

Image Classification Wrap-up Report

CV-2조(화이팅 해야조)

최현우, 안세희, 이동우, 전진하, 조명현

1. 프로젝트 개요

1-1. 프로젝트 주제

- 지난 3년간 COVID-19는 전 세계적으로 경제와 생산 활동에 큰 제약을 가져왔습니다. 이 바이러스는 주로 입과 호흡기에서 나오는 비말을 통해 전파되므로, 공공장소에서의 올바른 마스크 착용이 중요합니다. 공공장소에서 모든 사람의 마스크 착용 상태를 확인하는 것은 인력적 제약이 있으므로, 카메라를 통해 사람의 얼굴 이미지만으로 마스크 착용 여부를 자동으로 판별할 수 있는 시스템의 개발이 필요합니다.

1-2. 환경

- 컴퓨팅 환경 - 인당 1개의 V100 서버를 VS Code와 SSH로 연결하여 사용
- 협업 환경 - GitHub, Wandb
- 의사 소통 - Zoom, Slack
- 프로젝트 폴더 구조

Dataset

```
└─ datasets
  └─ eval
    └─ images
    └─ info.csv
  └─ train
    └─ images
    └─ train.csv
  └─ trainQA
    └─ images
    └─ train.csv
```

Project

```
└─ level1-imageclassification-cv-02
  └─ __pycache__
  └─ .git
  └─ .github
    └─ ISSUE_TEMPLATE
      └─ bug_report.md
      └─ feature_request.md
    └─ .keep
    └─ PULL_REQUEST_TEMPLATE.md
  └─ notebook
    └─ sample_submission.ipynb
  └─ output
  └─ redis_model_scheduler
    └─ redis_publisher.py
    └─ schedule_search.py
  └─ .gitignore
```

```

├── dataset.py
├── function.py
├── inference.py
├── k-fold-inference.py
├── loss.py
├── model.py
├── requirements.txt
└── train.py

```

2. 프로젝트 팀 구성 및 역할

전체	Wrap up report 작성, EDA, 모델 성능 실험
최현우	Redis Queue 기능 구현, wandb 실험 관리 적용, 학습 정보 로깅, 학습 소스 수정
안세희	Redis Queue 기능 구현, 모델 학습 코드 수정
이동우	모델 research 및 debugging
전진하	모델 research 및 debugging
조명현	Augmentation 성능 실험, Dataset Labeling 수정 및 테스트, Redis Queue 기능 구현

3. 프로젝트 수행 절차 및 방법

3-1. 프로젝트 일정

- 1주차 : 강의 전부 듣기, Redis를 사용한 메세지 큐 기반 모델 학습 서버 구현, 모델 리서치
- 2주차 : 모델 리서치, 모델 성능 실험, 데이터 갈래의 접근
- Timeline

12월

10	11	12	13	14	15	16
	Redis 모델 학습 서버 구축					
	강의 듣기 및 미션 수행					
		EDA				
					모델 성능 실험	
17	18	19	20	21	22	23
강의 듣기 ...	Redis 학습 서버 기능 수정			Wrap up report 작성		
모델 성능 실험						
		데이터 라벨 수정 및 Augmentation 실험				
24	25	26	27	28	29	30
Wrap up report 작성						

3-2. 프로젝트 기획

(1) Project Rule









- Zoom을 통해 팀 단위로 역할 분담 및 학습 모델 관련 변수 설정
- 학습 Model Run 전/후로 Configuration 및 결과 기록
- Git Convention - 유다시티

(2) Project 절차

1. 프로젝트 개발환경 구축(Github, Slack, WandB, Server, Redis)
2. EDA를 통해 Data 구조 파악 및 실험 진행방향 설정
3. 모델을 각각 나누어 모델 성능 실험 및 모델 선정
4. Data Imbalance 및 Miss labeling 보안을 위한 Data Augmentation & Labeling 수정
5. 리더보드 결과 상위 3개 Model 기반으로 성능 실험

