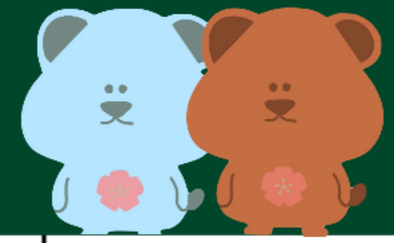
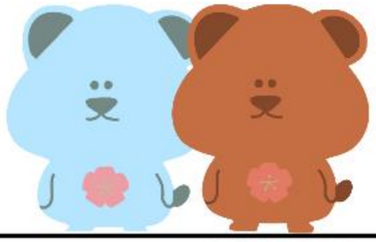


농업 환경 변화에 따른 작물 병해 진단 AI 경진대회

2팀 – 이지혜, 한예송, 홍재령

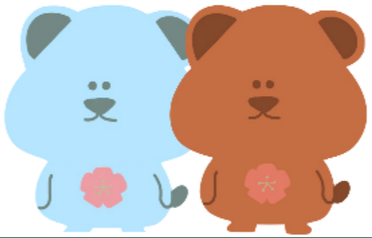
EWHA EURON

목차



EWHA EURON

Introduction



EWHA EURON

#1 대회 주제

농업 환경 변화에 따른 작물 병해 진단 AI 경진대회

병해 피해를 입은 작물 사진 & 작물의 생장 환경 데이터 → 작물의 병해를 진단하는
AI 모델 **작물의 종류 (crop), 병해의 종류 (disease), 병해의 진행 정도 (risk)**

각 폴더 당
- train : image, json, csv 파일
- test : image, csv 파일



EWHA EURON

#1 데이터셋

1. train : 학습용 데이터
    - 10001 : 데이터 고유 아이디
        - 10001.jpg : 이미지 파일
        - 10001.csv : 환경 데이터
            - 촬영 전 48 시간의 "측정 시각", "내부 온도", "내부 습도", "내부 이슬점", "내부 CO2", "외부 풍속", "외부 누적일사" 등의 환경 정보
        - 10001.json :
            - description
                - image : 이미지 파일 이름
                - date : 촬영 날짜
                - time : 촬영 시간
                - region : 촬영 지역
                - height : 이미지 높이
                - width : 이미지 너비
                - task : 데이터 종류 (질병/해충/병해/정상 구분)
            - annotations
                - disease : 작물 상태 코드
                - crop : 작물 코드
                - area : 작물 촬영 부위
                - grow : 작물의 생육 단계
                - risk : 질병 피해 정도
                - bbox : 주목 객체 바운딩 박스 (x, y, w, h 형태)
                - part : 병해 부위 바운딩 박스 (x, y, w, h 형태)
    - 10002
    - 10003
    - ...

#1 데이터셋

[추가] train.csv : train set에 대한 정답 파일
image : 이미지 파일 이름
label : "{작물 코드}_{작물 상태 코드}_{질병 피해 정도}" 형태의 문자열
"{crop}_{disease}_{risk}"

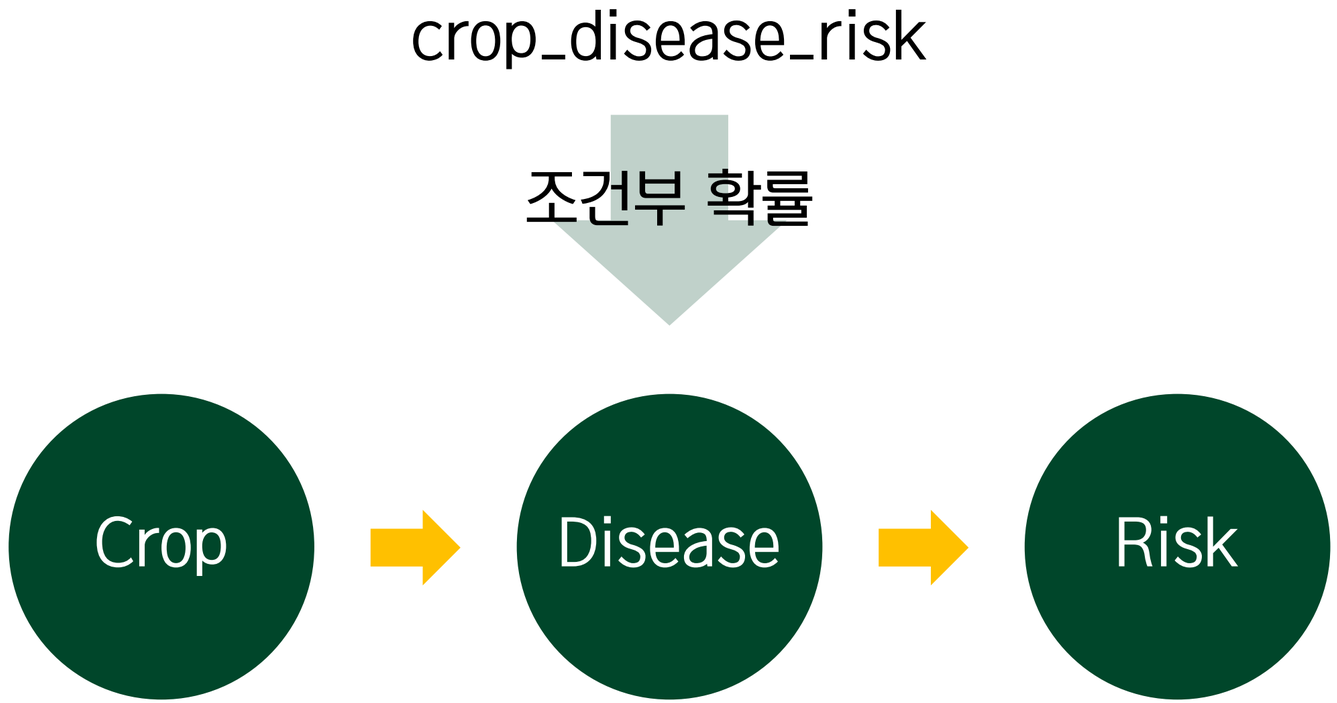
2. test : 평가용 데이터셋
├ 10001 : 데이터 고유 아이디
│ ├ 10001.csv : 환경 데이터
│ │ └ 촬영 전 48 시간의 "측정 시각", "내부 온도", "내부 습도", "내부 이슬점", "내부 CO2", "외부 풍속", "외부 누적일사" 등의 환경 정보
│ └ 10001.jpg : 이미지 파일
│
├ 10002
├ 10003
└ ...
3. sample_submission.csv : 제출용 양식

image : 이미지 파일 이름
label : "{작물 코드}_{작물 상태 코드}_{질병 피해 정도}" 형태의 문자열
"{crop}_{disease}_{risk}"
상세

