

예술작품으로 작가 예측 딥러닝 경진대회

4팀 : 권윤, 박유영, 정서현

목차

- 주제 설명
- 데이터 시각화
- EfficientNet
- EfficientNet_V2
- EfficientNet With Multi-Scale Pyramid Weighted Pooling
- 결과
- 그 외에 고려해본 모델

주제 설명



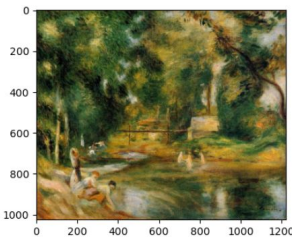
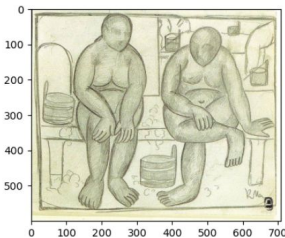
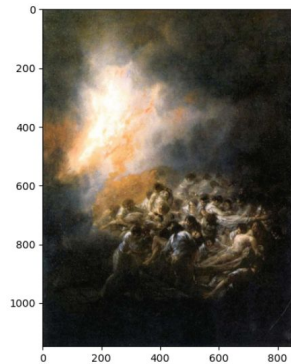
예술 작품 화가 분류 AI 경진대회

- 50명의 작가에 대한 예술작품 분류
- 사진의 1/4부분을 RandomCrop한 이미지로 Test



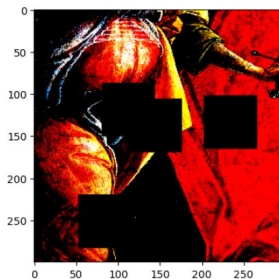
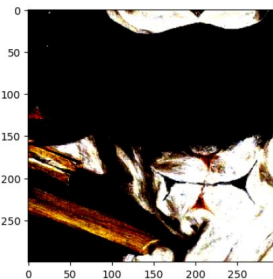
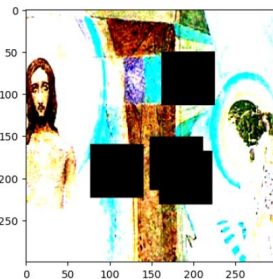
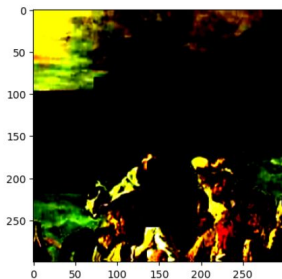
Vincent Van Gogh

데이터 시각화

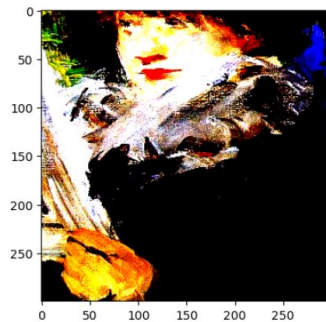
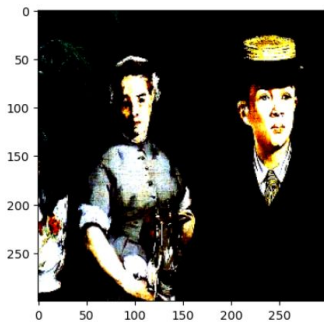
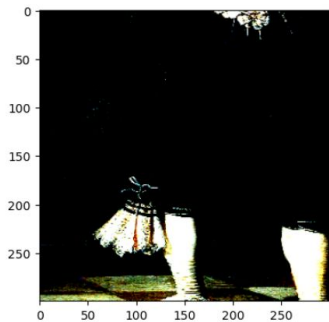
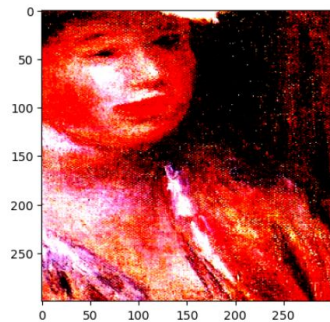


Train 데이터 (transform 전)

Train 데이터 (transform 후)

