

# [Dacon] 예술 작품 화가 분류 AI 경진대회

Team 3 | 이은준 이지운 정하윤 최지우

# CONTENTS

01

## 프로젝트 소개

- 프로젝트 주제
- 목표 소개

02

## EDA, 전처리

- Data Augmentation
- Weighted random sampling

03

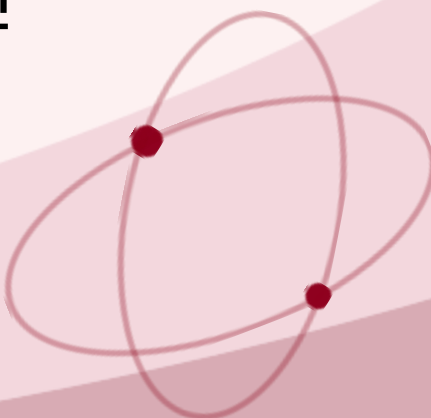
## 모델 선정, 적용

- EfficientNet B3
- Train / Test

04

## 결과

- 분석 결과 도출
- 결론 및 제언





# 01. 프로젝트 소개

# 01. 프로젝트 소개

월간 데이콘 예술 작품 화가 분류 AI 경진대회

알고리즘 | 비전 | 분류 | Macro f1 score

🏆 상금 : 인증서

🕒 2022.10.04 ~ 2022.11.14 09:59 [+ Google Calendar](#)

👤 820명 📅 마감

XP를 획득했어요!

🔗 참여중

대회안내

데이터

코드 공유

토크

리더보드

제출

☰ 개요

📄 규칙

🕒 일정

🏆 상금

📄 동의사항

[배경]

안녕하세요 여러분! 😊 월간 데이콘 예술 작품 화가 분류 AI 경진대회에 오신 것을 환영합니다.

화가 분류는 많은 연구가 이루어지고 있는 문제로, 주로 이미지 처리 및 기계 학습의 전통적인 접근 방식을 통해 꾸준히 연구되어왔습니다.

더 나아가 현재에는 화가의 작품을 흉내내거나, 직접 예술 작품을 창조해내는 GAN을 활용한 생성 모델 연구까지 이루어지고 있습니다.

이번 월간 데이콘 26은 예술 작품을 화가 별로 분류하는 대회입니다.

더 나아가 예술 작품의 일부분만을 가지고도 올바르게 분류해낼 수 있어야합니다.

예술 작품의 일부분만 주어지는 테스트 데이터셋에 대해 올바르게 화가를 분류해낼 수 있는 예술 작품의 전문가인 AI 모델을 만들어주세요.

[주제]

예술 작품을 화가 별로 분류하는 AI 모델 개발



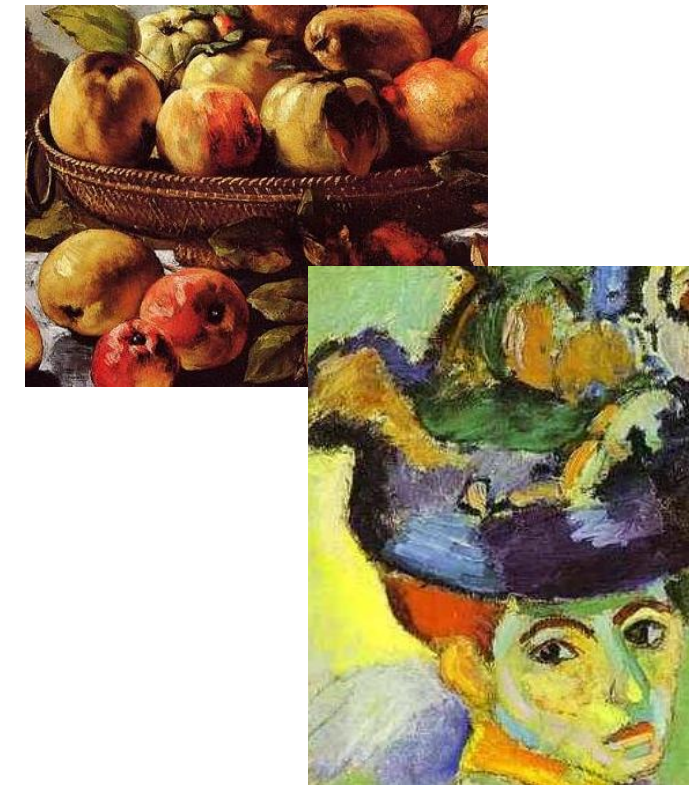
# 01. 프로젝트 소개

<Train data>



Artist: Vincent van Gogh

<Test data>



Artist: ???

이미지 분류 딥러닝 모델



Run !!

- Train.csv: Id, img\_path, artist로 구성
- Test.csv: Id, img\_path 로만 구성, artis를 잘 분류해내는 것이 목표!