4. 프로젝트 수행 결과

Public : 4등 / Private : 8등

4	CV_02조	   	0.7497	78.8413	54	1h
8 (4 ▾)	CV_02조	   	0.7300	78.1429	54	7h

저희 팀은 탐색적 데이터 분석(EDA) 결과를 기반으로 데이터 라벨링과 데이터 불균형 문제가 모델링 결과에 중요한 영향을 미친다고 판단했습니다. 이를 검증하기 위해 데이터 증강(Data Augmentation)과 라벨 수정 작업을 진행했으며, 이 과정과 결과를 W&B 로그를 통해 기록하고 검증했습니다.

데이터 증강의 경우, 다양한 Data Augmentation 방법을 ResNet50에 적용한 결과 Base Augmentation보다 좋은 성능을 보이진 못했고, 최종 선택한 모델에서 과적합 방지를 위해 성능은 떨어졌지만 복잡한 Augmentation이었던 ImgAugAugmentation 적용 결과 더 좋은 결과를 보여줬습니다.

한편, 데이터 라벨 수정 작업은 모델의 성능을 소폭 향상시키는 데 기여했습니다. 이는 라벨링의 정확성이 모델 성능에 중요한 요소임을 시사합니다.

이번 프로젝트를 통해, 특히 모델 선정과 이에 맞는 파라미터 튜닝의 중요성을 깊이 인식했습니다. 데이터 라벨링과 불균형 문제에 대한 접근이 일부 성공적이었음에도 불구하고, 최적의 모델과 파라미터를 결정하는 것이 전체적인 모델 성능에 결정적인 영향을 미친다는 점을 확인할 수 있었습니다. 이러한 경험은 향후 유사한 프로젝트를 진행할 때 중요한 교훈으로 작용할 것입니다.

