

Computer Vision Anomaly Detection

이미지 이상치 탐지 알고리즘 경진대회

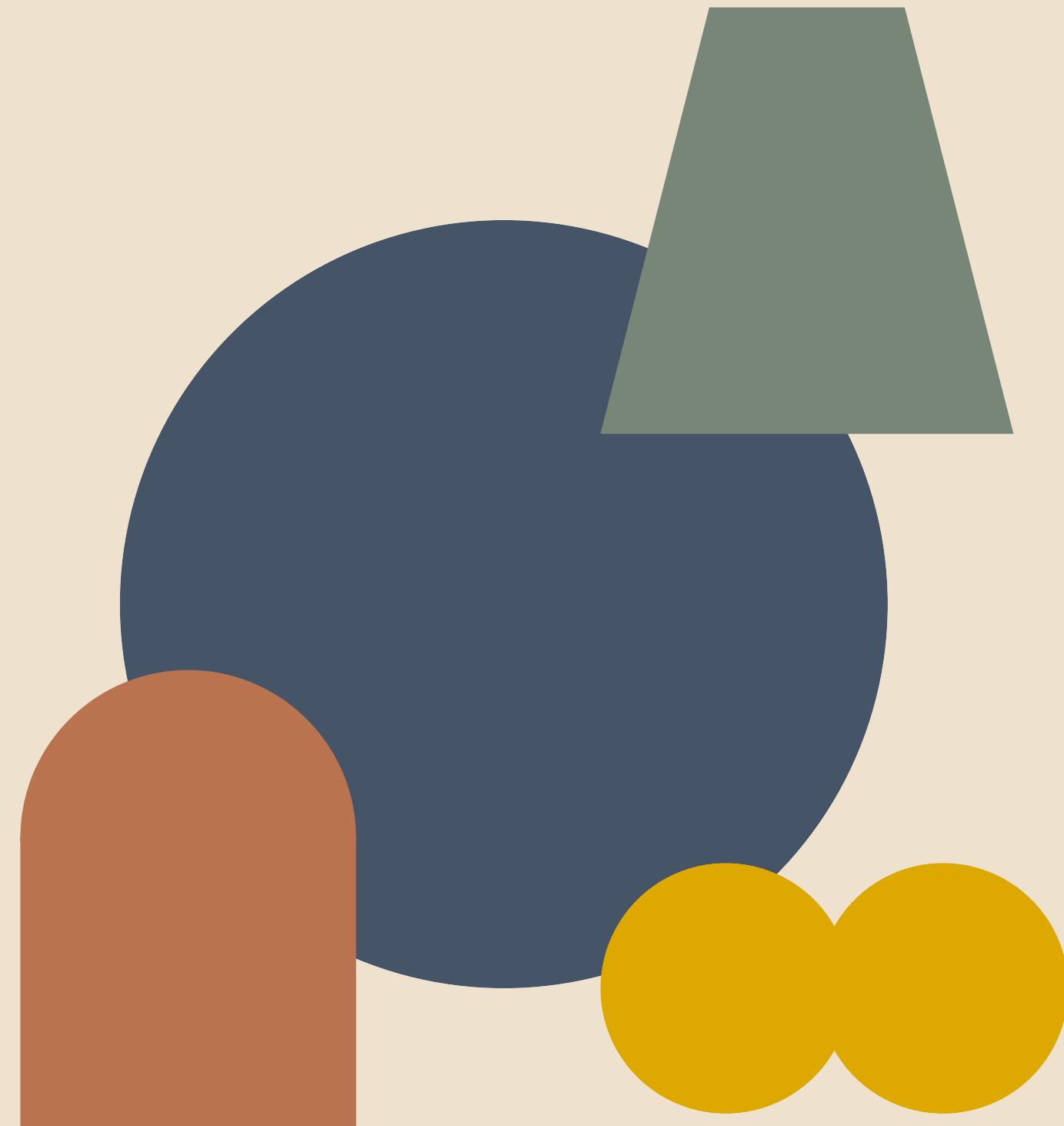
6조 : 서지연, 김태호, 조재성, 김민채

Contents

-
- 01 프로젝트 소개
 - 02 이상치 탐지
 - 03 데이터
 - 04 학습방법
 - 05 코드리뷰
 - 06 코드결과
 - 07 마무리

01

프로젝트 소개

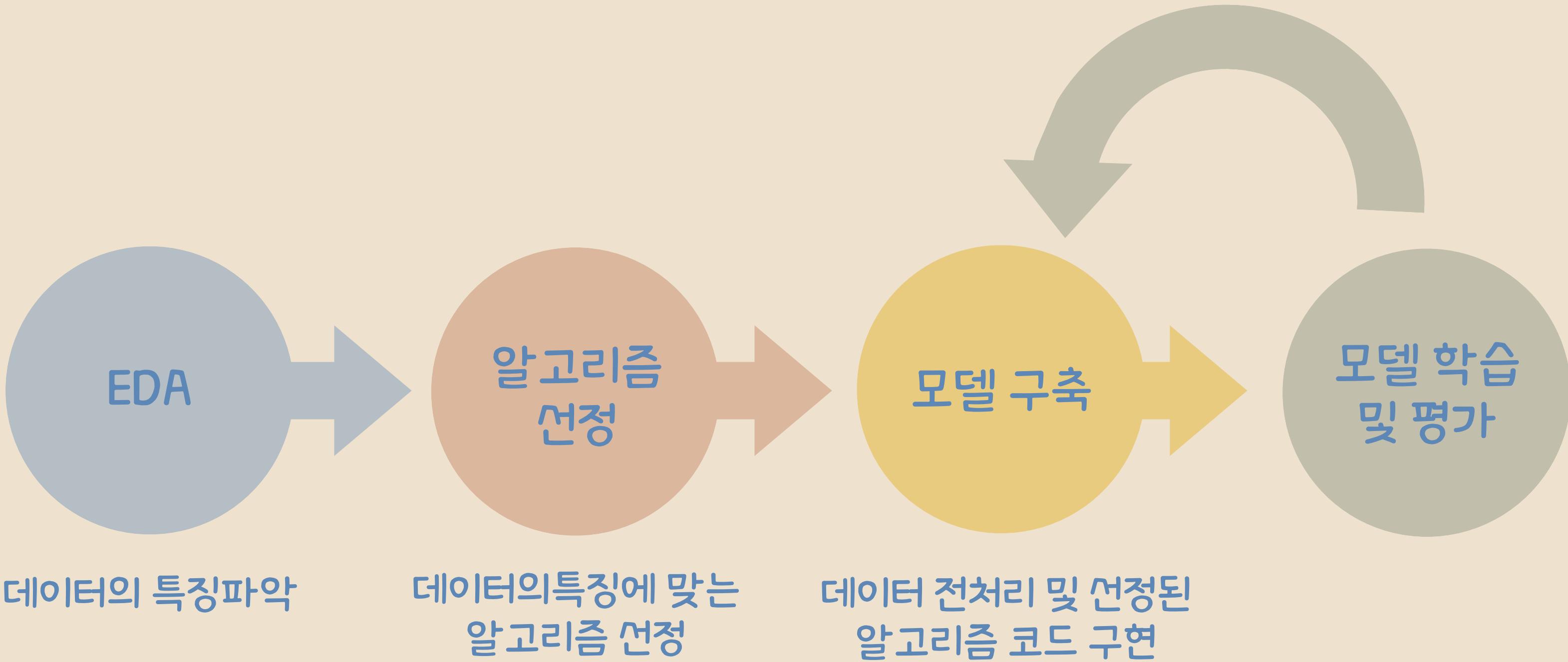


01 - 1. 프로젝트 개요

사물의 종류와 상태를 분류하는 컴퓨터 비전 알고리즘 개발

Dacon에서 제공하는 MVtec AD Dataset을 사용하여 사물의 종류를 분류하고, 정상 샘플과 비정상(이상치) 샘플을 분류

01 - 2. 프로세스



01 - 3. 역할분담

머지연

알고리즘 구축 및 데이터전처리, PPT 제작

김태호

알고리즘 구축 및 모델학습

조재성

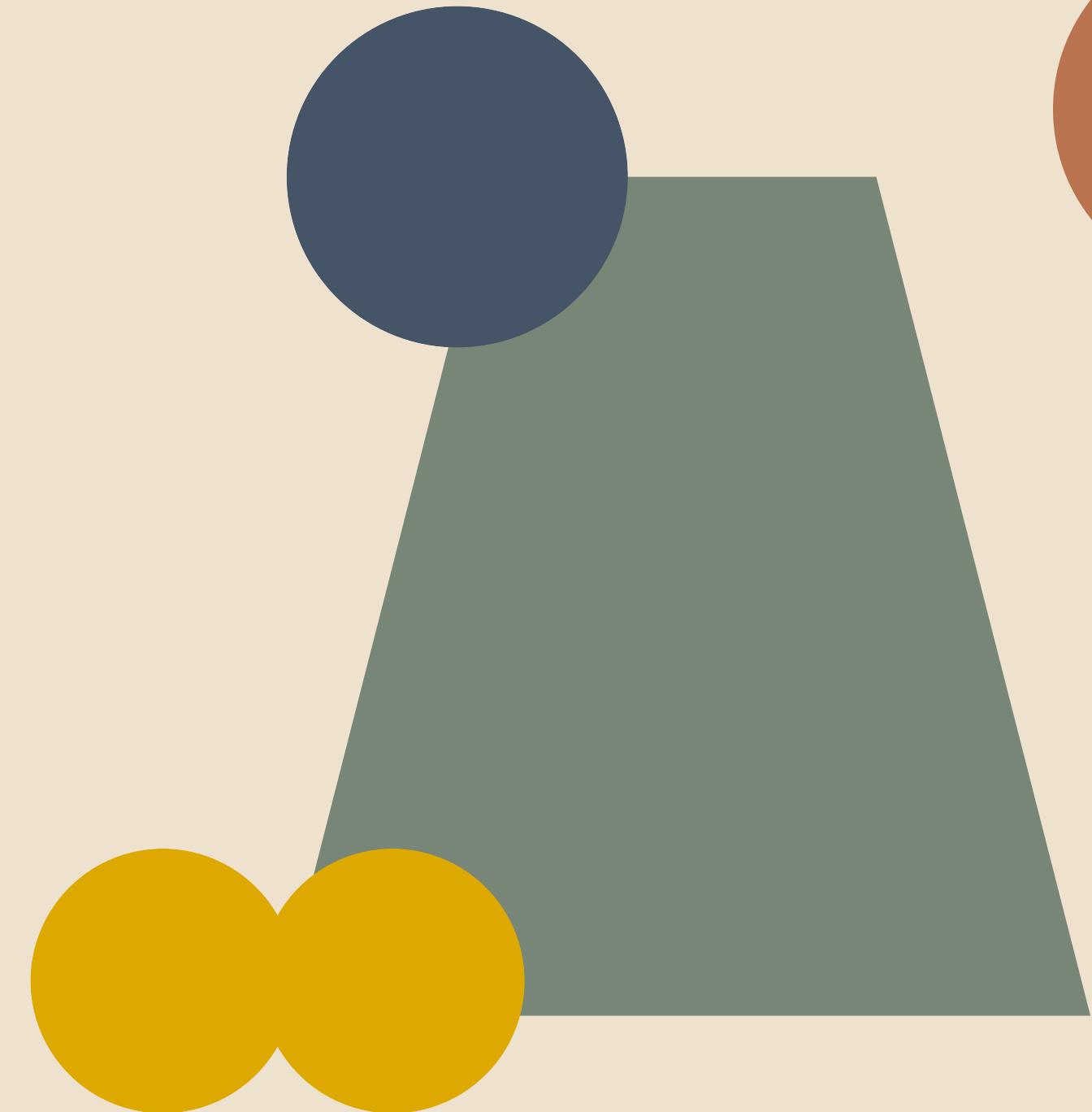
알고리즘 체크 및 모델학습

김민채

모델학습, 발표

02

이상치 탐지



02-1. 이상치 탐지란?

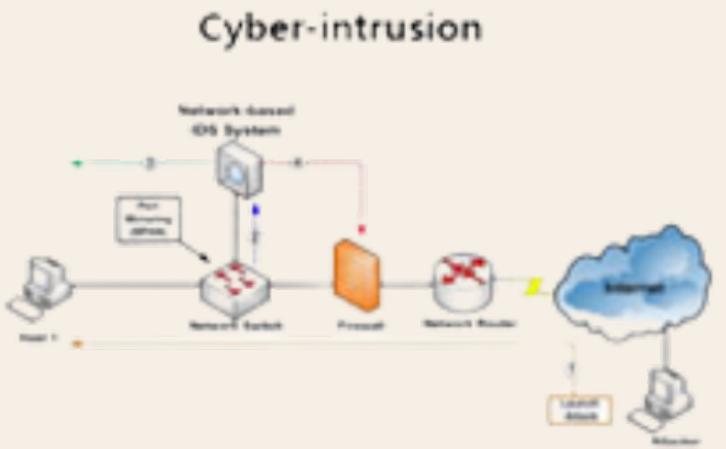
이상치 탐지란,

