CV-13 5penCV

Image Classification 대회 리뷰

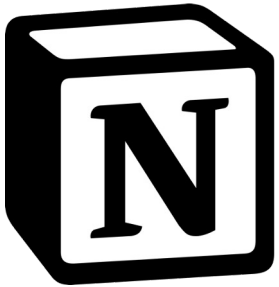
목차

1. 역할

	강정우	김윤지	송인성	정성혜	최홍록
Role	Modeling, Augmentation, Loss, Ensemble	Modeling, Augmentation, Loss, Stratify	EDA, Modeling, Augmentation, Loss	EDA, Modeling, Augmentation, Loss, Sampler	Modeling, Augmentation, Loss, Hyperparameter
Hash	#해피바이러스	#꼼꼼 박사	#기록 무새	#구현 박사	#튜닝 장인

2. 협업

Github
#코드공유
#버전관리



Notion
#실험계획
#결과정리

Slack
#소통
#이슈파악

WandB
#결과공유
#실험관리

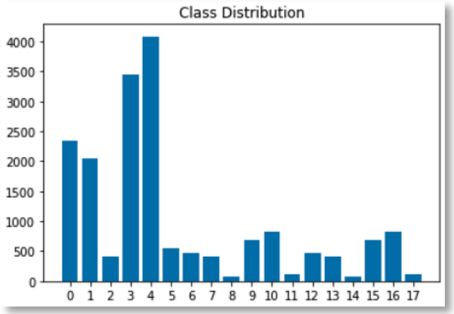
3. 과제 분석 - 목표

사람이 마스크를 **썼는지**, **안썼는지**, **이상하게 썼는지**와, **성별**, **나이대**(0~30, 30~60, 60~)를 판별하여 18개 클래스로 분류하는 문제이다.