4.1 Task Definition and Data analysis

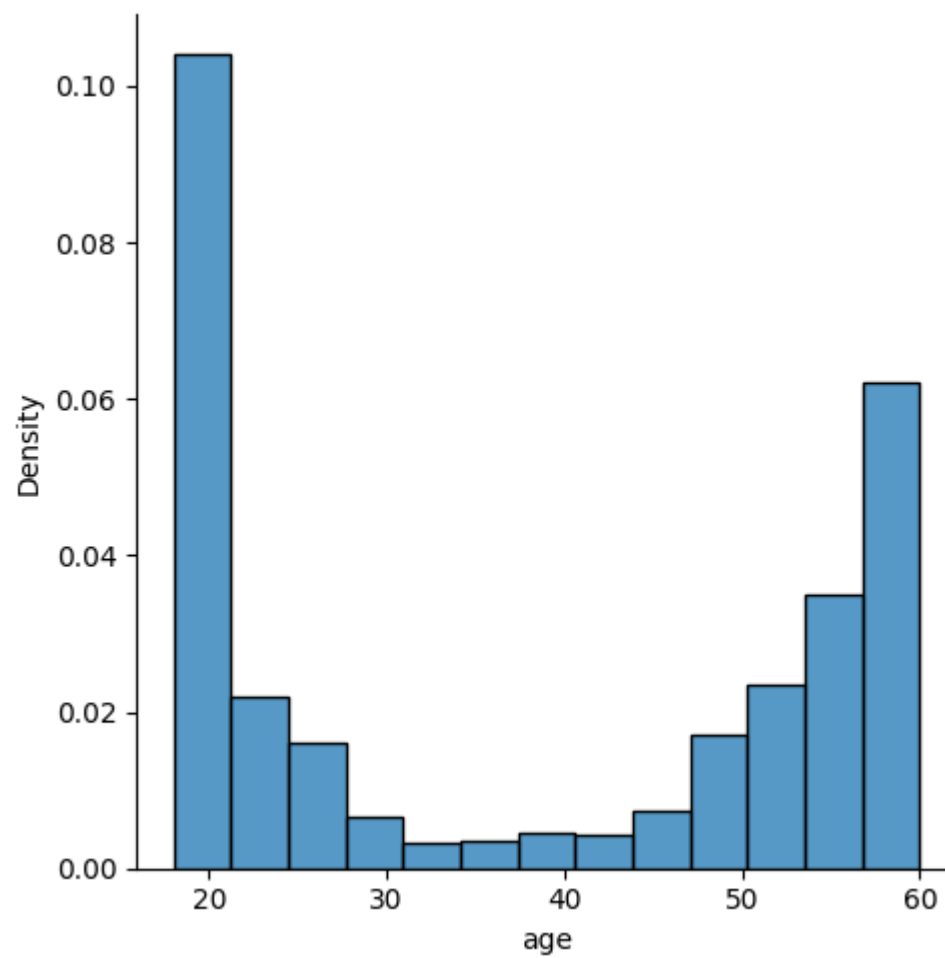
(1) 가설 설정

1. Data Labeling에 miss labeling과 데이터 불균형이 있기 때문에 이를 보완해준다면 좋은 결과를 기대할 수 있다.
2. Loss의 경우 Accuracy에 집중하는 CrossEntropy 보다 대회의 타겟 값인 f1 score에 집중하는 f1loss를 사용한다면 좋은 결과를 기대할 수 있다.
3. Augmentation을 통해 과적합을 예방하여 학습한다면 좋은 결과를 기대할 수 있다.
4. Stratified K-Fold 교차 검증을 이용하면 각 폴드에서 클래스 간 데이터 분포의 균일성을 보장하고, 전체 데이터셋에 대한 검증을 수행하기 때문에 모델의 일반화 성능 향상을 기대할 수 있다.

(2) 가설 검증 결과

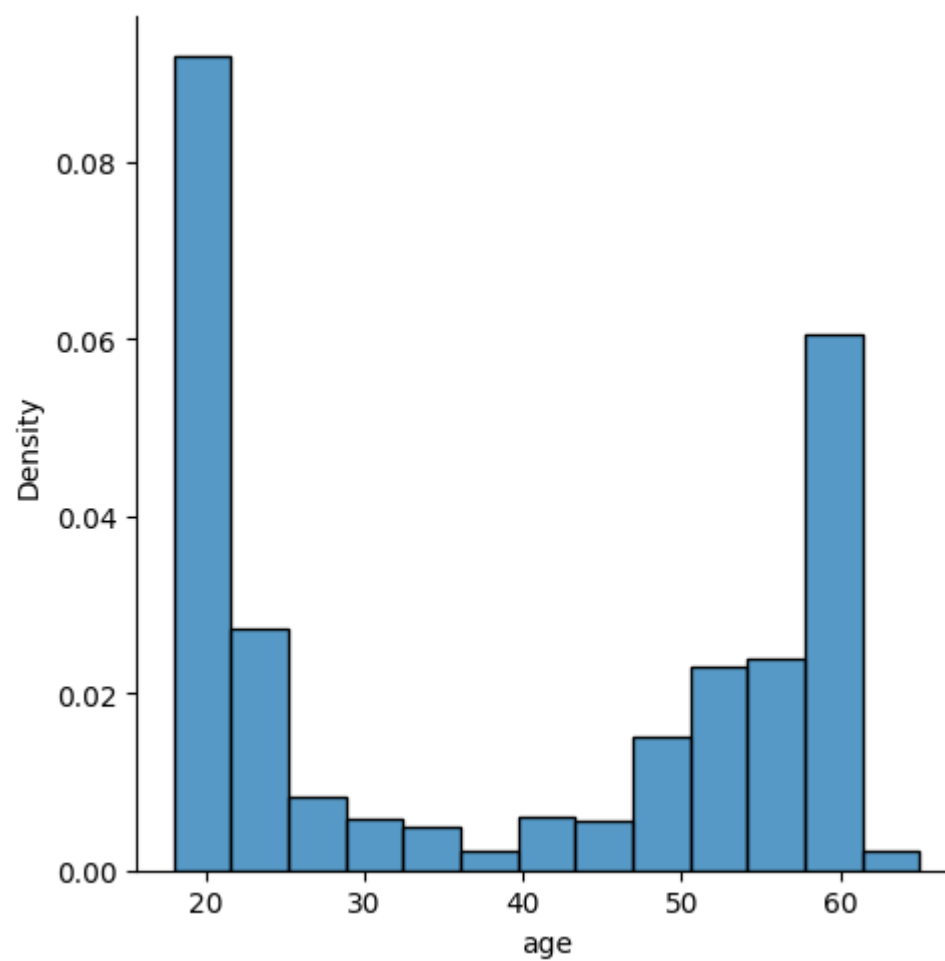
1. Re-Labeling Dataset에 EfficientNet-b2 모델 학습 결과