데이터안에서 *anomaly*, *outlier*, *abnormal* 과 같이 예상치 못한 패턴을 찾는 일련의 활동을 말합니다.

Anomaly Detection은 매우 많은 정상 데이터에서 극소수의 비정상 데이터를 구별하는 것이라 할 수 있습니다.



02- 2. 이상치 탐지 적용 분야



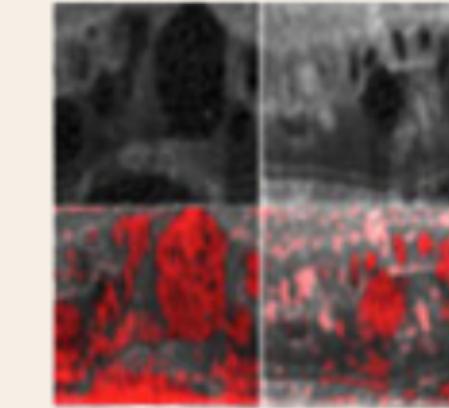
시스템 침입 탐지



보험 불법행위 검출



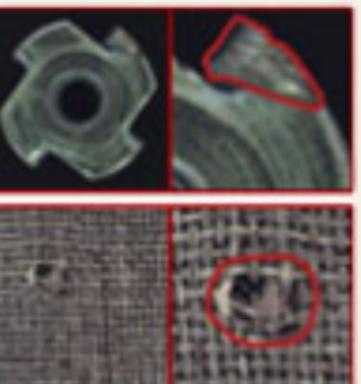
악성 코드 검출



의학데이터 이상 탐지



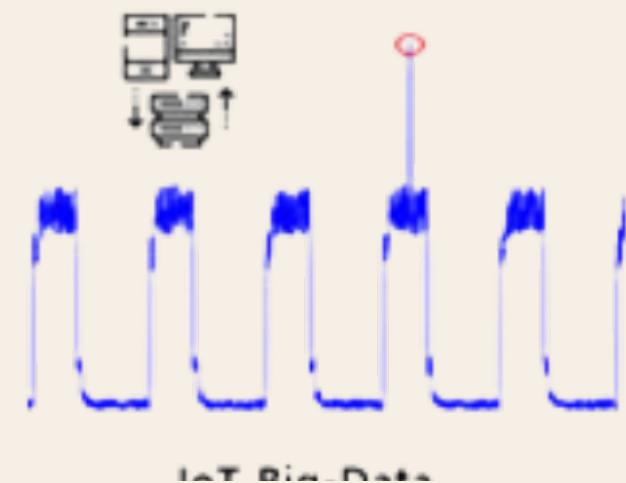
file



파일 이상 탐지



산업 제조업 이상 탐지



센서데이터 이상 탐지

02- 3. 이상치 탐지: 알고리즘

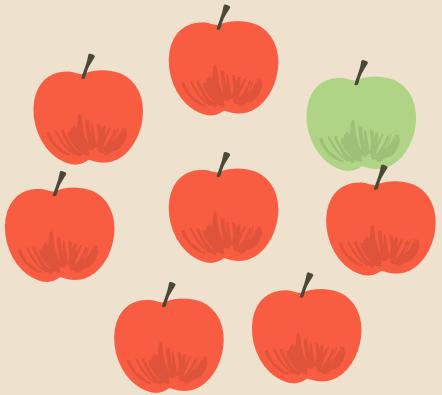
이상치 탐지 기법은 크게 분류기반, NN기반, 군집화기반, 통계적 기법, 스펙트럴 기법으로 나뉜다.

분류기반 - 신경망

학습(훈련) 할 데이터에 정답 label이 있어서 분류기(classifier)를 학습한 뒤, 학습된 모형으로 새로운 데이터가 각각의 클래스에 속할 확률을 예측하는 방법

정답 label의 개수에 따라 multi-class 와 one-class 로 나뉜다.