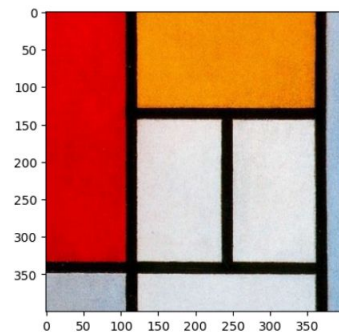
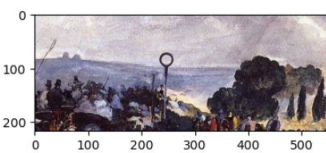
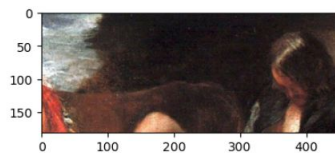
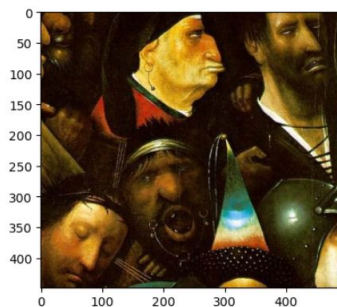


데이터 시각화



Valid 데이터

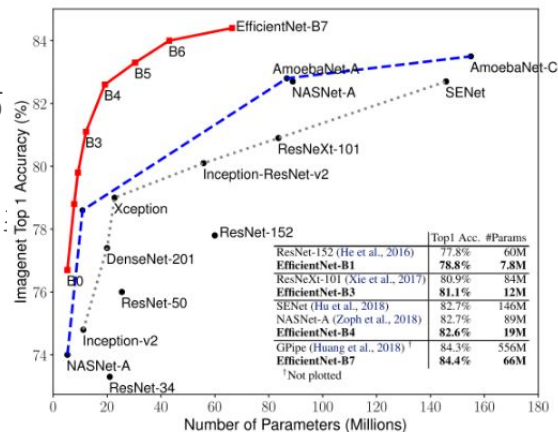
Test 데이터
(transform 전)



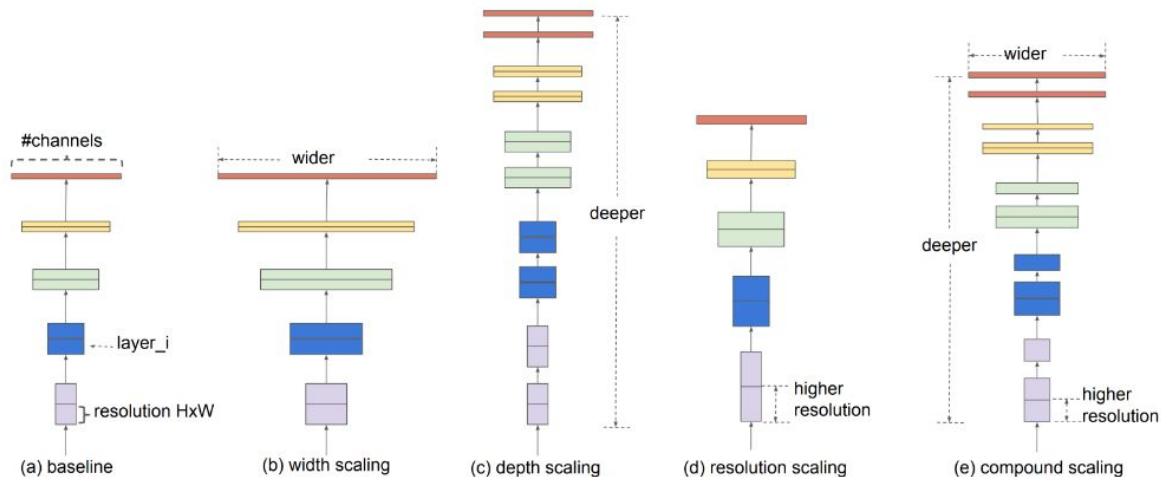
EfficientNet

하나의 베이스라인 구조(**NasNet**)를 가지고, 쉽게 원하는 만큼 모델을 더 복잡하게 만들 수 있도록, 복합적인 스케일링 방법을 구현한 딥러닝 네트워크