## 02. EDA / 전처리

## 2-1. 데이터 전처리

```
# 잘못된 라벨 처리하기
# id 3896과 3986이 모두 3896이라는 값을 가지고 있음
print(train_df['id'][3896])
print(train_df['id'][3986])

# 3986의 id를 3896으로 바꿔줌
train_df['id'][3986] = 3896
```

3896  
3896

```
print("변경 전:", train_df['artist'][3896]) # Titian으로 수정
train_df['artist'][3896] = 'Titian'
print("변경 후:", train_df['artist'][3896]) # 변경 후 확인
```

변경 전: Edgar Degas  
변경 후: Titian

```
print("변경 전:", train_df['artist'][3986]) # Algreed Sislye로 수정
train_df['artist'][3986] = 'Alfred Sisley'
print("변경 후:", train_df['artist'][3986])
```

변경 전: Titian  
변경 후: Alfred Sisley

<데이터 노이즈 처리>

```
[ ] #train 폴더 이미지 개수 확인
import glob
train_path = '/content/train'
train_files = glob.glob(train_path + '/*.jpg')
print(len(train_files))
```

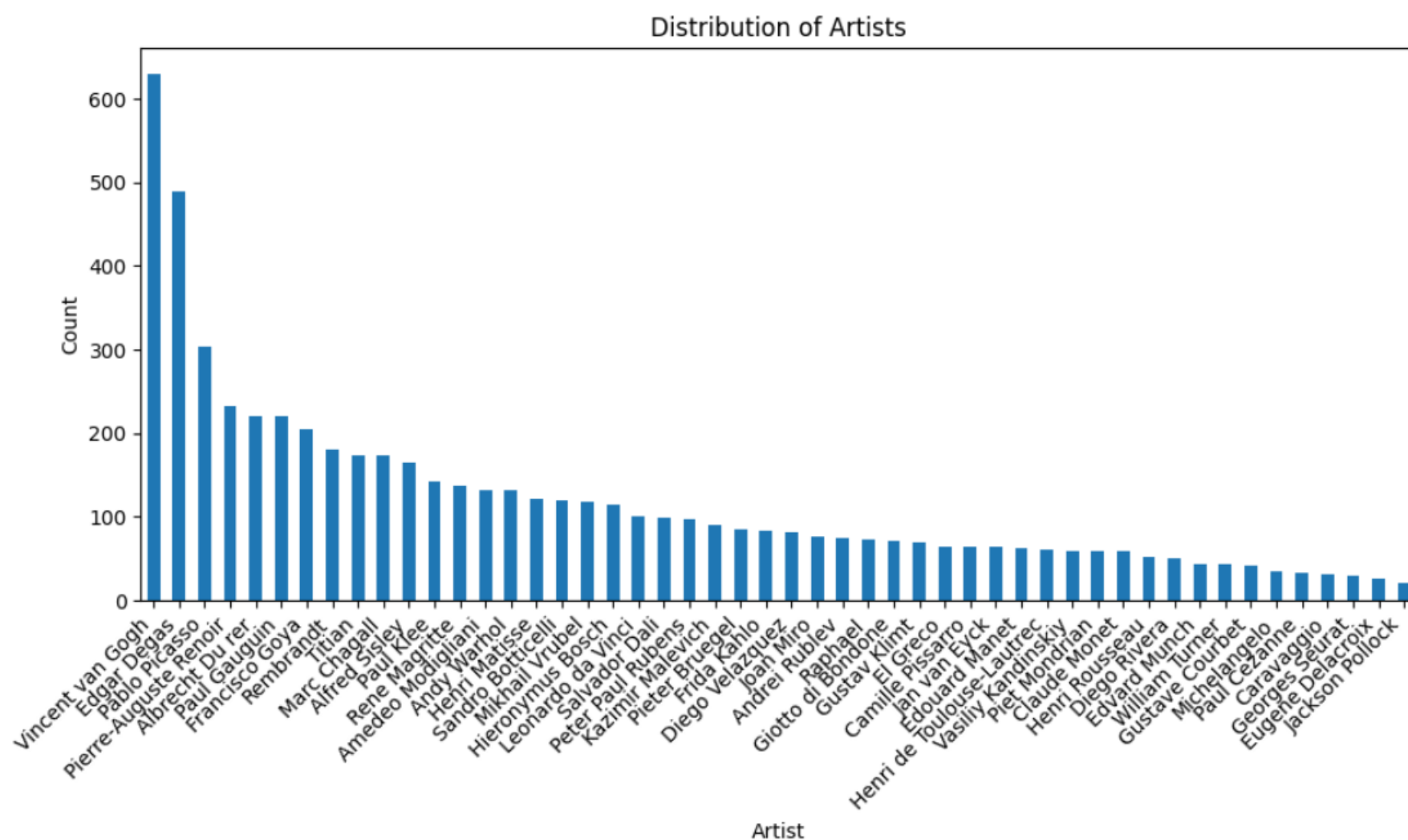
5911

```
[ ] #test 폴더 이미지 개수 확인
test_path = '/content/test'
test_files = glob.glob(test_path + '/*.jpg')
print(len(test_files))
```