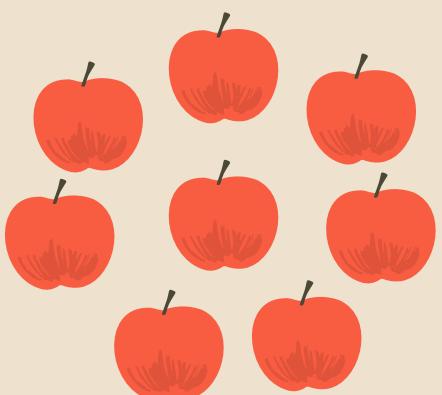
02- 3. 이상치 탐지: 학습 데이터 관점에서



Supervised Anomaly Detection

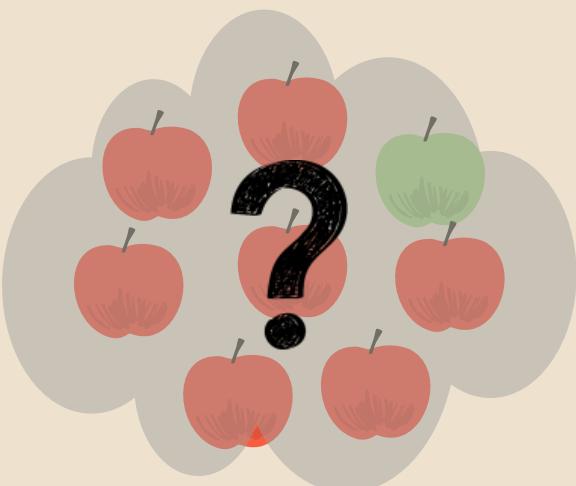
학습 시, Labeling이 된 정상/비정상 데이터를 모두 사용한 경우

- Data Imbalanced Problem 해결 필요



Semi-Supervised Anomaly Detection

학습 시, Labeling이 된 정상 데이터만을 사용한 경우



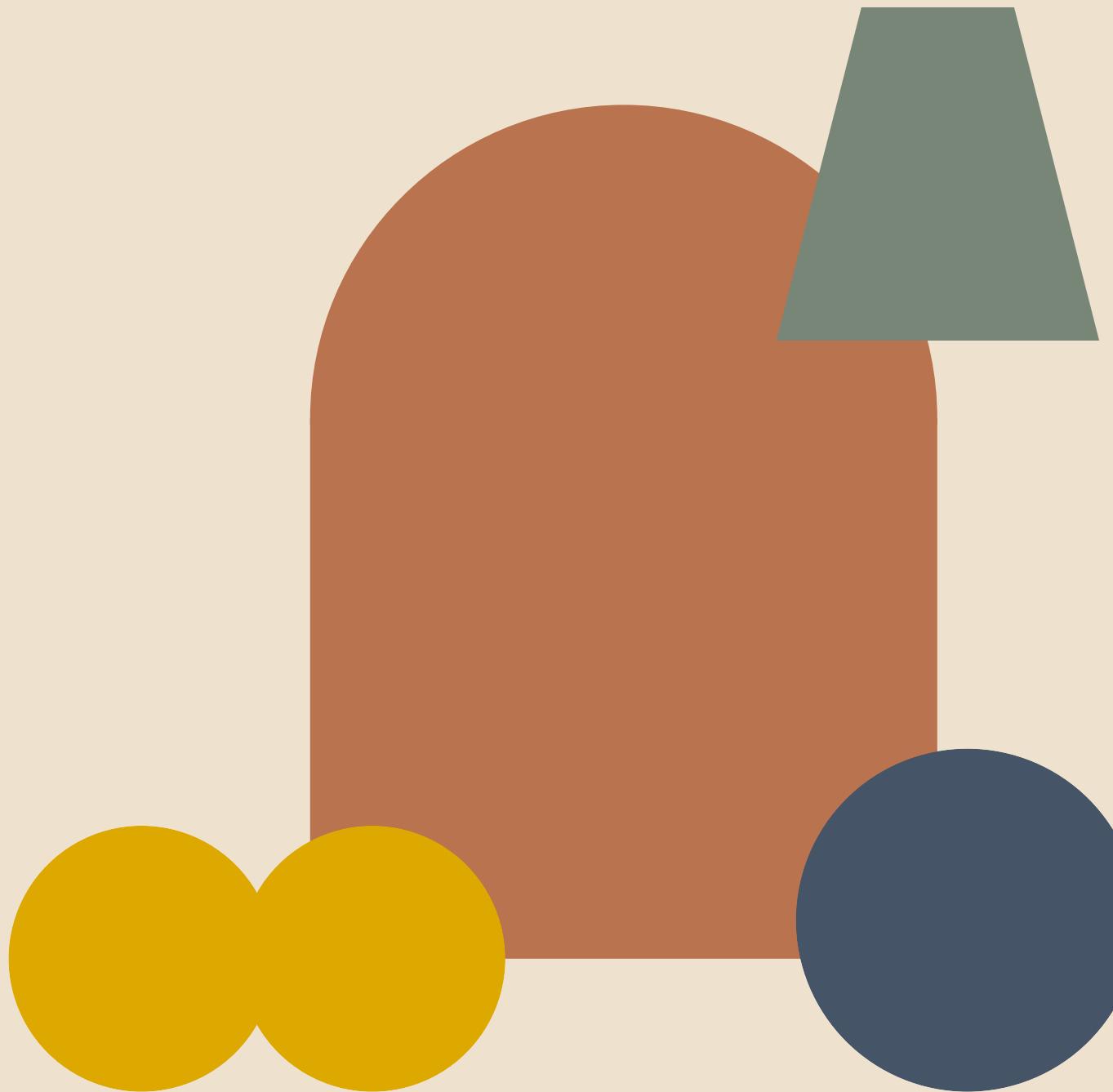
Unsupervised Anomaly Detection

학습 시, 데이터가 대부분 정상으로 이루어져 있다고 가정

- 데이터에 확실한 Labeling이 없을 때 용이함

03

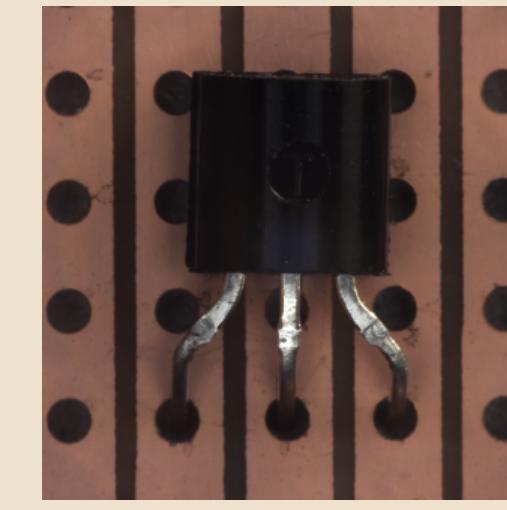
데이터



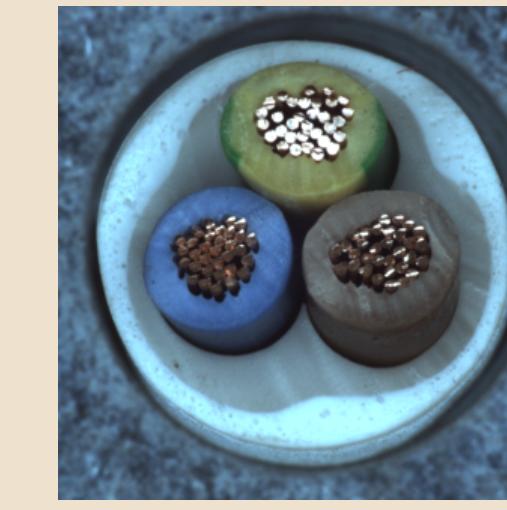
03- 1. 데이터 분석

제공된 데이터 속에는 15개의 class와 88개의 label로 분류되어 있다.

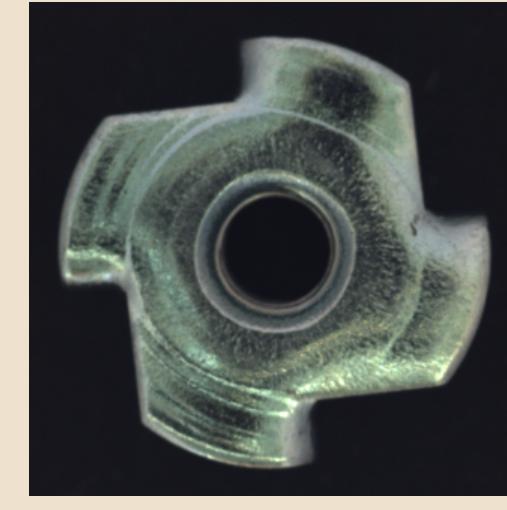
index	file_name	class	state	label
0	10000.png	transistor	good	transistor-good
1	10001.png	capsule	good	capsule-good
2	10002.png	transistor	good	transistor-good
...
8	10008.png	cable	bent_wire	cable-bent_wire
9	10009.png	transistor	good	transistor-good
10	10010.png	carpet	hole	carpet-hole