12455	1_00_0
12470	5_b6_1
12476	2_a5_2
12483	4_00_0
12485	5_00_0
12504	3_b3_1

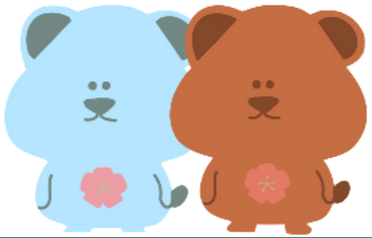
EWHA EURON

#1 접근 방식



Base Model + 이로부터 crop, disease, risk를 예측하는
Deep Neural Network + Catboost

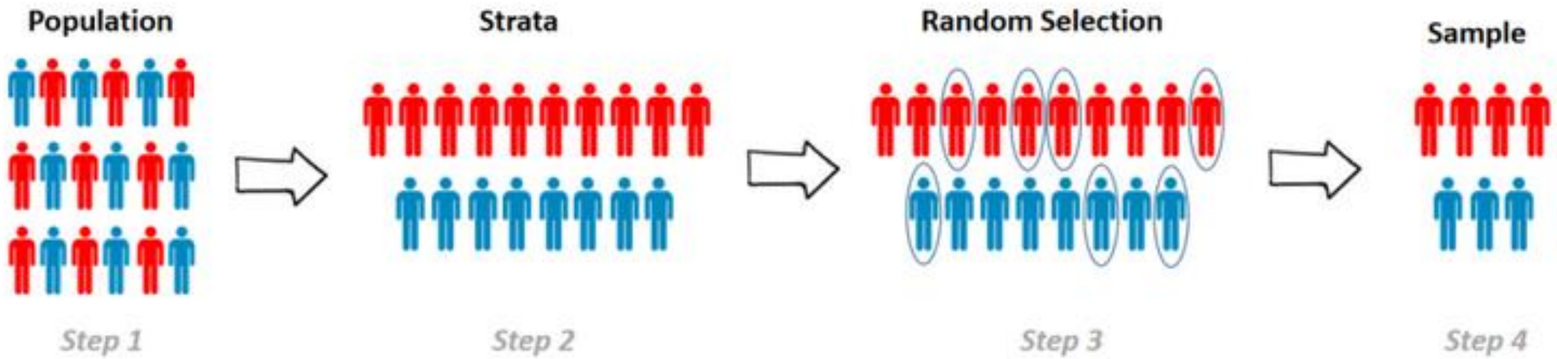
Data Preprocessing



EWHA EURON

#2 Stratified Sampling

데이터 분포 비율을 유지하면서 데이터를 샘플링(취득)하는 것



- **훈련 셋과 검증 셋 분리 과정**에서 데이터가 고르게 분포되도록 사용
- crop, disease, risk, **사진이 찍힌 부위, 생육 단계의** 5가지 정보를 기반으로 stratified sampling을 진행

#2 Train-Validation Split

```
df_dict['img_path'] = img_file
if json_file:
    with open(json_file, 'r', encoding='utf-8') as json_f:
        temp_dict = json.load(json_f)

        df_dict['crop'] = temp_dict['annotations']['crop'] - 1

        if df_dict['crop'] == 0 or df_dict['crop'] == 3:
            df_dict['disease'] = disease_encoding.index('0')
        else:
            df_dict['disease'] = disease_encoding.index(temp_dict['annotations']['disease'])
        df_dict['risk'] = temp_dict['annotations']['risk']
        xyhw = temp_dict['annotations']['bbox'][0]
        coordinate = [int(xyhw['x']), int(xyhw['y']), int(xyhw['w']), int(xyhw['h'])]
        df_dict['coordinate'] = coordinate
        df_dict['strat'] = str(temp_dict['annotations']['crop']) + str(temp_dict['annotations']['disease'])
        + str(temp_dict['annotations']['risk']) + str(temp_dict['annotations']['area']) + str(temp_dict['annotations']['grow'])
```

```
def strat_split_save(csv_file, save_folder, test_size=0.1, stratified='strat'):
    raw_data = pd.read_csv(csv_file)
    strat = raw_data[[stratified]] #strat
    target_data = raw_data.drop(columns=stratified)
    train_set, test_set, _, _ = train_test_split(target_data, strat, test_size=test_size, stratify=strat)

    train_set.to_csv(os.path.join(save_folder, 'train.csv'), index=False)
    test_set.to_csv(os.path.join(save_folder, 'test.csv'), index=False)
```

EWHA EURON

#2 결측치 제거

대부분의 데이터에 해당 열의 데이터가 없는 경우 아예 제거

	A	B	C	D	E	F	G	H
1	측정시각	내부 온도 1 평균	내부 온도 1 최고	내부 온도 1 최저	내부 온도 2 평균	내부 온도 2 최고	내부 온도 2 최저	내부 온도 3 평균
2	2021-06-24 14:30:00	28.3	28.3	28.1	-	-	-	-
3	2021-06-24 14:20:00	28.3	28.3	28.3	-	-	-	-
4	2021-06-24 14:10:00	28	28.2	27.8	-	-	-	-
5	2021-06-24 14:00:00	27.8	27.8	27.7	-	-	-	-
6	2021-06-24 13:50:00	27.7	27.8	27.6	-	-	-	-

```
def make_nan(x):
    if str(x) == '-':
        return np.nan
    return x

def return_dict_from_files(files_list_and_percentile_num: tuple):
    csv_file, json_file, img_file, percentile_num = files_list_and_percentile_num
    df_dict = {}

    temp = pd.read_csv(csv_file)
    temp = temp[use_columns]
    temp = temp.applymap(make_nan).astype(np.float64).describe(percentiles=[i/percentile_num for i in range(percentile_num+1)])
    temp_percentiles = temp.iloc[4:-1]
```

EWHA EURON

#2 결측치 제거

누적일사와 CO2의 경우 존재 여부를 추가
→ 없는 경우 이와 관련된 데이터를 전부 -1로 처리
(boosting 알고리즘을 사용할 예정이기에 가능한 방식)

```
for column in use_columns:
    if "누적일사" in column:
        if temp.loc['count', column] == 0:
            df_dict[column+'_is_exist'] = 0
        else:
            df_dict[column+'_is_exist'] = 1
        if np.isnan(temp_percentiles.loc['0%', column]):
            for i in range(len(temp_percentiles)):
                df_dict[column+f'_{i}'] = -1
        else:
            for i in range(len(temp_percentiles)):
                df_dict[column+f'_{i}'] = temp_percentiles.loc[:, column][i]
```

```
elif "CO2" in column:
    if temp.loc['count', column] == 0:
        df_dict[column+'_is_exist'] = 0
    else:
        df_dict[column+'_is_exist'] = 1
    if "최고" in column:
        if np.isnan(temp.loc['max', column]):
            df_dict[column+'_max'] = -1
        else:
            df_dict[column+'_max'] = temp.loc['max', column]
    elif "최저" in column:
        if np.isnan(temp.loc['min', column]):
            df_dict[column+'_min'] = -1
        else:
            df_dict[column+'_min'] = temp.loc['min', column]
    else:
        if np.isnan(temp_percentiles.loc['0%', column]):
            for i in range(len(temp_percentiles)):
                df_dict[column+f'_{i}'] = -1
        else:
            for i in range(len(temp_percentiles)):
                df_dict[column+f'_{i}'] = temp_percentiles.loc[:, column][i]
```

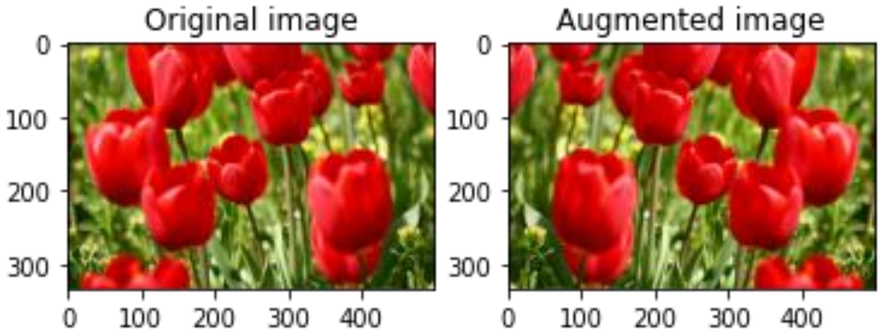
#2 Percentile

질병의 발생 여부는 환경의 극단적인 상황에 단시간 노출되기 보다는,
일정한 **한계점을 벗어난 환경에 일정 시간 이상** 노출되었을 때 발생할 것
→ 각 열의 값들에서 percentile을 구하고, 총 50등분
➔ 48시간 동안의 낮은 값부터 높은 값까지 순차적으로 나열하는 효과
➔ 각 값은 몇 시간 동안 그 값보다 높은 수치(혹은 낮은 수치)에 작물이 노출되었는지를 대표하는 값으로 볼 수 있음