12670

<train, test 데이터 개수 확인>

## 2-2. EDA



아티스트별 작품 수 불균형 확인

적게는 21개(Jackson Pollock),

많게는 629개(Van Gogh)까지 아티스트별 작품 수 차이가 큼

```
print(artist_counts)
```

Vincent van Gogh	629
Edgar Degas	489
Pablo Picasso	303
Pierre-Auguste Renoir	233
Albrecht Dürer	220
Paul Gauguin	220
Francisco Goya	204
Rembrandt	181
Titian	173
Marc Chagall	173
Alfred Sisley	165
Paul Klee	142
Rene Magritte	137
Amedeo Modigliani	132
Andy Warhol	132
Henri Matisse	121
Sandro Botticelli	120
Mikhail Vrubel	118
Hieronymus Bosch	115
Leonardo da Vinci	101
Salvador Dali	99
Peter Paul Rubens	97
Kazimir Malevich	91
Pieter Bruegel	85
Frida Kahlo	84
Diego Velazquez	81
Joan Miro	76
Andrei Rublev	74
Raphael	73
Giotto di Bondone	72
Gustav Klimt	69
El Greco	65
Camille Pissarro	64
Jan van Eyck	64
Edouard Manet	62
Henri de Toulouse-Lautrec	61
Vasiliy Kandinsky	60
Piet Mondrian	59
Claude Monet	59
Henri Rousseau	52
Diego Rivera	50
Edvard Munch	44
William Turner	44
Gustave Courbet	42
Michelangelo	34
Paul Cezanne	33
Caravaggio	32
Georges Seurat	30
Eugene Delacroix	26
Jackson Pollock	21

Name: artist, dtype: int64



## 2-2. EDA

```
# 각 이미지별로, 이미지의 크기가 다를 수 있음
# 예시로 앞에서부터 10개 이미지 뽑아서 크기 출력
for idx in range(10):
    image = Image.open(train_files[idx])
    image_size = image.size #원본 이미지 사이즈 확인

    print("Train Image size (width, height):", image_size)
```

```
Train Image size (width, height): (920, 1119)
Train Image size (width, height): (704, 1024)
Train Image size (width, height): (949, 766)
Train Image size (width, height): (686, 1092)
Train Image size (width, height): (1280, 946)
Train Image size (width, height): (2024, 1693)
Train Image size (width, height): (253, 913)
Train Image size (width, height): (1024, 822)
Train Image size (width, height): (829, 1024)
Train Image size (width, height): (1039, 850)
```

```
# test 이미지에 대해서도 확인.
# train 이미지에 비해 작은 크기 (가로 1/2, 세로 1/2 정도) 가 된 것을 확인할 수 있음
# test 데이터에서는 그림의 일부분만 주어진 것이 경진대회의 핵심 키!
```

```
# 예시로 앞에서부터 10개 이미지 뽑아서 크기 출력
for idx in range(10):
    image = Image.open(test_files[idx])
    image_size = image.size #원본 이미지 사이즈 확인

    print("Train Image size (width, height):", image_size)
```

```
Train Image size (width, height): (403, 512)
Train Image size (width, height): (453, 563)
Train Image size (width, height): (402, 512)
Train Image size (width, height): (455, 564)
Train Image size (width, height): (512, 720)
Train Image size (width, height): (334, 492)
Train Image size (width, height): (472, 383)
Train Image size (width, height): (444, 596)
Train Image size (width, height): (512, 402)
Train Image size (width, height): (284, 403)
```

### <이미지 크기 출력>

- train 이미지에 비해 test 이미지는 대체로 절반 크기
- Test 데이터셋에서는 작품의 일부분만 주어짐을 확인

## 2-2. EDA

### <흑백 이미지>

```
Black and white image found: 5613.jpg
Black and white image found: 2727.jpg
Black and white image found: 5152.jpg
Black and white image found: 2817.jpg
Black and white image found: 5615.jpg
Black and white image found: 5423.jpg
Black and white image found: 2532.jpg
Black and white image found: 0282.jpg
Black and white image found: 3560.jpg
Black and white image found: 0742.jpg
Black and white image found: 5292.jpg
Black and white image found: 1705.jpg
Black and white image found: 4599.jpg
Black and white image found: 4111.jpg
Black and white image found: 0789.jpg
Black and white image found: 5312.jpg
Black and white image found: 0544.jpg
Black and white image found: 5092.jpg
Black and white image found: 4127.jpg
Black and white image found: 3994.jpg
Black and white image found: 5517.jpg
Black and white image found: 1167.jpg
Black and white image found: 3193.jpg
Black and white image found: 5838.jpg
Black and white image found: 5625.jpg
Black and white image found: 4473.jpg
Black and white image found: 3002.jpg
Black and white image found: 4164.jpg
Black and white image found: 4516.jpg
Black and white image found: 5823.jpg
Black and white image found: 0555.jpg
Black and white image found: 4092.jpg
Black and white image found: 0969.jpg
Black and white image found: 4158.jpg
Black and white image found: 4172.jpg
Black and white image found: 4854.jpg
Black and white image found: 4699.jpg
Black and white image found: 3710.jpg
Black and white image found: 0786.jpg
Black and white image found: 1584.jpg
Black and white image found: 3602.jpg
Black and white image found: 5033.jpg
Black and white image found: 5349.jpg
```