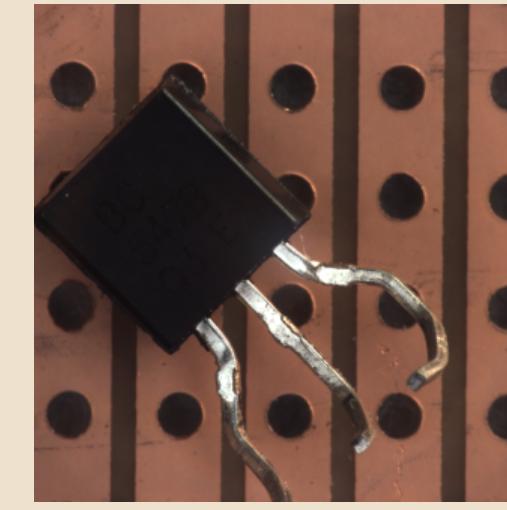
transistor-good



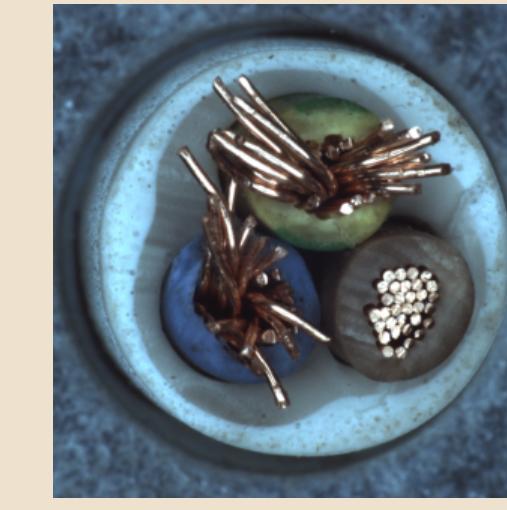
cable-good



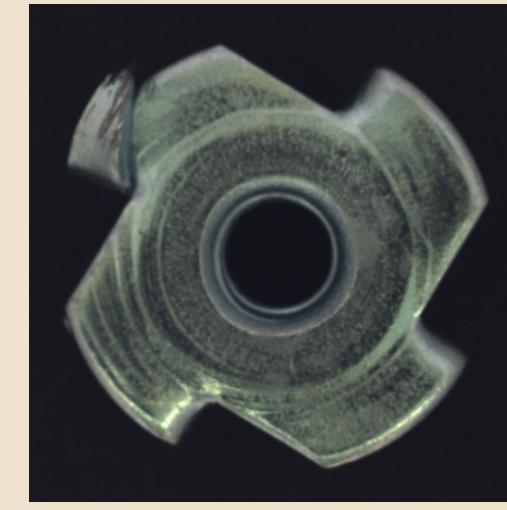
metal_nut-good



transistor-misplaced



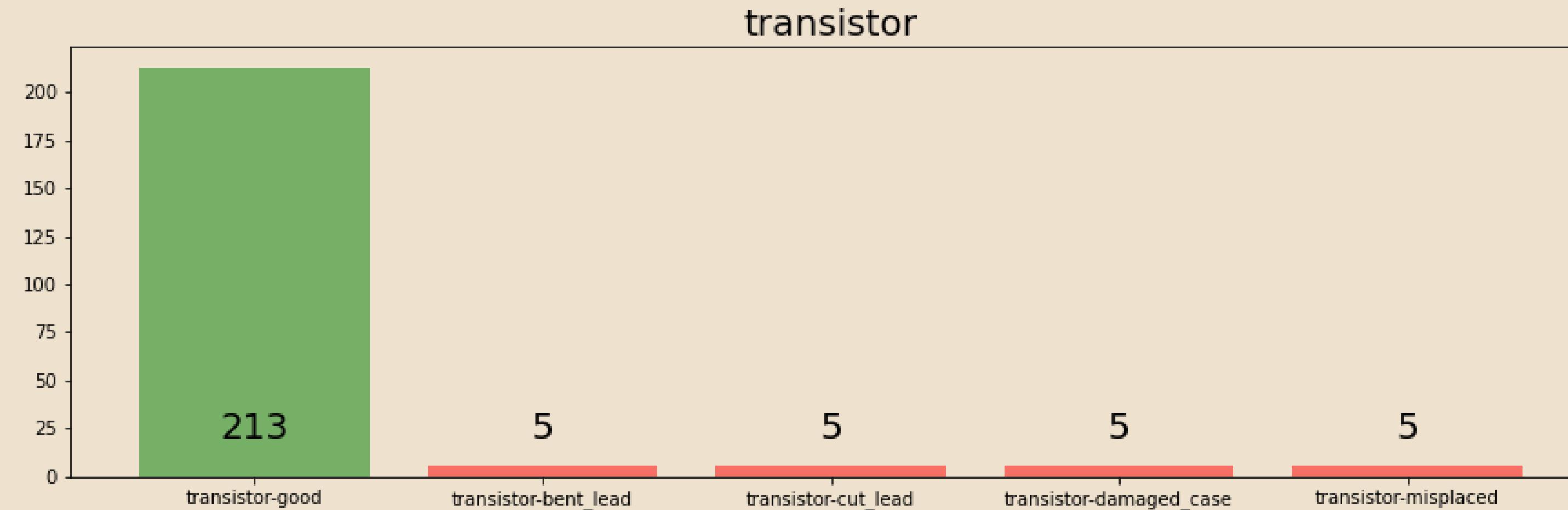
cable-bent_wire



metal_nut-bent

03- 1. 데이터 분석

Train Dataframe의 class별 분포 확인 결과,
정상, 비정상 데이터의 Data Imbalanced Problem 을 확인

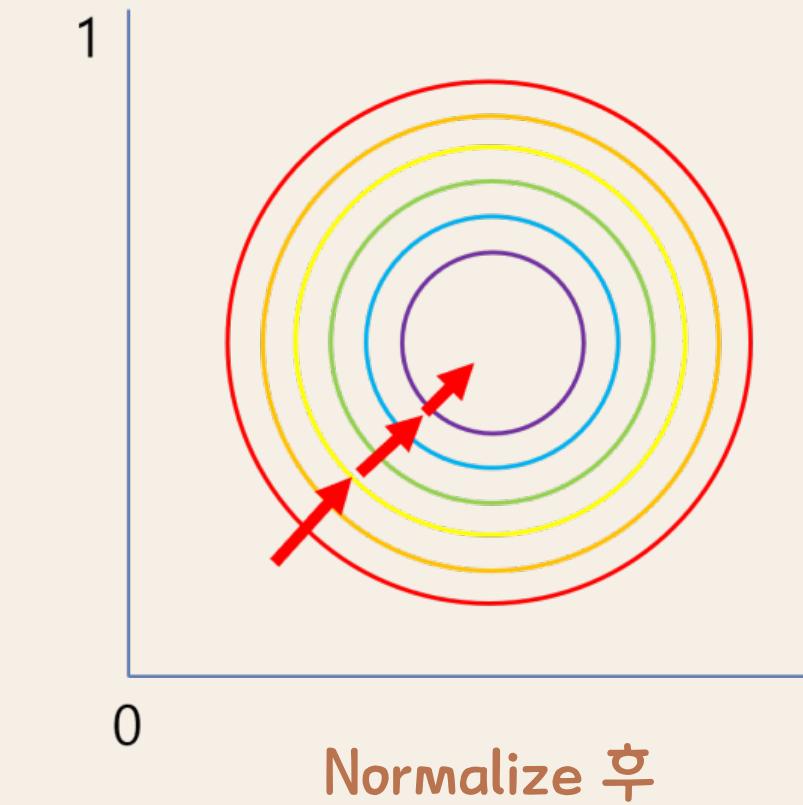


Train Dataframe의 transister 데이터 분포 확인 결과

03 - 2. 데이터 전처리

Normalize

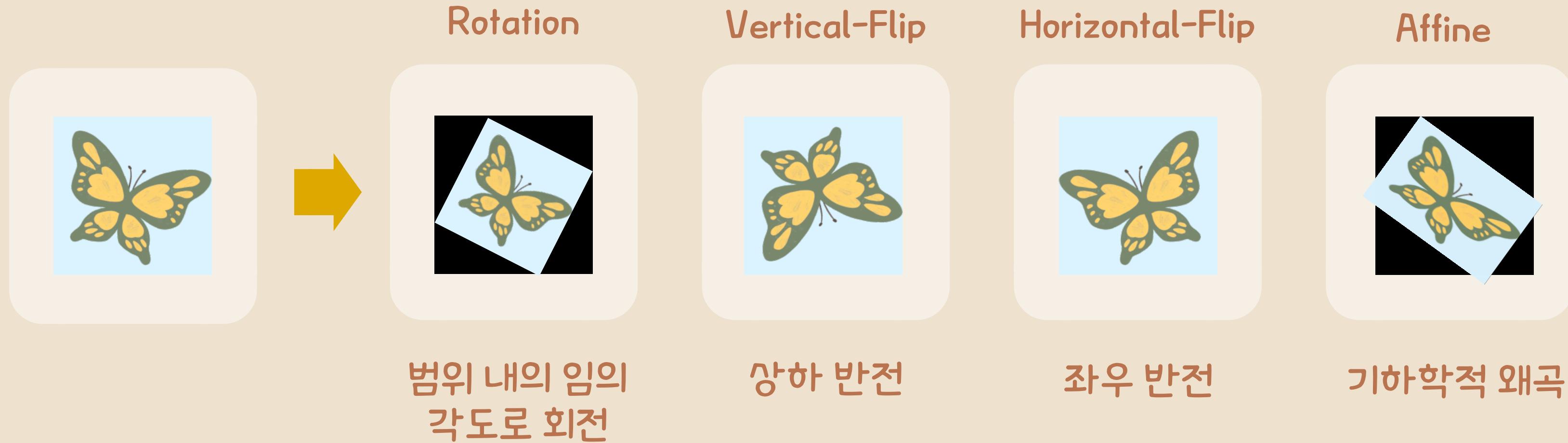
- local optimum(minimum)에 빠지는 것을 방지한다.