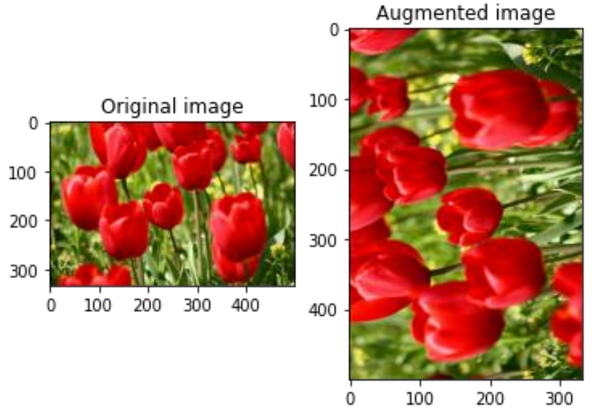
- percentile 개수를 줄이면 성능이 감소됨
- 데이터 누락이나 동일 데이터의 중복 등에 매우 강건

#2 Image Augmentation

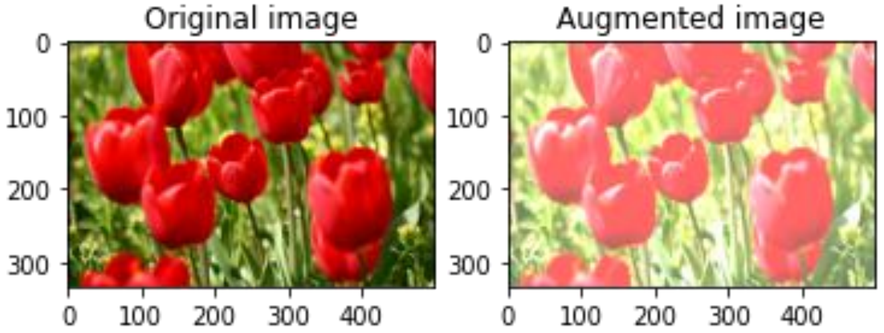
Flip



Rotation

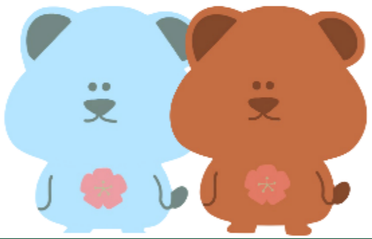


Brightness



#03. Technical 개념

(1) 수상자의 생각의 흐름
(2) BiT
(3) CatBoost



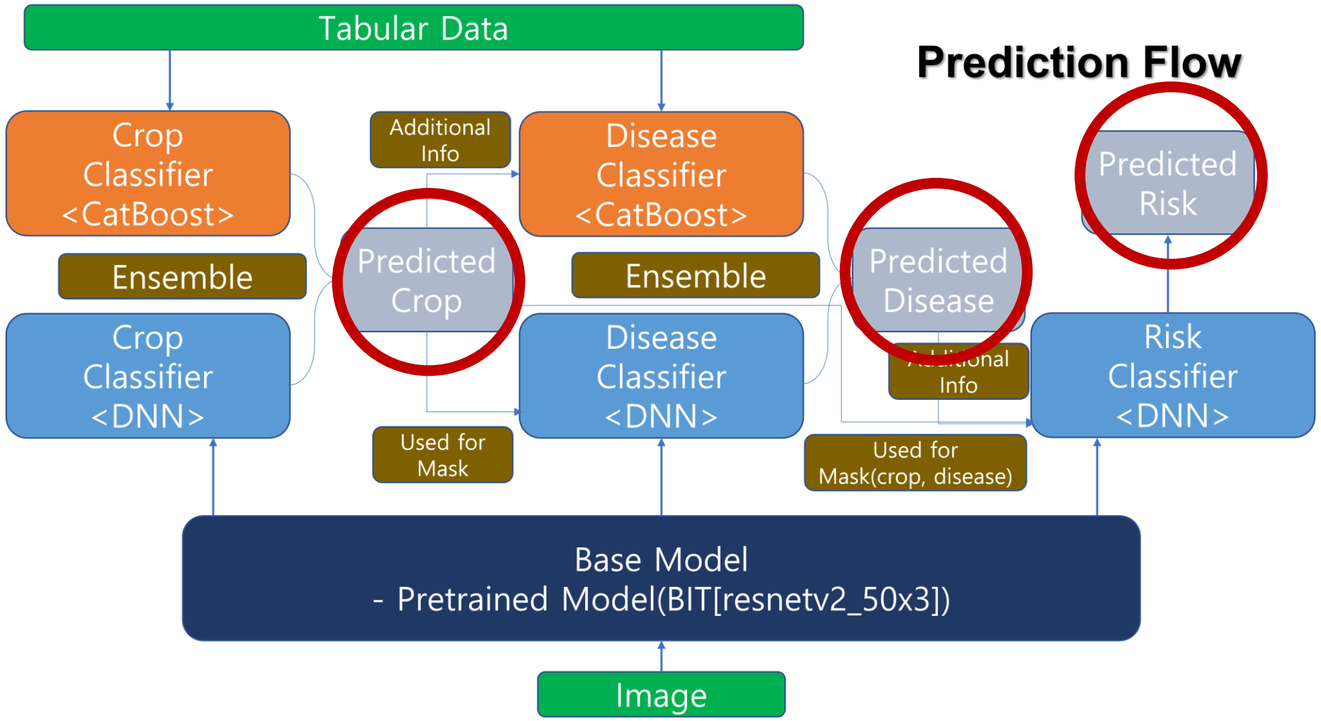
EWHA EURON

#3-1-1. 사용 모델 구조

✔1. [Crop]_[Disease]_[Risk]에서 각 라벨을 따로 예측

✔2. 수치 데이터 : CatBoost
이미지 데이터 : BiT (ResNet-v2)

✔3. training 중에는 실제 ground truth를, predict중에는 이전에 예측한 값 적용



#3-1-2. 수상자의 생각 흐름

✔ 1 모델의 layer이 많아도 성능이 오히려 안 좋음 =>
모델의 넓이가 중요 =>
저 수준의 feature이 중요

✔ 2 Image Net에 finetuning한 모델의 성능은 낮음
=> Image Net task보다 작물 병해가 **더 미세한 task**이기 때문

✔ 3. **이미지의 해상도**가 중요 (이미지 크기를 Image Net에 맞춰 (224 x 224)로 줄이니 성능 감소 => (384, 384)로 바꿈

결론 : BiT-ResNetv2_50x3

Width	Depth
모델의 layer의 개수	일반적으로 Conv layer의 channel #으로 계산
Complex Features (기대) Gradient Loss (문제)	Fine-grained features

#3-2-1. BiT : General Visual Representation Learning

✔ 논문의 의의 : 적은 데이터로 높은 성능의 Transfer Learning을 위한 효과적인 pre-training을 위한 인사이트 제공

✔ 논문의 결론 : 큰 데이터 + 큰 모델을 사용하자

1. 큰 자원이 필요하니 오래 학습 시키자 (데이터를 많이 사용하면서 짧게 학습시키면 오히려 악영향)
2. 수렴이 느려도 큰 weight decay를 사용하자
3. 데이터셋의 크기에 따라 model scaling은 매우 중요

✔ BiT의 구조 : **Upstream Pre-Training + Downstream Transfer Learning**

❓Weight Decay란?