### <artist>

```
img_path      artist
282 ./train/0282.jpg Francisco Goya
544 ./train/0544.jpg Francisco Goya
555 ./train/0555.jpg Francisco Goya
742 ./train/0742.jpg Francisco Goya
786 ./train/0786.jpg Francisco Goya
789 ./train/0789.jpg Francisco Goya
969 ./train/0969.jpg Francisco Goya
1167 ./train/1167.jpg Francisco Goya
1584 ./train/1584.jpg Francisco Goya
1705 ./train/1705.jpg Francisco Goya
2532 ./train/2532.jpg Francisco Goya
2727 ./train/2727.jpg Pablo Picasso
2817 ./train/2817.jpg Francisco Goya
3002 ./train/3002.jpg Francisco Goya
3193 ./train/3193.jpg Francisco Goya
3560 ./train/3560.jpg Francisco Goya
3602 ./train/3602.jpg Francisco Goya
3710 ./train/3710.jpg Francisco Goya
3994 ./train/3994.jpg Francisco Goya
4092 ./train/4092.jpg Francisco Goya
4111 ./train/4111.jpg Francisco Goya
4127 ./train/4127.jpg Francisco Goya
4158 ./train/4158.jpg Francisco Goya
4164 ./train/4164.jpg Albrecht Du rer
4172 ./train/4172.jpg Francisco Goya
4473 ./train/4473.jpg Francisco Goya
4516 ./train/4516.jpg Francisco Goya
4599 ./train/4599.jpg Titian
4699 ./train/4699.jpg Francisco Goya
4854 ./train/4854.jpg Francisco Goya
5033 ./train/5033.jpg Francisco Goya
5092 ./train/5092.jpg Francisco Goya
5152 ./train/5152.jpg Francisco Goya
5292 ./train/5292.jpg Francisco Goya
5312 ./train/5312.jpg Francisco Goya
5349 ./train/5349.jpg Pieter Bruegel
5423 ./train/5423.jpg Francisco Goya
5517 ./train/5517.jpg Albrecht Du rer
5613 ./train/5613.jpg Leonardo da Vinci
5615 ./train/5615.jpg Francisco Goya
5625 ./train/5625.jpg Francisco Goya
5823 ./train/5823.jpg Francisco Goya
5838 ./train/5838.jpg Francisco Goya
```

Original Black and White Image  
Shape: (984, 669)



Converted RGB Image  
Shape: (984, 669, 3)



