(1강) Competition with AI Stages!

Overview (문제 개요)

문제 개요를 읽고 할 수 있는 것

- 문제 정의의 구성
    - 주의 : 문제 정의 시 데이터를 통해 문제를 풀 수 있나? 고려할 것
    1. 문제 발견
    2. 문제의 입출력 확인
    3. 솔루션 적용 대상 및 적용 방법 고려
- 방향성 제시

Data Description

파일 형태, 메타 필드 소개 및 설명을 하는 데이터 스펙 요약본

Discussion

높은 등수보다 문제 해결을 목표로 대회를 참여하는 것이 좋다.

머신러닝 파이프 라인 순서

1. 도메인 이해 : 개요 읽기
2. 데이터 마이닝 : 데이터 수집
3. 데이터 분석
4. 데이터 전치리
5. 모델링
6. 학습
7. 배포

대회의 영역

1. 도메인 이해
2. 데이터 분석
3. 데이터 전치리
4. 모델링

5. 학습

마스크 착용 상태 분류 : 문제 정의

1. 문제 발견

> COVID-19는 치사율은 낮지만 전염력이 높다. 올바른 마스크 착용으로 감염 전파를 줄일 수 있어 마스크 착용 상태를 검사하는 것은 COVID-19로 인한 피해를 줄일 수 있다. 그러나 공공장소는 많은 사람이 밀집되어 마스크 착용 상태를 사람이 검사하면 많은 인적자원이 든다. 만약 사람의 얼굴 이미지만으로 마스크 착용 상태를 검사할 수 있다면 적은 인적자원 비용으로 COVID-19의 감염자를 줄일 수 있을 것이다.

2. 문제의 입출력 확인
    - 입력 : 마스크 착용/미착용/부분 착용 이미지
    - 출력 : 마스크 착용/미착용/부분 착용 여부, 성별, 나이를 구분한 클래스
3. 솔루션 적용 대상 및 적용 방법
    - 적용 대상 : 공공장소 이용객
    - 적용 방법 : 공공장소 입구에 솔루션 배치
4. 추가 활용
    - 밀집도에 따라 적절한 등급의 마스크 착용 여부를 확인하는 솔루션

(2강) Image Classification & EDA

데이터 분석

탐색적 데이터 분석(Exploratory Data Analysis) : 데이터를 이해하기 위한 노력

EDA는 아이디어가 새롭게 떠올라서 머신러닝 파이프 라인에서 무한 반복하는 작업이다.

마스크 착용 상태 분류 : EDA 대상

1. 중복 이미지 확인

> 1. 응용 프로그램 활용 : VisPics
> 2. 파이썬 활용 : https://towardsdatascience.com/finding-duplicate-images-with-pyth

2. 라벨과 이미지 일치 여부
3. 클래스 분포 확인
4. 이미지 사이즈 확인

5. 입력과 출력 분석 및 관계 확인

이미지 분류 태스크

이미지

정의 : 시각적 인식을 표현한 인공물

자료형 : unsigned int 8bit(숫자 범위 : 0 ~ 255)