- 2019년 Google의 딥러닝 연구팀에서 개발
- 비슷한 성능의 다른 모델들 대비 파라미터 개수가 현저히 낮음
- 쉽게 모델 구조를 확장시킬 수 있음
 - 하나의 하이퍼파라미터로 모델의 깊이, 필터 수 등을 자동적으로 스케일링
- 파이토치 라이브러리에 자체 구현이 되어있음
 - `torchvision.models.efficientnet_b{i}`
 - `i`: 0 ~ 8. 높아질수록 더 복잡한 구조
 - B0~B8로 이루어져 9가지 Resolution에 대응할 수 있음



EfficientNet



- 기존에는 모델의 성능을 높이기 위해 width, depth, input_size 중 한 가지만을 조절하는 시도를 해왔다.
- EfficientNet에서는, 정해진 공식을 통해 width, depth, input_size를 적절하게 조합하여, 연산량은 최소화하는 동시에 높은 성능을 보여준다.

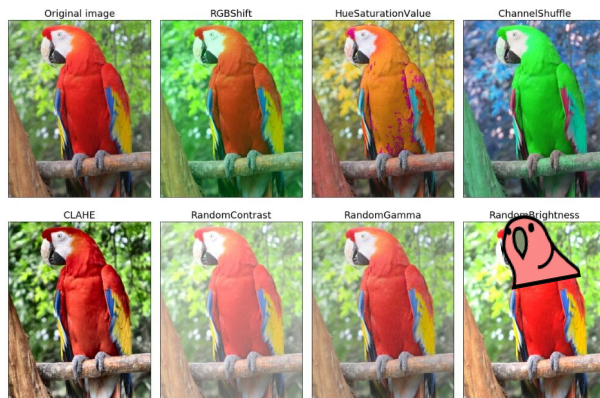
- 다양한 해상도를 지원하여 다양한 원본image의 해상도에 적용이 가능하다.
- 고해상도의 사진을 224*224로 Resize하였을 때 발생하는 정보 소실 문제를 더 고수준의 EfficientNet을 적용함으로 보완할 수 있다.

Model	Input Size	Parameter	Accuracy (%)
EfficientNetB0	224 × 224	4,057,253	97.95
EfficientNetB1	240 × 240	6,582,914	97.54
EfficientNetB2	260 × 260	7,777,012	96.04
EfficientNetB3	300 × 300	10,792,746	95.90
EfficientNetB4	380 × 380	17,684,570	94.13
EfficientNetB5	456 × 456	28,525,810	95.22
EfficientNetB6	528 × 528	40,973,969	98.22
EfficientNetB7	600 × 600	64,113,049	98.36

EfficientNet : 데이터 증강

고려사항

1. '예술작품'이라는 특성때문에, **Blur, Noise, Sharpness** 등을 적용하지 않는 것이 오히려 작가마다 다른 **feature**를 최대한 살릴 수 있다고 생각
-> 하지만 후에 **Overfit**문제..
2. 테스트 데이터셋은 예술 작품의 일부분(약 1/4)만 제공하기 때문에 **train**으로 주어진 데이터에서 **valid**데이터를 추출할 때 실제 **test**데이터처럼 재현해야 함 -> **crop**과정에서 왜곡을 어떻게 최소화할 것인지 고려해야 함



EfficientNet : 하이퍼파라미터 및 증강

```
1 # Train 데이터를 위한 증강 및 전처리
2 train_transform = A.Compose([
3     # test로 주어지는 데이터가 원본사이즈만큼의 1/4 RandomCrop인 점을 감안해 최대한 재현
4     A.Resize(CFG['IMG_SIZE']*2, CFG['IMG_SIZE']*2), # 300*300
5     A.RandomCrop(p=1, height=CFG['IMG_SIZE'], width=CFG['IMG_SIZE']),
6     A.CoarseDropout(max_holes=4, max_height=64, max_width=64, p=0.5),
7     A.OneOf([
8         A.MotionBlur(p=1),
9         A.OpticalDistortion(p=1),
10        A.GaussNoise(p=1),
11    ], p=0.3),
12    A.VerticalFlip(p=0.5),
13    A.HorizontalFlip(p=0.5),
14    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225),
15                max_pixel_value=255.0, always_apply=False, p=1.0),
16    ToTensorV2()
17 ])
```