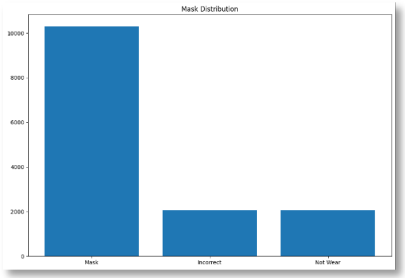



3. 과제 분석 - EDA

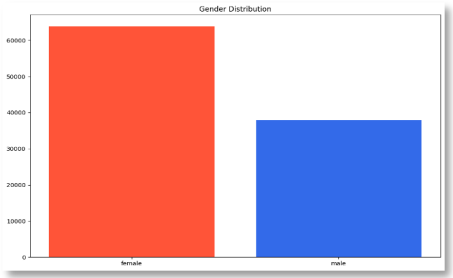
18개(마스크, 성별, 나이) 클래스의 분포



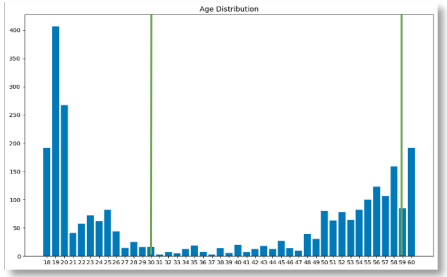
마스크 분포



성별 분포



나이 분포



Class Imbalance 문제 → 해결방법

1. Label smoothing
2. Sampler
3. Stratify

4. 실험 및 결과

4-1. Model

4-2. Loss

4-3. Data Augmentation

4-4. Data Preprocessing

4-5. Optimizer

4-6. Sampler

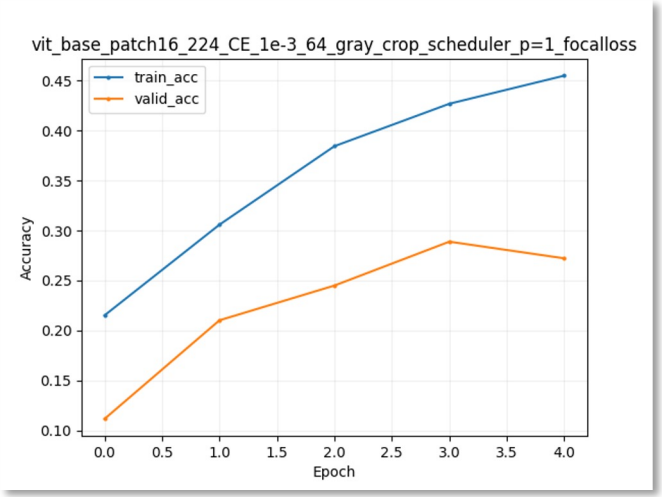
4-7. Stratify

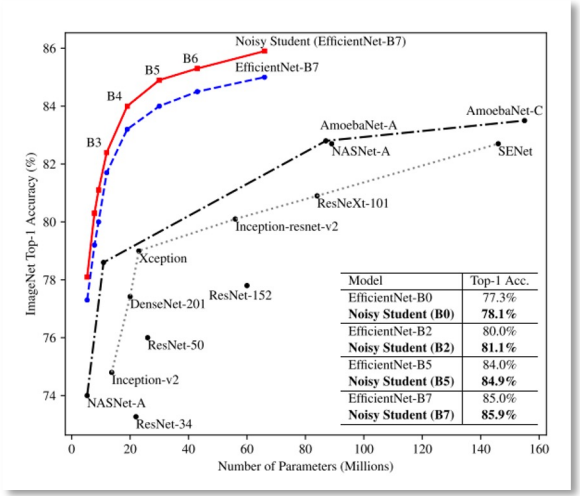
4-8. Ensemble

4. 실험 및 결과 - Model

다음과 같은 이유로 Transformer 모델이 아닌 CNN 기반의 모델을 선택했다.

- 학습 시켜야 될 파라미터가 많다.
- Transformer 는 많은 양의 데이터로 학습을 해야 이점이 드러난다.
- 실험 결과 accuracy가 0.2 주변으로 낮게 측정 되었다.





4. 실험 및 결과 - Model

timm 모듈의 pretrained 모델 중

-> Efficientnet_b0, Regnext32, ResNet34, ResNet50, Convnext_small 를 학습했다.

그중 valid_acc 가 가장 높고 valid_loss 가 가장 낮았던 **Efficientnet_b0**를 사용 하기로 했다.

4. 실험 및 결과 - Loss

데이터 불균형의 문제와 일반화 능력 향상을 위해

-> 기존 CrossEntropy Loss에 더하여 Focal Loss, F1 Loss, Labelsmoothing Loss를 적용해 실험했다.

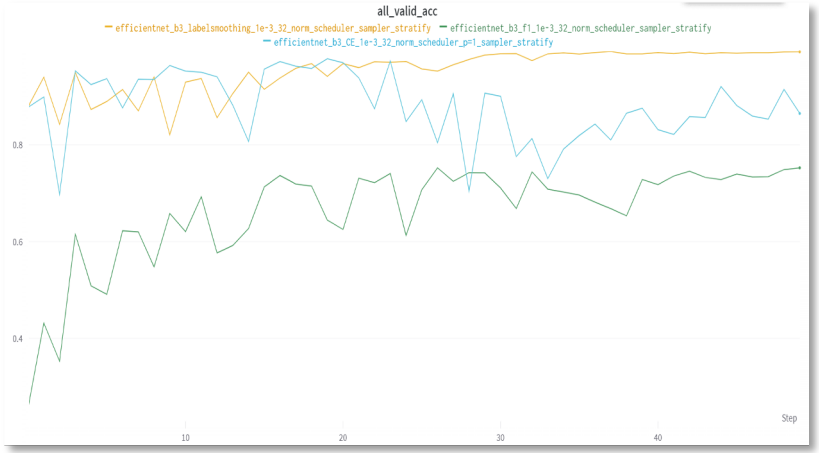
- Focal Loss : 손실함수의 가중치 계수를 조정해 불균형한 데이터셋에서 더 나은 결과를 보여줬다.