(3강) Dataset

데이터 사이언스 작업의 구성 = 전처리 80% + 모델링 20%

- 로그 데이터 : 노이즈 많음
- 이미지 데이터 : 용량이 큼, 이미지 변화 필요

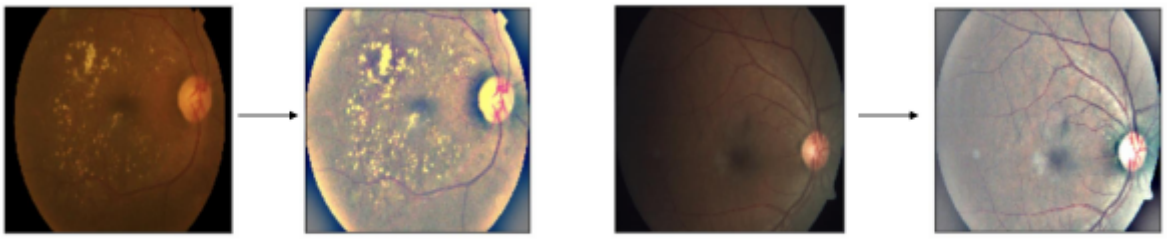
Pre-processing

Bounding box : 이미지의 메타 데이터, 테두리 제외는 노이즈

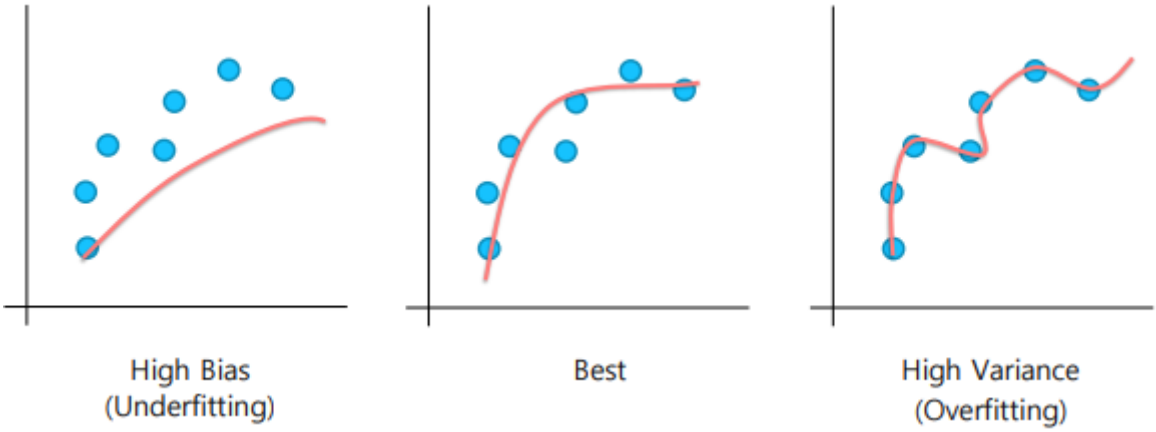
Resize : 빠른 계산을 위해 적당한 크기로 사이즈 변경 필요

도메인, 데이터 형식에 따라 다양한 케이스가 존재

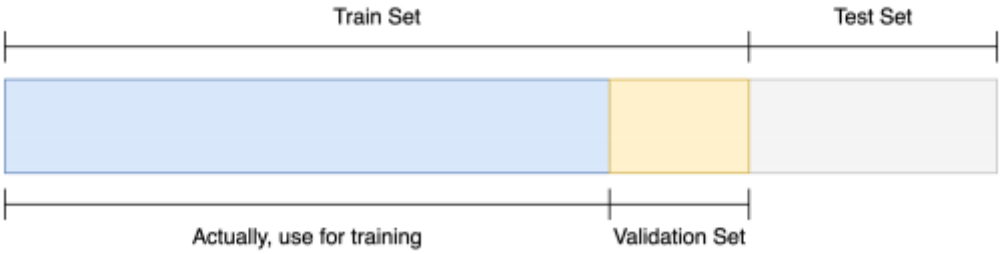
- 의료 데이터의 경우 전처리로 모델 학습과 문제 해결에 도움이 된다.
    - APTOS Blindness Detection : 검진 기계의 성능이 좋지 않아 이미지의 특징점을 구분하기 어려운 경우 이를 전처리하면 모델의 일반화에 좋은 영향을 미칠 수 있다.



Generalization



일반화 방법



Train / Validation : 훈련 데이터 중 일정 부분을 분리하고 검증 데이터셋으로 활용, 검증 데이터셋을 변경하는 방법도 있음

Data Augmentation : 문제가 만들어진 배경과 모델의 용도를 고려하면 노이즈에 관한 힌트를 얻을 수 있다. 힌트를 활용해 데이터의 경우와 상태를 다양하게 만든다.

- torchvision.transforms : 데이터 도메인을 생각해서 이미지 변경할 것
- Albumentations : torchvision.transforms보다 더 빠르고 다양하다.

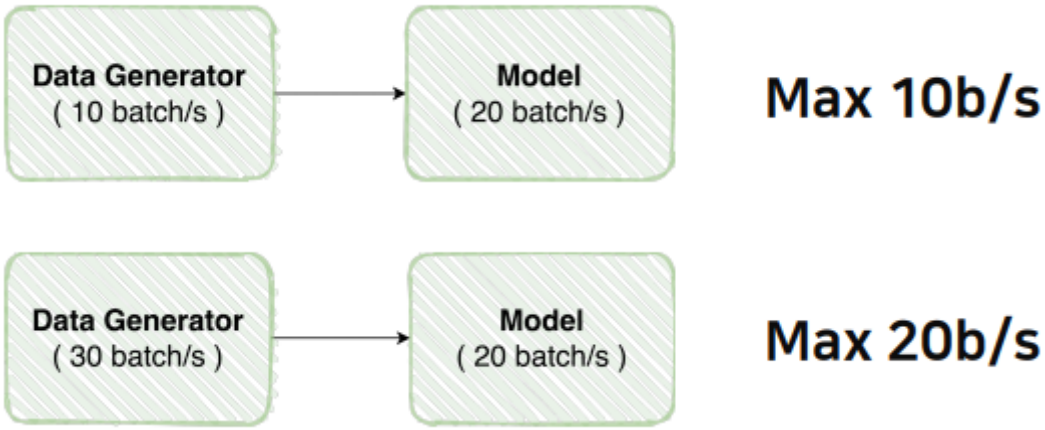
위 방법들이 일반화의 해결 방법이 아니다. 문제를 고찰하고 어떤 방법을 적용하면 데이터의 다양성을 늘릴 수 있는지 가정하고 **실험으로 증명**해야 한다.

(4강) Data Generation

Data Generation을 잘하면 GPU 사용 효율을 높이기도 함

커스텀 데이터셋 예제 : https://www.boostcourse.org/boostcampaitech2/lecture/1079694?isDesc=false

Data Feeding



데이터 생성기와 모델의 배치 차이 맞추는 것이 좋다.