<기존 Dataset, EfficientNet-b2>



36	Finished		0.7278	75.6508	상세 보기	2023-12-20 02:22			
----	----------	--	--------	---------	----------	------------------	--	--	--

<Re-Labeling Dataset, EfficientNet-b2>



41	Finished		0.7346	76.2857	상세 보기	2023-12-20 16:06			
----	----------	--	--------	---------	----------	------------------	--	--	--

- f1 score : 0.7278 → 0.7346 / Accuracy : 75.6508 → 76.2857




- Miss Labeling과 Data Imbalance를 보완하기 위한 Dataset Re-Labeling은 기존 Dataset 보다 좋은 결과를 보인다.

2. EfficientNet-b2 모델에서의 Loss function 변화에 대한 학습 결과



<Focal Loss>

45	Finished		0.7029	74.3492	상세 보기	2023-12-20 20:40		<input type="checkbox"/>	
----	----------	---	--------	---------	----------	------------------	---	--------------------------	---

<Label Smoothing Loss>

43	Finished		0.7050	75.0635	상세 보기	2023-12-20 20:17		<input type="checkbox"/>	
----	----------	---	--------	---------	----------	------------------	---	--------------------------	---


<F1 Loss>

41	Finished		0.7346	76.2857	상세 보기	2023-12-20 16:06		<input checked="" type="checkbox"/>	
----	----------	---	--------	---------	----------	------------------	---	-------------------------------------	---

- f1 score : 0.7346 / Accuracy : 76.2857
- F1 loss를 사용한 결과가 다른 loss function과 비교해서 가장 성능이 좋았다.

3. EfficientNet-b2 모델에서의 Augmentation 적용에 대한 학습 결과




<base augmentation>

38	Finished		0.7119	75.0317	상세 보기	2023-12-20 14:18		<input type="checkbox"/>	
----	----------	---	--------	---------	----------	------------------	---	--------------------------	---

<imgaug augmentation>

```
class ImgaugAugmentation:
    def __init__(self, resize, mean, std):
        self.imgaug_transform= iaa.Sequential([
            iaa.Affine(rotate=(-5, 5)), # 무작위 회전 (-5도부터 5도까지)
            iaa.Fliplr(0.5), # 50% 확률로 좌우 반전
            iaa.GaussianBlur(sigma=(0, 3.0)), # 가우시안 블러 적용
            iaa.AdditiveGaussianNoise(scale=(0, 0.1 * 255)), # 가우시안 노이즈 추가
            iaa.Resize({"height": resize[0], "width": resize[1]})
        ])





        self.torch_transform = Compose([
            ColorJitter(0.1, 0.1, 0.1, 0.1),
            ToTensor(),
            Normalize(mean=mean, std=std),
        ])
```

36	Finished		0.7278	75.6508	상세 보기	2023-12-20 02:22		<input type="checkbox"/>	
----	----------	---	--------	---------	----------	------------------	---	--------------------------	---





- f1 score: 0.7119 → 0.7278 / Accuracy: 75.0317 → 75.6508
- imgaug Augmentation을 적용했을 때 별도의 augmentation을 적용하지 않았을 때보다 더 성능이 향상되었다.