- 모든 입력을 표준 규모로 정규화함으로써 네트워크가 각 입력 노드에 대한 최적의 매개변수를 보다 빠르게 학습할 수 있다.



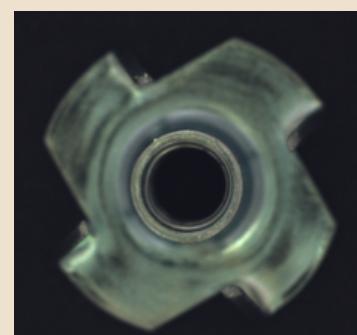
최적의 매개변수에 보다
빠르게 다가가는 것을 확인

03 - 2. 데이터 전처리

Augmentation : 데이터를 부풀려서 모델의 성능을 증폭시킨다.



※ metal_nut 데이터는 flip 이상치가 있으니 주의

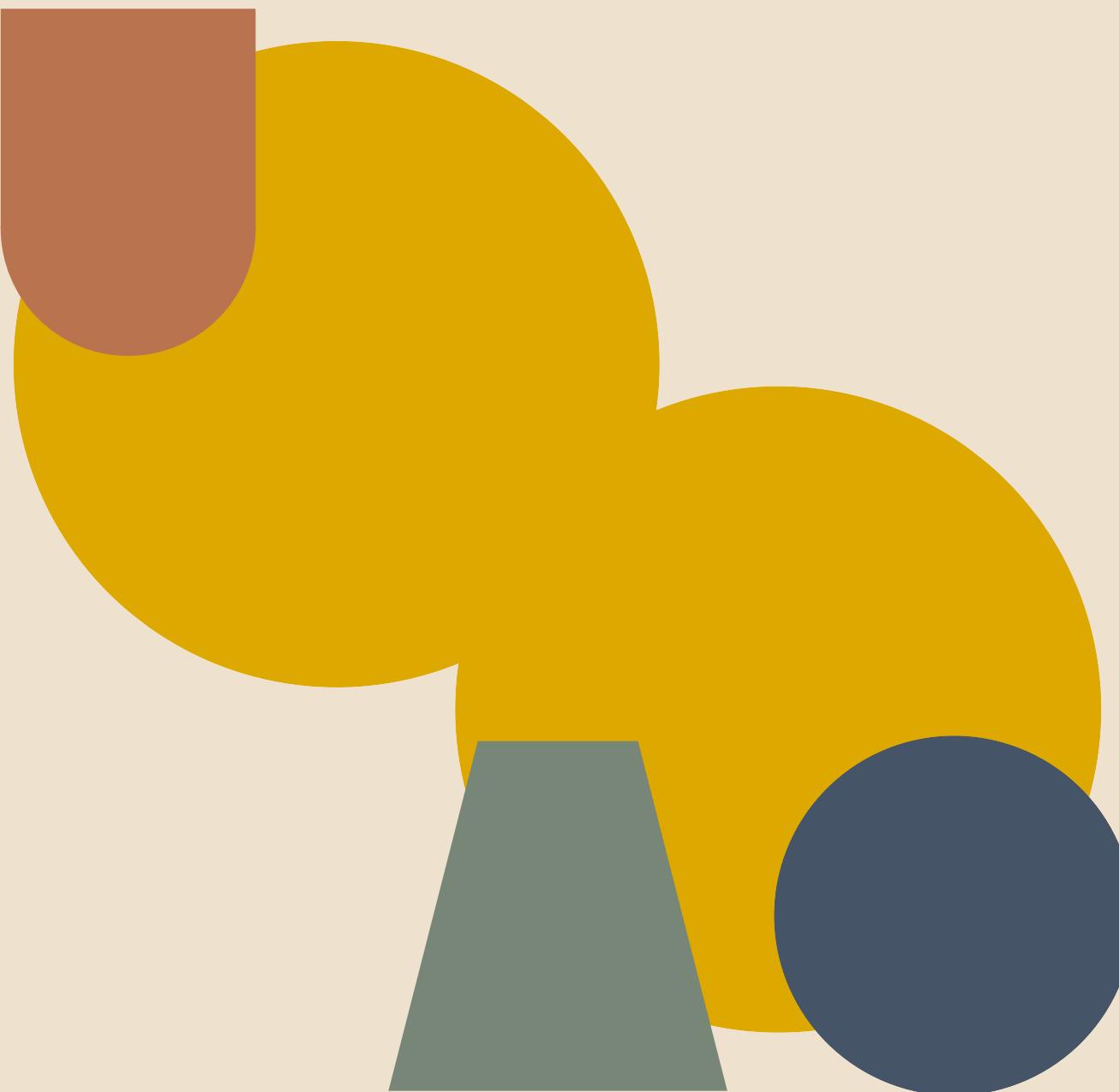


metal_nut-flip

 metal_nut의 경우 flip을 제외
하여 Augmentation 진행

04

학습방법



04 - 1. 교차검증

K-Fold,

K-fold 교차검증은 학습세트와 검증세트를
나눠 반복해서 검증한다.

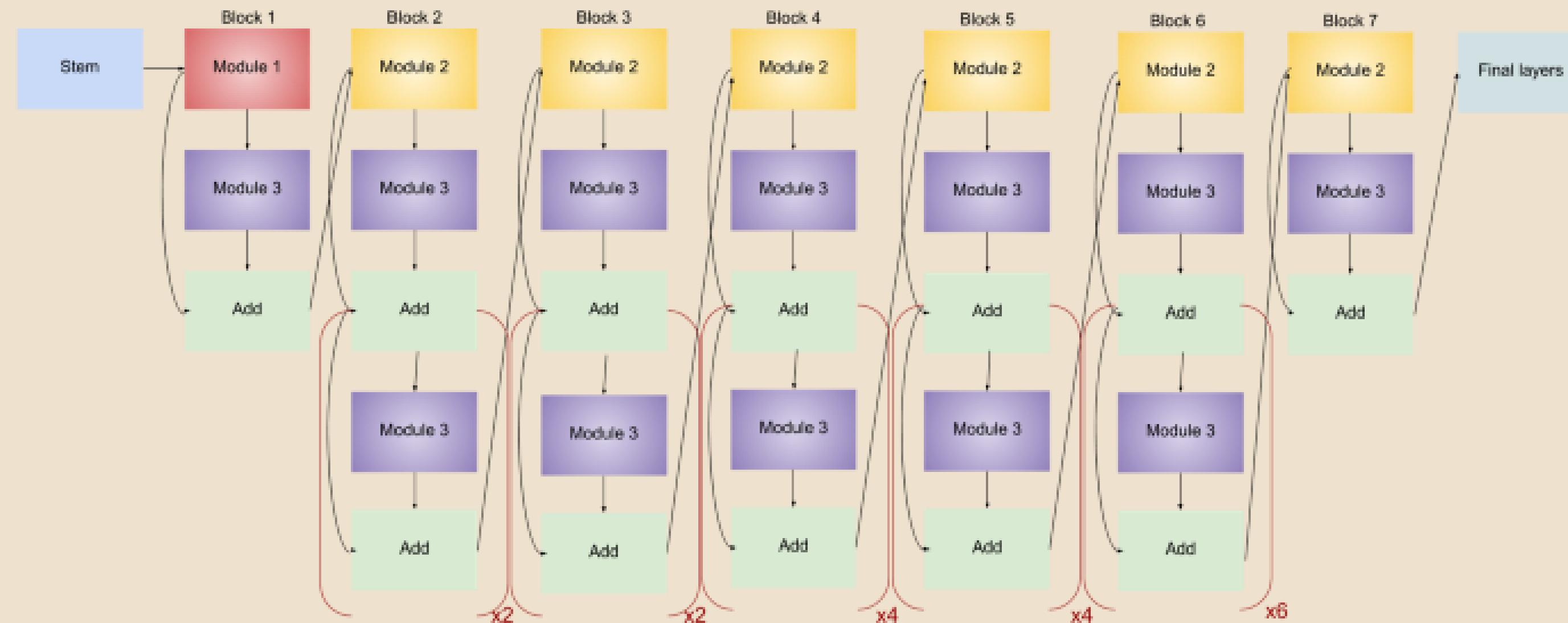
Stratified K-Fold,

target에 속성 값의 개수를 동일하게 가져
감으로써 K-fold의 문제점을 보완한다.



