이미지 분류 대회

환경 세팅

- vscode SSH 접속 세팅
- 개인 github private repository 생성 후 발급된 서버에 clone
- 서버에 Filezila 를 통해 SFTP 연결
- 서버에 XShell 을 통해 SSH 연결

이슈 사항

- RuntimeError: CUDA out of memory
  - 배치 사이즈 등을 너무 크게 잡을 경우 GPU 메모리 초과 오류가 발생하게 된다.
  - 적절한 배치 사이즈를 지정하여 해당 문제를 해결할 수 있다.

프로젝트 탬플릿 구축

- classes.csv 파일 생성
  - 18개의 class 에 해당되는 조건에 대한 정보를 갖는 파일
  - 각 이미지별로 해당 파일을 조회하여 label 값 생성
  - classes.csv 를 만들어서 라벨링을 하는 것이 코드는 깔끔하지만 속도가 느린 편이라 다른 방법으로 바꿀 필요가 있어보인다.
- labeling 방법 변경
  - classes.csv 파일을 읽는 방법 대신 각각의 성별, 나이대, 마스크 착용 여부 label에 가중치를 줘서 최종 label 을 만들어내는 방식으로 변경하였다.
  - 이를 통해 기존 방법에 비해 데이터를 생성하는 시간을 많이 단축시킬 수 있었다.
- 기본 학습 로직 구현
  - 모델 학습 시 예측값을 변환 안한 상태를 loss에 넣어줘야 하는데 argmax 를 적용한 값을 넣어서 예러가 발생여 이 부분에서 시간을 많이 잡아먹었다.
- f1 score 출력하기
  - 학습 과정에서 학습 성능을 평가할 수 있도록 대회의 평가 지표인 f1 score 를 계산하여 출력할 수 있도록 로직을 구현했다.
  - 1 epoch 에서의 f1 score 계산 시 각 배치별로 누적된 값을 전체 데이터 갯수로 나눠줘야 하는 loss, accuracy 와 달리, f1 score는 배치 횟수로 나눠줘야 하는 점이 달랐다.
  - 해당 방법을 aistage 게시판에 공유하여 다른 사람들도 참고할 수 있도록 했다.
- 증폭되어 사용되는 로직 모듈화
  - CustomDataset
    - train.py 와 inference.py 에서 동일한 dataset을 사용하도록 설정
  - CustomModel
    - train.py 와 inference.py 에서 동일한 model을 사용하도록 설정
- 모델 저장 naming 규칙 설정
  - save_name = f"{dt.now().strftime('%Y%m%d%H%M')}_{model.name}.pt"
- slack notification 설정
  - slack 의 webhook 을 이용하여 epoch 마다 학습 성능을 slack 으로 알림받을 수 있게 설정
  - 해당 방법을 aistage 게시판에 공유하여 다른 사람들도 참고할 수 있도록 했다.
- 모델 저장 방식 변경
  - 모델을 통째로 저장하는 방식에서 state_dict 을 저장하는 방식으로 변경
  - 맨 마지막 epoch 의 모델을 저장하는 것에서 f1 score 가 향상될때만 모델을 저장하는 방식으로 변경
- wandb 설정
  - 각 실험별 하이퍼파라미터를 관리할 수 있고, train, validation 각각의 loss, acc, f1 score 를 관리할 수 있다.
- early stopping 적용
  - validation f1 score 가 하락하는 경우가 2번 발생하면 early stop 하도록 설정
  - 이후 validation f1 score 가 개선될 때마다 모델을 저장하는 방식을 적용함으로서 early stopping 기능은 사용 안함
- team github repository 에 코드 병합
  - 각 캠퍼별 폴더를 만들어서 각자의 코드를 하나의 repository 에 병합
- baseline 분석
  - 멘토님이 구현하신 baseline 코드를 분석하여 전체적인 흐름 파악과 유용한 기능을 확인
- config.json 파일 생성
  - 모든 하이퍼파라미터들과 wandb 및 slack noti 와 같은 설정 정보들을 갖고 있는 config.json 파일을 생성
  - train.py 와 inference.py 에서 config.json 파일을 argparser 로 읽어들여 사용하도록 로직 수정
- 프로젝트 탬플릿 코드 정리
  - baseline 코드 중 유용한 부분을 참고하여 프로젝트 탬플릿 코드 재구성
    - make_data.py 파일의 내용을 dataset.py 에 dataset 클래스로 통합
    - custom_model.py 에 각 모델별 클래스를 생성하여 config.json 에서 선택하여 사용할 수 있도록 구현
    - custom_loss.py 에 여러 가지 종류의 losss 함수를 정의하여 config.json 에서 loss 를 선택하여 사용할 수 있도록 구현

