

Microsoft Ai School

PROJECT PRESENTATION

01

Team 5

2023.01.18

이성규
이승현
김민정
이주형
민안세

CHAPTER 1

- 프로젝트 설명
- 전체적 프로젝트 구조
- 사용된 모델

CHAPTER 2

- 프로젝트 시연

CHAPTER 3

- 프로젝트 제작 챌린지

CHAPTER 4

- 한계점 및 보안

- 프로젝트 설명

총 3개의 탭으로 이루어져 있음.

- Tap 1. Image Upload

- 원하는 이미지를 컴퓨터에서 업로드
- 예시 이미지 제공

- Tap 2. Using Webcam

- 원하는 이미지를 컴퓨터에서 업로드
- 예시 이미지 제공

- Tap 3. Making Image

- 원하는 종류의 이미지 제작

- 프로젝트 설명

총 3개의 탭으로 이루어져 있음.

- Tap 1. Image Upload

- 원하는 이미지를 컴퓨터에서 업로드
- 예시 이미지 제공

- Tap 2. Using Webcam

- 원하는 이미지를 컴퓨터에서 업로드
- 예시 이미지 제공

- Tap 3. Making Image

- 원하는 종류의 이미지 제작



주어진 이미지를 통해 어떤 종류의 쓰레기인지 판별

분류된 쓰레기의 분리 수거 방법 및 재활용 용도 확인 가능.

