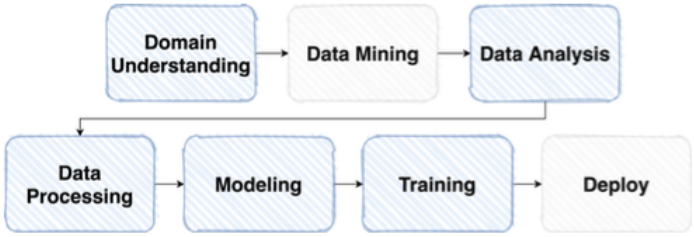
Wrap-up Report

부스트캠프 U-Stage가 끝나고 P-Stage 1st Competition으로 마스크 분류 대회를 마무리했다. 재미도 있고 실력도 정말 많이 향상된 것 같지만 아직 갈길이 멀다는 것을 느낀 대회였다. 이번 글은 개인적으로 대회를 하면서 느낀점과 기술적으로 어떤 기법들을 사용했는지 정리해보는 회고글이다.

실제 ML/DL 서비스를 만드는 Pipeline은 아래 그림과 같으나 Competition에서는 Dataset이 제공되고 Deploy하는 과정이 생략된다.



Problem Definition : domain understanding

멘토님들께서 Competition에서 가장 중요하다고 했던 Problem Definition으로 대회를 시작했다. 처음 Overview를 읽었을 때는 단순히 3개의 class를 분류하는 문제로 인식했다. `마스크를 쓴 사람`, `마스크를 안 쓴 사람`, `마스크를 이상하게 쓴 사람`

을 구별하는 거라 생각하고 쉽다고 생각했지만, 아니었다. 이번 대회는 총 18개의 Class로 분류하는 문제였다. 이미지를 보면 쉽게 이해할 수 있다.

Class Description:

마스크 착용여부, 성별, 나이를 기준으로 총 18개의 클래스가 있습니다.

Class 1	Mask	Gender	Age
0	Wear	Male	< 30
1	Wear	Male	>= 30 and < 60
2	Wear	Male	>= 60
3	Wear	Female	< 30
4	Wear	Female	>= 30 and < 60
5	Wear	Female	>= 60
6	Incorrect	Male	< 30
7	Incorrect	Male	>= 30 and < 60
8	Incorrect	Male	>= 60
9	Incorrect	Female	< 30
10	Incorrect	Female	>= 30 and < 60
11	Incorrect	Female	>= 60
12	Not Wear	Male	< 30
13	Not Wear	Male	>= 30 and < 60
14	Not Wear	Male	>= 60
15	Not Wear	Female	< 30
16	Not Wear	Female	>= 30 and < 60
17	Not Wear	Female	>= 60

여기서 2가지 방법론을 떠올려 볼 수 있었다. 하나의 모델이 18개의 클래스를 모두 분류하는 모델 or 3개의 모델 (마스크 분류 모델, 나이 예측 모델, 성별 예측 모델) 을 사용하는 방법이 있었다. 당시 후자는 생각을 못해서 전자로 방향을 잡고 진행했다.

Data Analysis

흔히 EDA (Exploratory Data Analysis)는 데이터를 분석하여 필요한 데이터와 필요하지 않은 데이터를 확인, 데이터가 존재하지 않거나 오류가 있는 데이터를 걸러내는 등 전체적인 데이터를 분석하는 일이다. 일반적으로 Text 정형 데이터는 분석할게 꽤 있지만 이미지 데이터는 어떤것을 분석해야 할지 몰랐다. 멘토님께서 분석하신 내용을 보고 나서야 이미지 데이터는 이렇게 하는구나. 라는 감이 잡혔다.

멘토님께서는 아래 내용을 EDA 하셨다.

1. 이미지 RGB 정보, 사이즈
2. Y 값 독립적 분포 확인

1번은 RGB 정보의 mean, std를 구해서 dataset `Normalize` 에 활용하셨다. 2번은 Gender와 Age의 분포를 확인함으로써 남성, 여성이 몇 명씩 존재하는지와 나이의 분포를 시각적으로 확인해볼 수 있었다.

Data Processing

모델에 집어넣을 데이터를 가공하는 단계이다. 이 단계에서 전체 소요시간의 40% 정도를 소모한 것 같다. 하지만 가장 후회되는 것이 Data Processing 단계를 가볍게 여겼다는 것이다. **이번 대회에서 나는 어떤 모델을 사용할 것인지를 메인으로 두고 진행했는데 제대로 된 데이터가 들어오지 않으면 아무리 좋은 모델도 제대로 학습할 수 없다는 것을 알게 되었다**. 어찌보면 너무 당연한 것이지만 간과했던 것이 이번 대회에서 가장 실수했던 부분이 아닌가 싶다.

처음 데이터를 접하고 당황했던 것은 일반적으로 train/test 로 폴더링이 되어있는데 이번 대회의 데이터는 아래 형태로 폴더링이 되어있었다.

```
|-- train
    |-- data
        |-- eval
        |-- train
            |-- train.csv
            |-- images
                |-- 000001_female_Asian_45
                |-- 000002_female_Asian_25
                |-- 000003_male_Asian_60
                    |-- 'mask1.jpg'
                    |-- 'mask2.jpg'
                    |-- 'mask3.jpg'
                    |-- 'mask4.jpg'
                    |-- 'mask5.jpg'
                    |-- 'incorrect.jpg'
                    |-- 'normal.jpg'
```

- **처음 작성했던 코드의 오류**
  - image들을 하나하나 꺼내서 데이터셋을 만듬
  - 이렇게 만든 데이터로 train을 하면 validation에서 같은 사람의 다른 사진이 나옴. 결과적으로 validation 단계에서 이미 봤던 사람을 또 보기 때문에 val-score가 꽤 높게 나옴. 하지만 실제 leader board에 제출하면 새로운 사람이 등장하기 때문에 f1-score가 심하게 낮게 나옴.
- **개선한 코드**
  - 데이터를 사람별로 불러옴
  - 000001 ~ 012345 까지의 사람을 train에 쓰고 그 이후부터 val에 쓴다고 가정하면 validation에서는 새로운 사람을 보고 평가하기 때문에 Leader Board와 유사하게 구성할 수 있었다.

