부스트캠프 랩업 리포트

<기술적인 도전>

본인의 점수 및 순위

`public accuracy 76.5873%, f1 0.7021` 125등

- 검증(Validation) 전략
    1. baseline 코드로 제시된 dataset 방법 중 val data를 무작위로 나눈게 아닌 프로파일별로 나누는 MaskSplitByProfileDataset 로 이미 만들어진 클래스를 사용하였고, 학습할 땐 val_ratio를 0.1로 하였습니다.

- 사용한 모델 아키텍처 및 하이퍼 파라미터
    1. 아키텍처: Vgg19BasedModel
        a. LB 점수 : `acc  65.1429%, f1 0.58`
        b. training time augmentation
            i. transform = A.Compose([
            ii. A.CenterCrop(355,355),
            iii. A.Resize(224,224),
            iv. A.HorizontalFlip(),
            v. A.ColorJitter(),
            vi. A.Rotate((-10,10)),
            vii. ToTensor()
            viii.])
        c. img_size = 224 x 224
        d. 추가 시도
            i. albumentations에서 지원하는 간단한 rotate, colorjitter 말고도 별 이상한 변환들(opticaldistortion, griddistortion, elastictransform…)도 넣어봤으나 오히려 성능 저하가 일어났습니다.
            ii. 데이터 불균형이라 저도 upscalling같은걸 써보고 싶었으나 어떻게 사용할지 모르고 복잡해서 그냥 정의되어 있는 focal loss를 사용했습니다. 성능이 미미하게 향상되었던 기억이 납니다. 사실 미리 정의되어있는 focal loss말고 catalyst라는 library에서 모든 데이터를 골고루 뽑는다는 dynamic balance class sampler라는 메소드를 써보고 싶었는데 사용법을 도저히 못찾았습니다.
            iii. 학습할 때 처음에 데이터 불균형으로 이상한 곳에 빠지면 빠져나오지 못하니 learning rate를 1e-5로 아주 낮은상태로 시작하니까 비율이 많은 데이터를 보고 그쪽으로 편향되어 학습되기 전에 비율이 적은 데이터도 보고 학습해서 올바른 방향으로 가는 것을 확인했습니다. 그래서 항상 overfitting이 일어나 val acc이 0%로 고정되던걸 해결했습니다.

iv.

2. 아키텍처: EfficientNet_b4
    a. LB 점수 : acc 76.7302%, f1 0.7067
    b. training time augmentation
        i. A.Compose([
        ii. A.CenterCrop(355,355),
        iii. A.Resize(260,260),
        iv. A.RandomCrop(resize[0],resize[1]),
        v. A.HorizontalFlip(),
        vi. A.ColorJitter(),
        vii. A.Rotate((-10,10)),
        viii. A.Normalize(mean=mean, std=std),
        ix. ToTensor()
        x.])
    c. Img_size = 224 x 224
    d. optimizer : AdamP
    e. scheduler : StepLR
    f. 기타 시도
        i. 그냥 adam보다 adamp를 쓰니 학습이 더 빨리 되었던 것 같습니다.
        ii. Loss를 focal loss만 사용했었는데 loss가 더 빨리 줄었던 걸로 기억합니다.
        iii. 여기서 앙상블을 시도했는데, mask, gender, age각각에 대한 efficientnet_b4를 따로 만들어 수행했습니다. 성능은 더 떨어졌습니다.
        iv. 단일 모델인데 저렇게 점수가 높은건, 그냥 모델 돌려놓고 자고 일어나서 epoch 40짜리로 오래된걸 돌렸더니 점수가 높게 나왔고 끝낼때까지 저 점수를 이기지 못했습니다. 이때 그냥 모델 좋고 훈련 많이하는게 최고라는걸 느꼈습니다.
3. 아키텍처: EfficientNet_b4(앙상블)
    a. LB 점수 : acc 75.4444%, f1 0.6995
    b. training time augmentation
        i. A.Compose([
        ii. A.CenterCrop(355,355),
        iii. A.Resize(260,260),
        iv. A.RandomCrop(resize[0],resize[1]),
        v. A.HorizontalFlip(),
        vi. A.ColorJitter(),
        vii. A.Rotate((-10,10)),
        viii. A.Normalize(mean=mean, std=std),
        ix. ToTensor()
        x.])])
    c. Img_size = 224 x 224
    d. optimizer : AdamP
    e. scheduler : StepLR
    f. 기타 시도
        i. Age를 계산할 때 60살이 제일 적어 데이터 불균형이 있으니 label로 구하지 말고 선형으로 구한 뒤 쪼개는 방법이 어떻겠냐

는 의견이 마스터세션 시간에 나와서 실험해봤습니다. 그냥 outlabel dimension을 1로 하고 측정했더니 결과가 평균값을 찾아내려고 해서 그냥 최대 나이에 맞춰 61개의 one-hot vector class로 예측하게 했습니다. 그 결과 나이 기준에 따른 label을 0<31<59로 하는게 제일 낫다는 결과를 얻었습니다. (f1 0.6802 -> 0.6995 향상)

ii. 점수 기준이 그냥 acc에서 f1으로 바뀌었으니 baseline code에 정의되어있던 f1 loss를 사용하면 좋지 않을까 라는 생각에 f1 loss를 써봤지만 성능이 매우 떨어졌던 것 같습니다. 그냥 cross entropy가 좋다는 생각이 들었습니다.

iii. 전에 age label을 0<31<59로 하는게 좋겠다고 결론내려 0<30<59와 0<31<59로 했지만 결국 0<30<59가 더 좋았던 걸로 기억합니다. 즉 뻘짓을 한거죠. 이로써 처음부터 one-hot vector 61class로 예측하고 내가 구간을 나누는 거랑 처음부터 3개의 class로 하는건 다른 예측이란걸 짐작했습니다.