- 전체적 프로젝트 구조

데이터 수집



Kaggle

- 전체적 프로젝트 구조

데이터 수집

Kaggle

직접 촬영

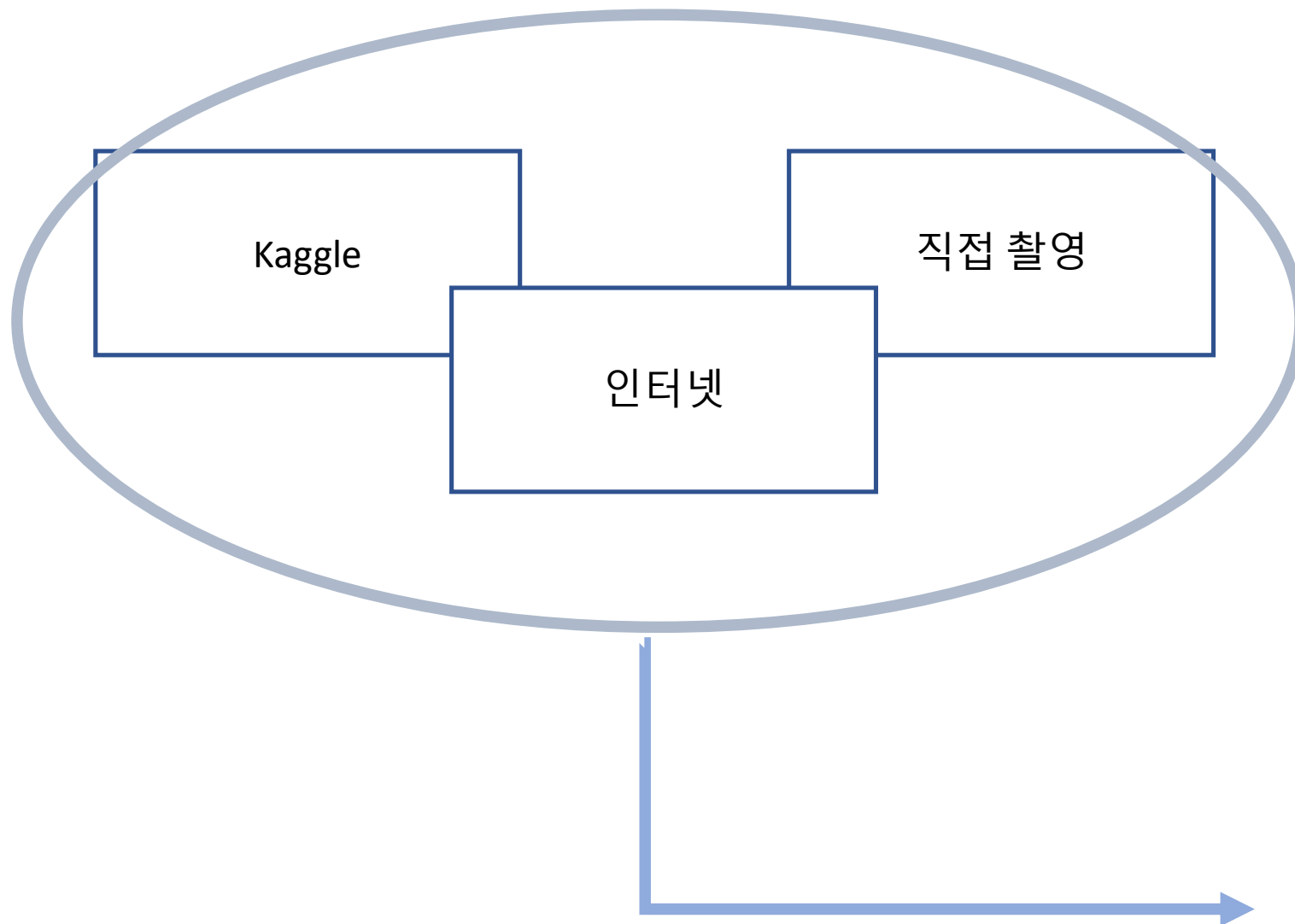
- 전체적 프로젝트 구조

데이터 수집



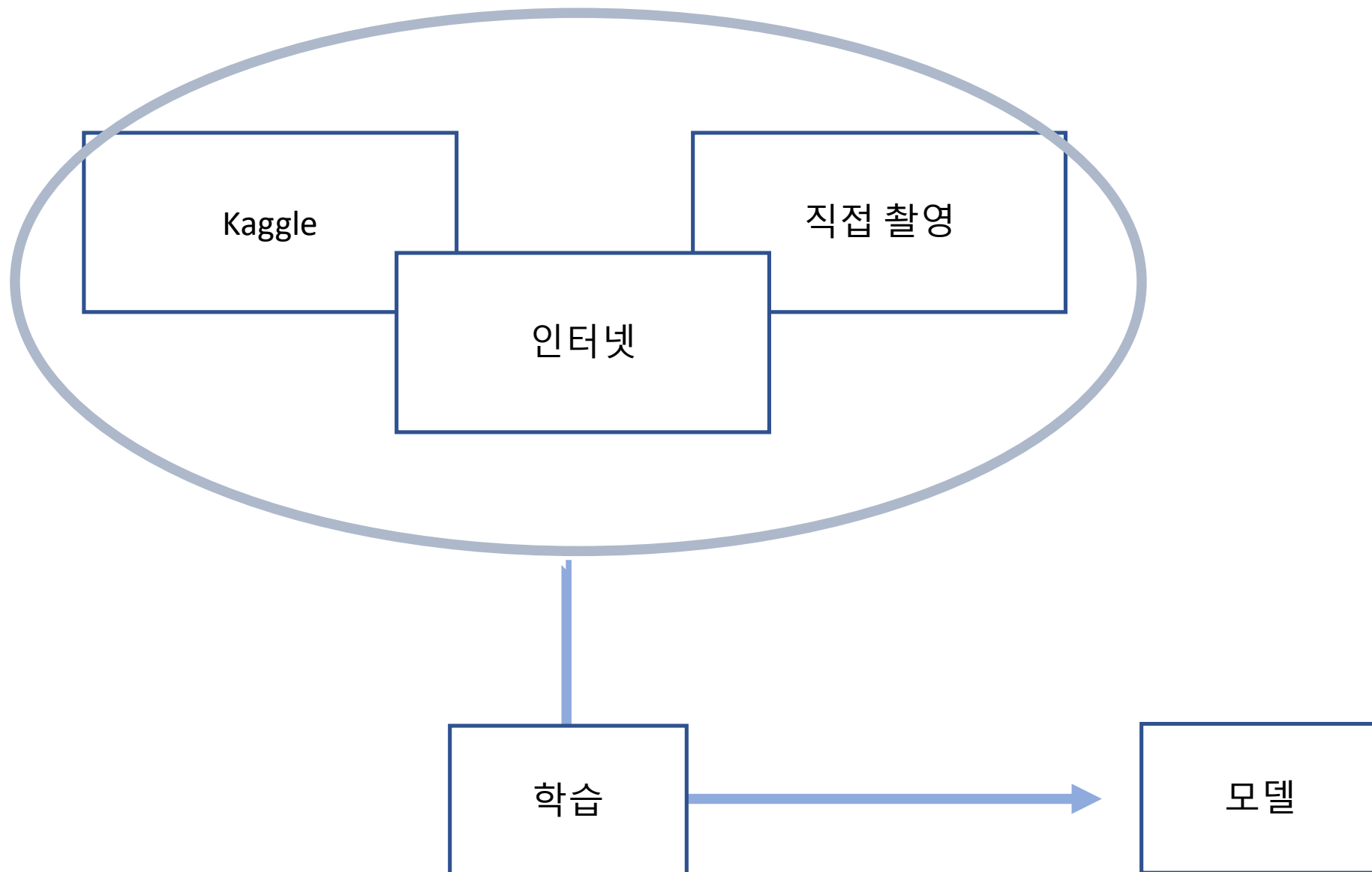
- 전체적 프로젝트 구조

데이터 수집



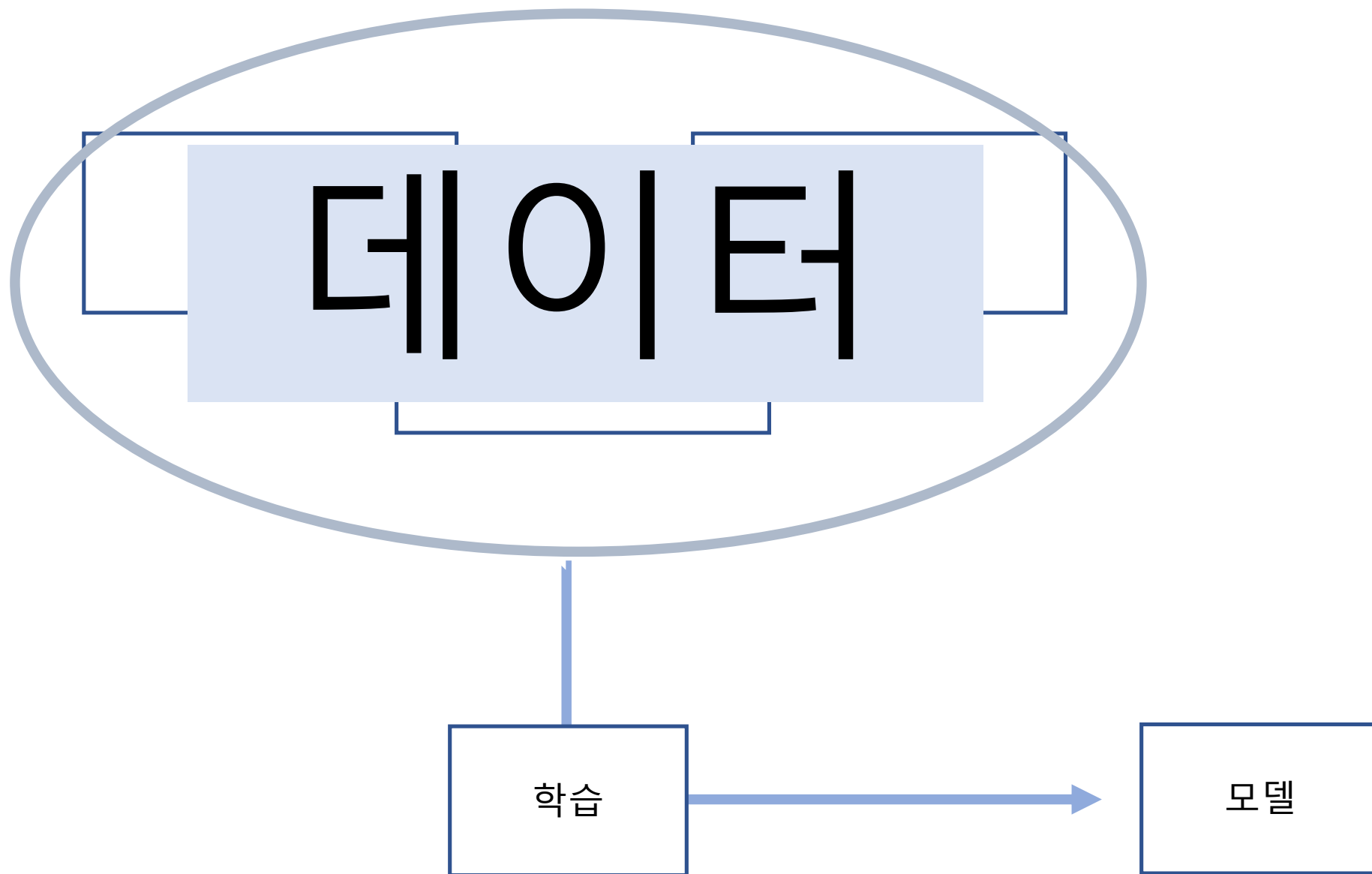
- 전체적 프로젝트 구조

데이터 수집

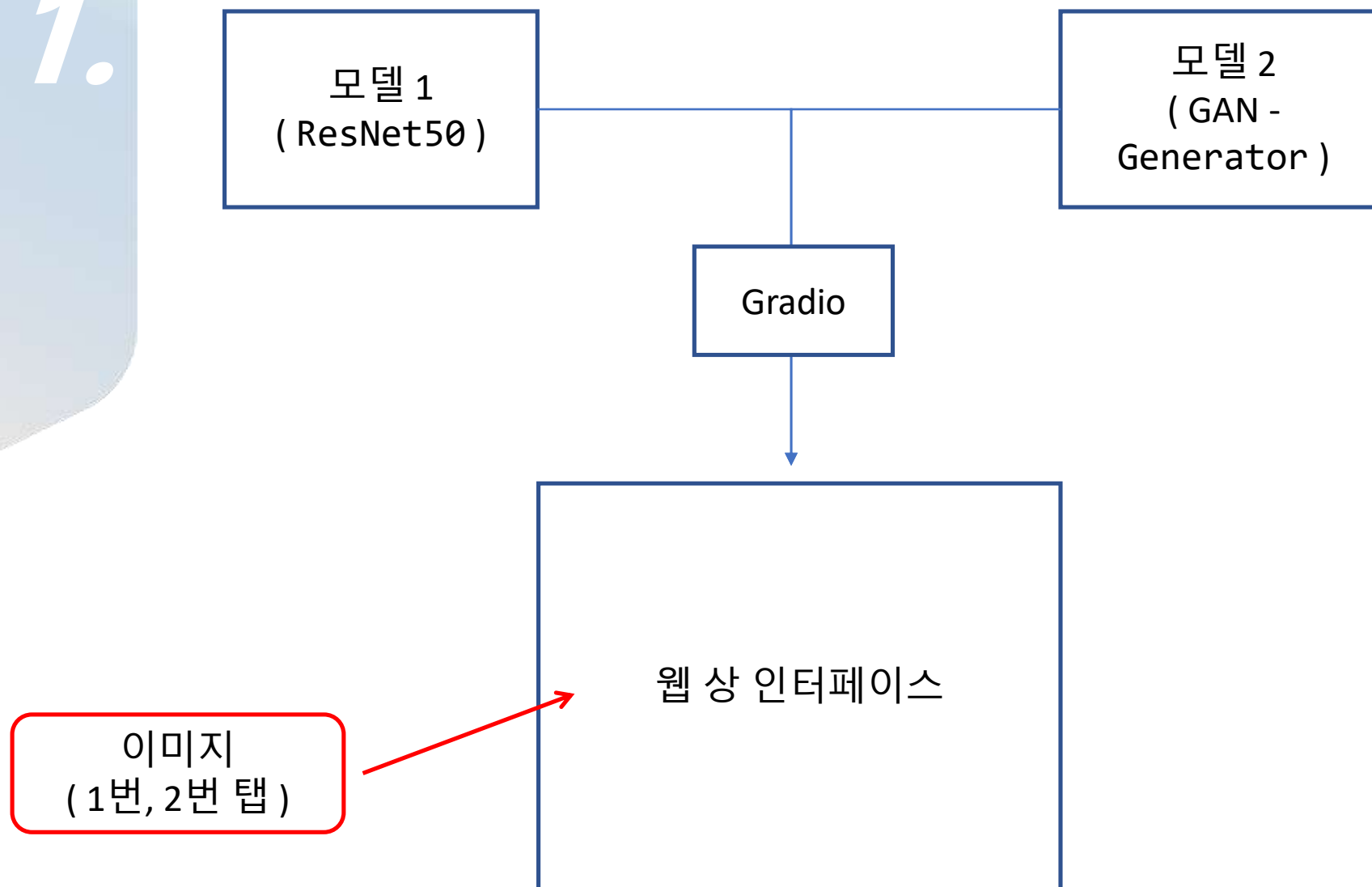


- 전체적 프로젝트 구조

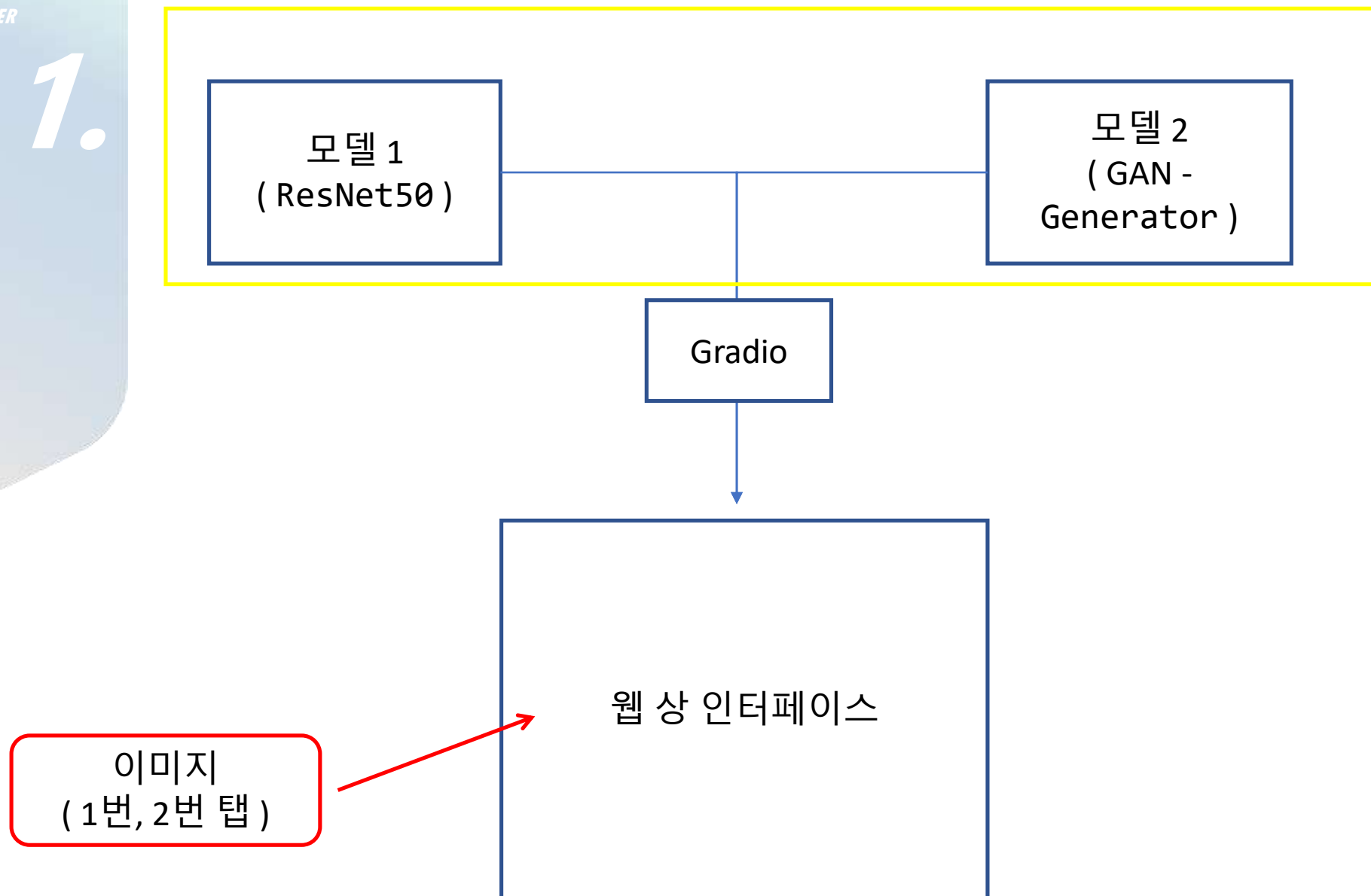
데이터 수집



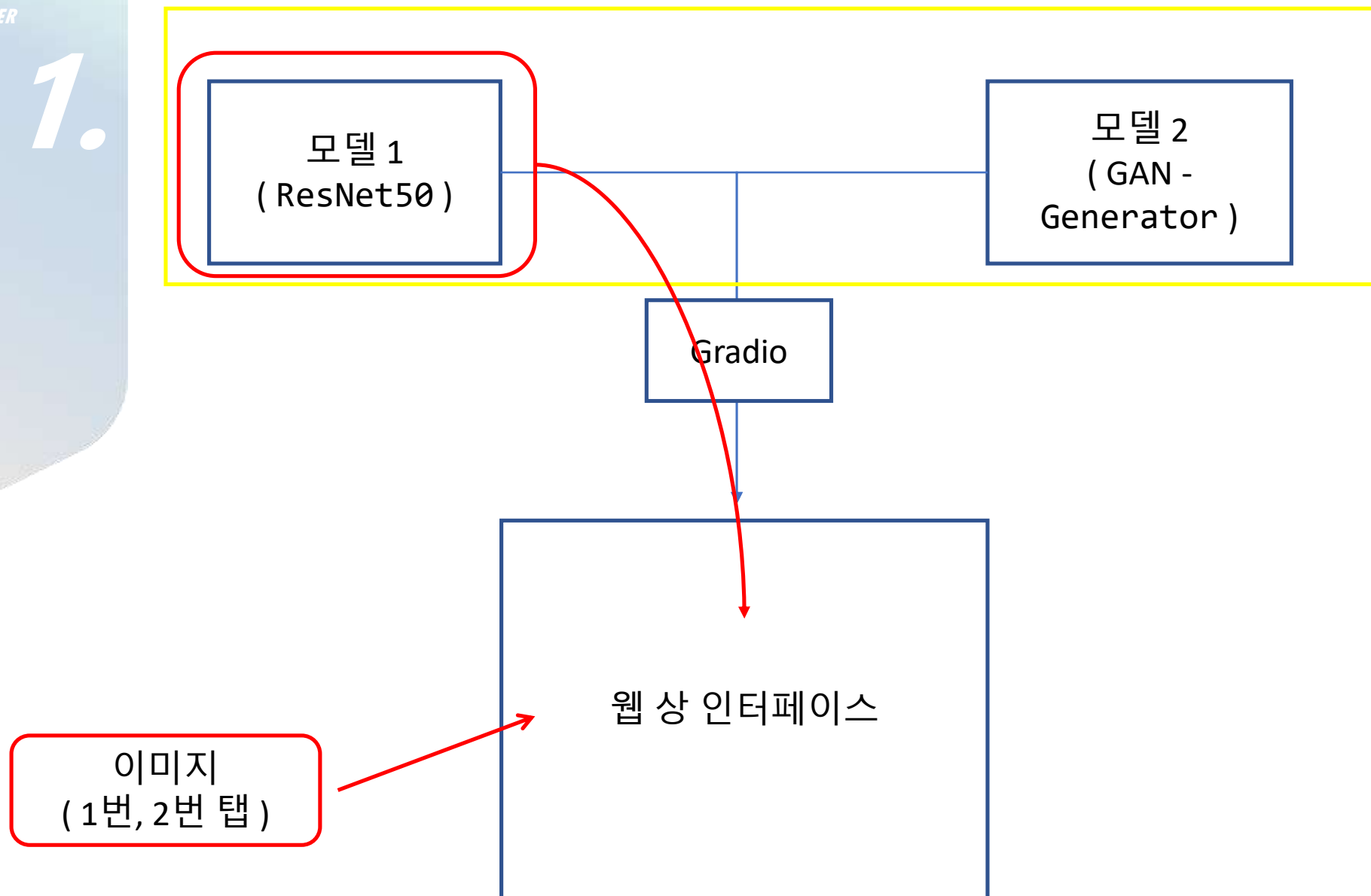
- 전체적 프로젝트 구조



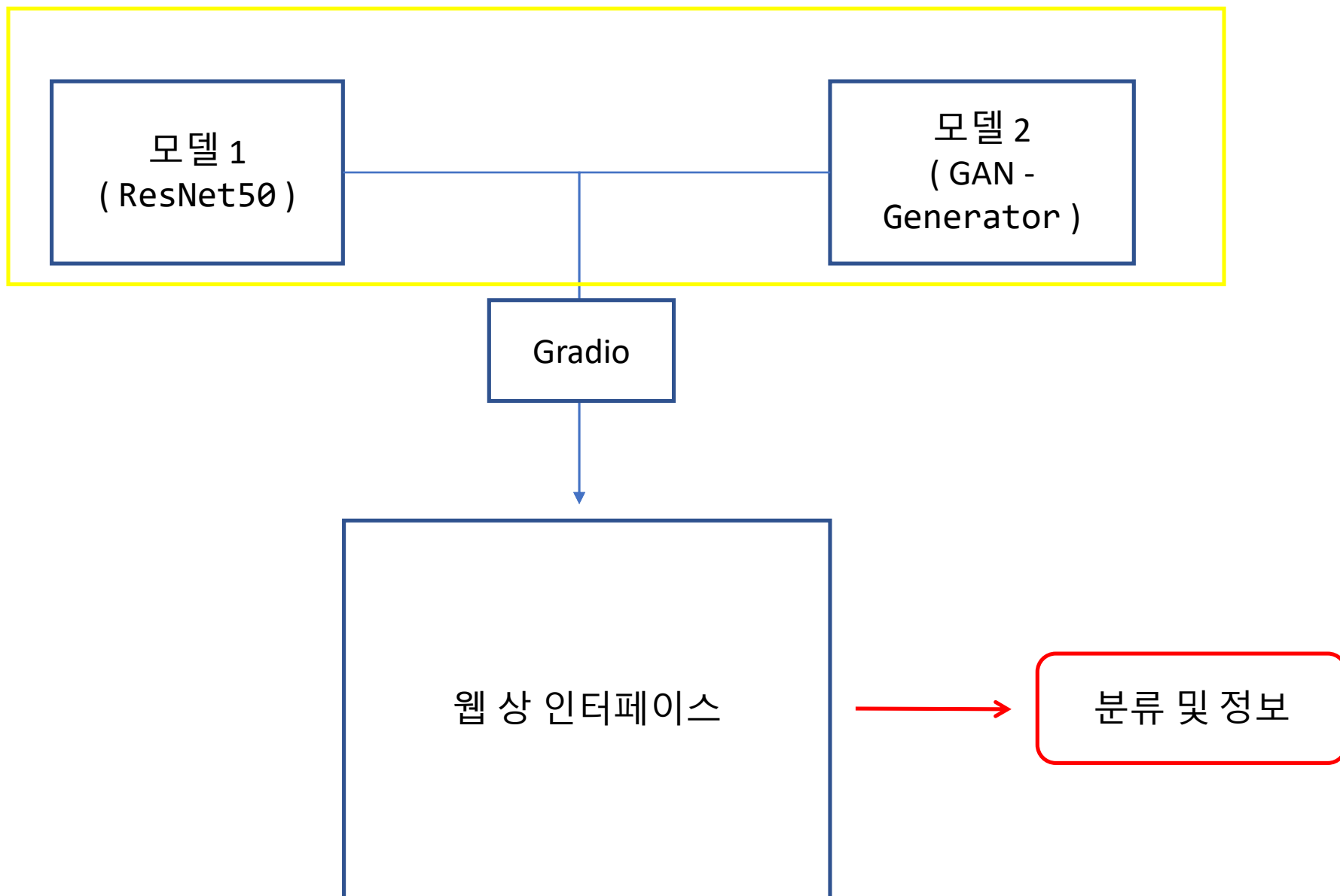
- 전체적 프로젝트 구조



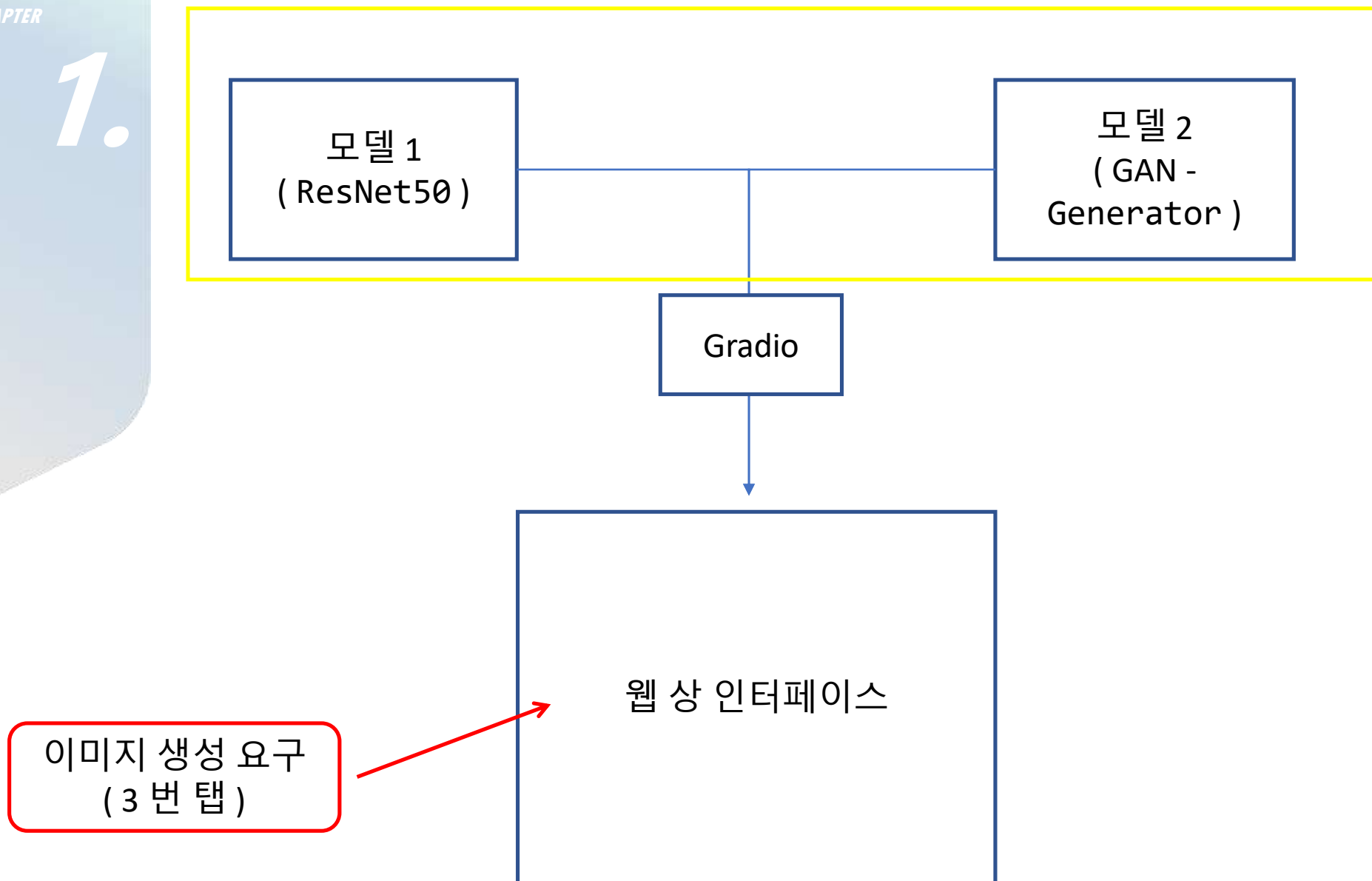
- 전체적 프로젝트 구조



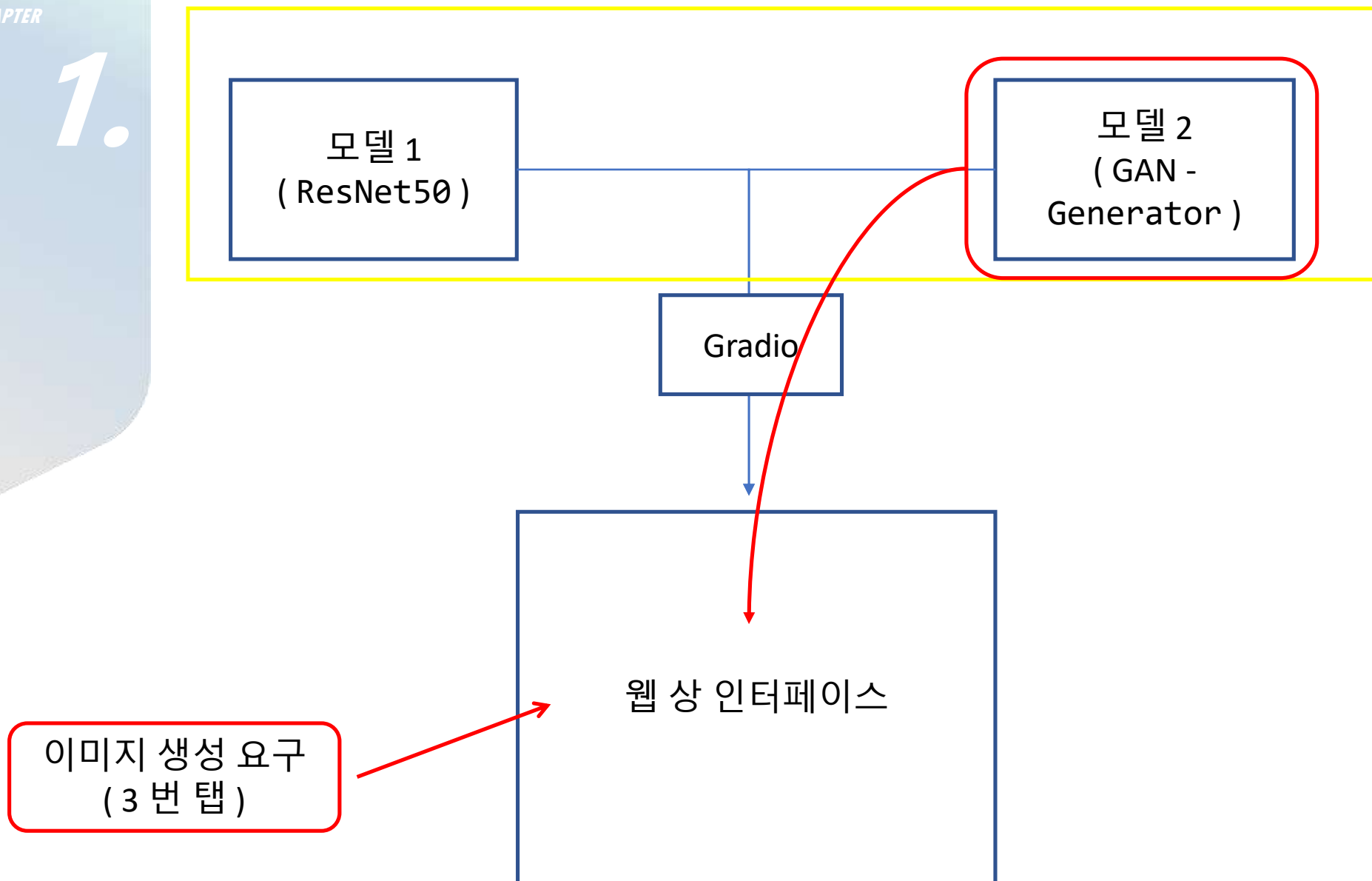
- 전체적 프로젝트 구조