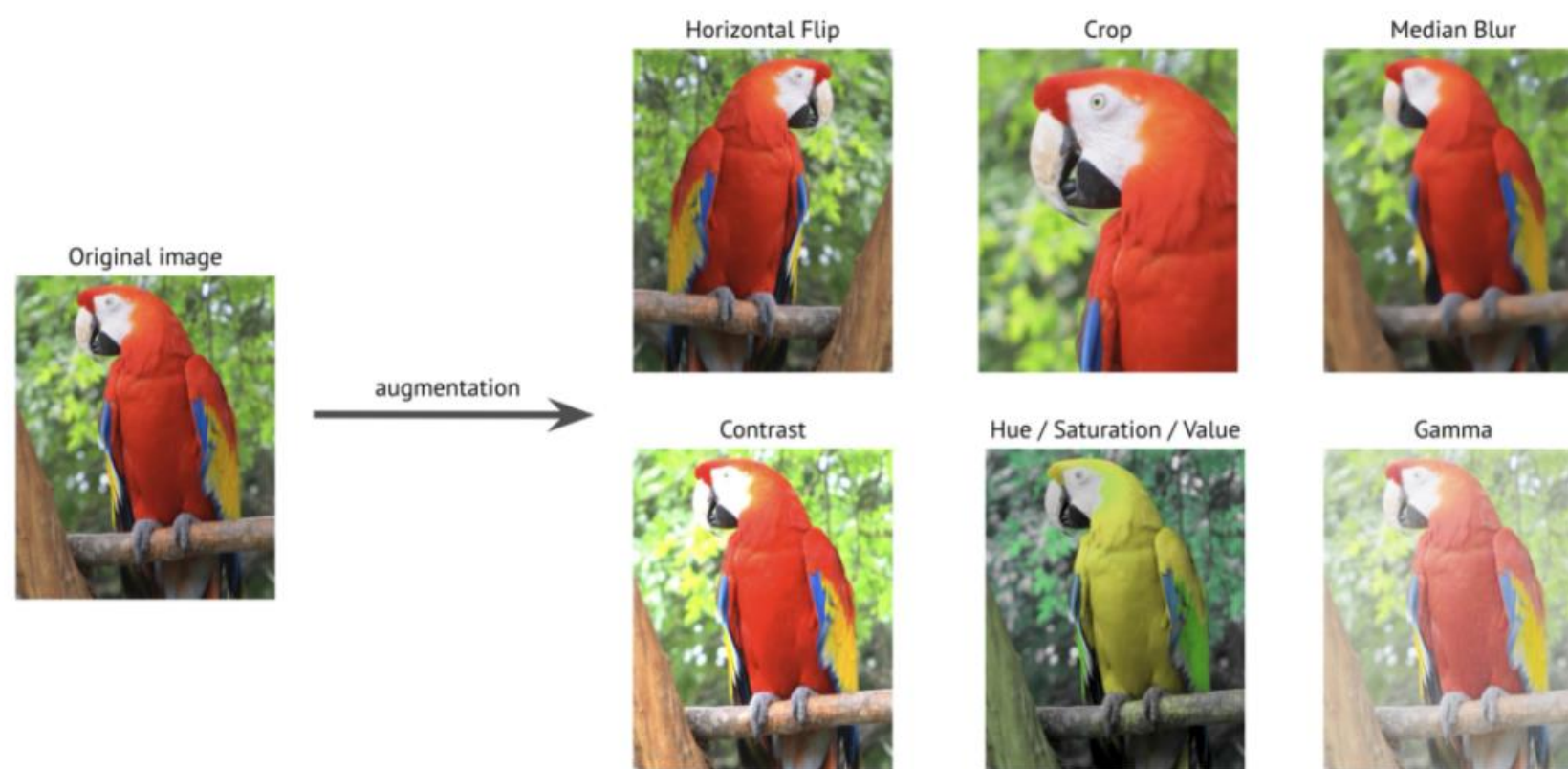
모델에 같은 차원의 이미지를 넣기 위해  
RGB로 convert, 시각적 차이 확인

흑백 이미지, 해당 작품의 artist 출력  
Francisco Goya의 작품이 가장 많음

## 2-3. DATA Augmentation

Data Augmentation을 적용하는 이유?

- 적은 양의 데이터로 심층 신경망을 효과적으로 학습시키기 위해 사용!
- 원본 데이터에서 인위적인 과정을 통해 새로운 데이터를 추가로 만들어 사용
- 다양한 변형을 통한 데이터셋 제공 → 오버피팅 완화, 모델 성능 향상



## 2-3. DATA Augmentation

```
train_transforms = A.Compose([
    A.RandomResizedCrop(CFG['IMG_SIZE'], CFG['IMG_SIZE'], scale=(0.25, 0.25)), #입력 데이터의 1/4 크기로 crop된 이미지 데이터를 생성
    #A.Resize(CFG['IMG_SIZE']*2,CFG['IMG_SIZE']*2)
    #A.RandomCrop(CFG['IMG_SIZE'],CFG['IMG_SIZE'])
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.ShiftScaleRotate(p=0.5), #아래 설명 참고
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), max_pixel_value=255.0, always_apply=False, p=1.0),
    ToTensorV2()
])

val_transforms = A.Compose([
    A.RandomResizedCrop(CFG['IMG_SIZE'], CFG['IMG_SIZE'], scale=(0.25, 0.25)), #입력 데이터의 1/4 크기로 crop된 이미지 데이터를 생성
    #A.Resize(CFG['IMG_SIZE']*2,CFG['IMG_SIZE']*2)
    #A.RandomCrop(CFG['IMG_SIZE'],CFG['IMG_SIZE'])
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225), max_pixel_value=255.0, always_apply=False, p=1.0),
    ToTensorV2()
])
```

RandomResizedCrop : 입력 이미지를 1/4 크기로 crop

HorizontalFlip : 좌우반전 // VerticalFlip : 상하반전

ShiftScaleRotate : 이미지 무작위 이동 + 크기 조절 + 회전

Normalize : R,G,B 평균값을 0으로. 각 pixel 값 - 평균 pixel 값

→각각의 augmentation 진행할 확률을 0.5로 설정

\* test\_transforms은 **별도로 적용!**

Test image는 이미  
가로 1/2, 세로 1/2 크기

→ crop 적용할 필요 x



## 2-4. Weighted Random sampling

```
def make_weights(labels, nclasses): # 클래스 불균형을 다루기 위해 가중치 생성
    labels = np.array(labels)
    weight_arr = np.zeros_like(labels)

    _, counts = np.unique(labels, return_counts=True) #레이블에서 고유한 클래스 찾고 클래스별로 이미지 개수 카운트
    for cls in range(nclasses):
        weight_arr = np.where(labels == cls, 1/counts[cls], weight_arr)
        # 각 클래스의 인덱스를 산출하여 해당 클래스 개수의 역수를 확률로 할당한다.
        # 이를 통해 각 클래스의 전체 가중치를 동일하게 한다.

    return weight_arr

weights = make_weights(train_labels, len(np.unique(train_labels)))
weights = torch.DoubleTensor(weights)
```