단순히 원본데이터에서 Resize: 0.578



2배 크기로 Resize한 후 Crop : 0.725

Blur, Noise, Sharpness와 같이
이미지의 질감, 선명도 등을 직접적으로
건드리는 feature를 사용한 전처리가 오히려
더 높은 분류성능을 보여줌

```
1 # Valid 데이터를 위한 증강 및 전처리
2 valid_transform = A.Compose([
3     A.Resize(CFG['IMG_SIZE']*2, CFG['IMG_SIZE']*2), # test로 주어지는 데이터
4     A.RandomCrop(p=1, height=CFG['IMG_SIZE'], width=CFG['IMG_SIZE']),
5     A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225),
6                max_pixel_value=255.0, always_apply=False, p=1.0),
7     ToTensorV2()
8 ])
9
10 # Test 데이터를 위한 증강 및 전처리
11 test_transform = A.Compose([
12     A.Resize(CFG['IMG_SIZE'], CFG['IMG_SIZE']),
13     A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225),
14                max_pixel_value=255.0, always_apply=False, p=1.0),
15     ToTensorV2()
16 ])
```

EfficientNet : 모델 생성 훈련 및 검증

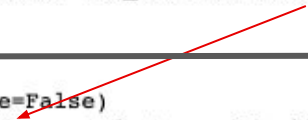
```
class BaseModel(nn.Module):
    def __init__(self, num_classes=50):
        super(BaseModel, self).__init__()
        self.backbone = models.efficientnet_b3(pretrained=True)
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.3),
            nn.Linear(in_features=1000, out_features=num_classes),
        )

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

```
(avgpool): AdaptiveAvgPool2d(output_size=1)
(classifier): Sequential(
  (0): Dropout(p=0.3, inplace=True)
  (1): Linear(in_features=1536, out_features=1000, bias=True)
)

```

```
(classifier): Sequential(
  (0): Dropout(p=0.3, inplace=False)
  (1): Linear(in_features=1000, out_features=50, bias=True)
)
```

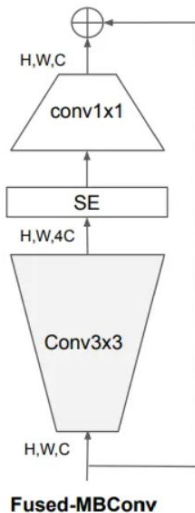
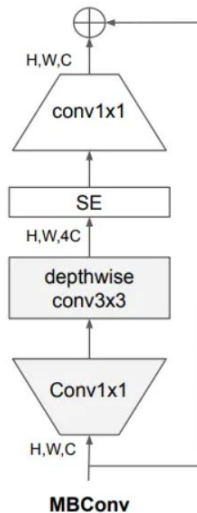


Macro F1 score

모델	데이터 증강	Epoch 수	제출점수	등수
Eff_b3	Crop, Dropout, OneOf	30	0.76365	36/301
Eff_b3	Crop, Dropout, OneOf	40	0.76855	34/301
Eff_b3	Crop, Dropout, OneOf	60	0.77779	33/301
Eff_b3	Crop, Dropout, OneOf	80	0.77958	32/301
Eff_b3	Crop, Dropout, OneOf	110	0.77792	33/301

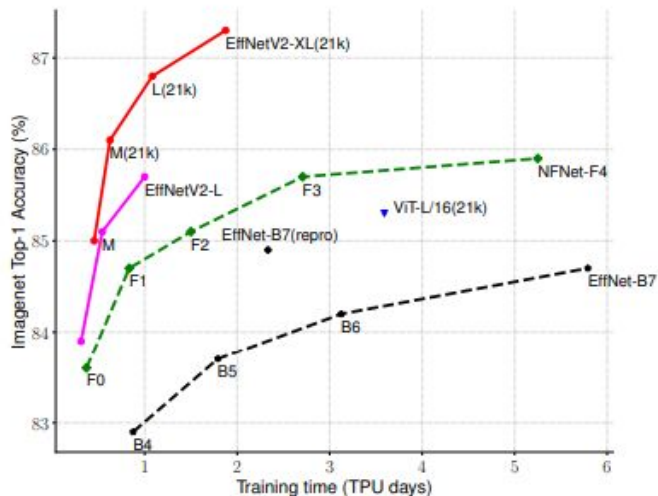
EfficientNet -> EfficientNetV2

- 모델을 구성하는 기본 단위인 MBConv 블록의 depthwise convolution 연산이 병목을 발생시킴
-> **MBConv블럭 + Fused-MBConv블럭**
- B0에서 B7까지 다양한 모델을 생성하기 위해 높이, 너비, 이미지 해상도에 동일한 스케일링 적용
-> **Progressive learning with adaptive regularization:**
저해상도에서는 규제 강도를 줄이고, 고해상도에서는 규제 강도를 높임
- 고해상도의 이미지를 사용해 모델을 훈련할 때 전체 과정이 느려짐
-> **GPU/TPU** 메모리 사용량을 줄이기 위해 최대 이미지 크기를 제한하여 교육 속도 높임



EfficientNetV2 우수성

모델의 크기(parameter 수)와 학습 속도 측면에서 효율성 극대화



(a) Training efficiency.

	Acc. (%)	Params (M)	FLOPs (B)	TrainTime (h)	InferTime (ms)
V1-B7	85.0	66	38	54	170
V2-M (ours)	85.1	55 (-17%)	24 (-37%)	13 (-76%)	57 (-66%)

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

EfficientNetV2 주요 하이퍼파라미터

- A.CLAHE: 이미지를 작은 블록 단위로 나누어 각 블록 내에서 보다 정확하게 밝기와 대비 조정