데이터셋 생성 속도는 이미지 사이즈가 클수록 transform이 늘어날 수록 줄어든다.

torch.utils.data

Dataset

데이터 셋 정의 : 바닐라 데이터를 원하는 모델의 입력 형태로 변환, 치환하는 클래스

[ ] ↳ 숨겨진 셀 4개

DataLoader

데이터 로더 정의 : 내가 만든 데이터 셋을 효율적으로 사용 가능하게 기능 추가하는 것, 좋은 CPU일 수록 데이터 로더 속도 빨라짐

```
from torch.utils.data import DataLoader

train_loder = DataLoader(train_set, batch_size=batch_size, num_workers=num_workers, drop_last=Tru
```

(5강) Model 1

모델 정의 : 시스템 정보의 표현

모델링 정의 : 시스템의 모형을 설계

파이토치 vs 케라스

- 파이토치 : 어렵지만 수정이 쉽다.
- 케라스 : 쉽지만 수정이 어렵다.

파이토치의 구성

- 파이토치 모델의 모든 레이어는 nn.Module 클래스를 따른다.

코드 실습

[ ] ↳ 숨겨진 셀 *11*개

(6강) Model 2

컴퓨터 비전의 발전 토대 : ImageNet 데이터셋 덕분

알고리즘 개발 및 검증에는 높은 품질의 데이터 셋은 필수적이다.

ImageNet은 스탠포드 대학에서 2006년부터 축적된 데이터셋이다. 2011년부터 ImageNet 챌린지 덕분에 성능이 좋은 컴퓨터 비전 알고리즘이 개발되었다.

ImageNet의 구성

- 이미지 개수 : 1400만 개
- 카테고리 개수 : 2만 개

Pretrained Model의 배경

- 모델 일반화를 위해 매번 학습시키는 것은 비효율적이다.
- 대신 고품질, 대용량 데이터로 미리 학습한 모델을 목적에 맞게 다듬어서 사용하자.

torchvision.models, timm

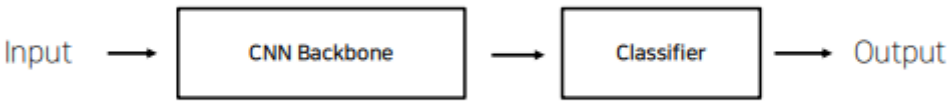
- 쉽게 모델 구조와 Pretrained 가중치를 다운로드 가능하다.
- 토치비전 모델 : https://pytorch.org/vision/stable/models.html
- timm : https://github.com/rwightman/pytorch-image-models#models

```
import torchvision.models as models
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/resnext50_32x4d-7cdf4587.pth" to /root/.cac
100%                                                95.8M/95.8M [00:01<00:00, 75.8MB/s]
```

Transfer Learning

CNN 기본 모델 구조 : 입력 + CNN 백본 + 분류기 => 출력



fc == fully connected layer == classifier

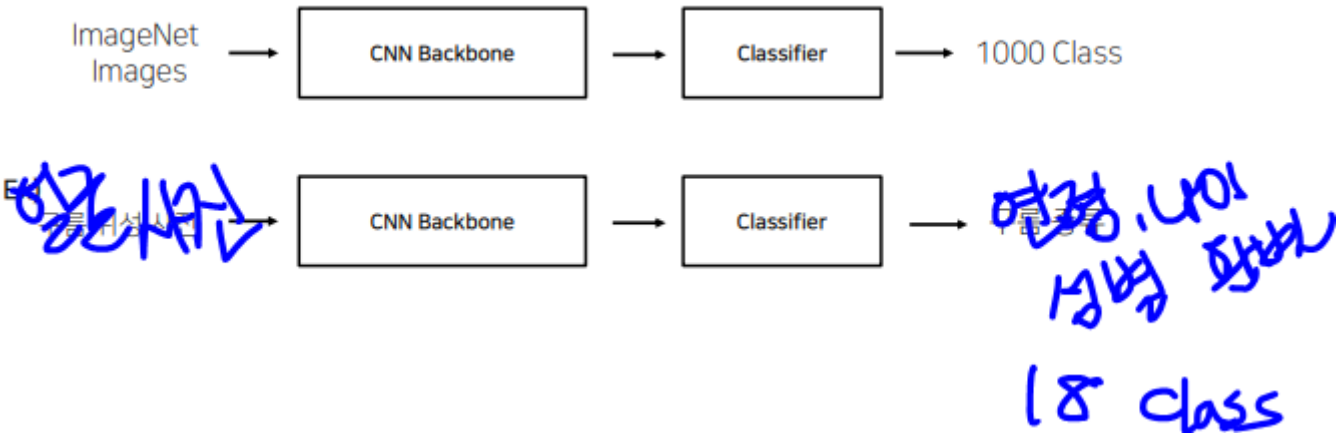
```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
print(resnet18)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torc
100%                                                 44.7M/44.7M [00:00<00:00, 77.3MB/s]
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
```