실험

- baseline 모델
  - **pretrain된 resnet18 모델을 불러와서 학습시킨 결과 test 데이터에 대한 f1 score 가 0.684 가 나왔다.**
- validation set 생성
  - validation set 생성 및 validation set 을 통한 평가 로직 추가
  - 모든 이미지에 대해 랜덤으로 샘플링하여 데이터 불균형이 더 심해져서 성능이 떨어지는 것 같다.
  - 사람 단위의 폴더 기준으로 validation set 을 나누도록 로직을 수정해야 할 것 같다.
  - validationo set 생성 시 유의 사항
    - validation set 폴더 단위를 기준으로 분리하기
    - 데이터가 한 사람 당 incorrect 1개, mask 5개, normal 1개로 이루어져 있다.
    - 이 전체 데이터를 대상으로 검증 세트를 임의로 분리하면 데이터 불균형이 더 심해진다.
    - 7개의 이미지를 포함하고 있는 폴더 단위로 검증 세트를 분리해야 원본 데이터와 같은 정도의 데이터 불균형을 유지할 수 있을 것 같다.
- train 과 inference 에 동일한 augmentation 적용
  - train 시 사용되는 augmentation 과 inference 시 사용되는 augmentation 을 동일하게 맞춰줬다.
  - 하지만 멘토링 질문을 통해 꼭 두 개의 augmentation 을 동일하게 맞춰줄 필요는 없다는 것을 알게 되었다.
- 60개 이상에 해당하는 label 에 가중치 부여
  - 데이터 분포를 확인해보면 나이대의 label 중 60세 이상에 해당하는 old label 의 데이터 개수가 현저히 적은 것을 확인할 수 있다.
  - loss 계산 시 60세 이상에 해당하는 label 에 가중치를 부여
  - **이 방법을 통해 test 데이터에 대한 f1 score 가 0.688 로 약간 개선되었다.**
- pretrained model 변경
  - baseline 에서 사용했던 ResNet18 대신 EfficientNet 을 사용하도록 변경
  - GPU 메모리 초과 오류가 발생하여 배치사이즈를 128에서 64로 줄여서 학습을 진행하였다.
  - resnet18 의 1 epoch 당 학습 속도가 2분 30초였던 것에 비해 efficientNet 은 3분 30초로 약간 증가했다.
  - **이 방법을 통해 test 데이터에 대한 f1 score 가 0.718 로 꽤 많은 성능 향상을 얻게 되었다.**
- ViT 모델 사용
  - 입력 이미지의 크기를 224x224 로 맞춰줘야 해서 augmentation 기법 중 Resize 사용
  - **적용 결과 오히려 efficientNet 보다 안 좋은 성능을 보였다.**
- LR Scheduler 사용
  - 종류
    - StepLR()
    - ReduceLROnPlateau()
  - **ReduceLROnPlateau 를 사용해 뒀지만 오히려 성능이 개선되지 않았다.**
- Gradient Accumulation 적용
  - 매번 배치가 끝날 때마다 gradient 를 업데이트하는 것이 아닌 특정 배치 횟수를 지정하여 gradient 를 누적하여 업데이트하도록 하는 기법
  - 그래프 상으로는 어느 정도 성능이 향상한 것으로 보임
  - 
  - **test 데이터에 대해서는 acc는 증가했지만 f1 score 는 감소하는 성능을 보였다.**
- augmentation 기법 적용
  - baseline 에 있는 여러 가지 종류의 augmentation 기법 적용
  - 
  - **적용 결과 단순히 ToTensor() 만 사용했을 때보다 안 좋은 성능을 보였다.**
- Stratified K-Fold Cross Validation 기법 적용
  - train.py 파일의 내용의 큰 틀을 수정해야 해서 별도의 train.py 를 만들어서 구현했다.
  - 기본적인 모델의 성능이 별로 안 좋은 상태에서 단순히 cross validation 만 하는 것은 성능 향상에 큰 효과가 없을 것 같다는 생각을 하게 됐다.
- 다양한 loss function 사용
  - 종류
    - Cross Entropy Loss
    - Focal Loss
    - Label Smoothing Loss
    - F1 Loss
  - **내가 loss 함수들의 특징을 제대로 이해하지 못한 채 사용해서 그런 지 별다른 성능 향상을 얻진 못했다.**
- cutmix 기법 적용
  - 종류
    - random cutmix
      - 기존 이미지에 붙일 새로운 이미지의 크기를 임의로 선택하여 cutmix 실시
    - quarter cutmix
      - 기존 이미지에 붙일 새로운 이미지의 크기를 이미지 크기의 1/4 로 제한하여 cutmix 실시
  - 기존의 배치 사이즈 64로 학습 시 GPU OOM 이 발생하여 배치 사이즈를 32로 감소시켜 주었다.
  - **random cutmix, quarter cutmix 모두 적용해봤지만 성능 향상은 미비했다.**
- soft voting ensemble
  - 7명의 팀원들 모두 각각 제일 좋은 성능을 나타낸 모델들을 soft voting 방식으로 ensemble 을 진행했다.
  - ensemble 을 하기 위해 예측 결과를 softmax 를 적용하여 출력값을 만든 후 soft voting 을 진행했다.
  - **이 방법을 통해 test 데이터에 대한 f1 score 가 0.753 로 꽤 많은 성능 향상을 얻게 되었다.**
  - 이를 통해 적절한 ensemble 을 통해 모델의 성능을 꽤 많이 끌어올릴 수 있다는 것을 깨닫게 되었다.