```
train_transform = A.Compose([
    A.Resize(CFG['IMG_SIZE'], CFG['IMG_SIZE']),
    # A.RandomCrop(CFG['IMG_SIZE'], CFG['IMG_SIZE']),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.ShiftScaleRotate(shift_limit=0.1, # -0.1~0.1 범위에서 무작위로 이동
                        scale_limit=0.2, # -0.2~0.2 범위에서 무작위로 스케일
                        rotate_limit=30, # -30~30도 범위에서 무작위로 회전
                        p=0.5),

    A.OneOf([
        A.MotionBlur(p=1),
        A.OpticalDistortion(p=1),
        A.GaussNoise(p=1),
        A.CLAHE(clip_limit=2, tile_grid_size=(8,8), p=1),
        A.Rotate(limit=(45, 90), p=1, border_mode=cv2.BORDER_CONSTANT)
    ], p=0.5),
    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),
    ToTensorV2()
])
```

고정 이미지 사이즈로 **resize(good)**

```
train_transform = A.Compose([
    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(shift_limit=0.1, # -0.1~0.1 범위에서 무작위로 이동
                        scale_limit=0.2, # -0.2~0.2 범위에서 무작위로 스케일
                        rotate_limit=30, # -30~30도 범위에서 무작위로 회전
                        p=0.5),

    A.OneOf([
        A.MotionBlur(p=1),
        A.OpticalDistortion(p=1),
        A.GaussNoise(p=1),
        A.CLAHE(clip_limit=2, tile_grid_size=(8,8), p=1),
        A.Rotate(limit=(45, 90), p=1, border_mode=cv2.BORDER_CONSTANT)
    ], p=0.5),
    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),
    ToTensorV2()
])
```

고정 이미지 사이즈 2배로 **resize** 했다가 **crop(bad)**

EfficientNetV2 모델 훈련

전이학습 된 `efficientnet_v2_m` 모델 사용

Loss 값의 변화가 너무 미미해서 `scheduler patience=1`로 조정

```
Eff_V2.eval()
optimizer_Eff_V2 = torch.optim.Adam(params=Eff_V2.parameters(), lr=CFG['LEARNING_RATE'])
scheduler_Eff_V2 = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer_Eff_V2,
                                                         mode='max', factor=0.1,
                                                         patience=1, verbose=True)

class BaseModel(nn.Module):
    def __init__(self, num_classes=50):
        super(BaseModel, self).__init__()
        self.backbone = models.efficientnet_v2_m(weights=True)
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(in_features=1000, out_features=num_classes),
        )

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














EfficientNetV2 모델 성능

- ❖ epochs=**20**, learning_rate=0.001, batch_size=16, **random_crop O**
 - Train_Loss = 0.66246, Valid_Loss = 1.06703, F1_Score = 0.68354
 - Test_Score = 0.47715...
- ❖ epochs=**30**, learning_rate=0.001, batch_size=16, **random_crop X**
 - Train_Loss = 0.22785, Valid_Loss = 0.99751, F1_Score = 0.74366
 - Test_Score = 0.30700...