앞서 확인했듯, 아티스트별 작품 수가 불균형함

→ 작가별 작품 균등하게 학습하기 위한 weighted random sampling 진행

각 클래스 데이터수의 역수를 확률로 설정, label이 선택될 가중치를 동일하게 전체 라벨에 할당



## 03. 모델 선정

## 03. 모델 선정

### <다양한 모델 시도>

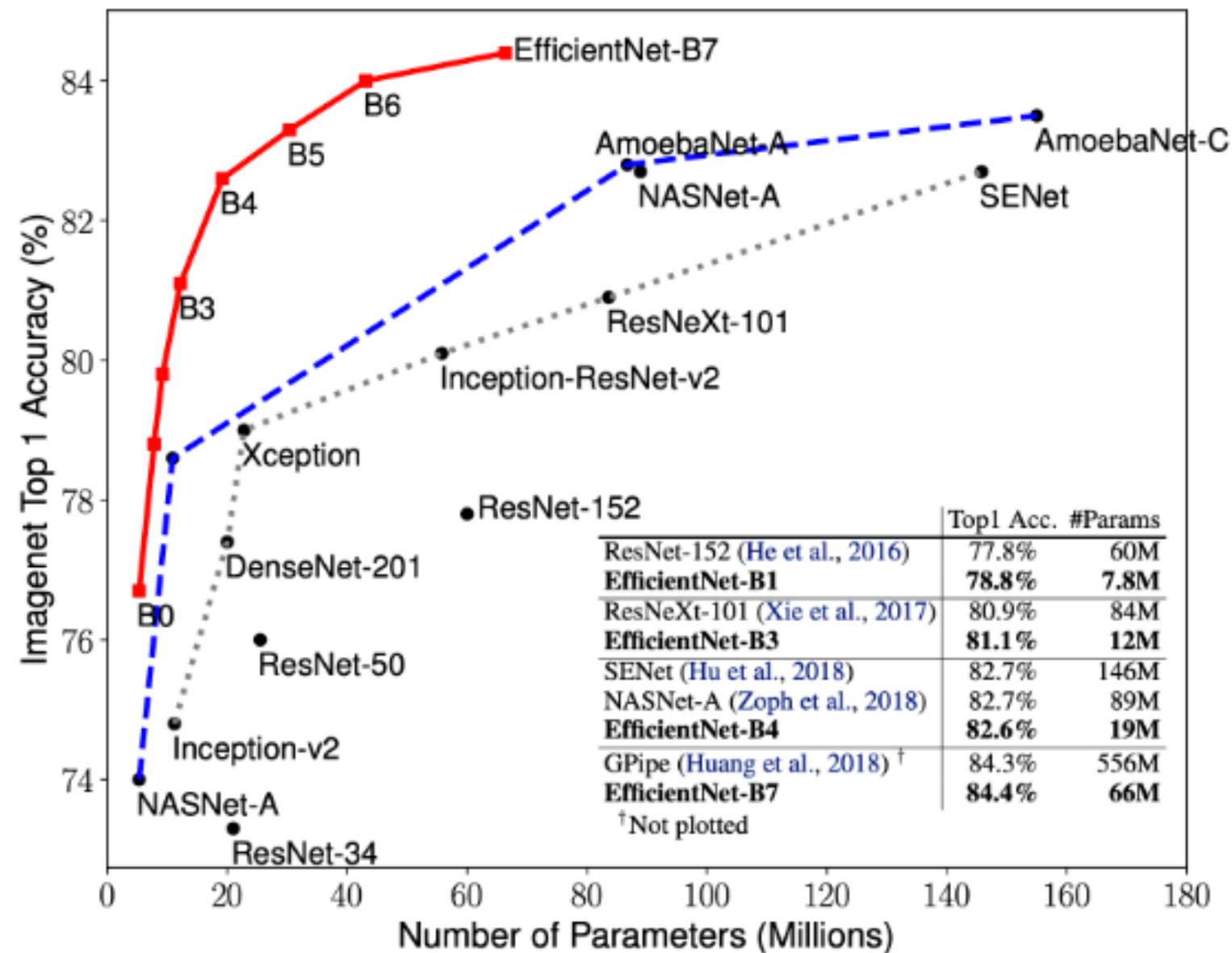
VGGNet, GoogLeNet
ResNet
EfficientNet b0
EfficientNet b3
EfficientNet b4
... etc