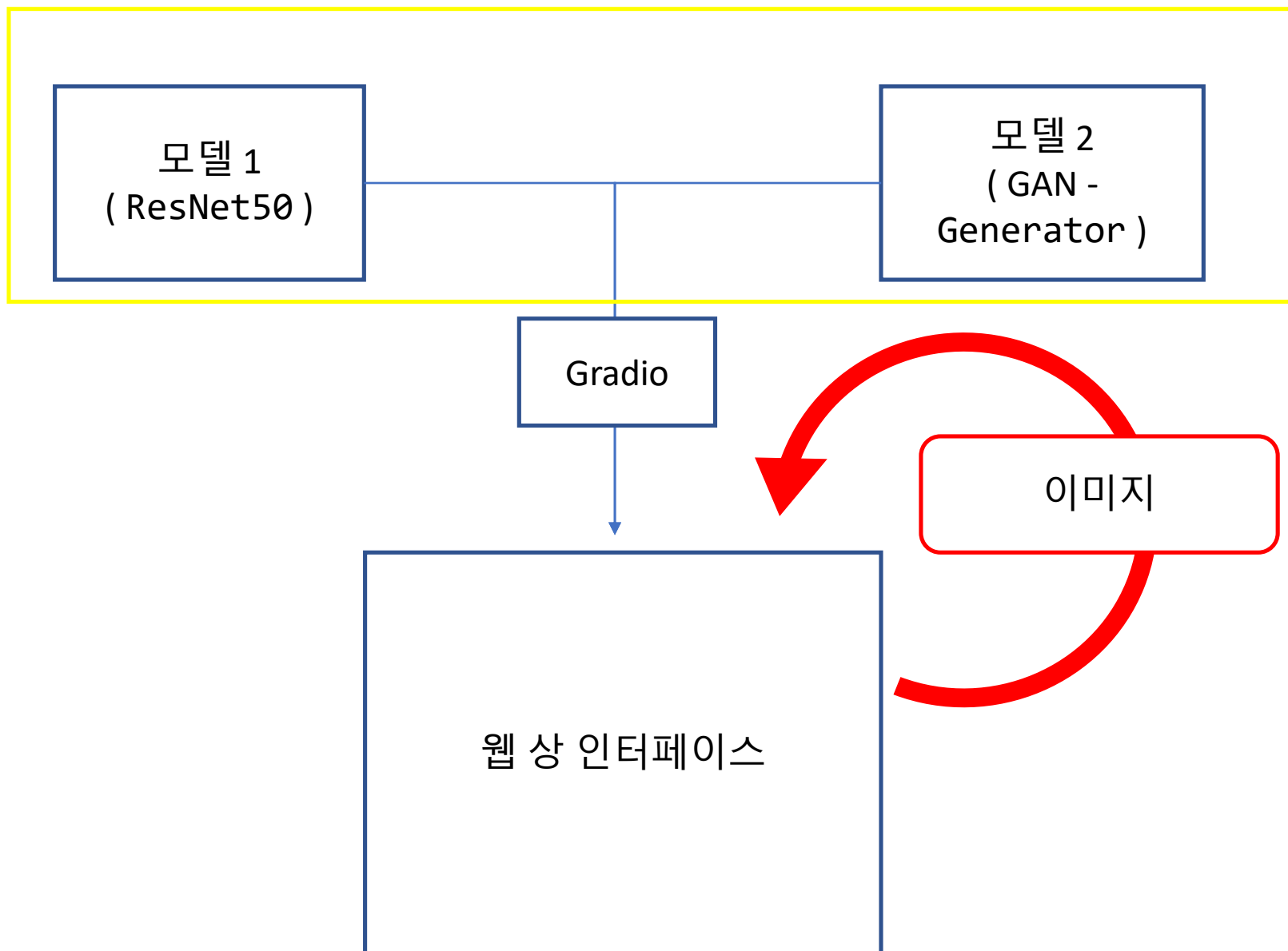
- 전체적 프로젝트 구조



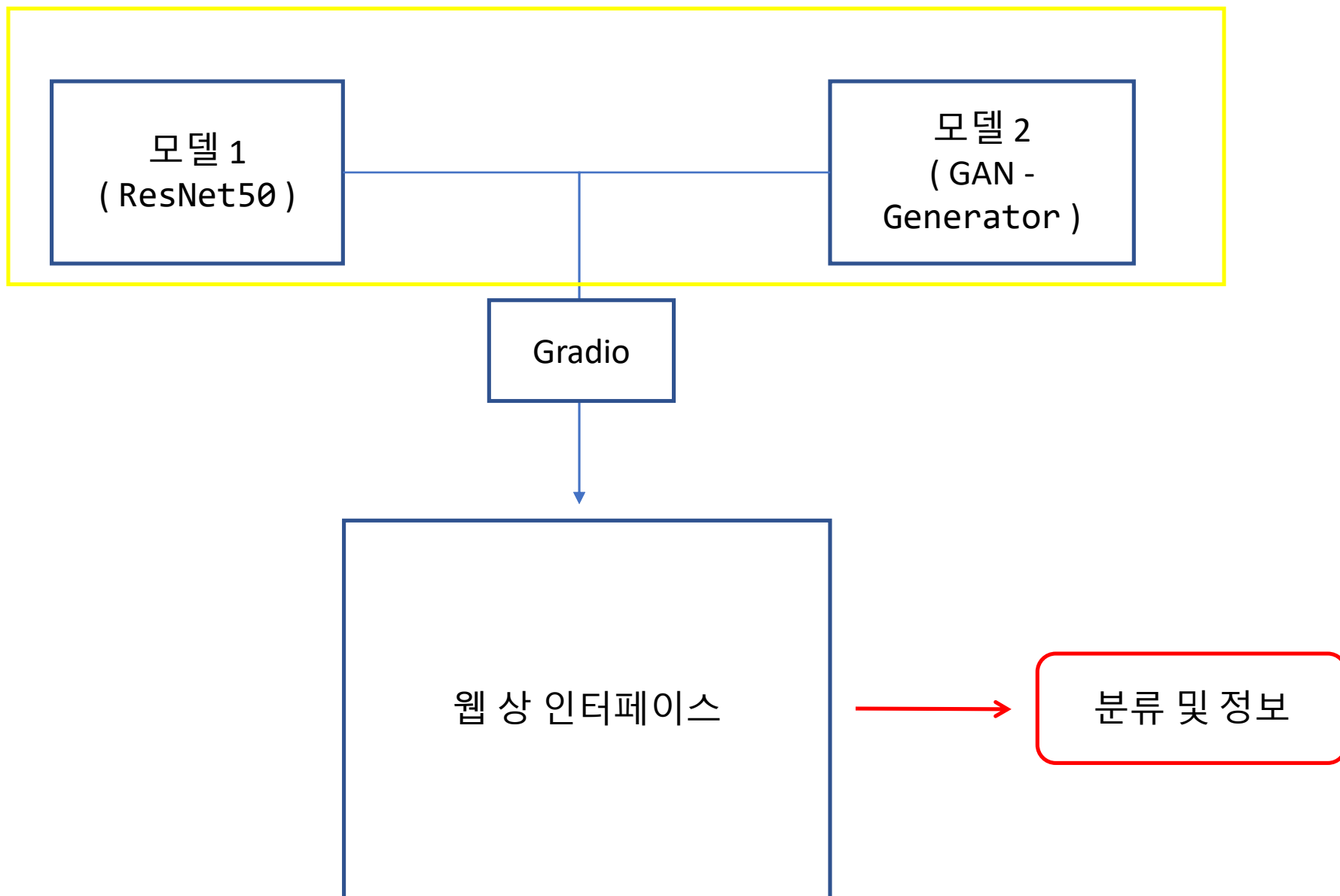
- 전체적 프로젝트 구조



- 전체적 프로젝트 구조

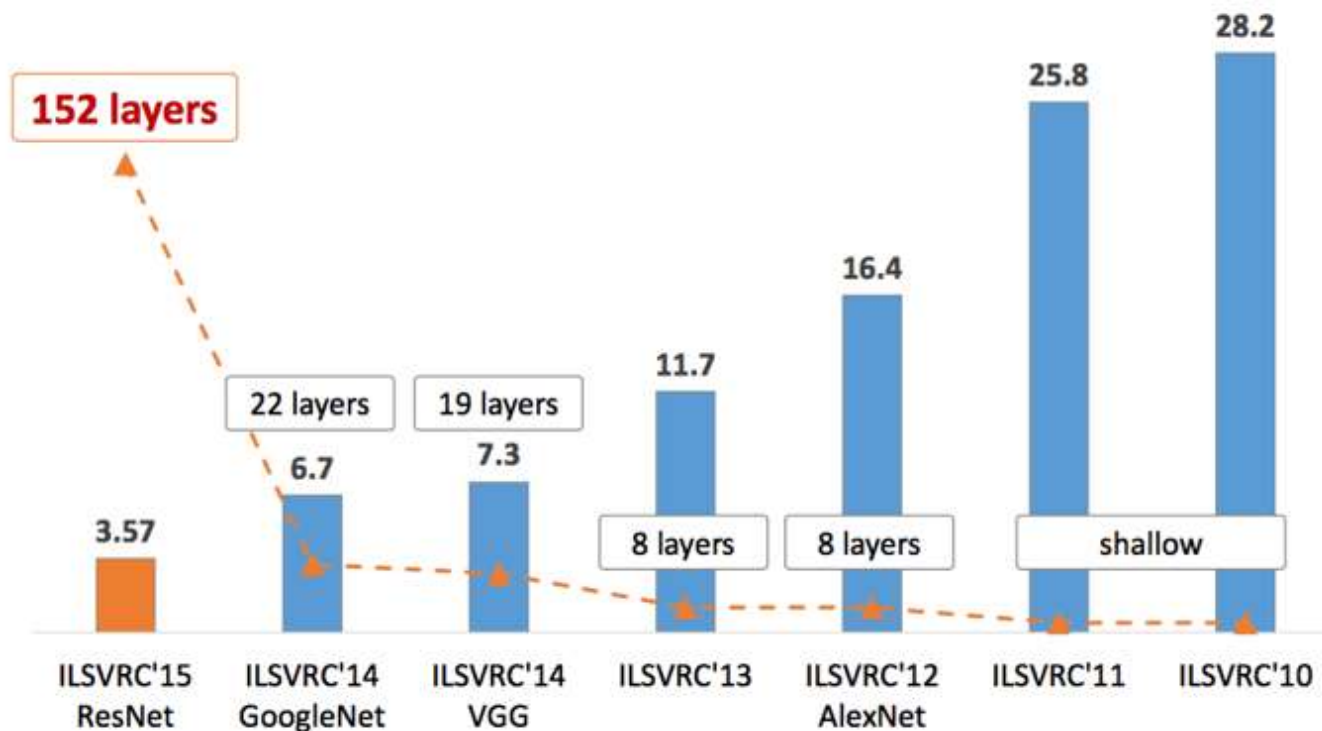


- 전체적 프로젝트 구조



모델 1
(ResNet50)

- 사용된 모델



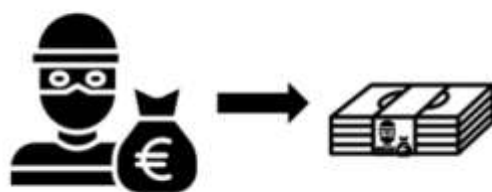
ResNet이 나오기 전에는 네트워크가 깊어지면 학습이 잘 되지 않는 현상이 있어서
깊이 있는 신경망이 많지 않았으나,

Identity map(skip connection)을 추가하여 깊이 있게 쌓을 수 있도록 해준 모델.

모델 2 (GAN - Generator)

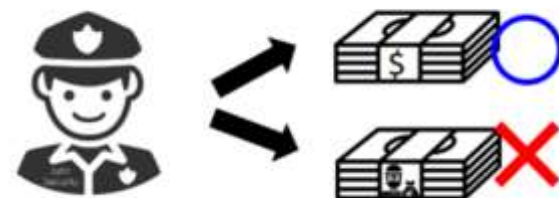
- 사용된 모델

- Ian Goodfellow(2014)에서 제안한 네트워크 모델
- 비지도학습의 대표적 알고리즘
- 서로 대립하는 역할의 두 모델이 경쟁하여 학습하는 방법론



목표 : 진짜 지폐와 최대한 비슷한
위조 지폐를 만들자 !

생성자(Generator)



목표 : 진짜 지폐와 위조 지폐를