4. k-fold 교차 검증

<k-fold 미적용>

41	Finished		0.7346	76.2857	상세 보기	2023-12-20 16:06			
----	----------	---	--------	---------	----------	------------------	---	---	---

<k-fold 적용>

50	Finished		0.7280	77.0000	상세 보기	2023-12-21 15:29			
----	----------	---	--------	---------	----------	------------------	---	---	---

- f1 score : 0.7346 → 0.7280 / Accuracy : 76.2857 → 77.0000
- 기존 가장 높은 성능을 보인 모델과 동일한 조건에 k-fold(k=5) 적용 결과 가설과 달리 Accuracy는 좋아졌지만 f1 score의 결과는 떨어졌다.

(3)모델개요

Resnet

ResNet의 핵심 아이디어는 스킵 연결(skip connection) 또는 잔차 연결(residual connection)을 도입하여 네트워크의 깊이가 증가함에 따라 발생하는 그래디언트 소실 문제를 해결하는 것입니다. 깊은 신경망에서는 정보가 전달되는 동안 그래디언트가 작아져 학습이 어려워지는데, ResNet은 이를 스킵 연결을 통해 우회함으로써 이 문제를 완화합니다. ResNet의 주요 특징은 다음과 같습니다:

1. **Residual Block**: 입력과 출력 간의 잔차를 학습하는 기본 블록입니다.
2. **Bottleneck Architecture**: ResNet은 병목 구조를 사용하여 계산 비용을 줄이고 효율성을 높입니다.
3. **Skip Connection (Shortcut Connection)**: 입력을 출력에 직접 연결하여 그래디언트의 소실 문제를 해결합니다.
4. **깊은 네트워크 학습 가능**: ResNet은 수백 개의 레이어로 이루어진 매우 깊은 네트워크를 효과적으로 학습할 수 있습니다.

선택 이유 : CNN기반의 깊은 네트워크 학습을 통해서 이미지의 특징을 파악하고 상대적으로 어려운 나이 예측을 잘 할 것이라고 생각했습니다.

ViT

기존의 CNN(Convolutional Neural Network) 기반의 아키텍처와는 다르게, ViT는 어텐션 메커니즘을 사용하여 이미지를 처리합니다. ViT의 주요 특징은 다음과 같습니다:

1. **어텐션 메커니즘 사용**: 이미지를 나타내는 패치들에 대한 어텐션 메커니즘을 사용하여 전역적인 정보를 캡처합니다.
2. **패치 임베딩**: 이미지는 격자 모양의 패치로 나뉘어 각 패치는 임베딩 과정을 거쳐 특정 차원의 벡터로 변환됩니다. 이렇게 임베딩된 패치들이 트랜스포머의 입력으로 사용됩니다.
3. **프리트레이닝 없는 전이학습(Pre-training-free Transfer Learning)**: ViT는 대규모 이미지 데이터셋에서 사전 훈련된 CNN과 달리, 대량의 데이터를 필요로 하지 않고 적은 양의 레이블된 데이터에서도 좋은 성능을 보일 수 있습니다.
4. **큰 모델 깊이(Depth)**: ViT는 깊은 트랜스포머 아키텍처를 사용하여 대규모 모델을 만들 수 있습니다. 이는 많은 파라미터를 학습하면서도 학습 데이터 부족 문제를 어느 정도 극복할 수 있게 해줍니다.

선택 이유 : 학습 데이터 부족 문제 극복을 기대하며 ViT 모델을 선택해 실험에 적용했습니다.