6개 + α 의 모델로 데이터셋 학습 시도

Accuracy 문제로 VGGNet, GoogLeNet, ResNet 등은 배제

EfficientNet 중에서 비교하여 모델 선정

## 03. 모델 선정: EfficientNet B3



효율적인 모델 구축을 위해 depth, width, resolution을 한번에 스케일링하는 방법 제안

→ **Compound scaling**

MnasNet에서 사용하는 MBConvBlock,

SENet에서 제안한 squeeze and excitation 사용



## 03. 모델 선정: EfficientNet B3

Table 1. EfficientNet-B0 baseline network – Each row describes a stage  $i$  with  $\hat{L}_i$  layers, with input resolution  $\langle \hat{H}_i, \hat{W}_i \rangle$  and output channels  $\hat{C}_i$ . Notations are adopted from equation 2.

Stage $i$	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	$224 \times 224$	32	1
2	MBConv1, k3x3	$112 \times 112$	16	1
3	MBConv6, k3x3	$112 \times 112$	24	2
4	MBConv6, k5x5	$56 \times 56$	40	2
5	MBConv6, k3x3	$28 \times 28$	80	3
6	MBConv6, k5x5	$14 \times 14$	112	3
7	MBConv6, k5x5	$14 \times 14$	192	4
8	MBConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

B0 모델 기반, compound coefficient( $\phi$ )에 맞는 최적의  $\alpha, \beta, \gamma$  값을 설정 -> scaling

## 03. 모델 선정: EfficientNet B3

Model	Top-1 Acc.	Top-5 Acc.	#Params
<b>EfficientNet-B0</b>	<b>77.1%</b>	<b>93.3%</b>	<b>5.3M</b>
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M
<b>EfficientNet-B1</b>	<b>79.1%</b>	<b>94.4%</b>	<b>7.8M</b>
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M
Xception (Chollet, 2017)	79.0%	94.5%	23M
<b>EfficientNet-B2</b>	<b>80.1%</b>	<b>94.9%</b>	<b>9.2M</b>
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M
<b>EfficientNet-B3</b>	<b>81.6%</b>	<b>95.7%</b>	<b>12M</b>
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M
<b>EfficientNet-B4</b>	<b>82.9%</b>	<b>96.4%</b>	<b>19M</b>
SENet (Hu et al., 2018)	82.7%	96.2%	146M
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M
<b>EfficientNet-B5</b>	<b>83.6%</b>	<b>96.7%</b>	<b>30M</b>
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M
<b>EfficientNet-B6</b>	<b>84.0%</b>	<b>96.8%</b>	<b>43M</b>
<b>EfficientNet-B7</b>	<b>84.3%</b>	<b>97.0%</b>	<b>66M</b>
GPipe (Huang et al., 2018)	84.3%	97.0%	557M

Base model	resolution
EfficientNetB0	224
EfficientNetB1	240
EfficientNetB2	260
EfficientNetB3	300
EfficientNetB4	380
EfficientNetB5	456
EfficientNetB6	528
EfficientNetB7	600

- B4 이후로는 파라미터가 커지는 데 비해 큰 성능 향상이 없는 것을 확인
- 모델마다 적합한 이미지 데이터의 해상도가 존재함 (ex: b0\_224 \* 224)
- B4 해상도의 경우 cuda 메모리 부족 문제로 실행 x,  
→ **EfficientNet B3로 최종 결정**

## 03. 모델 선정: EfficientNet B3

```
[ ] class efficientnet_b3(nn.Module):
    def __init__(self, num_classes=len(1e.classes_), fine_tune=True, dropout_rate=0.2):
        super(efficientnet_b3, self).__init__()
        self.backbone = EfficientNet.from_pretrained('efficientnet-b3')

        # Set whether to fine-tune or freeze the backbone
        for param in self.backbone.parameters():
            param.requires_grad = fine_tune

        # Add dropout layer 모델이 커져서 과적합 발생하므로 dropout 추가
        self.dropout = nn.Dropout(p=dropout_rate)

        # Classifier layer
        self.classifier = nn.Linear(1000, num_classes)

    def forward(self, x):
        # Backbone
        x = self.backbone(x)

        # Dropout
        x = self.dropout(x)

        # Classifier
        x = self.classifier(x)

        return x
```

- Pytorch에 내장된 efficientnet-b3를 이용  
(해상도 300 \* 300, 기본 Dropout 포함 )
- 분류기 작동 이전에 추가로 Dropout layer  
적용 → 과적합 방지, 모델 성능 향상



## 04. Train / Test



## 04. Train

```
# 손실 함수 및 옵티마이저 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 학습을 위한 장치 설정 (GPU 사용 가능한 경우)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
```

분류 문제이므로 손실함수로 CrossEntropyLoss 사용

Optimizer: Adam 사용, lr = 0.001

## 04. Train

```
from torch.optim.lr_scheduler import StepLR
scheduler = StepLR(optimizer, step_size=10, gamma=0.1) # epoch 10마다 0.1배로

# 스케줄러 설정값 기준 성능이 크게 향상됨 -> lr이 매우 큰 역할을 함!
```