잘 구별 해내자 !

판별자(Discriminator)

Generator 모델(G)은 Discriminator 모델(D)을 속이기 위해 최대한 진짜 같은 데이터 제작.

D모델은 위조 데이터와 진짜 데이터를 감별.

D의 예측 결과에 따라 각 모델의 Loss가 결정되고 서로 학습을 반복.

이런 경쟁 구도 속에서 두 모델의 능력이 상호 발전,

결론적으로는 G모델은 D모델이 구별하기 힘들 정도로 진짜 같은 데이터 만듦.

CHAPTER

2.

- 프로젝트 시연

<https://00564af7-9687-472f.gradio.live>

1. 데이터 수집

- Kaggle에서 주제에 맞는 12개 라벨의 이미지를 가져왔는데 이미지 사이즈가 64x64 이하로 매우 작아 식별이 어렵거나, 라벨과 맞지 않는 이미지를 발견하였습니다. 사이즈가 작은 것은 일괄 삭제가 가능하나, 라벨과 맞지 않거나 해상도가 낮은 이미지는 일일이 확인해야 하는 어려움이 있었습니다. 또한 각 라벨 별 이미지 개수의 균형을 맞추기 위해 각 데이터당 1000개를 기준으로 수집을 하였는데, 부족한 데이터를 추가하는 과정에서 각 1000개 이상의 데이터를 얻기가 쉽지 않아 오랜 시간이 소요되었습니다.