❗학습된 모델의 복잡도를 줄이는 목적으로 가중치가 너무 큰 값을 갖지 않도록 L1, L2 decay같은 패널티 추가

#3-2-2. Upstream Pre-Training

- ✔ 모델 Architecture : ResNet-v2
- ✔ Optimizer : SGD (momentum = 0.9, learning rate = 3e-2)
- ✔ Image Size : (224, 224) Batch size : 4096
- ✔ Train Scheduler : BiT 모델에 사용된 **데이터의 크기별로** 다르게 적용
- ✔Data Augmentation : Downstream Task에 따라 다르게 사용
- ✔ Warm Up : 매 스텝마다 linear 하게 learning rate 증가

❓왜 SGD를? Adam이 RMSProp + momentum이고 빠르게 최적의 값으로 수렴하기 때문에 거의 항상 Adam을 사용했었는데 ,

❗사실 SOTA달성은 SGD가 많이 하며, SGD가 convex한 train function에서 매우 stable + Adam같은 adaptive계열은 train performanc는 좋아도 test에서 배신

#3-2-3. ResNet-v2 : Pre-Activation ResNet

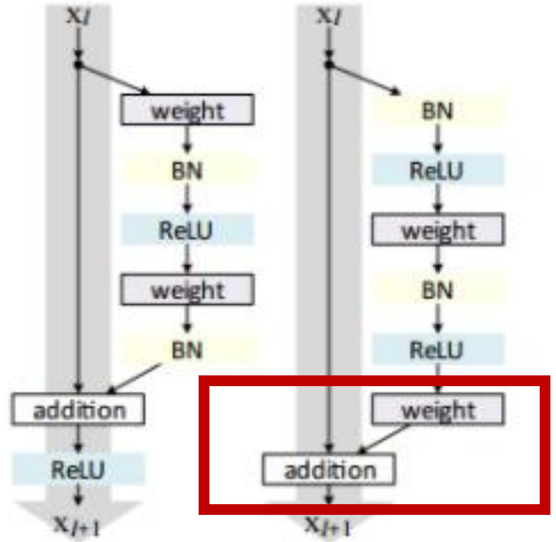
✔ Deeper Conv Layer의 Gradient 손실 방지를 위해 Identity Mapping을 한 것이 ResNet

✔활성화 함수의 위치를 바꿔서 Identity Mapping이 가능하게 함

✔ after-add mapping 함수를 identity mapping으로 바꾼다면 순전파와 역전파시에 신호가 보존 가능

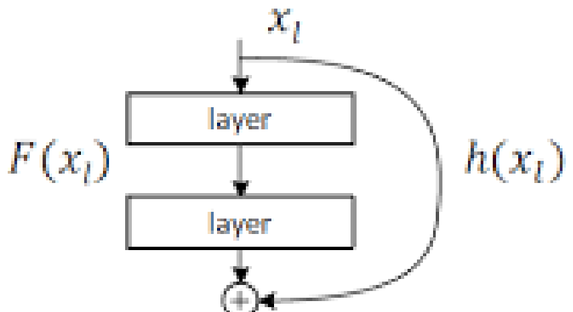
❓Identity Mapping이란?

❗입력값을 identity mapping함수를 사용해 정보가 손실되지 않도록 출력값에 전달해 gradient 손실 방지



(a) original (b) proposed

- shortcut mapping: h = identity
- after-add mapping: f = ReLU

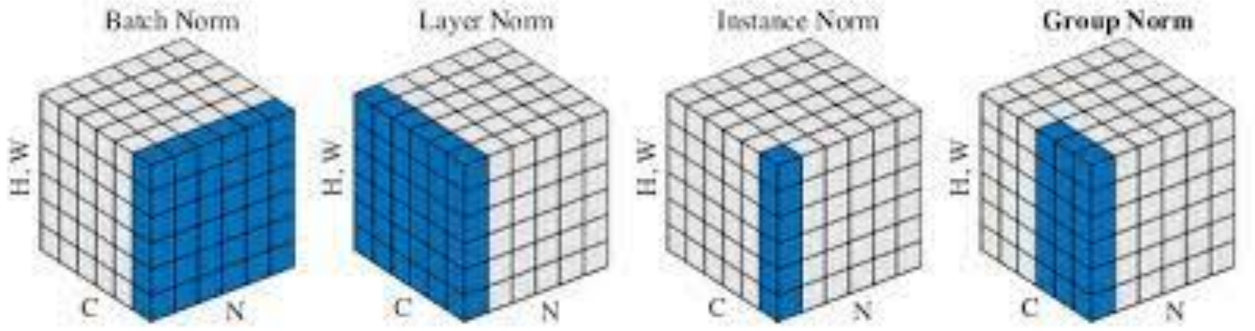


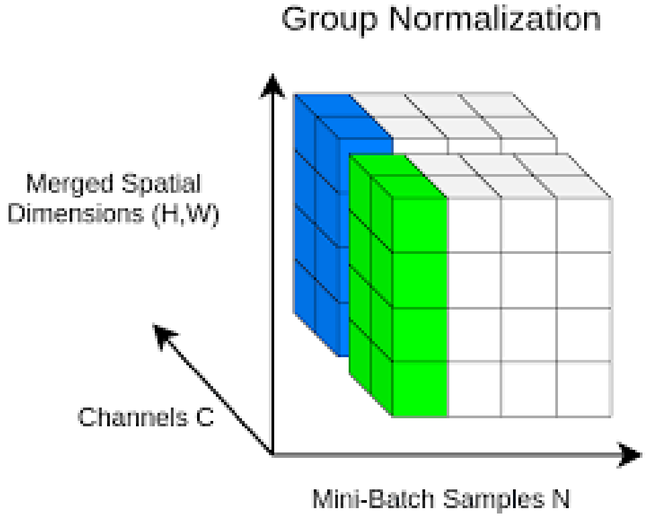
$$x_{l+1} = f(h(x_l) + F(x_l))$$

#3-2-4. Batch Norm → Group Norm

- ✔ Batch Norm은 batch size에 영향을 많이 받음
- ✔ Batch Norm은 계속해서 통계량을 업데이트 하는 원리 -> 연산량 증가
- ✔ Group Norm은 **하나의 샘플이 갖는 channel을 그룹지어** 평균과 표준편차를 계산하고 normalization

❗ 어떻게 샘플을 정햐느냐의 차이인데, Group Norm은 그룹의 개수를 정해주면 전체 채널수 / group 만큼의 샘플로 normalize를 한다.





#3-2-5. Weight Standardization

✔ Normalization은 데이터의 불필요한 정보 제거에 필수

✔ batch size에 제한이 없음

✔ Loss function의 최소화를 위해서는 이계도함수를 줄여야 하고, 이를 위해서는 loss function이 smooth해야 함

✔ Conv Filter의 Weight에 대해서 정규화

```
class StdConv2d(nn.Conv2d):
  def forward(self, x):
    w = self.weight
    v, m = torch.var_mean(w, dim = [1,2,3], keepdim = True,
unbiased = False) ## variance(분산), mean(평균)
    w = (w-m)/torch.sqrt(v + 1e-10)

    return F.conv2d(x, w, self.bias, self.stride,
self.padding, self.dilation, self.groups)
```

$$\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{W}}_{i,j} \mid \hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}} + \epsilon}\right] \tag{2}$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x} \tag{3}$$

where

$$\mu_{\boldsymbol{W}_{i,\cdot}} = \frac{1}{I}\sum_{j=1}^{I} \boldsymbol{W}_{i,j}, \quad \sigma_{\boldsymbol{W}_{i,\cdot}} = \sqrt{\frac{1}{I}\sum_{i=1}^{I}(\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}})^2} \tag{4}$$

EWHA EURON

#3-3-1. Downstream Fine-Tuning

- ✔ 2만장 : Small Task/ 50만장 미만 : Medium Task/ 50만장 이상 : Large Task
- ✔ Batch size : 512
- ✔ Optimizer : SGD (momentum = 0.9, learning rate = 3e-3)
- ✔ Image size :
  - 96 x 96 보다 작은 경우 : 160 x 160 → 128 crop
  - 448 x 448 보다 큰 경우 : 384 crop
- ✔ Fine-Tuning Schedule :
  - 공통 : 30, 60, 90% 에서 lr decay (10 factor : learning rate /= 10)
  - Small Task : 500 steps [100, 200, 300, 400, 500]
  - Medium Task : 1만 steps [500, 3000, 6000, 9000, 10000]
  - Large Task : 2만 steps [500, 6000, 12000, 18000, 20000]