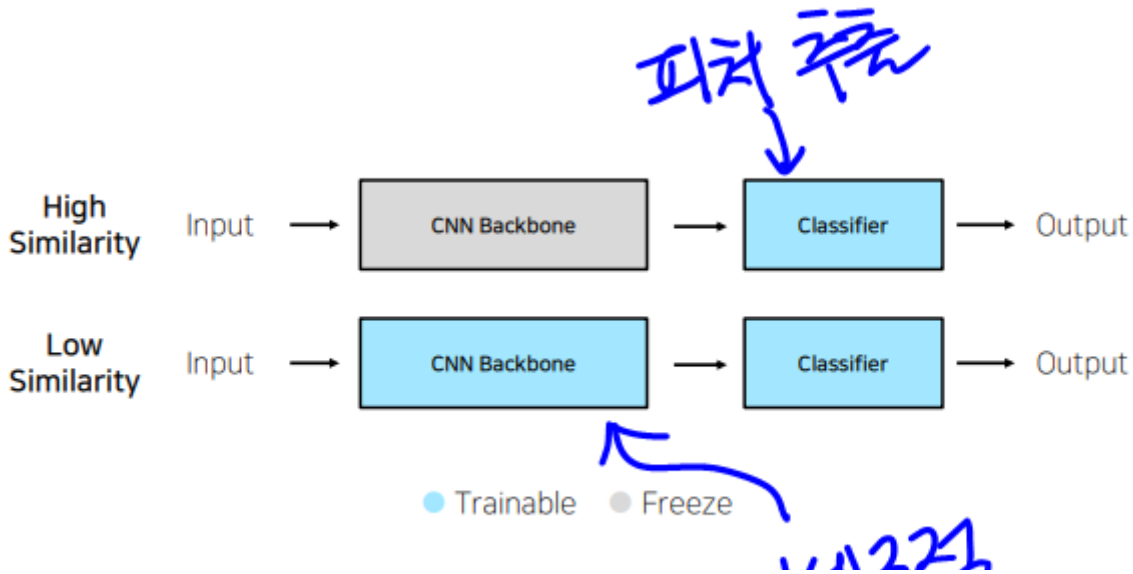
Transfer Learning 순서

1. Pretraining에서 설정했던 문제와 내 문제와의 유사성을 알아내자.
    - ImageNet Pretraining : 실생활 이미지를 1000개의 클래스로 구분함
    - 내 문제 : 인간 정면 사진을 18개의 클래스로 구분해야 함

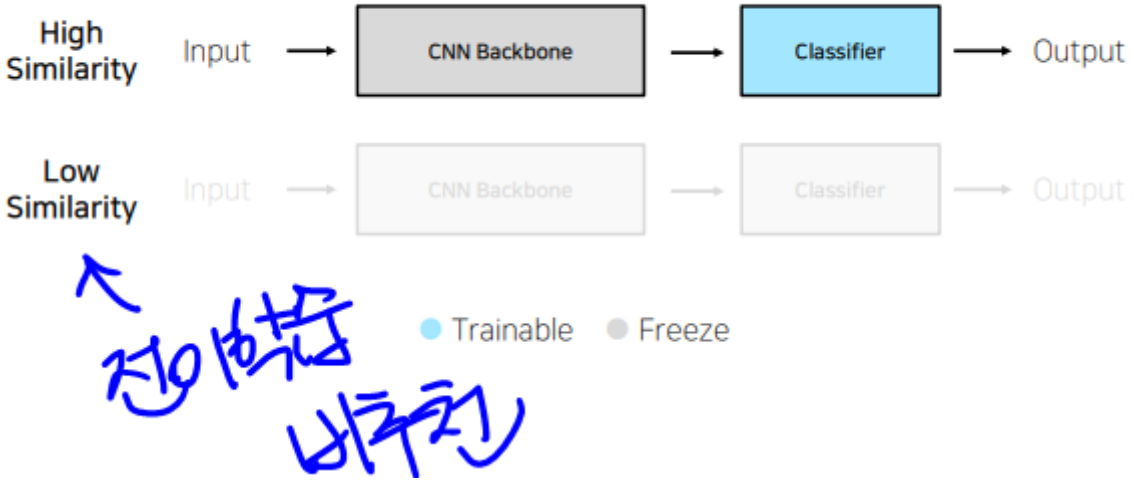


2. 내 문제를 풀기 위한 학습 데이터가 많을 때

- 이미지넷의 학습 방식이 마스크 착용여부, 성별, 연령까지 구분할 수 있을까? => 아니오, CNN 백본까지 학습

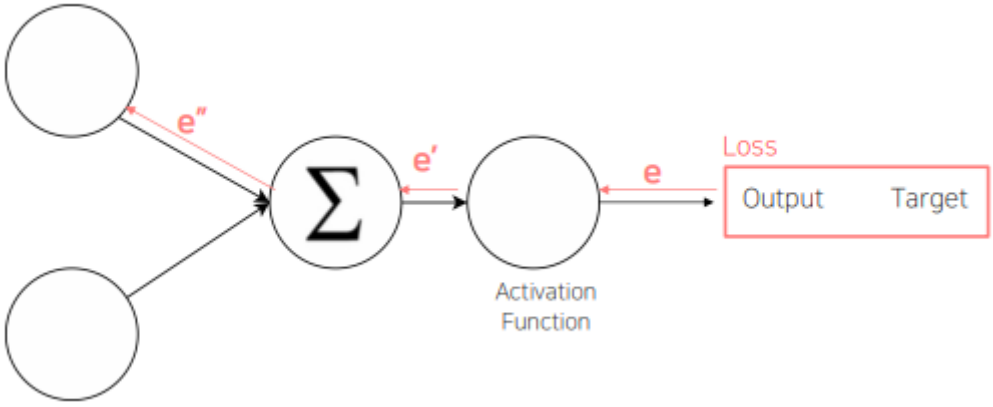


3. 내 문제를 풀기 위한 학습 데이터가 부족할 때



(7강) Training & Inference 1

학습에 필요한 요소 : Loss, Optimizer, Metric



오차 역전파

손실함수 == 비용함수 == 오차함수

loss.backward()

loss.backward() 함수가 실행되면 모델 파라미터의 grad 값이 업데이트된다.