추가 데이터는 Kaggle, Ai Hub에서 비슷한 주제의 data를 구하고 웹 크롤링과 직접 촬영하는 방식으로 수집하였습니다. Kaggle에서 data 추가 수집 시, 기존 data와 중복된 data가 올라와 있어 주의가 필요했습니다.

2. DataSplit

train : valid : test = 8 : 1 : 1 비율로 나누어
dataset / label / image 구조로 저장함

이미지의 비율을 유지하기 위해 가로 세로 중 짧은 쪽은 검은 배경
으로 채워 정사각형 이미지를 만듦

간혹 cv2.imread()로 읽을 수 없는 이미지가 있어 예외처리 했음

3. 커스텀 데이터

`__init__` 에서 데이터셋의 경로, `task('train', 'valid', 'test' 중 하나)`, 를 입력 받아 해당하는 이미지의 경로들을 객체 변수에 할당
`transforms`를 받아 객체 변수에 할당

```
# Load Dataset
```

```
dataset_path = os.path.join('../', 'dataset')
```

```
train_dataset = CustomDataset(dataset_path, task='train', transforms=train_transforms)
```

```
valid_dataset = CustomDataset(dataset_path, task='valid', transforms=valid_transforms)
```

```
test_dataset = CustomDataset(dataset_path, task='test', transforms=valid_transforms)
```


3. 커스텀 데이터

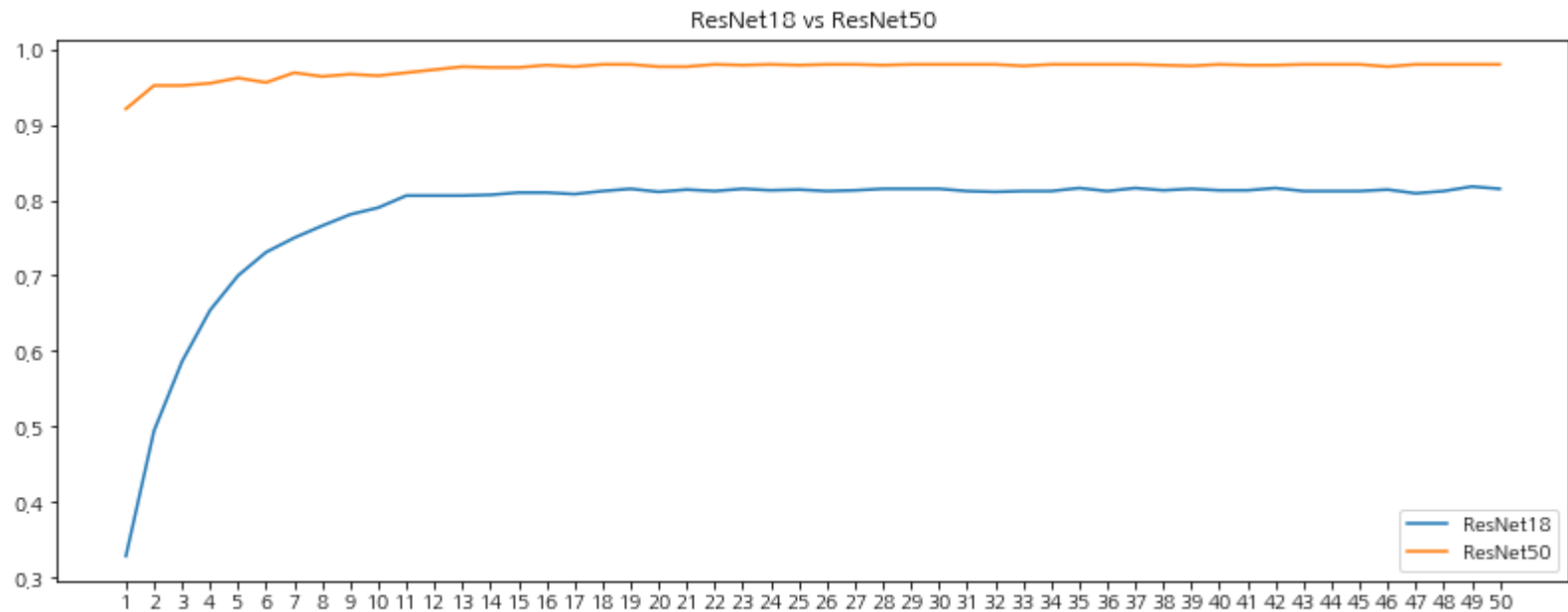
dataset 경로의 폴더 목록으로 부터 label명을 추출하여 순서대로 index 부여하고 dict자료형으로 객체 변수에 할당

`__getitem__` 은 객체 변수와 입력 받은 index로 이미지 경로의 이미지와 label을 추출하여 반환

```
class CustomDataset(Dataset):
    def __init__(self, data_path, task, transforms=None):
        image_dir = os.path.join(data_path, task)
        image_path_list = glob.glob(os.path.join(image_dir, '*', '*.png'))

        self.image_path_list = image_path_list
        self.label_dict = {label: index for index, label in enumerate(os.listdir(image_dir))}
        self.transforms = transforms
        # print(self.label_dict)
```

4. 모델 학습



재활용 쓰레기 데이터에 대하여 학습을 진행 했을 때
ResNet50의 검증 정확도가 ResNet18에 비해 앞서있어 ResNet50을 선택

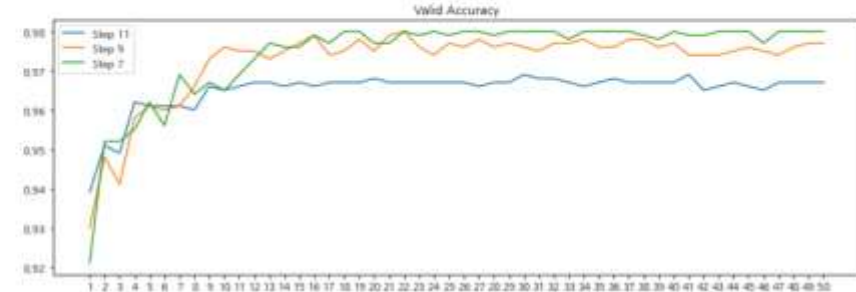
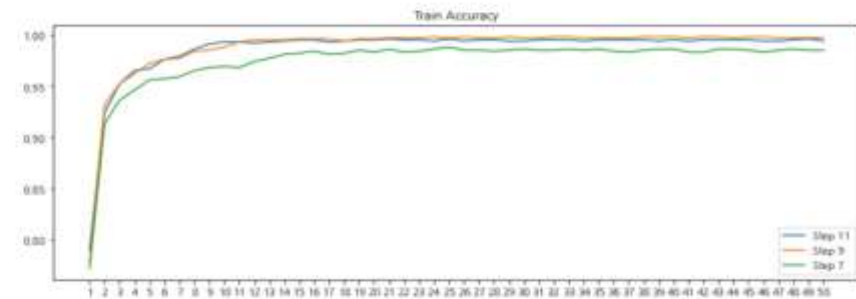
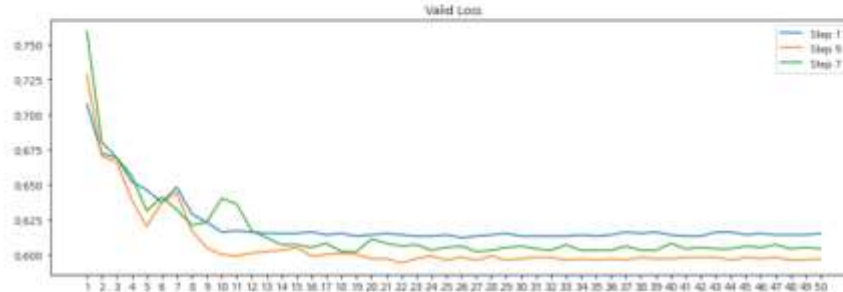
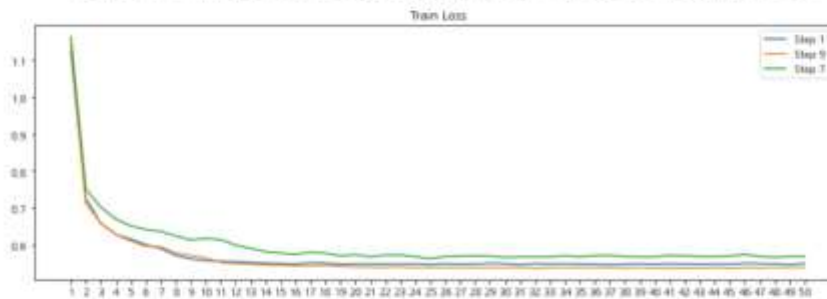
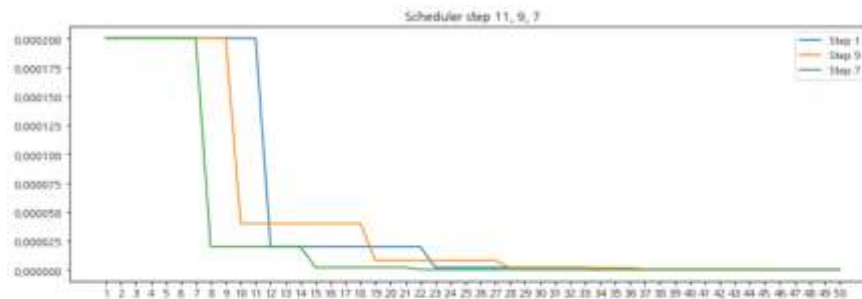
- 프로그램 제작 Challenge

CHAPTER

3

4. 모델 학습

Scheduler(step, gamma), Learning Rate



- 프로그램 제작 Challenge

CHAPTER

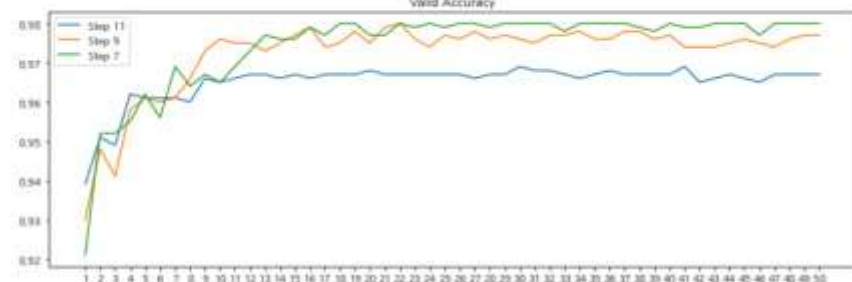
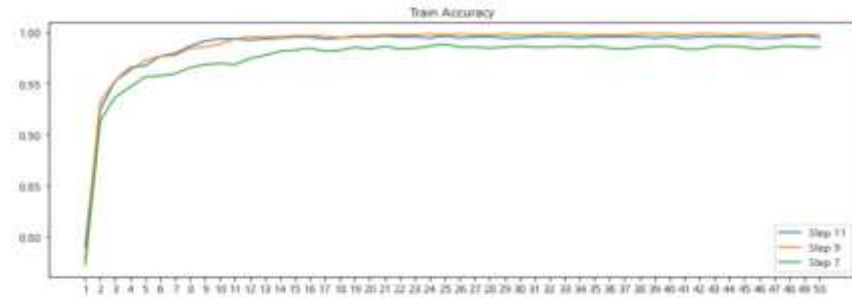
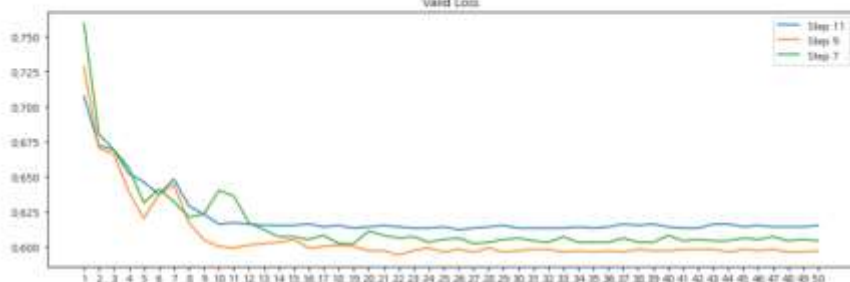
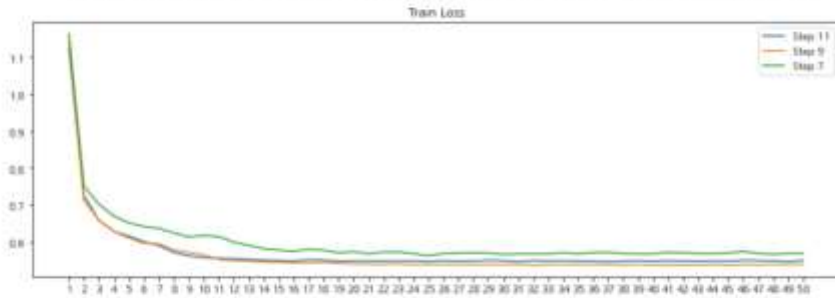
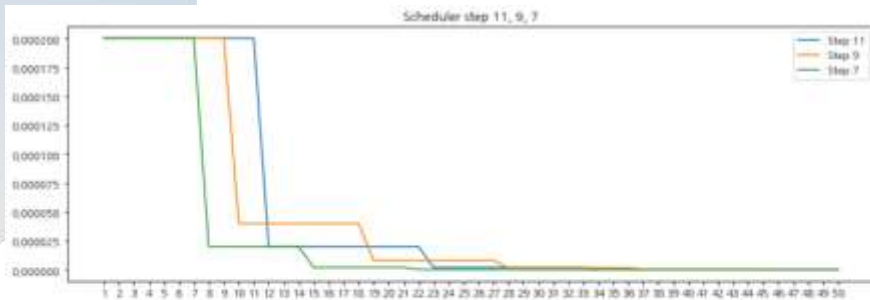
3

4. 모델 학습

Scheduler(step, gamma), Learning Rate

gamma는 0.1이었고,
step_size는 7, 9, 11로 실험했는데,
9일 때 전반적으로 좋은 성능을 보였지만

테스트 정확도는 11일 때 가장 높았기 때문에
이 모델을 사용했습니다



- 프로그램 제작 Challenge

4. 모델 학습

Augmentations

확보한 자료가 충분치 않았기 때문에
(클래스 당 1000장 내외) 다양한 효과를
주어 만회하려고 했으나, 결과적으로
검증 정확도가 상당히 높았던 모델임에도
기대와 다른 예측을 보여주는 경우가 있
었음.