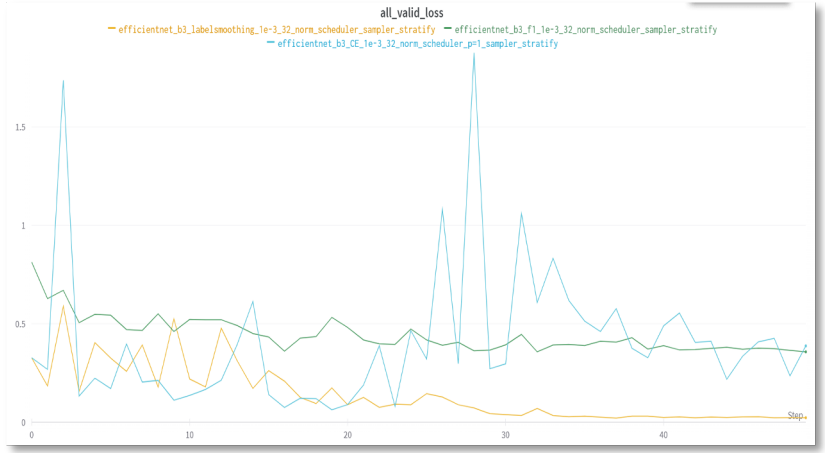
- F1 Loss : 정밀도와 재현율의 조화 평균을 이용하여 계산해 클래스의 불균형을 고려했다.

- Labelsmoothing Loss : 정답 클래스에 대한 softmax 확률을 1.0으로 고정하는 것이 아니라, 다른 클래스에 대한 확률을 약간 높이고 정답 클래스의 확률을 약간 낮추어 overfitting을 방지하고 모델의 일반화 능력을 향상시켰다.

4. 실험 및 결과 - Loss

CrossEntropy Loss, F1 Loss, LabelSmoothing Loss를 비교하여 실험을 진행하였다.

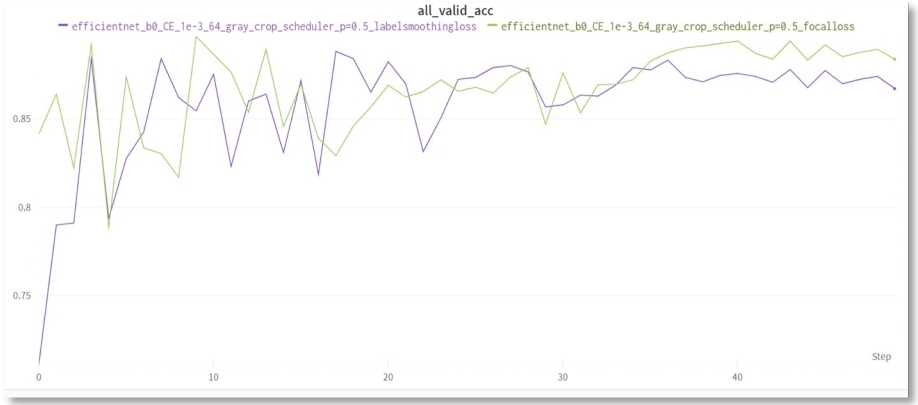


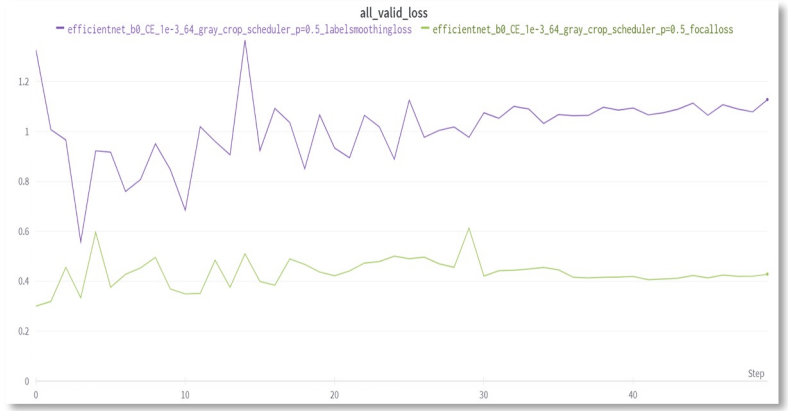


- 성능 : F1 Loss < CrossEntropy Loss < LabelSmoothing Loss
- 거의 모든 epoch에서 LabelSmoothing Loss의 성능이 좋았다.