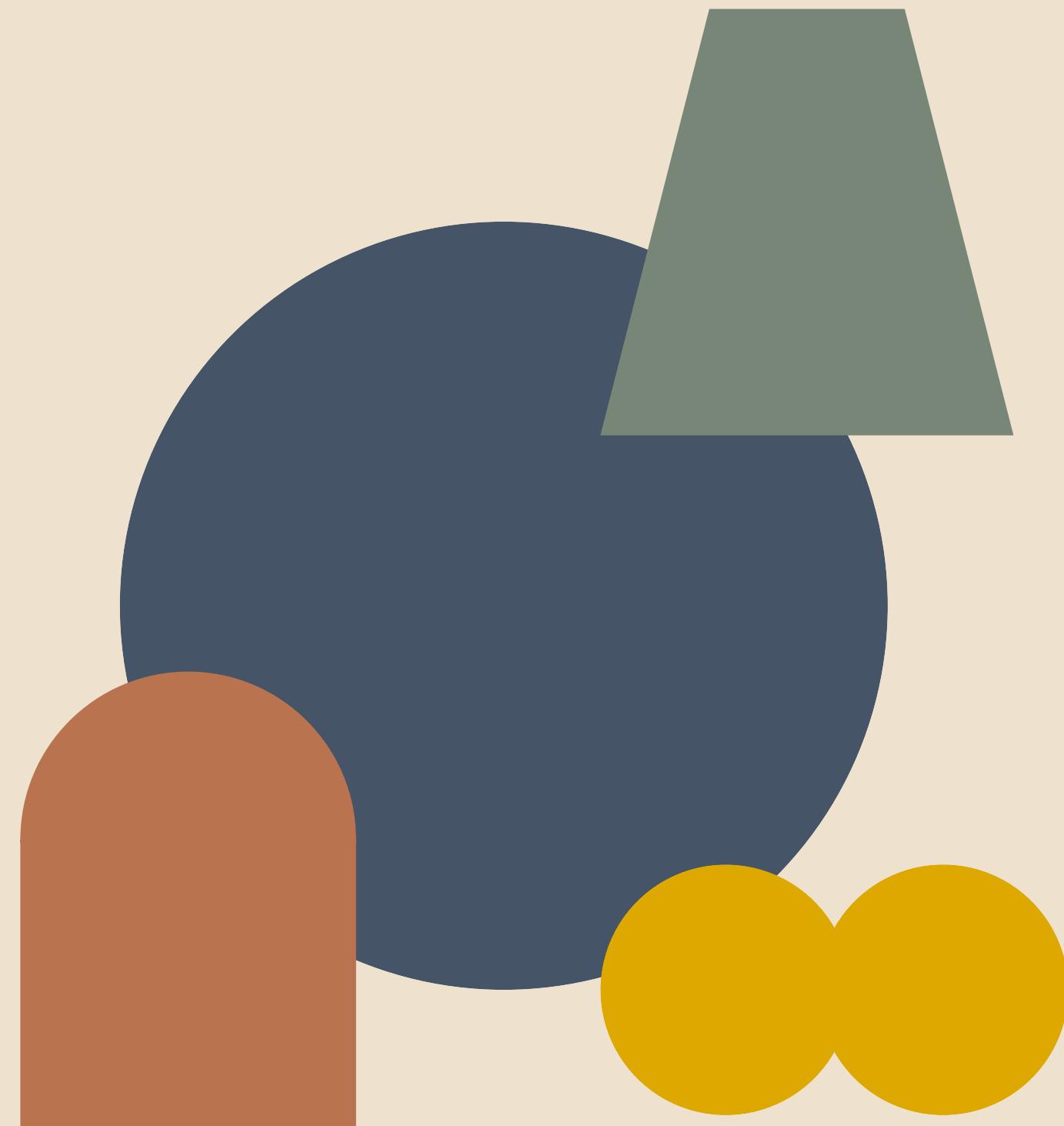
04 - 2. Pre-Trained Model

efficient B4 model 구조



05

코드리뷰



05 - 1. Settings



```
# timm으로 Model Load
import timm

# Pytorch로 Data Load
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torchvision.transforms as transforms

# sklearn Model Load
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, accuracy_score
```

모듈 선언

05 - 2. Data Loading

```
● ● ●

def img_load(path):
    img = cv2.imread(path)[:, :, ::-1]
    img = cv2.resize(img, (384, 384), interpolation = cv2.INTER_AREA)
    return img

# glob으로 만들었던 이미지 file 이름 list를 사용하여 load하고 리스트를 만든다.
train_imgs = [img_load(m) for m in tqdm(train_png)]
test_imgs = [img_load(n) for n in tqdm(test_png)]

# 위에서 만든 이미지 리스트를 npy 형식으로 저장한다.
np.save(path + 'data/train_imgs_384', np.array(train_imgs))
np.save(path + 'data/test_imgs_384', np.array(test_imgs))

# 저장한 이미지를 불러온다
train_imgs = np.load(path + 'train_imgs_384.npy')
test_imgs = np.load(path + 'test_imgs_384.npy')
```

Data Loading 전처리

05 - 2. Data Loading

```
● ● ●

# RGB 평균((384, 384, 3) => (3)) 리스트
# 리스트의 각 RGB별 평균을 내서 사용

meanRGB = [np.mean(x, axis=(0,1)) for x in train_imgs]
stdRGB = [np.std(x, axis=(0,1)) for x in train_imgs]

meanR = np.mean([m[0] for m in meanRGB])/255
meanG = np.mean([m[1] for m in meanRGB])/255
meanB = np.mean([m[2] for m in meanRGB])/255

stdR = np.mean([s[0] for s in stdRGB])/255
stdG = np.mean([s[1] for s in stdRGB])/255
stdB = np.mean([s[2] for s in stdRGB])/255

print("train 평균",meanR, meanG, meanB)
print("train 표준편차",stdR, stdG, stdB)
```

Normalize 값 계산

05 - 3. Class 정의 ()

```
● ● ●

class Custom_dataset(Dataset):
    ... (생략)...
    # train mode에서 idx==7 (metal_nut) 일 경우에는 flip작업을 제외한다.
    if self.classes[idx]==7:
        train_transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean = [0.433038, 0.403458, 0.394151],
                                std = [0.181572, 0.174035, 0.163234]),
            transforms.RandomAffine((-45, 45)),
            transforms.RandomRotation((0,80)) # 이미지 Rotate
        ])
        img = train_transform(img)
    else :
        train_transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean = [0.433038, 0.403458, 0.394151],
                                std = [0.181572, 0.174035, 0.163234]),
            transforms.RandomAffine((-45, 45)),
            transforms.RandomVerticalFlip(p=0.5), # 이미지 수직 대칭 허용
            transforms.RandomHorizontalFlip(p=0.5), # 이미지 수평 대칭 허용
            transforms.RandomRotation((0,80)) # 이미지 Rotate
        ])
        img = train_transform(img)
```

Dataset 정의 Class

05 - 3. Class 정의 (2)

```
● ● ●

class Network(nn.Module):
    def __init__(self, mode = 'train'):
        super(Network, self).__init__()
        self.mode = mode
    if self.mode == 'train':
        # 전이학습 모델 efficientnet_b4 사용
        self.model = timm.create_model('efficientnet_b4', pretrained=True,
                                       num_classes=88, drop_path_rate = 0.2) # dropout 20%
    if self.mode == 'test':
        self.model = timm.create_model('efficientnet_b4', pretrained=True,
                                       num_classes=88, drop_path_rate = 0)

    def forward(self, x):
        x = self.model(x)
        return
```