(예: 건전지가 캔으로 분류되는 경우)

데이터 중 색깔에 영향을 받을 것으로 예
상되는 클래스가 있었기 때문에 색깔에 영향을
주는 효과를 제거했고, 비슷한 변형을 주는 효과
들을 서로 묶어 그 중 하나만 반영되도록 했음

```
temp_transforms = A.Compose([
    A.Resize(width=224, height=224),
    A.OneOf([
        A.ShiftScaleRotate(p=1),
        A.RandomRotate90(p=1),
    ]),
    A.OneOf([
        A.Spatter(mode='rain', p=1),
        A.Spatter(mode='mud', p=1),
    ]),
    A.OneOf([
        A.RandomBrightnessContrast(p=1),
        A.RandomToneCurve(p=1),
    ]),
    A.OneOf([
        A.GaussianBlur(p=1),
        A.GaussNoise(p=1),
    ]),
    A.RandomShadow(),
    A.HorizontalFlip(),
    A.VerticalFlip(),
    A.Normalize(),
    ToTensorV2(),
])
```

4. 모델 학습

Batch size, numworkers, pin-memory

배치의 크기가 너무 작은 경우 학습이 제대로 진행되지 않는 현상을 보여서 메모리가 감당할 수 있는 선에서 최대한으로 지정함

학습 진행 속도를 좀 더 빠르게 하기 위해

numworkers=4, pin-memory=True로 설정하여 GPU자원을 최대한 활용

5. 인터페이스 제작

모든 팀원들이 파이썬 관련 GUI 사용 경험이 없었음.
이에, 구현한 모델을 서비스 하기 위한 플랫폼 확보에 어려움을 겪음.

처음에는 3일이란 한정적인 시간을 생각하여 터미널에 input을 통해 프로그램을 진행할까 생각하였으나.
(int 값을 받아서 그 int값이 무엇이냐에따라 분기하며 print문으로 설명 생각.)

그래도 보여지는 게 있는게 좋다고 판단하여 빠르게 구현할 수 있는 방법을 생각하였음.

→ PyQt, Flask 등 다른 프레임워크도 생각하였으나 Gradio가 구현이 앞서 두 예시보다 훨씬 간편하고 머신러닝 앱에 특화된 기능임을 생각하여 채택하게 됨.

6. GAN 모델 사용

GAN이란 모델을 처음 접해 봄으로써, 여러 챌린지들이 발생하게 되었음.


1. 다른 py 파일에서의 모델 호출 : GAN의 Generator는 들어오는 이미지의 크기와 latent_dim를 인자로 줘야 했는데, 처음 GAN코드에 익숙치 않다보니 찾는데 살짝 시간을 소비하였음.
2. Generator의 모델은 아웃풋 값으로 Tensor를 내보내주는데, 이를 이미지로 바꾸는 것이 은근히 어려웠었음.
이미지에서 Tensor로 바꾸는 실습은 많이 해보았으나 Tensor에서 이미지로 바꾸는 경우는 처음이 였으며 생각보다 정규화때문에 조금 애를 먹었던 것으로 기억.

1. Gradio 기능 숙련도 미숙

Image UploadUsing WebCamMake IMG

사진이나 가진 쓰레기가 없다 구요?

걱정 마세요! 원하는 쓰레기를 무료로 만들어드립니다!

Made IMG

원하는 생성 쓰레기 선택

☒ 옷☐ 강통☐ 병☐ 종이☐ 신발

☐ 금속

← Image Making! :D

↓ 구현 예정이 있음! ↓

↓ 바로 적용 해보기 ↓

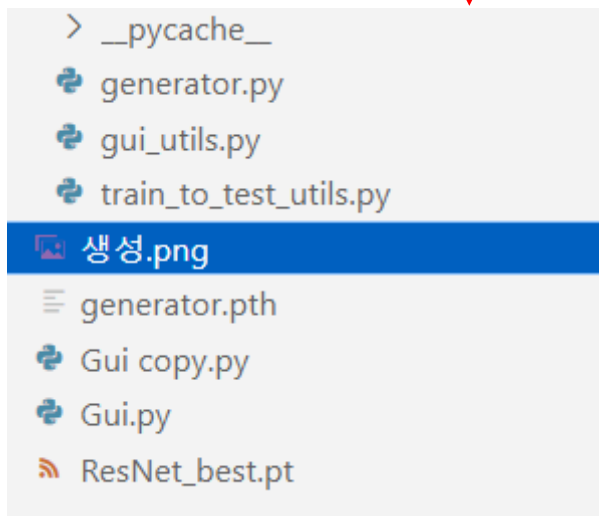
Save Image

Save New Image

1. Gradio 기능 숙련도 미숙

```
31
32 gen_images = GANmodel(z)
33
34 save_image(gen_images.data, '생성.png', normalize=True)
35
36 |
```

GAN 모델 생성 이미지 저장 가능! But...



웹을 열었던 개인의 PC에 저장하는 것이 아닌,
서버를 열고 있는 그 사람의 로컬 컴퓨터에 저장!

> 구현 실패

2. GAN 모델 관련 한계점

1. GAN 성능

처음 GAN 모델코드를 구현하고 epoch를 300 정도 돌린 결과값이 다음과 같음



2. GAN 모델 관련 한계점

1. GAN 성능

처음 GAN 모델코드를 구현하고 epoch를 300 정도 돌린 결과값이 다음과 같음



이는 성능 측면으로 상상했던 것과 다르게 많이 부족하다 생각이 들었고,

이에 성능을 보다 높이기 위해 무작정 epoch를 1000으로 늘려서 학습을 진행해 봄.

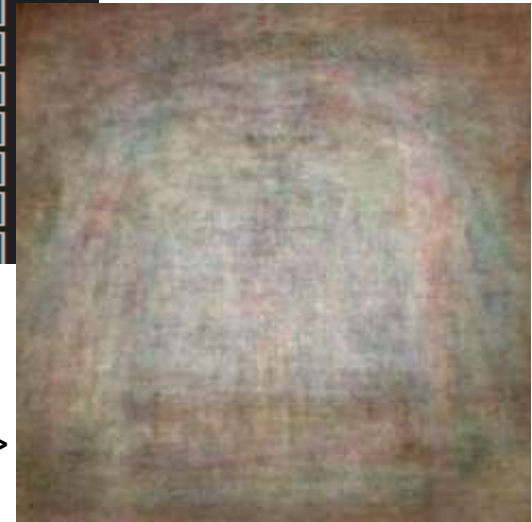
2. GAN 모델 관련 한계점

1. GAN 성능

Loss 값이 떨어질 것이라는 예상과는 다르게
오히려 Loss값은 어느 순간부터 오르기 시작했고,
생성한 이미지의 성능도 매우 떨어져 보였음.

[Epoch 844/1000]	[Batch 40/84]	[Loss_G : 8.2671]	[Loss_D : 0.1733]
[Epoch 844/1000]	[Batch 60/84]	[Loss_G : 9.2800]	[Loss_D : 0.0256]
[Epoch 844/1000]	[Batch 80/84]	[Loss_G : 7.7417]	[Loss_D : 0.0174]
[Epoch 845/1000]	[Batch 0/84]	[Loss_G : 18.8233]	[Loss_D : 0.1252]
[Epoch 845/1000]	[Batch 20/84]	[Loss_G : 11.4779]	[Loss_D : 0.0162]
[Epoch 845/1000]	[Batch 40/84]	[Loss_G : 10.3525]	[Loss_D : 0.0898]
[Epoch 845/1000]	[Batch 60/84]	[Loss_G : 8.5970]	[Loss_D : 0.0465]
[Epoch 845/1000]	[Batch 80/84]	[Loss_G : 18.4447]	[Loss_D : 0.9708]
[Epoch 846/1000]	[Batch 0/84]	[Loss_G : 16.9495]	[Loss_D : 3.1977]
[Epoch 846/1000]	[Batch 20/84]	[Loss_G : 7.0386]	[Loss_D : 0.0792]

Epoch에 따라 오히려
서서히 형체가 더 흐려짐 ->



2. GAN 모델 관련 한계점

1. GAN 성능

이에 **learning rate, decay of first order momentum of gradient** 등 하이퍼 파라미터들을 조금씩 바꾸어 학습을 진행해 보았으나

눈에 띄게 성능이 좋아지진 않았음.

그나마 500 epoch에 $5e-4$ LR 정도가 성능이 좋다고 판단하였고, 이 모델을 이번 프로젝트에 채용하였음.

바지, 치마 등 적어도 형태는 구별이 가능함. >>>



- 한계점 및 보안

CHAPTER

4.

2. GAN 모델 관련 한계점

1. GAN 성능

어째서 일까?

2. GAN 모델 관련 한계점

1. GAN 성능

어째서 일까?

1. GAN의 최신 모델을 사용해 보려고 노력하였으나
현 실력으로는 힘들다고 판단되어,
상대적으로 따라서 코드 작성하기가 쉬운 GAN의 초기 모델 GIt을 참고하여
GAN모델을 구현하였다.

-> 초기 GAN과 최신 GAN의 성능차이가 있어서 원하는 기대 값 만큼
(구별하기 힘들 정도로 완전 진짜 같은 이미지)
결과가 안 나온 것이 아닐까 판단.

2. GAN 모델 관련 한계점

1. GAN 성능

어째서 일까?

2. 원본 Git에서의 GAN 코드와 구현한 GAN의 코드는 input 이미지 사이즈가 다르다.

(현 학습에 사용된 이미지 크기에 따라 input 이미지의 크기를 변경하여 모델을 학습하였다.)

-> Git에 최적화로 기본 설정되어 있는 파라미터들이 우리가 사용한 이미지 크기에 최적인 값이 아닐 것이다.

2. GAN 모델 관련 한계점

2. 시간 부족에 따른 GAN 적응

쓰레기 분리수거 프로그램을 사용해 보면 알겠지만,
모든 라벨 값에 대해 이미지 생성이 구현되어 있진 않다.



병 이미지를 생성하려 하면 다음과 같은 팝업이 뜬다.

2. GAN 모델 관련 한계점

2. 시간 부족에 따른 GAN 적용

실력 대비 시간이 부족하여 모든 라벨에 GAN 모델 적용을 하지 못하였습니다.

1. GAN모델은 기본적으로 300이상의 epoch를 거쳐야 어느 정도 성능이 나온다.
(모델 학습 시간이 매우 길다.)
2. 주어진 시간이 3일 정도였으나, 모델 학습 코드 구현과 추가적으로 인터페이스 관련 코드 학습도 병행하느라 미처 모든 라벨 값에 GAN모델을 학습시키지 못했습니다.
3. GAN 모델 자체를 학습 및 구현하는데도 시간소요가 조금 많이 되었습니다.

→ 다음에 시간이 된다면 추가적으로 모든 라벨 값에 대해 이미지 생성 모델을 적용시키고 싶습니다.