4. 실험 및 결과 - Loss

Focal Loss, LabelSmoothing Loss를 비교하여 실험을 진행하였다.





- Validation Dataset에 대해서는 Focal Loss가 성능이 우세했으나,
- Test Dataset에서 f1 score 0.0006, accuracy 0.1만 향상되어, 큰 향상을 보이지 못했다.

4. 실험 및 결과 - Data Augmentation

적은 양의 데이터에 다양한 변환을 적용하여 데이터셋의 규모를 키우는 기법

Overfitting을 방지하는 효과를 얻을 수 있다.

```python
transform_dict = {
    'None': None,
    'random_gray': transforms.RandomApply(torch.nn.ModuleList([
                        transforms.Grayscale(num_output_channels=3)
                    ]), p=p),
    'random_all': torch.nn.Sequential(
        transforms.RandomApply(torch.nn.ModuleList([
            transforms.Grayscale(num_output_channels=3),
            transforms.ColorJitter(0.5),
            transforms.RandomHorizontalFlip(p=p),
            # transforms.CenterCrop(size=(384,384)),
            transforms.GaussianBlur((5,9), sigma=(0.1,5)),
            ]), p=p)
        ),
    'random_all1': torch.nn.Sequential(
        transforms.RandomApply(torch.nn.ModuleList([
            transforms.Grayscale(num_output_channels=3)
        ]), p=p),
        transforms.RandomApply(torch.nn.ModuleList([
            transforms.ColorJitter(0.5)
        ]), p=p),
        transforms.RandomApply(torch.nn.ModuleList([
            transforms.RandomHorizontalFlip(p=p)
        ]), p=p),
        transforms.RandomApply(torch.nn.ModuleList([
            transforms.GaussianBlur((5,9), sigma=(0.1,5))
        ]), p=p),
        ),
}
```

Train Dataset에 다양한 Transform을 랜덤으로 적용

- Grayscale: 흑백 처리
- ColorJitter: 밝기/채도/대비/색조 조절
- Center-crop: 이미지 중앙 부분만 잘라내어 사용
- Gaussian Blur: 노이즈 제거
- HorizontalFlip: 좌우대칭

Transform이 적용되는 확률을 조절해가며 효과 비교

- random_all: 모든 변환을 p 확률로 적용
- random_all1: 각 변환을 p 확률로 적용

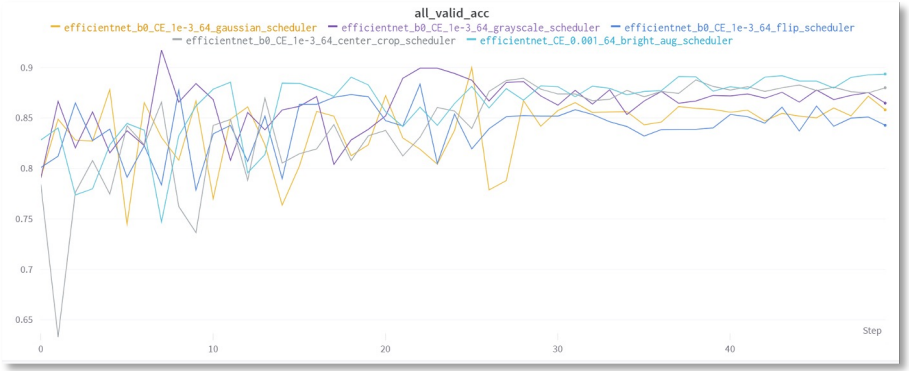
4. 실험 및 결과 - Data Preprocessing

기존 데이터에 추가로 data augmentation을 통해 data를 생성하지 않고, data augmentation에 적용한 기법을 Train Dataset과 Validation Dataset 전체에 적용하고 Test Dataset에도 preprocessing을 적용했다.