특별한 loss

오차를 만들어내는 과정에 양념을 더하는 방법

- Focal loss : 클래스가 불균형한 경우, 맞출 확률이 높은 클래스는 loss를 작게 부여하고 맞출 확률이 낮은 클래스의 loss를 훨씬 크게 부여한다.
- Label smoothing loss : 클래스 타겟 라벨을 원핫벡터([0,1,0,0,0...)로 사용하지 않고 조금 soft 하게 표현([0.025,0.9,0.025,0.025 ...])해서 일반화 성능을 높인다.

Optimizer

$$w' = w - \eta \frac{\partial E}{\partial w}$$

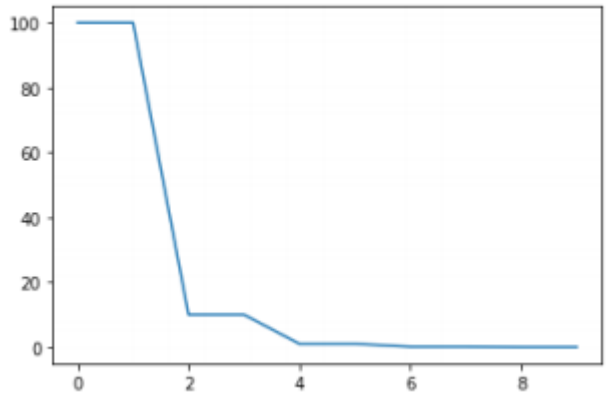
Learning rate (학습률)

방향

LR 스케쥴러

StepLR

특정 Step마다 LR 감소

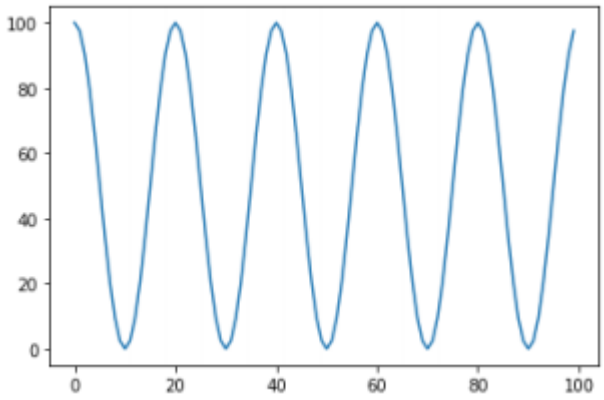


- CosineAnnealingLR

  코사인 함수 형태로 LR을 급격히 변경한다.

  loss크면 많이 이동

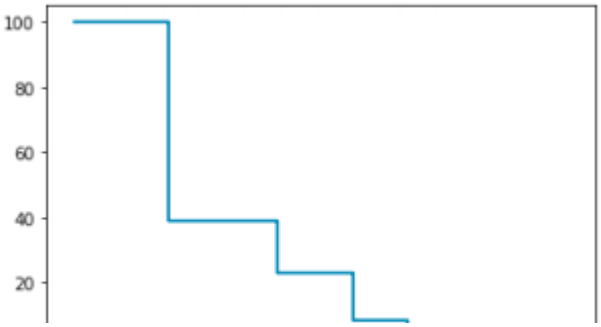
  지역해 빠르게 탈출 가능



- ReduceLROnPlateau

  성능 향상이 없을 때 LR 감소시킨다.

  일반적으로 많이 사용한다.



Metric

모델 평가 지표 : 학습된 모델을 객관적으로 평가하는 지표 필요

- 분류 문제 : Accuracy, F1-score, precision, recall, ROC & AUC
- 회귀 문제 : MAE, MSE
- 추천(랭킹, 순서 중요) 문제 : MRR, NDCG, MAP

Class 분포 편향 X : Accuracy

Class 분포 편향 O : F1-score

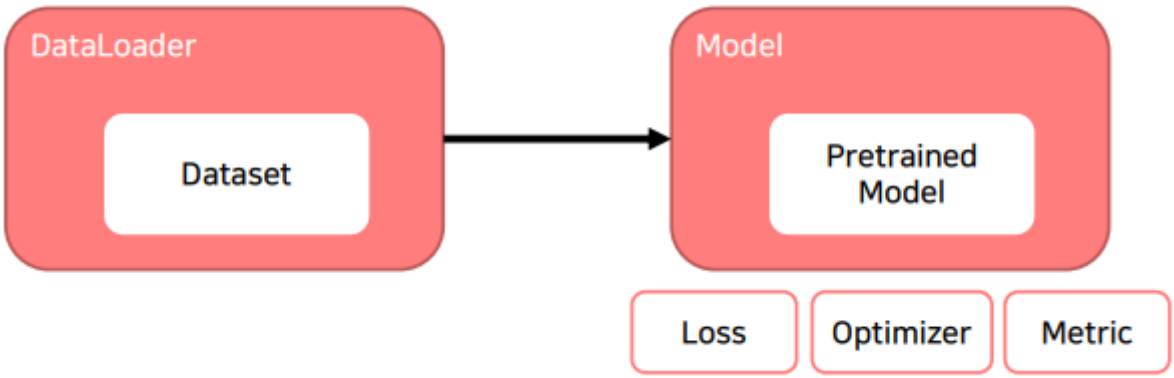
코드 실습

[ ] ↳ 숨겨진 셀 5개

(8강) Training & Inference 2

Training Process

트레이닝 준비



- model.train() : 모델을 학습한다. <-> model.eval()