최적의 hyperparameter을 찾기 위한 Heuristic Rule

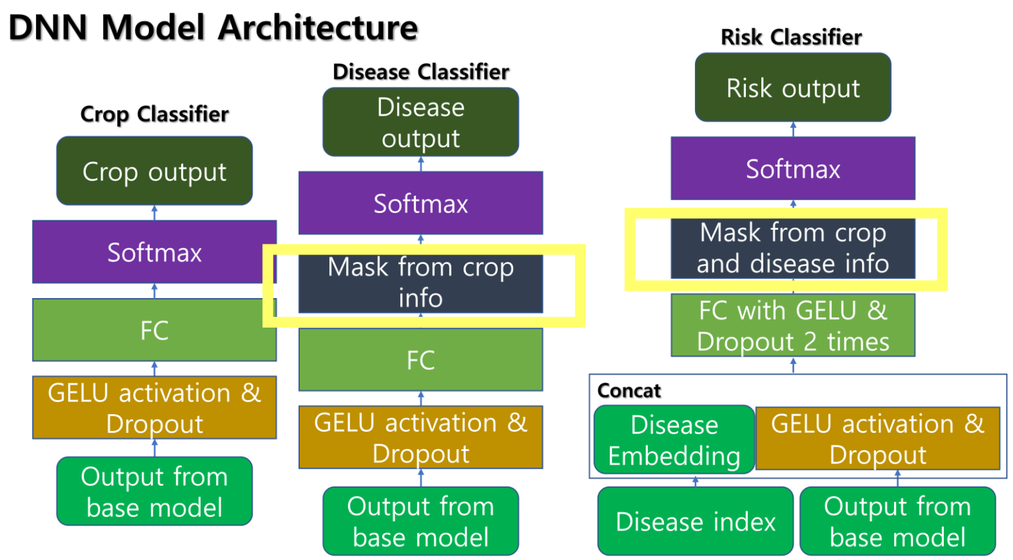
Downstream Tuning중에는 weight decay나 dropout등의 regularization 없이도 우수한 성능

#3-4. 최종 Image사용 Net 구조

- ✔ Disease label은 embedding layer을 사용해서 추가 정보로 입력
- ✔ 마지막 risk는 오직 DNN만을 사용해서 예측 (CatBoost사용 안함)

✔ `loss_fn_disease_risk = FocalLoss(ignore_index=0)`

✔
```
disease_mask = {}
for key, values in disease_dict.items():
  disease_mask[key] = [True] * len(disease_decoder)
  for v in values:
    disease_mask[key][disease_encoder.index(v)] = False
```




```
class RiskHeadClassifier(nn.Module):
  def __init__(self, num_base_features=1000, hidden_feature=200, num_classes=4, drop_p=0.1, embedding_dim=10):
    super().__init__()
    self.linear = nn.Linear(num_base_features + embedding_dim, hidden_feature)
    self.classifier = nn.Linear(hidden_feature, num_classes)
    self.act = nn.GELU()
    self.drop = nn.Dropout(drop_p)
    self.embedding = nn.Embedding(num_embeddings=len(disease_encoder), embedding_dim=embedding_dim)

  @torch.cuda.amp.autocast()
  def forward(self, x, disease_code, mask):
    x = self.act(self.drop(x))
    y = self.drop(self.embedding(disease_code))
    concat = torch.cat([x, y], dim=1)
    out = self.act(self.drop(self.linear(concat)))
    out = self.classifier(out)
    out.masked_fill_(mask, -10000.)
    return out
```

#3-5-1. CatBoost : unbiased Boosting with Categorical Features

✔특정 종류의 target leakage로 인한 OVERFIT 해결이 주 목적

✔ Boosting 계열의 머신 러닝 알고리즘
✔ 결정 트리에서의 Gradient Boosting Algorithm (Level-wise로 트리 구축)

✔ 데이터가 시간이 지나면서 매우 변화하는 특성일 때

✔ 현실 세계의 시계열 데이터에서 특정 기간동안만 나타나는 특성이 있을 수 있음
=> 이러한 기간을 나누어서 개별 모델을 만들어서 학습시키고 sum_models 메소드로 여러 모델들을 결합 가능

```
class CatBoostClassifier(iterations = None,task_type =
      'GPU', devices = '0', thread_count = 16,
      learning_rate = None, border_count = None,
      random_strength = None,
      leaf_estimation_iteration = None,
      auto_class_weights = None,
      max_depth = None, eval_metric = None,
      use_best_model = True,
      cat_features = None, verbose = True)
```

EWHA EURON

#3-5-1. CatBoost : unbiased Boosting with Categorical Features

✔ <u>Ordered Boosting + Random Premutation</u>
: 기존 알고리즘에 대한 순열 기반의 대안

- 임의적으로 시계열을 부여하기 때문에 순서가 있어 “Ordered“

$$h^t = \arg\min_{h \in H} \frac{1}{n} \sum_{k=1}^{n} \left(-g^t(\mathbf{x}_k, y_k) - h(\mathbf{x}_k) \right)^2 .$$

기존의 부스팅 모델 : 일괄적으로 모든 훈련 데이터를 대상으로 잔차 계산
Catboost : **<u>일부만 가지고 잔차계산을</u>** 한 뒤 모델 생성 -> 이 모델로 다른 데이터의 잔차를 예측 (위의 과정 반복)

✔ <u>Categorical Features를 위한 Ordered TS</u>
: 범주형 feature처리에 집중한 알고리즘

✔흔히 사용하는 one-hot encoding의 cardinality 증가로 인한 메모리 문제 해결

TS : 목표 통계량

<u>범주를 제한된 수로 clustering 하기 위해서</u> 각 범주의 기대 목표값을 추정하는 TS별로 범주를 그룹화

현재 데이터를 인코딩 하기 위해 이전 데이터의 인코딩 된 값을 사용 -> Data Leakage 방지

EWHA EURON

#3-5-2. CatBoost를 위한 수치 데이터 처리

✔ 존재하지 않는 수치는 -1로 설정해 두는 것이
boosting algorithm을 사용하였기 때문에 가능

✔전체 관측 데이터 중에서 제일 비율이 작은 값부터 큰 값까지 나열
=> 특정 온도보다 낮은 환경에서, 혹은 특정 누적 일사보다
낮은 환경에서 몇시간 있었는지를 확인

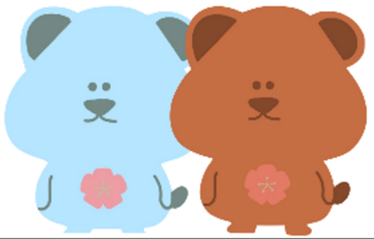
❗ 이러한 시도는 식물의 질병이 한순간에 벌어지는 것이아니라 지속적으로 특정 환경에 노출 되었기 때문이라는 가정에 의해서 한 것이다.

```
df=df.applymap(make_nan).astype(np.float64).describe(percen
tiles=[i/percentile_num for i in range(percentile_num+1)])

# (1) '-'인 비어있는 값은 nan으로 바꿈
# (2) percentileslist-like of numbers, optionalThe
percentiles to include in the output. All should fall
between 0 and 1. The default is [.25, .5, .75], which
returns the 25th, 50th, and 75th percentiles.
```

EWHA EURON

Train & Predict



EWHA EURON

Masking: train set

disease_dict

key: crop	value: disease					
0	0	‘정상군’ 만 존재				
1	00	a5				
2	00	a9	b3	b6	b7	b8
3	0	‘정상군’ 만 존재				
4	00	a7	b6	b7	b8	
5	00	a11	a12	b4	b5	

real_targets [crop_disease_risk]