모델정의 Class

05 - 4. Model

```
● ● ●

cv = StratifiedKFold(n_splits = 5, random_state = 2022, shuffle=True)
batch_size = 32
epochs = 70

for idx, (train_idx, val_idx) in enumerate(cv.split(train_imgs, np.array(train_labels))):

    # Train
    train_dataset = Custom_dataset(np.array(train_imgs), np.array(train_labels), np.array(train_classes))
    train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)

    # Val
    val_dataset = Custom_dataset(np.array(val_imgs), np.array(val_labels), np.array([]), mode='test')
    val_loader = DataLoader(val_dataset, shuffle=True, batch_size=batch_size)

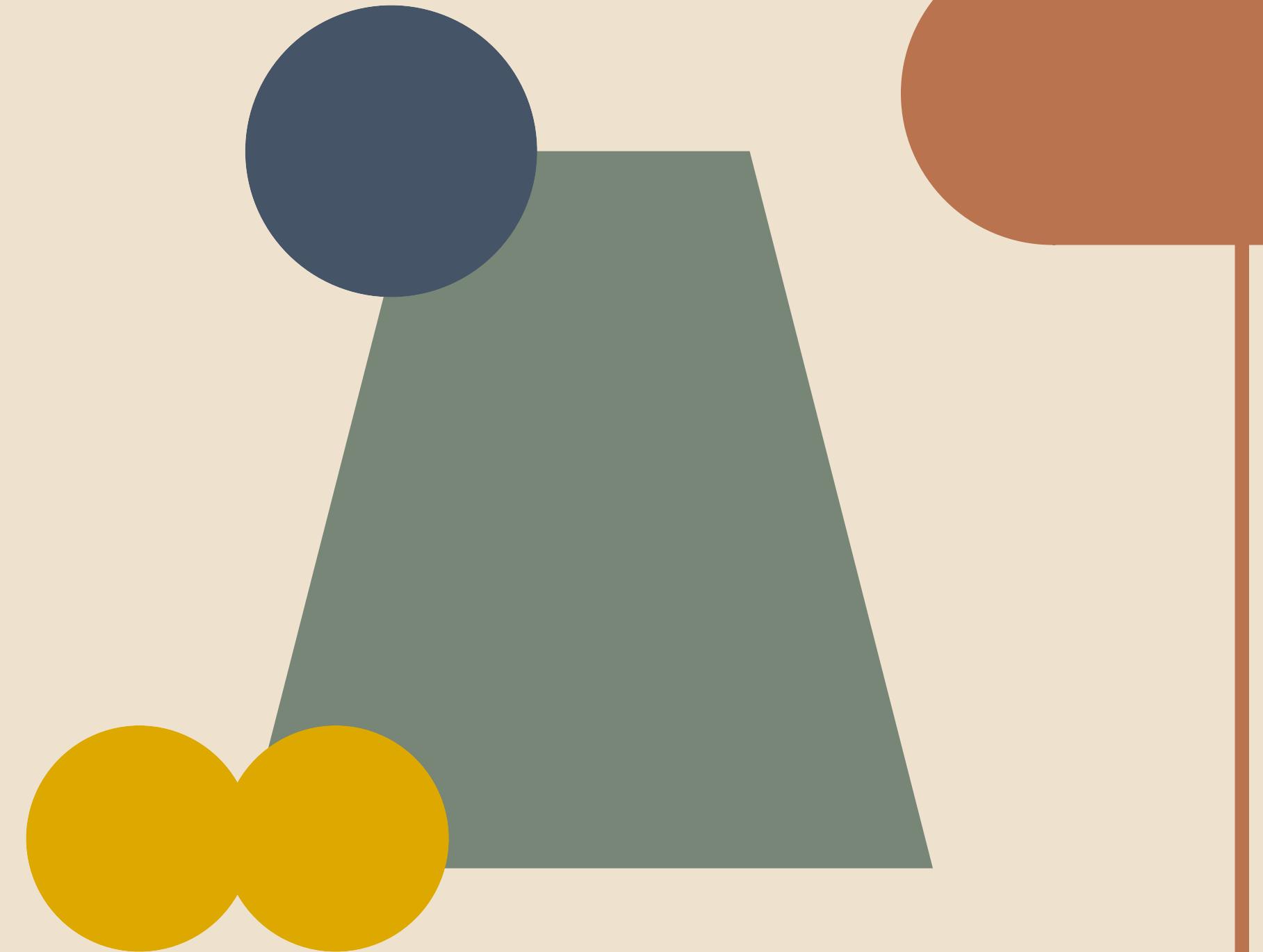
    model = Network().to(device)
    optimizer = torch.optim.Adadelta(model.parameters())

    for epoch in range(epochs):
        for batch in (train_loader):
            ... (학습) ...
        for batch in (val_loader):
            ... (평가, torch_save) ...
        if early_stopping == 20:
            break
```