사용한 기법 : Grayscale, ColorJitter, Center-crop, Gaussian Blur, HorizontalFlip

사용하지 않은 기법: RandomRotation, VerticalFlip

- 사람 얼굴은 돌리거나 위아래로 뒤집으면 제대로 된 방향이 아니기 때문이다.

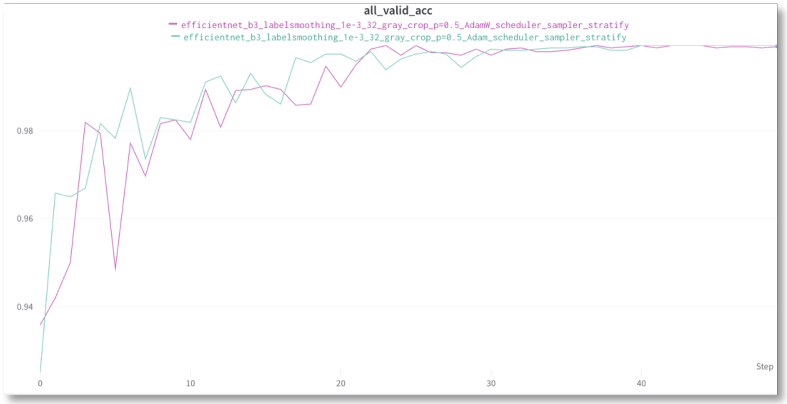


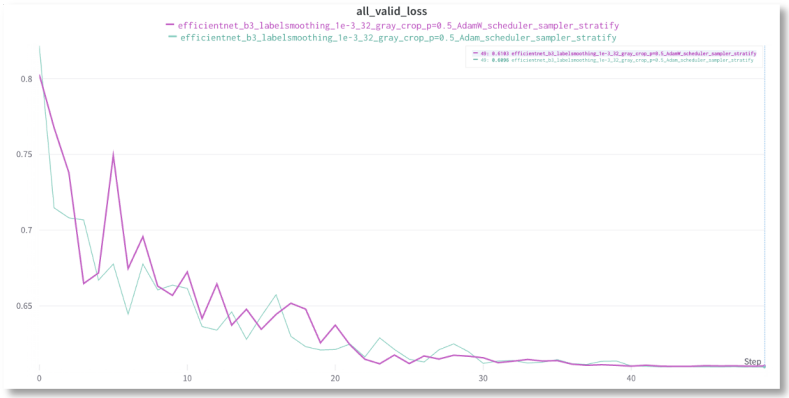
4. 실험 및 결과 - Optimizer

Adam과 AdamW 두가지의 Optimizer를 비교했다.

-> 학습 속도와 정확성을 모두 고려하여 학습의 방향과 크기를 모두 개선한 기법으로 가장 많이 사용되어 오던 Adam 을 사용하였다. 그리고 Adam에 weight decay 일반화 방법을 적용해 업데이트 하는 과정을 고려하는 AdamW와 성능을 비교해 봤다.

- 결과적으로 유의미한 성능차이가 없었고 수렴속도가 더 빠른 Adam 을 사용해서 실험을 진행했다.

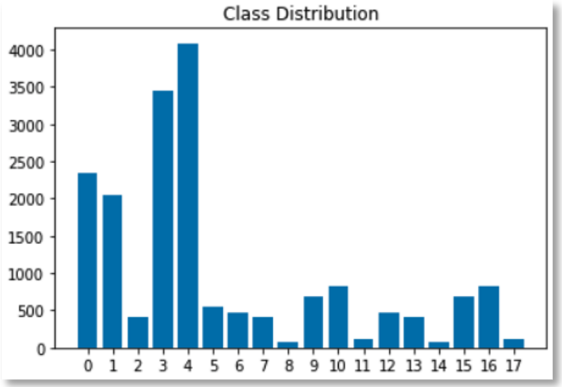




4. 실험 및 결과 - Sampler

Train Dataset의 클래스 불균형 문제를 해결하기 위해 **WeightedRandomSampler**를 사용했다.