{

‘1_00_0’ ‘정상군’ 만 존재

'2_00_0', '2_a5_2', 질병의 위험도 중 ‘중기’ 만 존재

‘b3’ , ‘b6’ , ‘b7’ , ‘b8’ 질병 모두 ' 전기’ 의 위험도만 존재

'3_00_0', '3_a9_1', '3_a9_2', '3_a9_3', '3_b3_1', '3_b6_1', '3_b7_1', '3_b8_1’ ,

‘4_00_0’ ‘정상군’ 만 존재

'5_00_0', '5_a7_2', '5_b6_1', '5_b7_1', '5_b8_1’ ,

'6_00_0', '6_a11_1', '6_a11_2', '6_a12_1', '6_a12_2', '6_b4_1', '6_b4_3', '6_b5_1’

}

```
disease_dict = {0:['0'],
                1:['00', 'a5'],
                2:['00', 'a9', 'b3', 'b6', 'b7', 'b8'],
                3:['0'],
                4:['00', 'a7', 'b6', 'b7', 'b8'],
                5:['00', 'a11', 'a12', 'b4', 'b5'],}
real_targets = ['1_00_0', '2_00_0', '2_a5_2', '3_00_0', '3_a9_1', '3_a9_2', '3_a9_3', '3_b3_1', '3_b6_1', '3_b7_1', '3_b8_1', '4_00_0', '5_00_0', '5_a7_2', '
```

Masking

disease_mask

disease

	[0] '0'	[1] '00'	[2] 'a5'	[3] 'a7'	[4] 'a9'	[5] 'a11'	[6] 'a12'	[7] 'b3'	[8] 'b4'	[9] 'b5'	[10] 'b6'	[11] 'b7'	[12] 'b8'
crop 0	0	1	1	1	1	1	1	1	1	1	1	1	1
crop 1	1	0	0	1	1	1	1	1	1	1	1	1	1
crop 2	1	0	1	1	0	1	1	0	1	1	0	0	0
crop 3	0	1	1	1	1	1	1	1	1	1	1	1	1
crop 4	1	0	1	0	1	1	1	1	1	1	0	0	0
crop 5	1	0	1	1	1	0	0	1	0	0	1	1	1

real_targets [crop_disease_risk]

{ '1_00_0' ,
'2_00_0', '2_a5_2' ,
'3_00_0', '3_a9_1', '3_a9_2', '3_a9_3', '3_b3_1', '3_b6_1', '3_b7_1', '3_b8_1' ,
'4_00_0' ,
'5_00_0', '5_a7_2', '5_b6_1', '5_b7_1', '5_b8_1' ,
'6_00_0', '6_a11_1', '6_a11_2', '6_a12_1', '6_a12_2', '6_b4_1', '6_b4_3', '6_b5_1' }

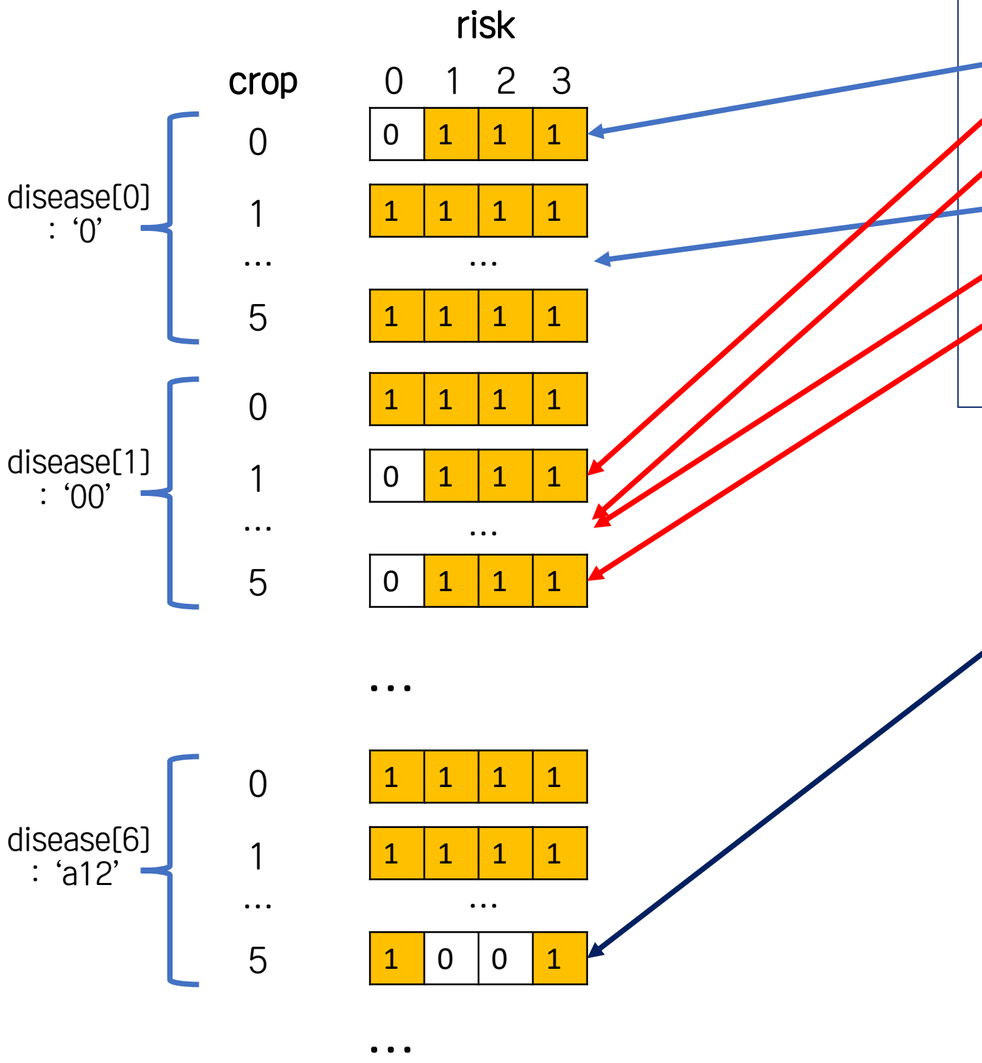
```
disease_encoding = []
for disease_codes in disease_dict.values():
    disease_encoding.extend(disease_codes)
disease_encoding = sorted(list(set(disease_encoding)))
disease_decoding = {}
for k, v in enumerate(disease_encoding):
    disease_decoding[k] = v

disease_mask = {}
for k, values in disease_dict.items():
    disease_mask[k] = [True] * len(disease_encoding)
    for v in values:
        disease_mask[k][disease_encoding.index(v)] = False

percentile_num = 50
```

Masking

risk_with_crop_mask



real_targets [crop_disease_risk]