- optimzer.zero_grad() : 이전 배치의 grad가 현재 배치의 grad의 초기값이 되는 것을 방지한다. 옵티마이저가 모델의 파라미터를 제어하기 때문에 가능하다.
- criterion = torch.nn.CrossEntropyLoss() : criterion은 nn.Module 상속
- loss = criterion(outputs, labels) : loss를 마지막으로 chain 생성
- optimizer.step() : grad 업데이트

코드 실습

Gradient Accumulation

[ ] ↳ 숨겨진 셀 2개

Inference Process

- model.eval() : 추론 모드로 모듈을 설정
- with torch.no_grad() : 파라미터 업데이트하지 않음
- validation : 추론 과정에 검증 데이터 셋이 들어가면 검증임

코드 실습

[ ] ↳ 숨겨진 셀 6개

Pytorch Lightning

간단하게 학습할 수 있어 생산성이 좋다.

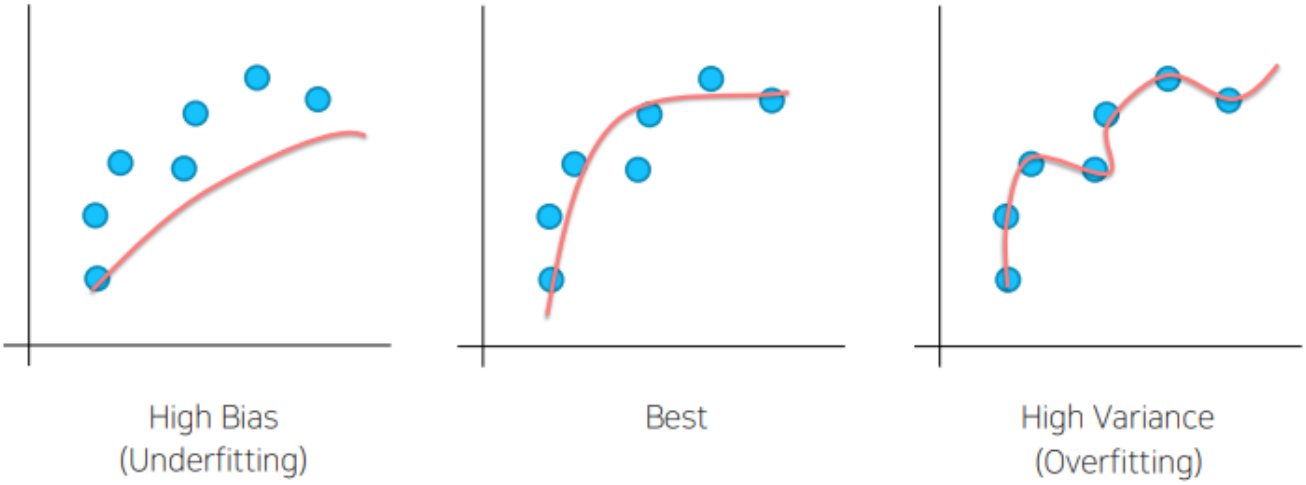
(9강) Ensemble

Ensemble

앙상블 : 더 나은 성능을 위해 서로 다른 여러 개의 학습 모델을 사용하는 것, 속도가 느려서 현업에서 많이 사용하지 않음, 머신러닝 프로세스에서 마지막 단에서 실행

- 부스팅 : High bias일 때 사용, 틀린 문제에 가중치를 두어서 학습
  - 예 : 그래디언트 부스팅
- 배깅 : High variance일 때 사용, 데이터셋의 샘플을 추출해서 학습

- 예 : 랜덤 포레스트



Model Averaging(Voting)

Voting이 잘 되는 이유 : 서로 다른 모델이 똑같은 오류를 발생시키는 경우가 보통 없다.