Stratified 5-Fold 알고리즘을 사용한 모델 구현

06

코드결과



06- 0. 코드 평가 기준

f1-score : Recall · Precision의 조화평균, 이상치 탐지 평가에 적합

Recall : 실제로 True인 데이터를 모델이 True라고 인식한 데이터의 수

Precision : 모델이 True로 예측한 데이터 중 실제로 True인 데이터의 수

$$F1\ Score = 2 \times \frac{recall \times precision}{recall + precision}$$

06- 1. Non-CV vs 5-Fold

efficientb3 Non-CV f1-score : 0.755853178

efficientb3 5-Fold f1-score : 0.8159042465

Non-CV보다 5-Fold의 결과가 좋으니 최종 모델로는 5-Fold로 결정

06- 2. Efficient B3 vs B4

adadelta Non-CV

efficientb3 Non-CV f1-score : 0.7957753882

efficientb4 Non-CV f1-score : 0.8113580182

B3보다 B4의 결과가 더 좋으니 B4로 결정

06 - 3. Non-Flip vs Flip

efficientb4 adadelta Non-CV

non_flip f1-score : 0.6808946396

all_flip f1-score : 0.7979795186

all_flip_except_nut f1-score : 0.8113580182

원본데이터 학습 시, Augmentation을 진행하여 성능 비교
Nut를 제외한 모든데이터에서 flip데이터를 생성한 결과점수가 가장 높음

06 - 4. optimizer 성능

efficientb4 Non-CV flipped

AdamW f1-score : 0.6808946396

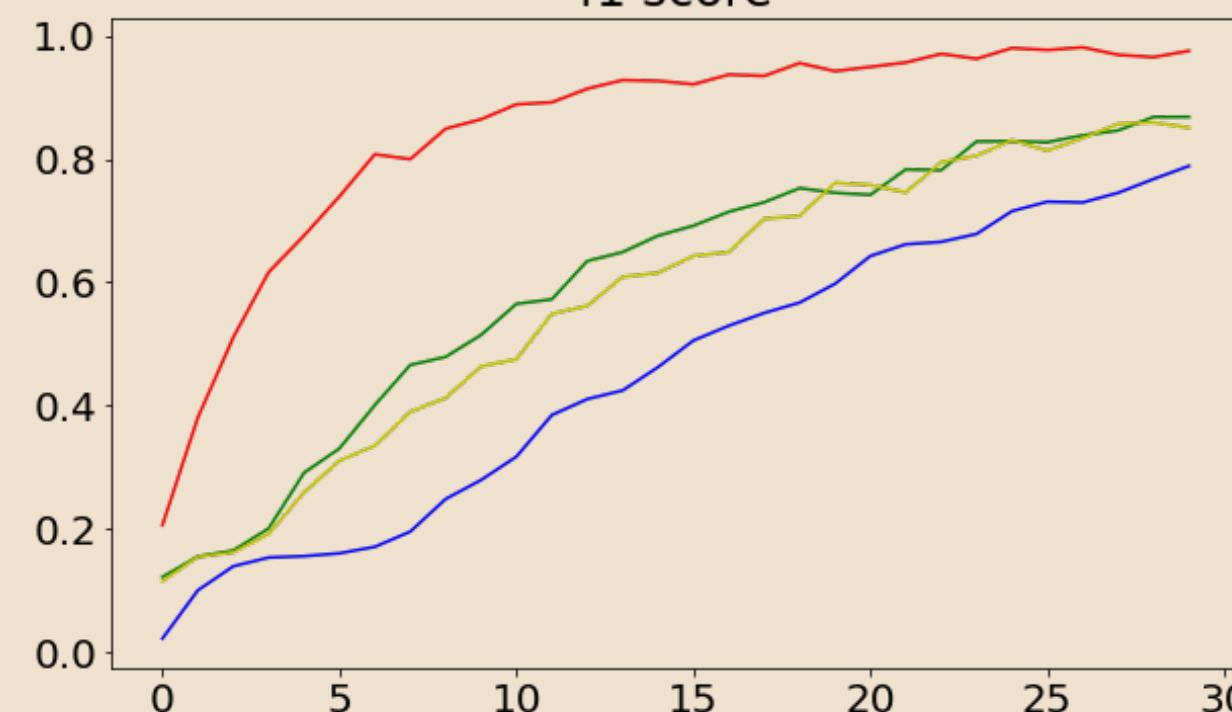
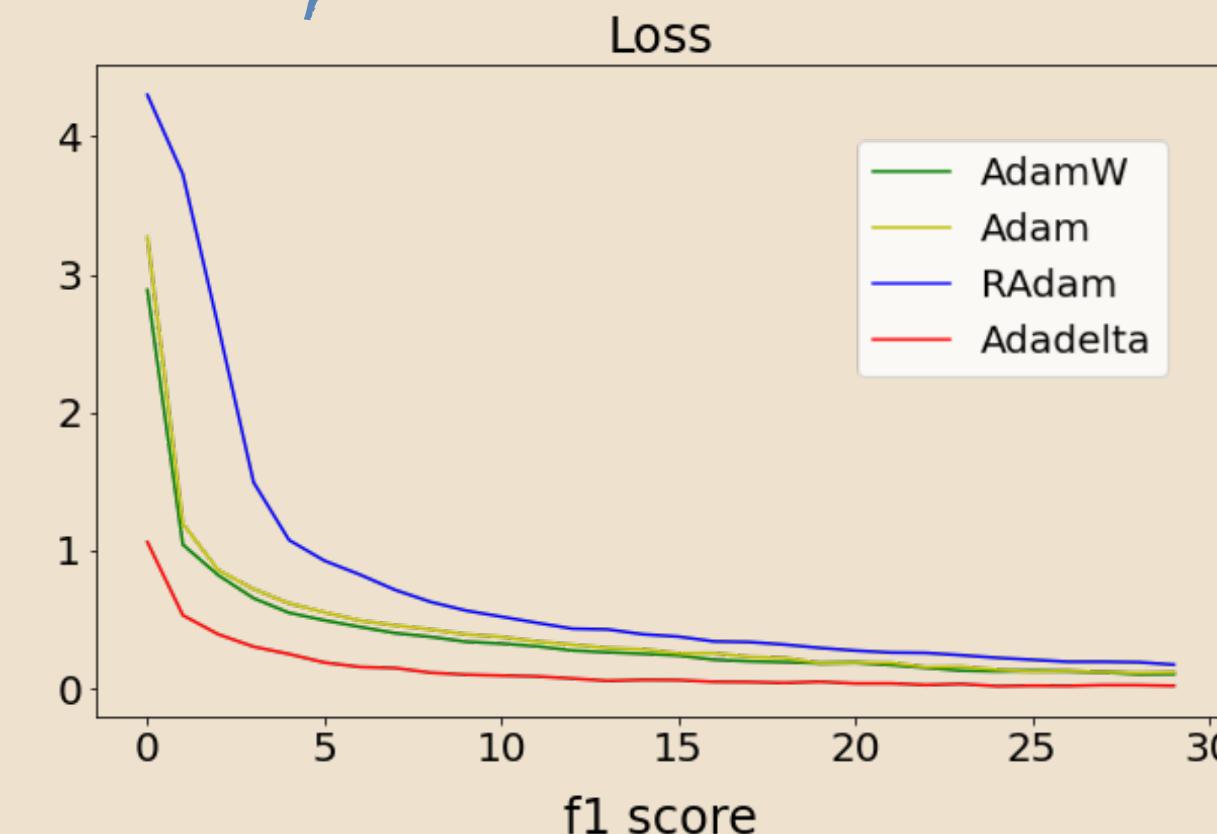
RAdam f1-score : 0.6181746729

Adam f1-score : 0.6955826575

Adadelta f1-score : 0.8113580182

Adadelta를 optimizer로 결정

Adadelta (발간선) 의 성능이 가장 좋음



06- 5. 최종코드 성능

Efficient B4, 5-Fold, Adadelta, Flipped Data f1-score : **0.8562280927**

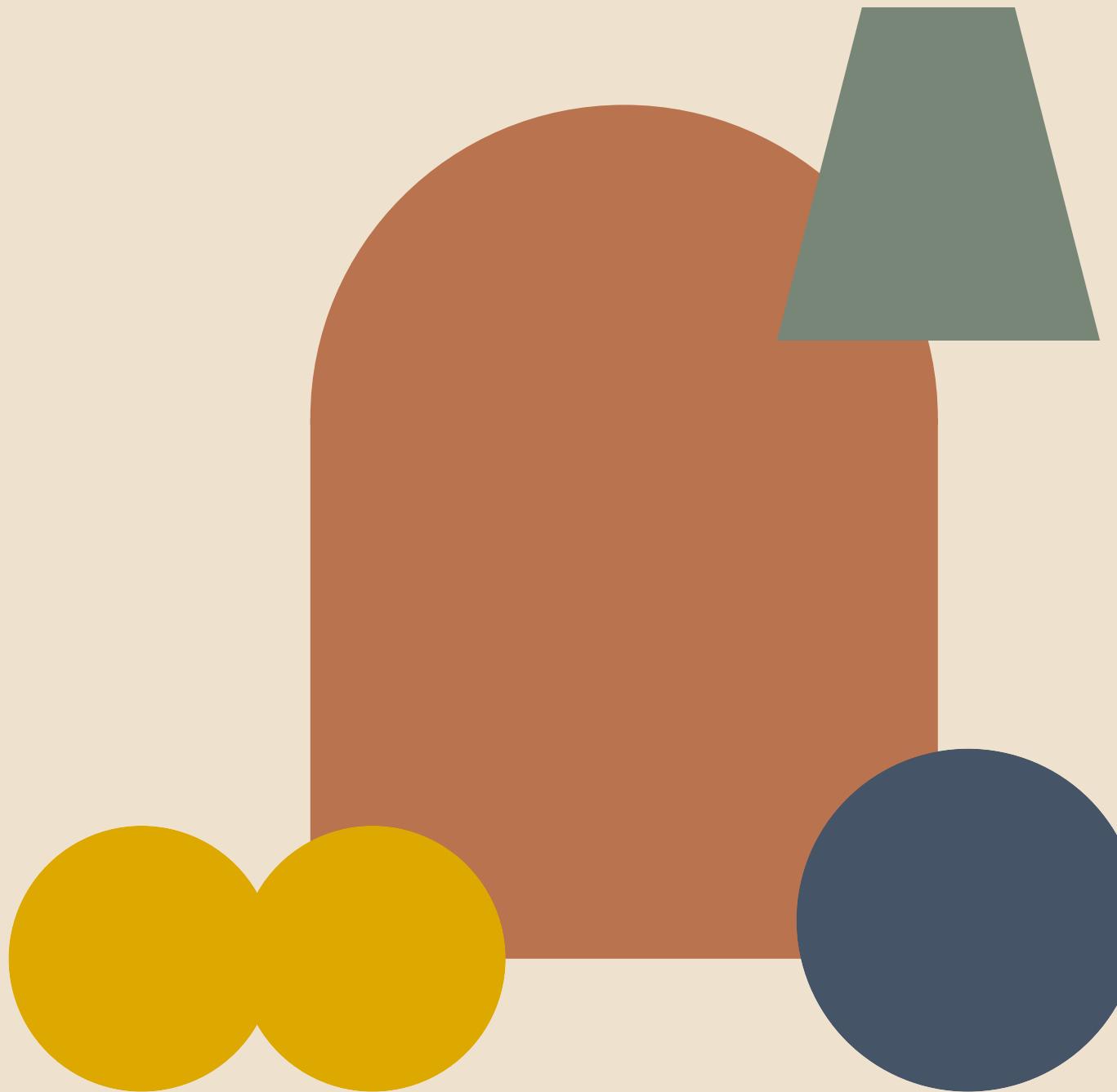


상위 6%

의 점수

07

마무리



07- 0. 느낀점

머지연

딥러닝 신경망을 실제로 적용해보니 프로젝트를 해보고 딥러닝은 아주 큰 세계라는 것을 느꼈습니다.. 딥러닝 공부와 함께 다양한 데이터를 접해보니 딥러닝 활용 능력을 길러야겠다는 필요성을 느끼게 되었습니다. 또한, 딥러닝은 장비발이라는 것을 체감하였습니다.

김태호

첫 딥러닝 프로젝트여서 그런지 배운 것을 활용하는 것 조차 쉽지 않았습니다. 또한 이상치 탐지라는 색다른 주제 때문에 많이 힘들었습니다. 이후에도 많은 프로젝트를 하면서 발전해나 한다는 필요성을 얻었습니다.

조재성

딥러닝 첫 프로젝트인데 강의를 듣는 것과 실제 프로젝트는 다르다는 걸 확실히 느끼게 되었습니다. 더 많이 공부해야 할 것 같고, 항상 생각처럼 잘 되지는 않는다는 걸 명심해야 할 것 같습니다..

김민채

강의도 어렵게 느껴지고 낯설었지만 코드를 사용해보니 프로젝트를 하려니 걱정부터 앞섰고 너무 부족하다고 느꼈습니다. 복습을 더 잘 해야하고 열심히 해야겠다고 생각했습니다.



Thank You