(아쉬운 부분) Train과 Test의 결과로 봤을 때 과적합인 것을 알 수 있었지만, 할당된 GPU를 다 이용해서 Dropout이나 배치 정규화의 조정을 다 할 수 없었다.

다른 영화 작가별 분류 접근 방법

Kevin Alfianto Jangtjik, Mei-Chen Yeh, and Kai-Lung Hua. 2016. Artist-based Classification via Deep Learning with Multi-scale Weighted Pooling. In Proceedings of the 24th ACM international conference on Multimedia (MM '16). Association for Computing Machinery, New York, NY, USA, 635–639. <https://doi.org/10.1145/2964284.2967299>

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11	Class 12	Class 13
												

Multi-scale Pyramid Weighted Pooling



Layer 1

2



2

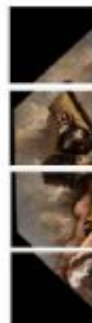
5



11

13

Layer 2



2

2

5

5



5

5

5

7



12

13

13

13



13

13

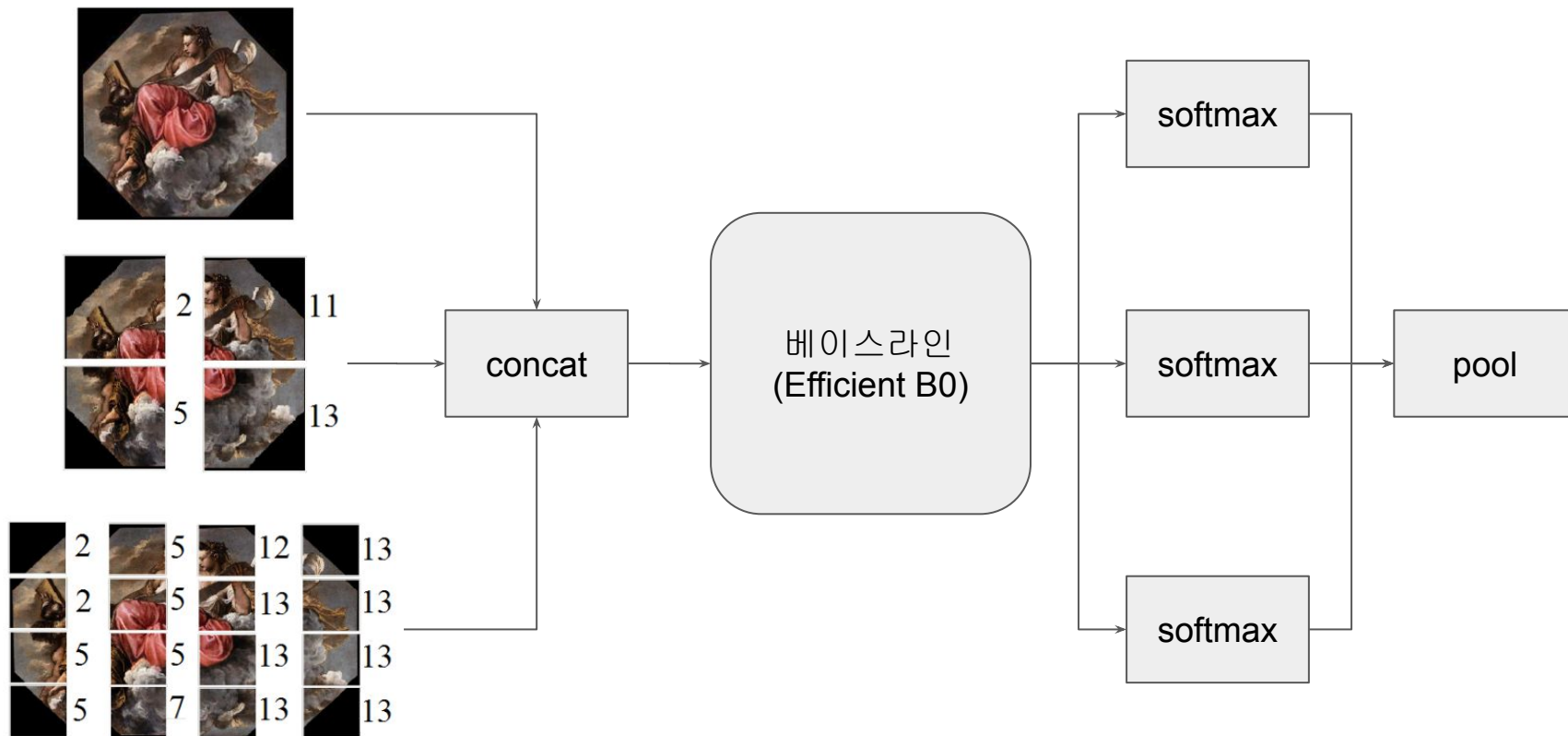
13

13

Layer 3

각 레이어 별로 예측값을 (각 타깃값의 확률) 구한 뒤에, 레이어별 가중치를 곱해서 더하는 방식으로 예측값을 취합함

모델 구조



구현 방식

```
class PyramidNet(nn.Module):
    def __init__(self, baseline, baseline_kwargs, num_baseline_output, num_classes):
        super().__init__()
        self.network = nn.Sequential(
            baseline(**baseline_kwargs),
            nn.ReLU(),
            nn.Linear(in_features=num_baseline_output, out_features=num_classes)
        )

    def __call__(self, x):
        return self.forward(x)

    def forward(self, x):
        softmax = nn.Softmax(dim=0)
        x = self.network(x)
        layers = torch.split(x, split_size_or_sections=[1, 4, 25])

        return 25 * softmax(layers[0].flatten()) + 4 * softmax(layers[1]).mean(dim=0) + softmax(layers[2]).mean(dim=0)
```

```

def __getitem__(self, idx):
    image_path = self.file_label_frame.index[idx]
    image_label = self.file_label_frame.loc[image_path, 'artist']