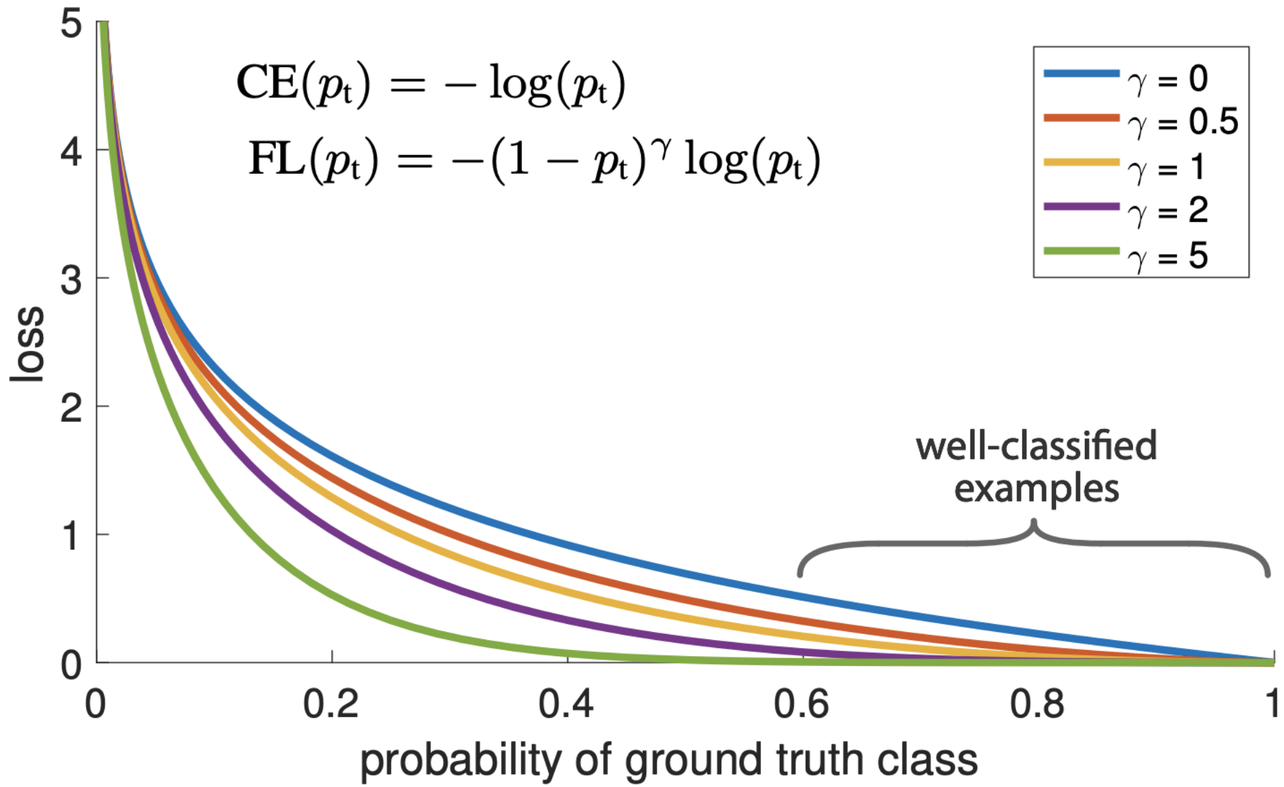
{
'1_00_0',
'2_00_0', '2_a5_2',
'3_00_0', '3_a9_1', '3_a9_2', '3_a9_3', '3_b3_1', '3_b6_1', '3_b7_1', '3_b8_1',
'4_00_0'
'5_00_0', '5_a7_2', '5_b6_1', '5_b7_1', '5_b8_1',
'6_00_0', '6_a11_1', '6_a11_2', '6_a12_1', '6_a12_2' '6_b4_1', '6_b4_3', '6_b5_1'
}

```
risk_with_crop_mask = {}
for target_name in real_targets:
    crop_i, disease_i, risk_i = target_name.split('_')
    crop_i = int(crop_i) - 1
    if crop_i == 0 or crop_i == 3:
        disease_i = disease_encoding.index('0')
    else:
        disease_i = disease_encoding.index(disease_i)
    risk_i = int(risk_i)
    risk_with_crop_mask[disease_i*10 + crop_i] = [True, True, True, True]
for target_name in real_targets:
    crop_i, disease_i, risk_i = target_name.split('_')
    crop_i = int(crop_i) - 1
    if crop_i == 0 or crop_i == 3:
        disease_i = disease_encoding.index('0')
    else:
        disease_i = disease_encoding.index(disease_i)
    risk_i = int(risk_i)
    risk_with_crop_mask[disease_i*10 + crop_i][risk_i] = False
```

Loss: focal loss

CE: 기존의 cross entropy criterion
FL: focal loss

γ: focusing parameter
예측이 쉬운 example에 대한 loss의 비중을 낮추는 역할



분류가 잘 된 example에 대한 loss ⬇
분류가 잘 안된(혹은 어려운) example에 대한 loss ⬆
분류가 잘 안된 example에 초점을 두어 학습하게끔 해준다.

Training (train_torch.py)

```python
loss_fn = FocalLoss()
loss_fn_disease_risk = FocalLoss(ignore_index=0)
optimizer = torch.optim.AdamW([
    {'params': base_model.parameters()},
    {'params': crop_model.parameters()},
    {'params': disease_model.parameters()},
    {'params': risk_model.parameters()}], lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)
writer = SummaryWriter(os.path.join(save_folder, 'log'))
```

각각의 모델을 독립적으로 training 하고 나서 (crop)-(disease)-(risk)의 조건부 확률을 고려하기 위해 세 모델의 loss를 추합

Train set으로 세 분류 모델을 각각 training

```python
while True:
    stop_count += 1
    current_epoch += 1
    base_model.train()
    crop_model.train()
    disease_model.train()
    risk_model.train()

    for data in train_dataloader:
        img, crop, disease, risk, disease_mask, risk_mask = data

        with torch.cuda.amp.autocast():
            img = img.to(DEVICE)
            crop = crop.to(DEVICE)
            disease = disease.to(DEVICE)
            risk = risk.to(DEVICE)
            disease_mask = disease_mask.to(DEVICE)
            risk_mask = risk_mask.to(DEVICE)
            img_processed = base_model(img)
            crop_logits = crop_model(img_processed)
            disease_logits = disease_model(img_processed, disease_mask)
            risk_logits = risk_model(img_processed, disease, risk_mask)
            loss = loss_fn(crop_logits, crop) + loss_fn_disease_risk(disease_logits, disease) + loss_fn_disease_risk(risk_logits, risk) #1.5, 15

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()

        writer.add_scalar(f'Loss/train_step_{k_fold}', loss, step)
        step += 1

    base_model.eval()
    crop_model.eval()
    disease_model.eval()
    risk_model.eval()
```

각 모델 training

각 모델의 Loss 값 계산

EWHA EURON

Training (train_torch.py)

Test set으로 세 분류 모델의 loss값 추합

Catboost를 이용하여 테이블형 정형 데이터를 활용하여 crop과 disease를 예측(risk 예측은 f1 score가 낮아 사용X)

DNN과 catboost의 crop, disease 예측 결과를 앙상블하여 사용(단순 평균)

```
total_loss = 0
preds_crop = []
preds_disease = []
preds_risk = []
preds_crop_ensem = []
preds_disease_ensem = []
preds_risk_ensem = []
with torch.no_grad():
    for idx, data in enumerate(test_dataloader):
        img, crop, disease, risk, disease_mask, risk_mask = data
        img = img.to(DEVICE)
        crop = crop.to(DEVICE)
        disease = disease.to(DEVICE)
        risk = risk.to(DEVICE)
        disease_mask_np = disease_mask.numpy()
        disease_mask = disease_mask.to(DEVICE)
        risk_mask = risk_mask.to(DEVICE)
        with torch.cuda.amp.autocast():
            img_processed = base_model(img)
            crop_logits = crop_model(img_processed)
            disease_logits = disease_model(img_processed, disease_mask)
            risk_logits = risk_model(img_processed, disease, risk_mask)

            loss = loss_fn(crop_logits, crop) + loss_fn_disease_risk(disease_logits, disease) + loss_fn_disease_risk(risk_logits, risk)

            target_df = test_dataset.df.iloc[BATCH_SIZE*idx: BATCH_SIZE*(idx + 1), :].drop(columns=['img_path', 'coordinate', 'crop', 'disease', 'risk'])
            crop_boost_pro = crop_cat.predict_proba(target_df)
            crop_DL_pro = crop_logits.softmax(dim=-1).detach().cpu().numpy()
            crop_idx_list = np.argmax(crop_DL_pro + crop_boost_pro, axis=-1).tolist()
            preds_crop_ensem.extend(crop_idx_list)


            disease_DL_pro = disease_logits.softmax(dim=-1).detach().cpu().numpy()
            target_df = test_dataset.df.iloc[BATCH_SIZE*idx: BATCH_SIZE*(idx + 1), :].drop(columns=['img_path', 'coordinate', 'disease', 'risk'])
            disease_boost_pro = disease_cat.predict_proba(target_df)
            disease_boost_pro = np.ma.MaskedArray(data=disease_boost_pro, mask=disease_mask_np)
            disease_idx_list = np.argmax(disease_DL_pro + disease_boost_pro.data, axis=-1).tolist()
            preds_disease_ensem.extend(disease_idx_list)


        total_loss += loss * len(img)
        preds_crop.extend(np.argmax(crop_DL_pro, axis=-1))
        preds_disease.extend(np.argmax(disease_DL_pro, axis=-1))
        preds_risk.extend(torch.argmax(risk_logits, dim=-1).detach().cpu().tolist())
```