-> 각 클래스의 이미지 데이터가 특정 확률에 따라 학습에 사용되도록 하는 기법이다.



Train Dataset 클래스 분포

특정 클래스에 대부분의 이미지가 몰려있고, 일부 클래스는 이미지 수가 100장 이하로 매우 적다는 문제점이 있다.

```python
num_samples = len(train_dataset)
train_labels = []
for _, label in train_dataset:
    train_labels.append(label['age'])
class_cnts = Counter(train_labels)

class_weights = [num_samples / class_cnts[i] for i in range(len(class_cnts))]
weights = [class_weights[t] for t in train_labels]
sampler = torch.utils.data.sampler.WeightedRandomSampler(weights, num_samples)
```

Sampler 구현 코드

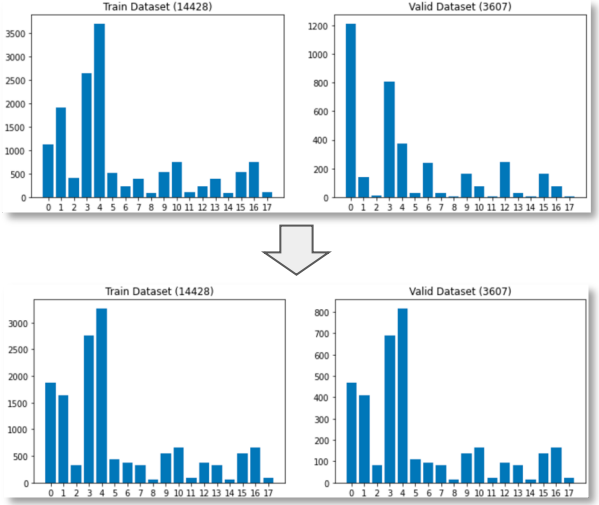
각 클래스 이미지 수 (calss_cnts)에 반비례하게 weight를 적용하여 모든 클래스의 이미지가 균등하게 학습에 사용되도록 했다.

4. 실험 및 결과 - Stratify

데이터 셋을 분리할 때 클래스 분포를 유지해주는 기법이다.

-> 특정 클래스의 데이터가 한쪽으로 많이 분배되는 것을 방지하기 위해 사용했다.

- Train Dataset에 대부분 분배되면, 검증할 데이터가 적어서 올바른 성능 측정이 어렵다는 문제가 발생한다.
- Validation Dataset에 대부분 분배되면, 학습할 데이터가 적어서 올바른 학습이 어렵다는 문제가 발생한다.



stratify 적용 전

Train, Validation Dataset의 클래스 분포가 다르다.

stratify 적용 후

Train, Validation Dataset의 클래스 분포가 동일하다.