Microsoft Ai School

Classification 문제 해결

02

Team 5

2023.01.18

이성규
이승현
김민정
이주형
민안세

1. 금속 표면

- Data 전 처리
- 사용 모델
- 모델 성능 비교
- 결과 분석

2. 얼굴 분류

- Data 전 처리
- Test 이미지 수집
- 사용 모델
- 모델 성능 비교
- 결과 분석

- Data 전 처리

금속 표면

1.

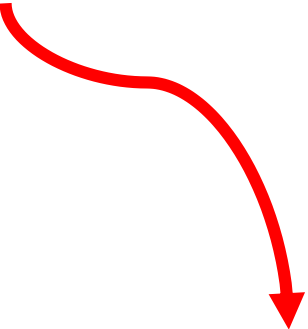
주어진 Json으로 부터 정보를 받아, Json 정보에 따라 이미지를 자른 후 검정색으로 패딩하여 이미지의 크기를 모두 통일 시켜 주었습니다.

```
label = item['label']  
bbox = item['bbox']  
# print(label, bbox)
```

```
xmin, ymin, xmax, ymax = bbox[0], bbox[1], bbox[2], bbox[3]  
# print(xmin, xmax, ymin, ymax)
```

```
image = cv2.imread(image_path)  
crop_image = image[ymin:ymax, xmin:xmax, :]
```

```
squire_image = np.zeros((x, y, channels), np.uint8)  
squire_image[int((y - height) / 2):int(y - (y - height) / 2),  
|      |      |      int((x - width) / 2):int(x - (x - width) / 2)] = origin_image
```



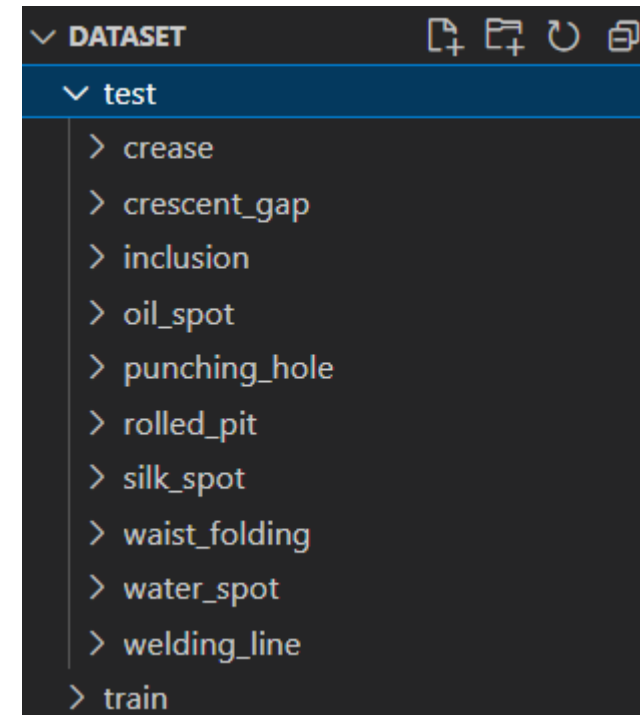
- Data 전 처리

금속 표면

1.

그 이후 Json에 있는 이미지의 Label 에 따라 폴더를 나눠 Train과 Test 이미지 분류를 해주었습니다.

```
def copy_data(task, data):  
    for index, image_dict in enumerate(data):  
        # print(image_dict)  
        # exit()  
  
        label = image_dict['label']  
        temp_image = image_dict['image']  
        image = make_square_image(temp_image)  
  
        dir_path = os.path.join('../', 'dataset', task, label)  
        dest_path = os.path.join(dir_path, '{}_{}.png'.format(label, index))  
  
        os.makedirs(dir_path, exist_ok=True)  
        cv2.imwrite(dest_path, image)
```

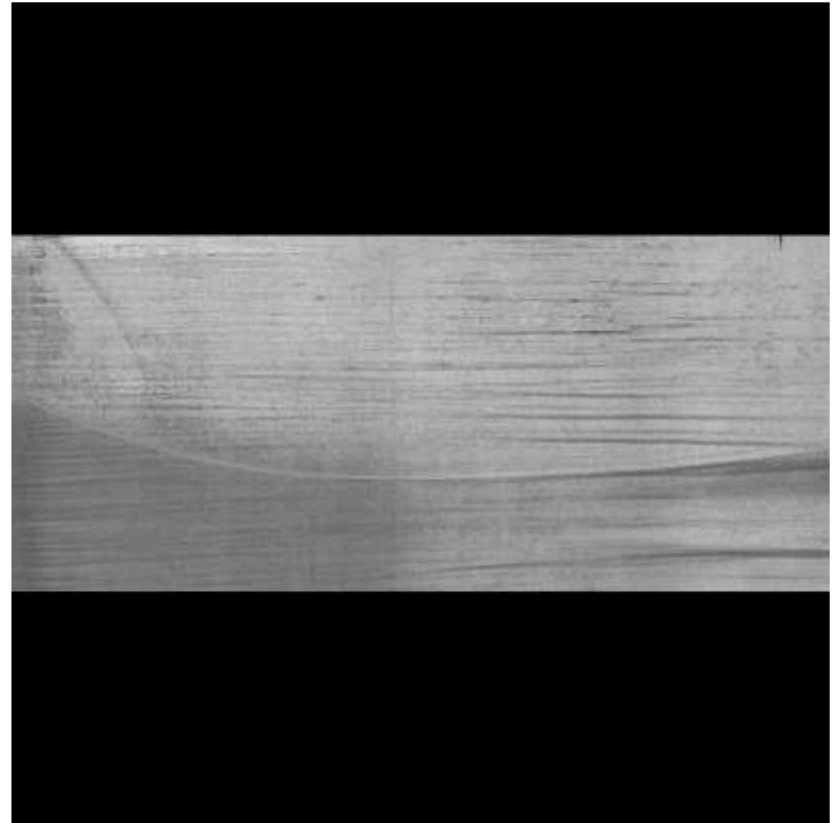
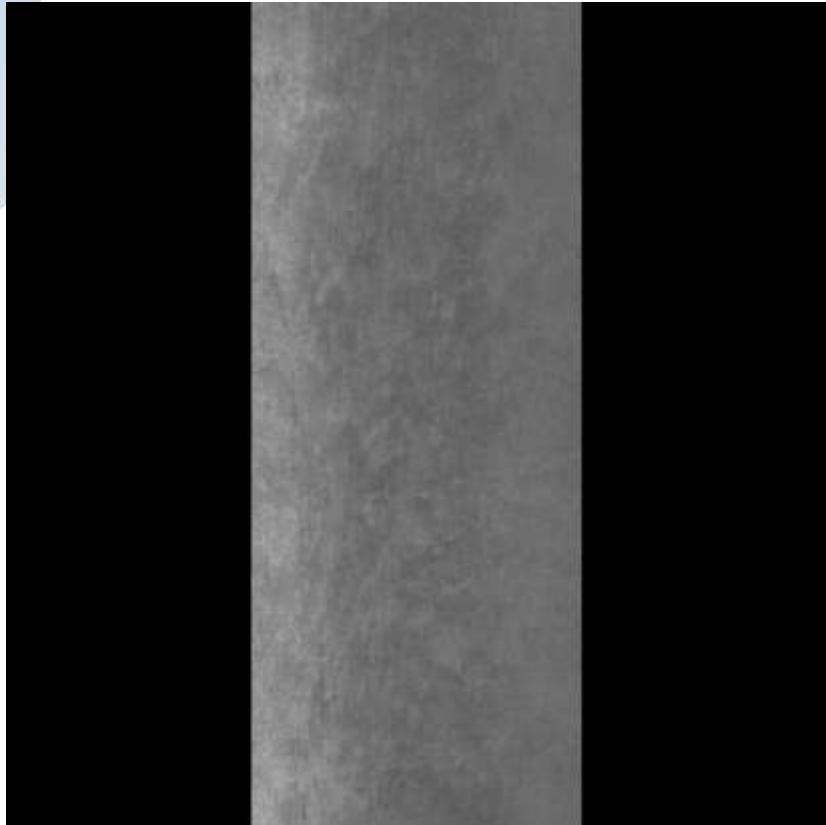


- Data 전 처리

금속 표면

1.