Training (train_torch.py)

```
preds_total = [disease_idx*100 + crop_idx*10 + risk_idx for crop_idx, disease_idx, risk_idx in zip(preds_crop, preds_disease, preds_risk)]
preds_total_ensem = [disease_idx*100 + crop_idx*10 + risk_idx for crop_idx, disease_idx, risk_idx in zip(preds_crop_ensem, preds_disease_ensem, preds_risk)]
f1_score_crop = f1_score(answer_crop, preds_crop, average='macro', labels=total_crop_labels)
f1_score_crop_ensem = f1_score(answer_crop, preds_crop_ensem, average='macro', labels=total_crop_labels)
f1_score_disease = f1_score(answer_disease, preds_disease, average='macro', labels=total_disease_labels)
f1_score_disease_ensem = f1_score(answer_disease, preds_disease_ensem, average='macro', labels=total_disease_labels)
f1_score_risk = f1_score(answer_risk, preds_risk, average='macro', labels=total_risk_labels)
f1_score_total = f1_score(answer_total, preds_total, average='macro')
f1_score_total_ensem = f1_score(answer_total, preds_total_ensem, average='macro')

acc_score_crop = accuracy_score(answer_crop, preds_crop)
acc_score_disease = accuracy_score(answer_disease, preds_disease)
acc_score_risk = accuracy_score(answer_risk, preds_risk)
#acc_score_total = accuracy_score(answer_total, preds_total)
```

각각의 classifier의 성능이 뛰어나다고 해서 결과인 (crop)-(disease)-(risk)를 정확히 맞춘다고 보장 X

-> validation 환경에서 이들을 다시 조합하여 f1 score를 산출하여 좋은 성능의 모델을 선택

```
if max_f1_total < f1_score_total:
    stop_count = 0
    max_f1_total = f1_score_total
    torch.save({'base_model': base_model.module.state_dict(),
                'crop_model': crop_model.module.state_dict(),
                'disease_model': disease_model.module.state_dict(),
                'risk_model': risk_model.module.state_dict(),}, os.path.join(save_folder, f'max_f1_total_{k_fold}.pt'))

if max_f1_ensemble < f1_score_total_ensem:
    stop_count = 0
    max_f1_ensemble = f1_score_total_ensem
    torch.save({'base_model': base_model.module.state_dict(),
                'crop_model': crop_model.module.state_dict(),
                'disease_model': disease_model.module.state_dict(),
                'risk_model': risk_model.module.state_dict(),}, os.path.join(save_folder, f'max_f1_ensem_{k_fold}.pt'))

if max_f1_total_risk_care < f1_score_total and f1_score_crop == 1 and f1_score_disease == 1:
    stop_count = 0
    max_f1_total_risk_care = f1_score_total
    print(f'DL only F1 Score: {max_f1_total_risk_care}')
    torch.save({'base_model': base_model.module.state_dict(),
                'crop_model': crop_model.module.state_dict(),
                'disease_model': disease_model.module.state_dict(),
                'risk_model': risk_model.module.state_dict(),}, os.path.join(save_folder, f'max_f1_total_risk_care_{k_fold}.pt'))

if max_f1_ensemble_risk_care < f1_score_total_ensem and f1_score_crop_ensem == 1 and f1_score_disease_ensem == 1:
    stop_count = 0
    max_f1_ensemble_risk_care = f1_score_total_ensem
    print(f'Ensemble F1 Score: {max_f1_ensemble_risk_care}')
    torch.save({'base_model': base_model.module.state_dict(),
                'crop_model': crop_model.module.state_dict(),
                'disease_model': disease_model.module.state_dict(),
                'risk_model': risk_model.module.state_dict(),}, os.path.join(save_folder, f'max_f1_ensem_risk_care_{k_fold}.pt'))
```

predict

crop classifier prediction

disease classifier prediction

risk classifier prediction

```
with torch.no_grad():
    with torch.cuda.amp.autocast():
        if current_label == 'crop':
            list_for_crop_prediction = []
            for img_idx, data in tqdm(enumerate(test_dataloader)):
                img = data
                img = img.to(DEVICE)
                img_processed = base_model(img)
                img_tensors.append(img_processed.detach().cpu())
                crop_DL_pro = crop_model(img_processed).softmax(dim=-1).detach().cpu().numpy()
                target_df = test_dataset.df.iloc[BATCH_SIZE*img_idx: BATCH_SIZE*(img_idx + 1), :].drop(columns=['img_path'])
                list_for_crop_prediction.append((crop_DL_pro, target_df, crop_cat))

            with Pool() as pool:
                preds_crop = pool.map(ensemble_cal, list_for_crop_prediction)
            preds_crop = [index for index_list in preds_crop for index in index_list]

        elif current_label == 'disease':
            list_for_disease_prediction  = []
            for img_idx, data in tqdm(enumerate(test_dataloader)):
                disease_mask = data
                img_processed = img_tensors[img_idx].to(DEVICE)
                disease_DL_pro = disease_model(img_processed, disease_mask.to(DEVICE)).softmax(dim=-1).detach().cpu().numpy()
                target_df = test_dataset.df.iloc[BATCH_SIZE*img_idx: BATCH_SIZE*(img_idx + 1), :].drop(columns=['img_path'])
                list_for_disease_prediction.append((disease_DL_pro, target_df, disease_cat, disease_mask))

            with Pool() as pool:
                preds_disease = pool.map(ensemble_cal_with_mask, list_for_disease_prediction)
            preds_disease = [index for index_list in preds_disease for index in index_list]

        elif current_label == 'risk':
            for img_idx, data in tqdm(enumerate(test_dataloader)):
                disease, risk_mask = data
                img_processed = img_tensors[img_idx].to(DEVICE)
                risk_mask_np = risk_mask.numpy()
                disease = disease.to(DEVICE)
                risk_mask = risk_mask.to(DEVICE)

                risk_idx_list = torch.argmax(risk_model(img_processed, disease, risk_mask), dim=-1).detach().cpu().tolist()
                preds_risk.extend(risk_idx_list)
```

Conclusion

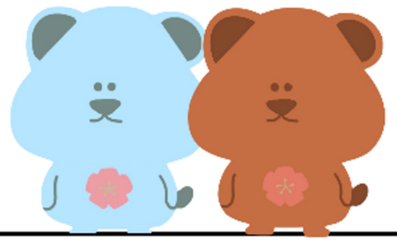
✓ 데이터 관찰 후, 예측 과제를 (crop)-(disease)-(risk) 순의 조건부 확률로 바꿔 예측 정확도를 높임

✓ Catboost
✓ Mask

} 를 사용하여 조건부 확률을 고려한 예측 정확도를 높임

✓ Focal_loss를 사용하여 잘 학습되지 않은 example에 초점을 맞추어 학습

THANK YOU



EWHA EURON