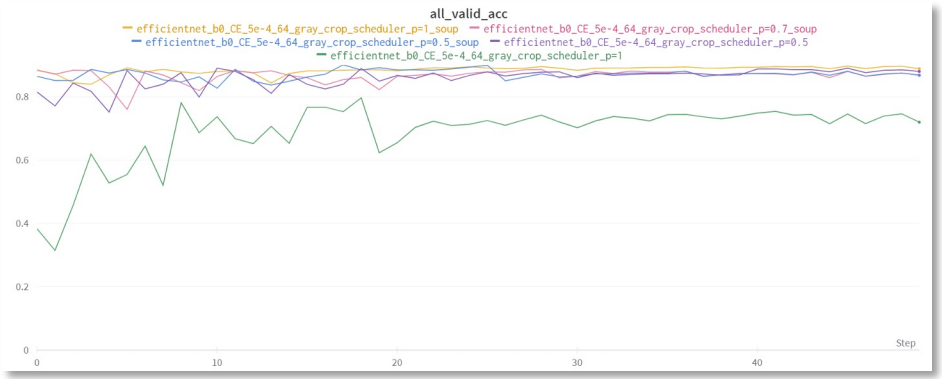
4. 실험 및 결과 - Ensemble

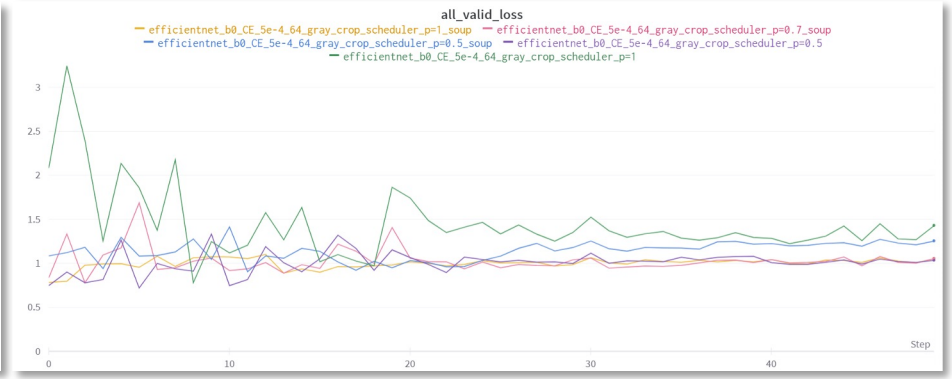
이전 실험결과 grayscale과 center-crop을 동시에 사용한 실험(이하 gray-crop)이 가장 성능이 좋았다.

그래서 해당 조합에 대해 적용되는 확률(p=0.5, 0.7, 1.0)을 조절하여 ensemble 실험을 진행했다.

모델의 학습 과정에서 저장한 checkpoint들의 weight를 앙상블하여 최종 weight를 계산해서 ensemble을 진행했다.

ensemble한 것중에는 preprocessing(p=1)된 모델이 제일 성능이 좋았지만, 이전 모델들에 비해 좋은 성능을 보이진 않았다.





5. 회고 및 개선방안

잘했던 점

- 빠르게 베이스라인 코드를 짜 팀원 모두가 빠른 이해와 함께 빠르게 실험을 진행 할 수 있었다.
- github으로 공유하면서 시간 절약과 코드 오류로 인한 시행착오를 줄일 수 있었다.
- 다같이 잦은 소통으로 의견들을 충분히 물어보고 최대한 절충할 수 있는 실험을 진행 할 수 있었다.
- 소통을 통해 매일매일 그 날 팀의 목표와 각자 실험을 할 내용을 정해 짧은 프로젝트 기간 동안 수월하게 진행되어 잘 마칠 수 있었다.
- 팀원들 모두 개인 역량이 뛰어나서 코드 구성에 무리 없이 새로운 것을 설계 하여 실험할 수 있었다.

아쉬웠던 점

- class distribution과 Train Dataset과 Validation Dataset간에 class 분포가 적절하게 되어있는지를 먼저 확인했으면 좋았을 것이다.
- WeightedRandomSampler만 적용한 것과 Augmentation을 같이 적용한 것의 비교를 못해본 것이 아쉽다.
- sampler 사용했을 때 왜 성능이 떨어지는지 분석해보고 싶다.
- Validation accuracy와 더불어 매 epoch마다 F1 score도 측정하여 제대로 학습되는지 확인해보고 싶다.

5. 회고 및 개선방안

개선 방안

- AdamW 로 실험을 더 해봤었을 것이다.
- F1 score 를 도입해서 제출 하기 전에 F1 score 를 비교 할 수 있어 성능 예측을 빠르게 할 수 있게 만들고 싶다.
- Hard voting 또는 soft voting을 해보고 싶다.
- 배경 제거 라이브러리를 통해 성능 향상을 시도해보고 싶다.
- 더 큰 모델을 사용해봤을 것이다.
- 클래스 별 추론 결과를 확인하여 모델이 잘 예측하지 못하는 클래스 특징을 파악해보고 싶다.
- backbone은 freeze하고 fc layer만 학습하는 방향을 시도해보고 싶다.