- Hard voting : 다수결 투표
- Soft voting : 다른 라벨의 점수도 고려 가능, 대게 성능이 좋다고 함

Hard Voting

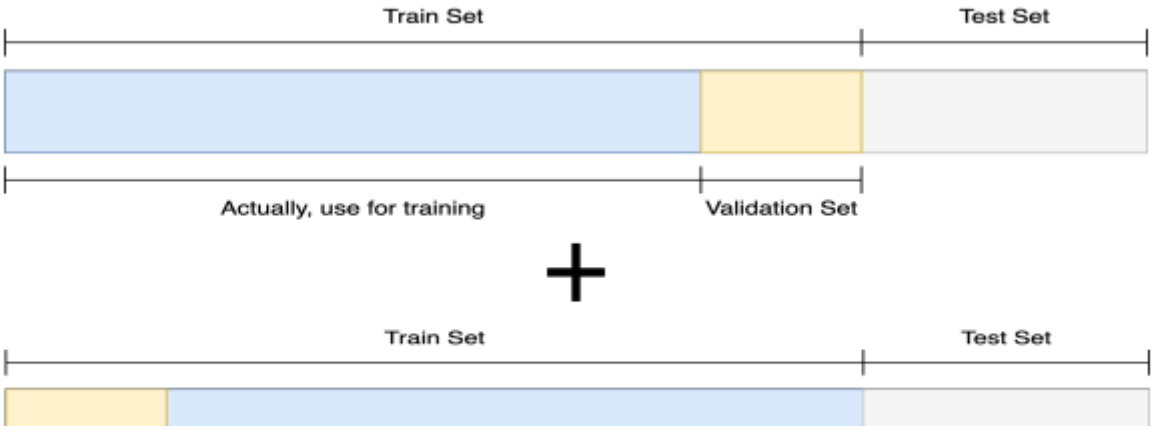
index	Model 1	Model 2	Model 3	Model 4	Model 5	result
0	1	1	2	3	4	1
1	3	3	1	3	1	3
2	1	2	4	3	4	4
3	3	2	1	3	3	3

Soft Voting

index	Model 1	Model 2	Model 3	Model 4	Model 5	result_avg
0	[0.1, 0.5, 0.2, 0.2]	[0.1, 0.6, 0.1, 0.2]	[0.1, 0.4, 0.3, 0.2]	[0.3, 0.3, 0.2, 0.2]	[0.1, 0.4, 0.1, 0.4]	2
1	[0.1, 0.1, 0.7, 0.1]	[0.3, 0.2, 0.3, 0.2]	[0.2, 0.1, 0.6, 0.1]	[0.1, 0.2, 0.5, 0.2]	[0.3, 0.3, 0.3, 0.1]	3
2	[0.1, 0.1, 0.1, 0.7]	[0.2, 0.2, 0.4, 0.2]	[0.1, 0.1, 0.2, 0.5]	[0.2, 0.1, 0.3, 0.4]	[0.1, 0.3, 0.2, 0.4]	4

Cross Validation

Validation Set을 고정하지 않고 다양한 데이터 샘플을 Validation Set으로 이용한다.



Stratified K-Fold Cross Validation

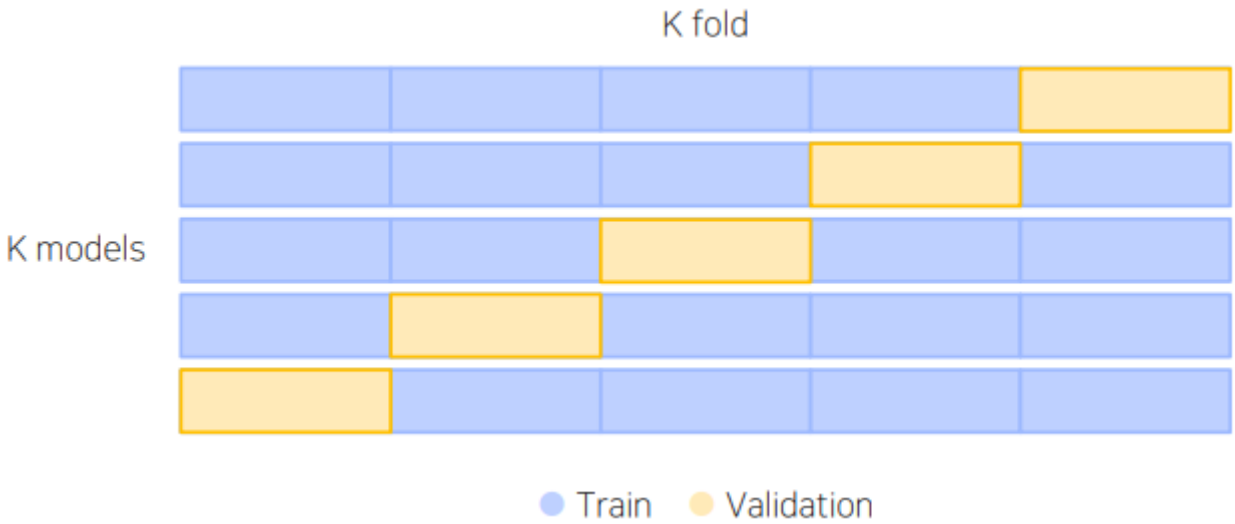
가능한 경우를 모두 고려 + 전체 Class 분포를 반영해서 Split한다.

K-Fold

- 일반화에는 좋지만 적절한 K를 찾는 것이 어렵다.
- 보통 K는 5 이상이다.
  - K = 3, 66% 학습에 사용, 일반화 잘 되는지 의문
  - K = 20, 95% 학습에 사용, 앙상블 해야되는 모델 너무 많아 시간 소요가 많다.