4. 아키텍처: EffieicientBasedModel(b4)
    a. LB 점수 : `acc 72.9048%, f1 0.6639`
    b. training time augmentation
        i. transforms.Compose([
        ii. Resize(resize, Image.BILINEAR),
        iii. ToTensor(),
        iv. Normalize(mean=mean, std=std)
        v.])
    c. img_size = 224 x 224
    d. optimizer : AdamP
    e. scheduler : StepLR
    f. 기타 시도
        i. Efficientnet_b4를 쓰는데output으로 18개의 class를 내는게 아니라 3class, 2class, 3class를 output하도록 맨 마지막 layer에 각각에 대한 linear를 연결시켰습니다. 이때 각각의 linear layer가 깊을수록 성능이 좋아졌습니다.(f1 0.6639 -> 0.6809 향상)
5. 아키텍처: EffieicientBasedModel(b0)
    a. LB 점수 : `acc 71.4921%, f1 0.6561`
    b. training time augmentation
        i. transforms.Compose([
        ii. Resize(resize, Image.BILINEAR),
        iii. ToTensor(),
        iv. Normalize(mean=mean, std=std)
        v.])
    c. img_size = 224 x 224
    d. optimizer : AdamP
    e. scheduler : StepLR
    f. 기타 시도
        i. 위에서는 `efficientnet_b4` 끝단에 `linear`만 연결한 방식이기 때문에 모델 후반부 쯔음 `cnn`에서 `linear`에 들어가기 전에 나눠서 `cnn`을 좀 거쳐가면 어떨까 해서 만든 모델입니다. `Train loss`와 `val loss`는 확실히 줄어들었지만 `public`

score는 더 떨어졌습니다. 원래 b4모델을 개조하고 싶었는데 분석할 시간이 없어 b0 모델을 개조해서 그런 것 같습니다.

ii. 그래도 cnn layer와 linear layer를 깊게 쌓을수록 성능이 올라가는건 변함없었습니다.

- 앙상블 방법
    1. Mask, gender, age 각각의 모델에 대해 완전히 다른 모델을 사용하고 나중에 결과를 합치는 방법을 사용했습니다.

- 시도했으나 잘 되지 않았던 것들
    1. F1 loss로 했으나 성능이 안좋았고, focal loss도 잘 신뢰하지 못하겠습니다.
    2. 앙상블이 잘 되지 않았습니다.
    3. Argumentation이 많을수록 좋을 줄 알았으나, 오히려 성능이 떨어졌습니다.
    4. Efficientnet b4가 아닌 b0을 쓰니 성능이 떨어졌습니다.

<학습과정에서의 교훈>

f1loss니 focal이니 하지말고 그냥 cross_entropy 쓰자.

Train을 너무 많이 하면 overfitting이 날 수도 있지만 val은 loss가 낮고 acc이 높을수록 무조건 좋은 것 같다. 그래서 많이 훈련시키는게 장땡이다.

왜인진 모르겠지만 cnn에서 linear로 연결시킬 때 파라미터 개수 맞춰도 에러가 뜬다. Flatten()으로 먼저 하고 해야 문제가 안생긴다.

앙상블이 무조건 좋은건 아니다. 각각의 클래스에 대해 다른 모델로 학습시키는 방식 말고 투표같은 방식을 생각해보자.

이미지 normalization은 모든곳에서 mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225] 으로 통일이다. 하지만 내 학습 데이터를 기준으로 만들고 테스트 데이터에 적용할 수도 있으니 참고하자.

Accuracy 와 f1loss는 서로 다른 느낌인 것 같다. 해결방법은 잘 모르겠음.

Ipynb Notebook 쓰지 말고 IDLE 쓰자. 진짜 너무 좋고 편리하다.

나도 EDA를 하지만 토론게시판에 유용한 정보들이 많이 나온다. 계속 확인하자.

그냥 성능좋다고 평가받은 좋은 모델에 많이 학습시키는게 최고로 좋다.

한번에 한 개의 모델만 GPU에 올리자. 2개 이상은 out of memory 뜬다.

내가 수정하려는 pretrained model에서 layer가 sequencital list로 되어있는지 아닌지 잘 보자. 안그러면 입력갯수 오류가 자꾸 뜬다. *로 각 원소를 저장하는 형식으로 받아야 한다.

class module 안에 순수 파이썬 list같은걸 쓰지 말자. cuda에 안들어가서 장소 다르다고 오류난다.

<마주한 한계와 도전숙제>

아쉬웠던 점들

1. Tensorboard 사용법을 못익혔다.
2. 한번 훈련하는데 오래걸리고 조급해지다 보니 하이퍼 파라미터 변화에 대한 성능 영향을 볼 수가 없었다. 그래서 NNI(AutoML)도 해보고 싶다.
3. 이제 끝나고 토론게시판을 보는데 진짜 상위권분들이 좋은 생각과 방법론들을 공유해주셨다. 왜 이제 이걸 봤을까.
4. 한 모델에 여러 개의 output을 낼 수 있다는 걸 마감 6시간 전에 깨달았다.

한계/교훈을 바탕으로 다음 스테이지에서 새롭게 시도해볼 것

1. NNI 해보기
2. 토론 게시판 잘 보기
3. 미리 학습된 모델을 나름대로 잘 수정해보기