Augmentation

데이터를 사람별로 불러오고나서 Augmentation을 적용했다. 얼굴 부분을 위주로 모델이 학습해야하기 때문에 `CenterCrop` 기법을 사용했다. **원본 이미지에서 가운데 얼굴만 있는 부분을 자르면 학습에 용이할 것이라 생각했는데 실제로 성능향상에 도움이 됐다.**

Tensor 연산을 할 수 있게 `ToTensor` 를 사용했고 `Normalize` 를 사용해서 feature 분포를 고르게 해주고 gradient 를 update하는 과정이 원활하게 진행되도록 해주었다.

`Gaussian Noise` 를 씌우는 방법은 효과가 있는지 없는지 잘 모르겠다.

Modeling

데이터를 학습시킬 모델을 정의하는 단계이다. 최신 image classification 모델들이 있는 `timm` 에서 `efficientnet` 을 가져와서 사용했다. 이 단계에서도 많은 시간을 소모했는데 그럴 필요가 있었나 싶다. Acc를 테스트해본 모델들을 기록해두지 않아 했던 모델을 또 하고 또 했던 것 같다. 시간낭비..

다음 Competition부터는 노션이나 엑셀 테이블을 사용해서 어떤 모델들을 동일한 조건으로 테스트했는지 기록을 꼼꼼하게 해둘 것이다. 이것도 했다 저것도 했다 하니 Backbone을 정하는 데에만 3일은 소요된 것 같다. **5개 정도의 모델을 돌려보면서 `Backbone` 으로 사용할 모델을 구하고 그 이후에 Layer을 추가하거나 Dropout기법 등 튜닝을 하는 것이 효과적일 것이라고 깨달았다.**

Training

모델링과 마찬가지로 쓸 데 없이 시간을 많이 소모했다. 10번이 넘는 테스트에서 `confusion matrix`를 살펴보면 `60대`를 잘 구별해내지 못하는 것을 찾았다. 처음에 고려했던 문제점은 `overfitting`이 되어서 잘 구별을 못하나? 였다. 따라서 overfitting을 해소하기위해 마지막 레이어에 `Dropout`을 추가해줘서 overfitting을 해소하고자 했다. 약간의 성능 상승이 있었지만 모델 문제인가 하고 모델을 계속 바꿨었는데 결과적으로 시간만 낭비했었다.

한 번에 하나만 하자

CenterCrop도 넣고, Gaussian Noise 씌우고, Learning Rate을 바꾸는 등 여러가지를 한 번에 적용하여 모델을 학습하면 어떤 Hyper-parameter나 Augmentation 기법이 효과적인지 알 수 가 없었다. 한 번에 하나씩 하면서 정리해서 알아보기 쉽게 했다면 논리적으로 결과를 도출할 수 있었을 것 같다.

앞으로는 2~3개의 Backbone 모델을 확실하게 정하고, 데이터를 모델이 잘 학습할 수 있도록 전처리 후, Hyper-parameter들을 수정해가면서 성능 향상을 측정해야겠다.

Learning Rate Scheduler

Scheduler의 효과에 대해 알게 되었다. Scheduler을 사용하지 않을 시 training accuracy를 93~4% 까지 밖에 올리지 못했는데 scheduler을 사용해서 점점 learning rate을 낮춰주니 99~100%까지 training 성능을 올릴 수 있었고 validation에서도 성능을 올릴 수 있었다. 이번엔 StepLR을 사용했는데 다양한 스케줄러를 테스트해보지 못한 것이 아쉽다

TTA (Test Time Augmentation)

성능을 많이 올려줬던 방법이다. 학습시에 이미지 가운데, 즉 사람 얼굴만 잘라서 학습을 해서 성능이 좋았는데 테스트시 이미지를 자르지 않고 테스트를 진행했었다. 당연히 성능이 잘 나올수가 없었다. 이미지 사이즈만 같지 Augmenation을 적용하지 않은 이미지는 feature가 다를 수 밖에 없는데 이 것을 대회 마지막에 알아차리게 되었다. Test time Augmentation을 적용해서 f1-score를 상당히 끌어 올릴 수 있었다.

Ensemble

앙상블, 학습된 여러 모델을 사용하여 제출할 output.csv를 생성했다. 쉽게 말해 A, B, C 라는 모델이 있을 때, 세 모델에서 동시에 output을 구해서 그 값들의 평균을 사용하는 것이다. 생각보다 성능을 많이 올릴 수 있었다. 3개의 모델보다 5개 정도의 모델을 앙상블하면 더 좋은 결과를 얻을 수 있었을 것 같다.

Summary

- 주피터 노트북 형태를 벗어나서 `Python` 파일로 작업할 수 있게 되었다.
- `Multi-Class Classification`에 관해 알게 됨.
- `os` 라이브러리를 사용해서 `directory` and `file` 접근에 관해 알게 됨.
- `Data Processiong` 에 보다 많은 시간을 쏟아서 잘 학습할 수 있는 데이터를 구축하자.
- `K-Fold Cross Validation` 해보기!
- 한 번의 하나의 `Augmentation` 이나 `Hyper-Parameter` 를 수정하면서 성능에 미치는 영향을 기록하자.