 다른 레이블도 보게 함으로써 일반화 성능을 올리게 하는 것이다. 즉 과적합이 방지 될 수 있다.아래 그림을 보면 실제로 레이블의 확률 값이 높아졌음을 확인이 가능하다. label smoothing시 적용시 다른 레이블도 고려를 하면서 자연스럽게 30~41대 label의 확률도 올라 같던 것 같다. 실제로 성능이 올랐던 이유는 과적합이 막아지면서 loss가 낮아져서 성능이 올랐다고 느꼈던 것 같다.