StepLR: 주어진 step\_size 마다 학습률을 감소 비율(gamma)만큼 조절

초기 학습률이  $10^{-3}$ 이라면, 10 epoch 후에는  $10^{-4}$ 로, 20 epoch 후에는  $10^{-5}$ 로 감소

실제 시도에서는 step size, gamma 여러 번 조정 -> scheduler 미사용시 보다 성능 크게 향상

## 04. Train

```
def train_model(model, criterion, optimizer, train_loader, val_loader, epochs=30, patience=5, save_path='best_model.pth'):
    best_val_loss = float('inf') # 최상의 검증 손실 초기화
    best_model_state = None # 최상의 모델 상태 초기화
    no_improvement = 0 # 개선되지 않은 에폭 수 초기화

    for epoch in range(epochs):
        model.train() # 모델을 학습 모드로 설정
        running_loss = 0.0

        for inputs, labels in train_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad() # 그래디언트 초기화
            outputs = model(inputs) # 순전파

            # 라벨을 one-hot 인코딩
            labels_onehot = F.one_hot(labels, num_classes=50).to(torch.float32)

            loss = criterion(outputs, labels_onehot)
            loss.backward() # 역전파
            optimizer.step() # 최적화

            running_loss += loss.item()

        print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")
```

라벨 one-hot encoding  
적용, 성능 향상

## 4-1. Train

```
with torch.no_grad(): # 그래디언트 계산 비활성화
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, F.one_hot(labels, num_classes=50).to(torch.float32))
        val_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

val_loss /= len(val_loader)
accuracy = 100 * correct / total
print(f"Validation Loss: {val_loss}, Accuracy: {accuracy}%")

# Early Stopping 적용, 5번 에포크 동안 개선 없으면 중단
if val_loss < best_val_loss:
    best_val_loss = val_loss
    best_model_state = model.state_dict() # 최상의 모델 상태 업데이트
    no_improvement = 0
else:
    no_improvement += 1

if no_improvement >= patience:
    print(f"No improvement for {patience} epochs. Early stopping.")
    break

scheduler.step()
```

```
# 최적의 모델 저장
torch.save(best_model_state, save_path)
print(f"Best model saved to {save_path}")

# 학습 실행 및 최적의 모델 저장
train_model(model, criterion, optimizer, train_loader, val_loader, epochs=30, patience=5, save_path='./data/bestmodel.pth')
```

best\_val\_loss 기반 best\_model\_state 정의

→ Early stopping 적용,

5epoch 동안 개선 없으면 중단

스케줄러 적용: 10 epoch 마다 lr 0.1배로 조정




## 4-2. Test

```
## 테스트 데이터셋 로드
test_dataset = CustomDataset(dataframe=test_df, img_dir=test_path, transform=test_transforms)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=4, pin_memory=True)

## 추론
y_preds = []
model.eval() # 모델을 평가 모드로 설정
with torch.no_grad(): # 그래디언트 계산 비활성화
    for x, _ in tqdm(test_loader):
        x = x.to(device)
        outputs = model(x)
        _, predicted = torch.max(outputs, dim=-1)
        y_preds += predicted.tolist()
```

100%

396/396 [09:19&lt;00:00, 1.74s/it]



## 05. 결론 및 제언

## 05. 결론 및 제언

### <분석 결과 도출>

993079      b3\_dropout.csv  
edit

2024-02-27 14:36:47

0.7665454188  
0.760134413



- 최종 모델: EfficientNet B3, dropout 0.2 적용, 이미지 사이즈 300 \* 300 (해상도 맞춤 조정)  
**private 0.76**, Dacon 리더보드 기준 상위 15% 이내의 결과!
- Weighted random sampling, RandomResizeCrop 등 데이터의 특성을 반영한 전처리의 효과가 좋았음!

## 05. 결론 및 제언

### <프로젝트 의의>

- 주어진 이미지 데이터셋의 특징 (데이터 불균형, train data와 test data의 크기 차이 등)을 EDA를 통해 확인하고, 그에 맞는 전처리 방법을 사용해 성능을 향상시켰음

(가중 랜덤 샘플링, transform 시 1/2 crop)

- EfficientNet 모델 기반으로 다양한 시도(dropout 적용, scheduler 적용 등)를 거듭하면서 최선의 이미지 분류모델을 찾아나감

### <프로젝트의 발전 방향>

- EfficientNet 외에 ViT, Noisy student 등 최신 이미지 분류모델 시도
- Colab 사용시 GPU 제한 문제, cuda out of memory 등 여러 시간적/공간적 제약이 있었음
- 모델 앙상블(스태킹, 보팅 등), 하이퍼파라미터 튜닝 등 다양한 추가 시도

→ 더 나은 분류 성능 기대



**Thank You**