    image = cv2.imread(self.root_dir + image_path)
    padded = self.padder(image=image)['image']
    transform = A.Compose([A.Resize(224,224), A.Normalize(), ToTensorV2()]])

    w = padded.shape[0]
    h = padded.shape[1]
    crops_layer2 = torch.cat([
        transform(image=padded[:w//2, :h//2, :])['image'],
        transform(image=padded[:w//2, h//2:, :])['image'],
        transform(image=padded[w//2:, :h//2, :])['image'],
        transform(image=padded[w//2:, h//2:, :])['image']
    ], dim=0)
    crops_layer3 = []
    for i in range(5):
        for j in range(5):
            start_w, start_h = i * (w // 5), j * (h // 5)
            end_w, end_h = (i + 1) * (w // 5), (j + 1) * (h // 5)
            crops_layer3.append(transform(image=padded[start_w:end_w, start_h:end_h, :])['image'])
    crops_layer3 = torch.cat(crops_layer3, dim=0)

    image = torch.Tensor(transform(image=padded)['image'])
    return torch.cat([image, crops_layer2, crops_layer3], dim=0).reshape(30, 3, 224, 224), image_label

```

기대되는 결과

Table 1: Performance Evaluation

Top-1			
Experiment	Precision	Recall	F Score
Pre trained model (baseline)	65.26%	66.15%	65.7%
Multi-scale pyramid (hard)	70.48%	71.92%	71.2%
Multi-scale pyramid (soft)	70.83%	71.92%	71.4%
Multi-scale pyramid (adaptive)	71.22%	72.30%	71.7%

(a)

Top-2			
Experiment	Precision	Recall	F Score
Pre trained model (baseline)	83.72%	83.46%	83.59%
Multi-scale pyramid (hard)	84.58%	85%	84.79%
Multi-scale pyramid (soft)	87.61%	87.69%	87.65%
Multi-scale pyramid (adaptive)	88.10%	88.08%	88.09%

(b)

- 튜닝 해볼만한 것들
 - 레이어마다 다른 증강방법 적용
 - 레이어 가중치 변경
 - 레이어마다 다른 구조의 모델 사용