Efficient-Net

EfficientNet은 모델의 크기를 조절함으로써 계산 비용을 최소화하면서 성능을 향상시킬 수 있는 아이디어에 기반하고 있습니다. EfficientNet의 특징은 다음과 같습니다:

1. **Compound Scaling**: EfficientNet은 모델의 세 가지 핵심 구성 요소(scale, depth, width)를 동시에 증가시킴으로써 성능을 향상시킵니다. 이러한 접근 방식은 네트워크의 크기를 조절하여 더 작은 모델에서 더 큰 모델로 쉽게 확장할 수 있도록 합니다.
2. **Bilinear Interpolation**: 이미지 크기를 조절하는 데 Bilinear Interpolation을 사용하여 성능 손실을 최소화합니다.
3. **Depthwise Separable Convolution**: 효율적인 계산을 위해 Depthwise Separable Convolution을 도입합니다. 이는 일반적인 Convolution 연산에 비해 적은 파라미터와 계산 비용을 가집니다.

4. **Swish Activation Function:** ReLU(Rectified Linear Unit) 대신 Swish 활성화 함수를 사용하여 더 부드러운 그래디언트 흐름을 유지하고 모델의 성능을 향상시킵니다.
5. **Feature Pyramid Network (FPN) 추가:** 다양한 스케일의 특징을 결합하여 객체 검출과 같은 작업에서 유용한 정보를 뽑아내기 위해 FPN이 추가되었습니다.

EfficientNet은 작은 모델부터 큰 모델까지 다양한 크기의 버전으로 제공되며, 이를 통해 사용자는 자신의 환경에 맞게 적절한 모델을 선택할 수 있습니다.

선정 이유 : 크기에 따른 다양한 모델을 테스트 해보며 저희 데이터셋에 적합한 크기의 모델을 선택할 수 있고, 효율적인 계산을 통해 더 짧은 시간 동안 많은 실험을 진행할 수 있을 것이라고 생각했습니다.

5. 자체 평가 의견

잘했던 점

- 줌과 슬랙을 통해 팀원 간의 원활한 소통으로 역할 분담과 실험 진행이 원활했다.
- 데이터의 라벨 불균형 문제를 파악하고 해결하려 노력했고, 실험에도 반영을 잘하였다.
- 모델 scheduler를 개발하여 효율적인 실험을 진행했다.

시도 했으나 잘 되지 않았던 것들

- K-fold를 활용해 모델의 성능 향상을 기대했으나, 결과가 예상과 다르게 성능 향상을 하지 못하였다.
- ViT 모델이 이미지의 위치 정보를 활용하여 이번 Task에서 좋은 성능을 낼 것이라 생각하였으나 실제로 성능이 다른 모델들에 비해 떨어졌다.
- 같은 조건에서 하이퍼파라미터를 하나씩 테스트하며 가장 성능이 좋은 파라미터를 골라 한번에 적용했으나 파라미터들 간의 조합에 따라 성능이 달라져 생각만큼 성능을 개선하지 못했다.
- 여러 loss를 적용해보았지만 각각의 loss들이 모델, epoch 등에 따라 내는 결과값이 다르고, loss 값의 스케일도 서로 달라 정확한 비교를 하는 것이 힘들었다.

아쉬웠던 점들

- 모델 성능 실험 과정에서 epoch이라는 변수에 대해 뒤늦게 고려한 점이 아쉬웠다.
- 초반 인프라 구축에 시간을 많이 들여 실험 횟수가 상대적으로 부족했다.
- 더 다양한 back bone모델을 시도해보지 못한 점이 아쉬웠다.
- competition이 끝나갈때쯤 K-fold, 앙상블 등 시간이 오래걸리는 실험들을 하게되어서 시간이 부족했습니다.
- 다양한 args를 추가하여 모델 실행을 더 편리하게 만들지 못했던 점이 뒤늦게 아쉬웠다.

프로젝트를 통해 배운 점

- Git, Github를 통한 협업 과정을 배웠다.
- WandB를 이용해 실험 관리하는 법에 대해 배웠다.
- Miss labeling과 Imbalance 인 데이터셋을 경험해볼수 있었습니다.
- 모델 학습 지표들을 보고 경향성을 읽고, 비교하는 법을 배웠습니다.