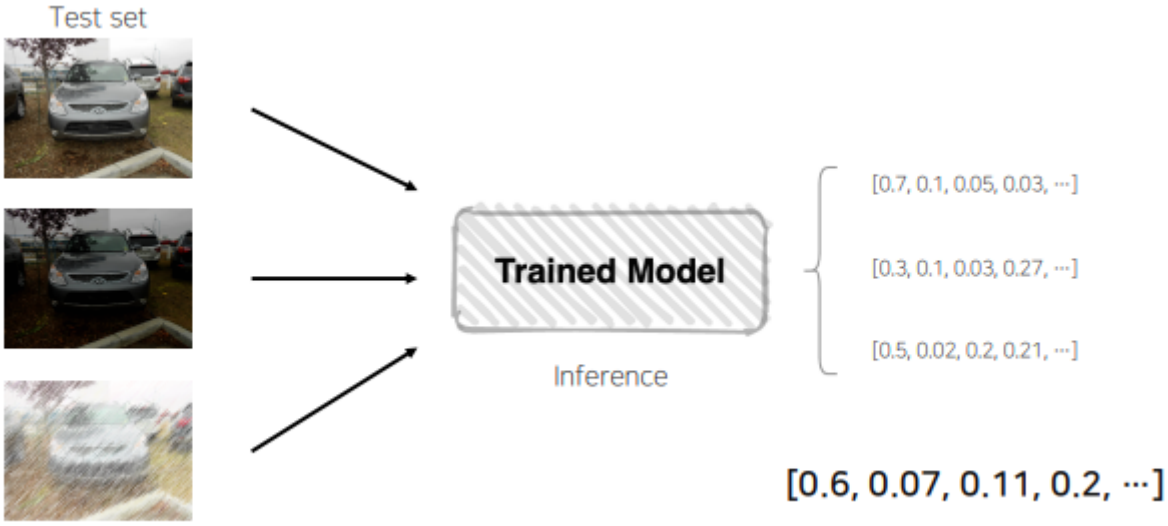
Stratified

- 전체 데이터 셋 Class 분포를 반영해서 학습과 검증 셋을 분류한다.



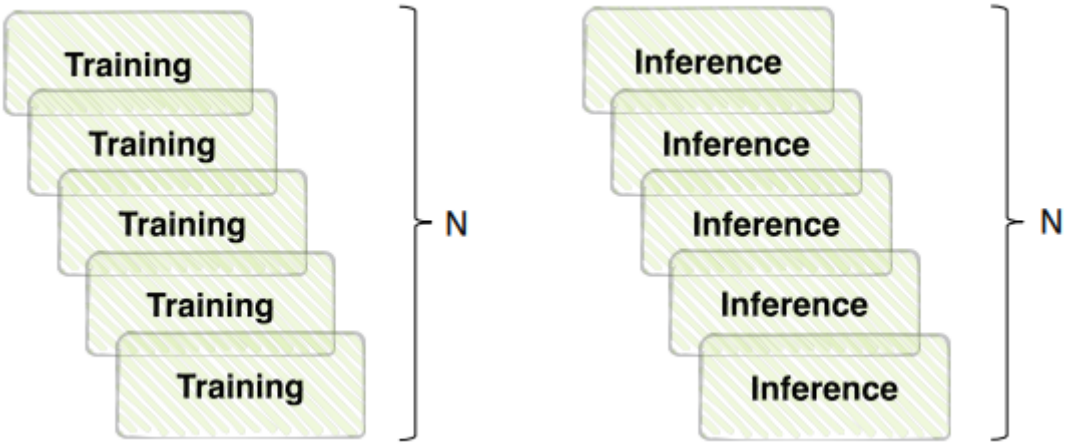
TTA (Test Time Augmentation)

테스트 이미지를 Augmentation 후 모델 추론, 출력된 여러가지 결과를 앙상블



성능과 효율의 Trade-off

앙상블은 효과는 확실히 있지만 학습과 추론 시간이 배로 증가한다.



Hyperparameter Optimization

초매개변수 최적화는 모델 앙상블만큼 시간이 들지만 앙상블만큼 효과가 좋지는 않다.

시스템에 영향을 주는 초매개변수

- Learning rate

- Batch size
- Hidden Layer 갯수
- Loss 파라미터
- Optimizer 파라미터
- k-fold에서 k
- Dropout
- Regularization

Optimization 기법

- Grid Search
- Randomized Search
- Bayesian optimization

참고 문서

- Optuna github
- Hyperparameter Opt 포스팅
- AutoML 관련 포스팅

Optuna : 파라미터 범위를 주고 그 안에서 시도횟수 만큼 시행

코드 실습

[ ] ↳ 숨겨진 셀 3개

(10강) Experiment Toolkits & Tips

Training Visualization

Tensorboard

사용법

```
tensorboard --logdir [log 저장 경로] --hots [주소] --port [포트번호]
```

wandb

wandb 홈페이지 : https://wandb.ai/site

코드 실습

[ ] ↳ 숨겨진 셀 *10*개

Machine Learning Project

Paperswithcode : https://paperswithcode.com/

Jupyter Notebook

- 장점 : Cell 단위 실행 가능 -> EDA와 전처리가 편함
- 단점 : 학습 진행 도중 노트북 창이 꺼지면 트래킹이 불가함

Python IDLE

- 장점 : 코드 재사용과 디버깅, 실험이 편함

```
python train.py --config ./config.json
```

Tips

분석 코드보다 필자의 설명글(이유, 방향성)을 주의 깊게 읽을 것