Train 및 Test 이미지 예시



- 사용 모델

1. efficientnet_b0

2. ResNet50

3. DenseNet201

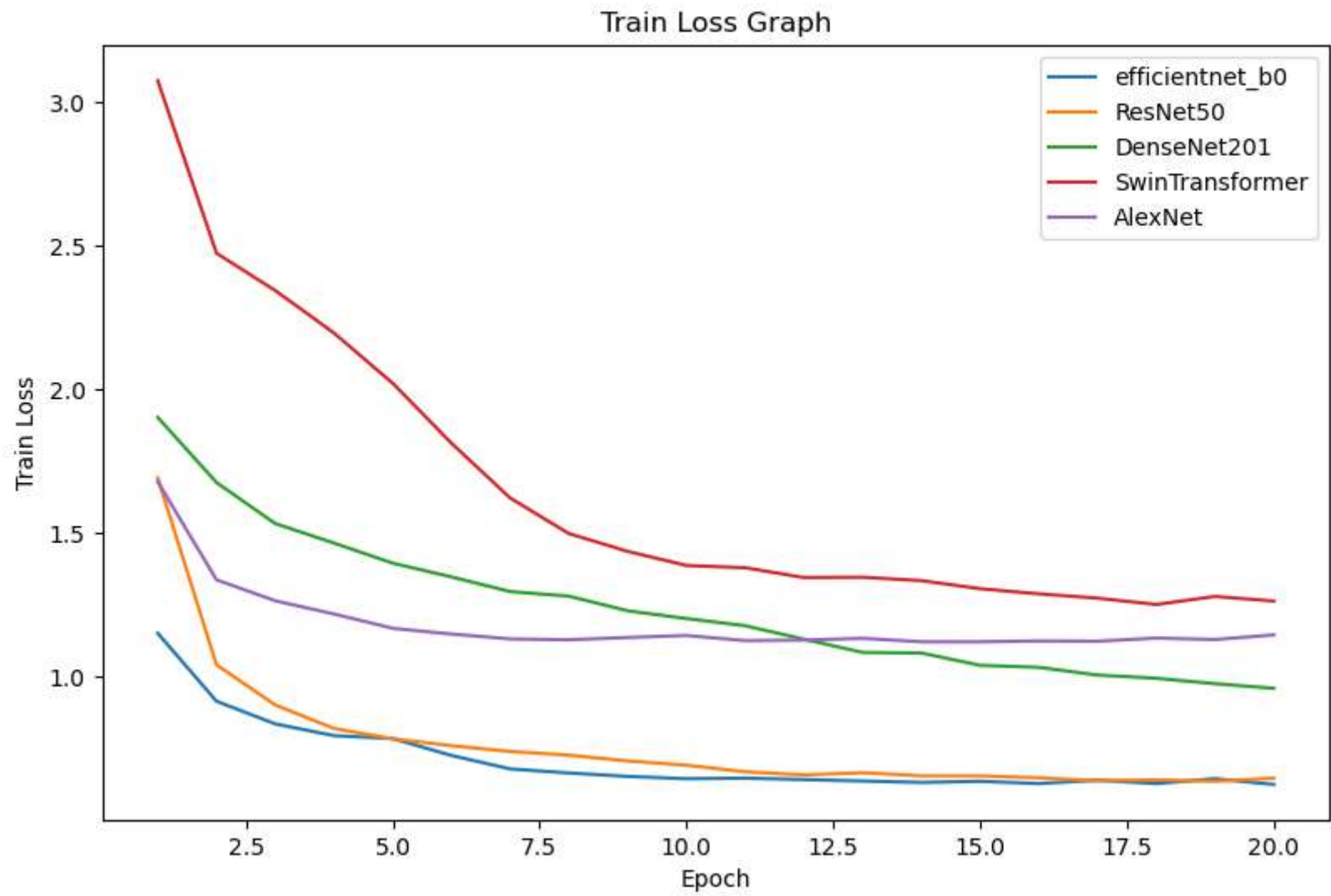
4. SwinTransformer

5. AlexNet

- 모델 성능 비교

금속 표면

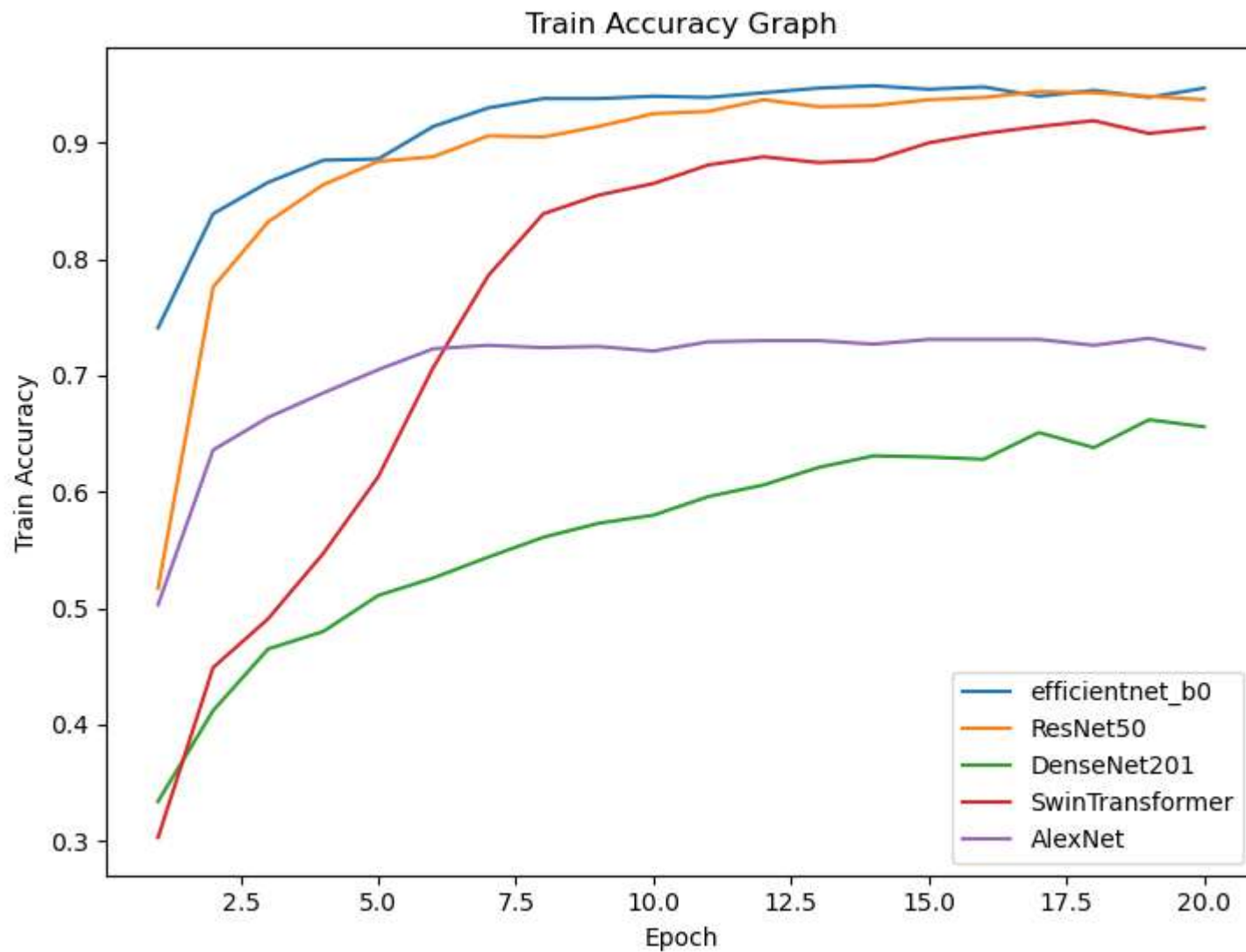
1.



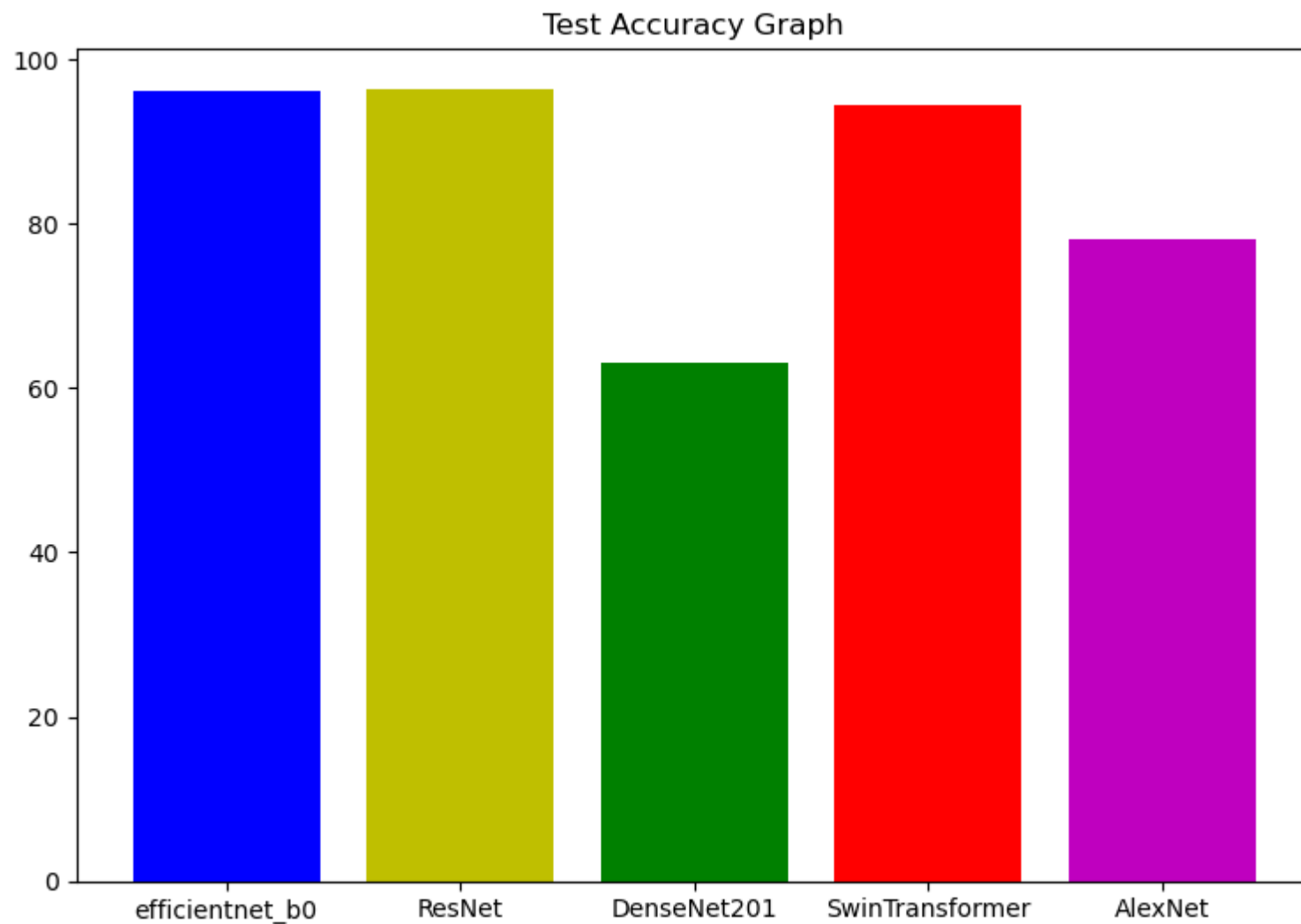
- 모델 성능 비교

금속 표면

1.



- 모델 성능 비교



efficientnet_b0 = 96.111

ResNet50 = 96.38

DenseNet201 = 63.05

SwinTransformer = 94.44

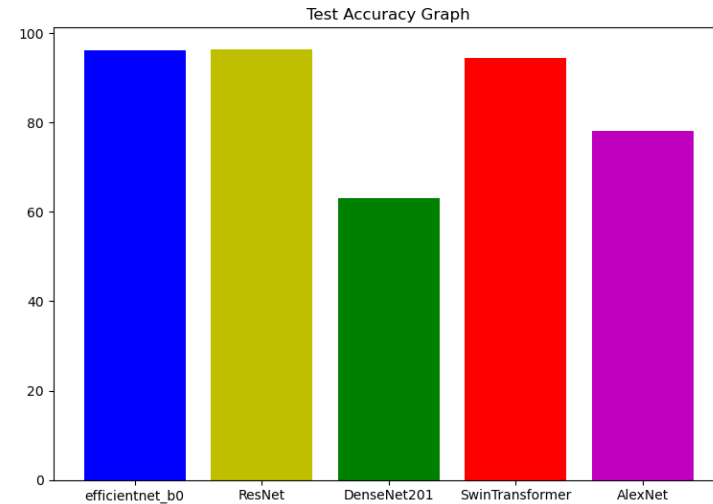
AlexNet = 78.05

- 결과 분석

금속 표면

1.

efficientnet_b0 = 96.111
ResNet50 = 96.38
DenseNet201 = 63.05
SwinTransformer = 94.44
AlexNet = 78.05



1. 최종적 Test Acc만 보았을 땐 성능이 대략적으로 다음과 같이 나왔습니다.

ResNet >= efficientnet_b0 > SwinTransformer > AlexNet > DenseNet201

- 결과 분석

금속 표면

1.

2. ResNet50 vs efficientnet_b0

ResNet과 efficientnet은 성능 측면에서 매우 우수하고 비슷하게 결과가 나왔습니다.

Efficientnet 은 기본적으로 ResNet50을 기반으로 layer수를 더욱 늘리며 정확도 뿐 아니라 추론속도도 향상된 모델이라고 소개가 나옵니다.

[Compared with the widely used [ResNet-50](#), our EfficientNet-B4 improves the top-1 accuracy from 76.3% of ResNet-50 to 82.6% (+6.3%), under similar FLOPS constraint.

출처 : <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>]

저희 실험에서도 Loss 그래프와 Acc 그래프를 보면, ResNet50 보다 더 빠르게 성능이 향상되는 efficientnet을 보실 수 있습니다.

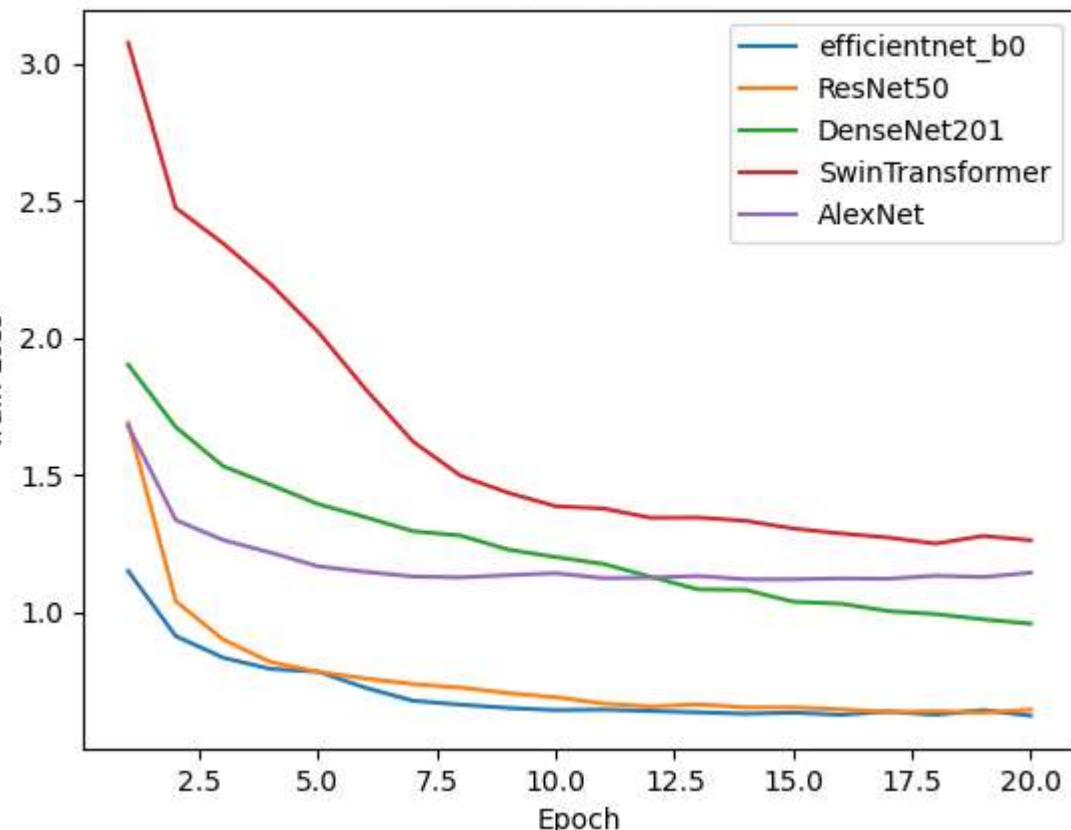
- 결과 분석

금속 표면

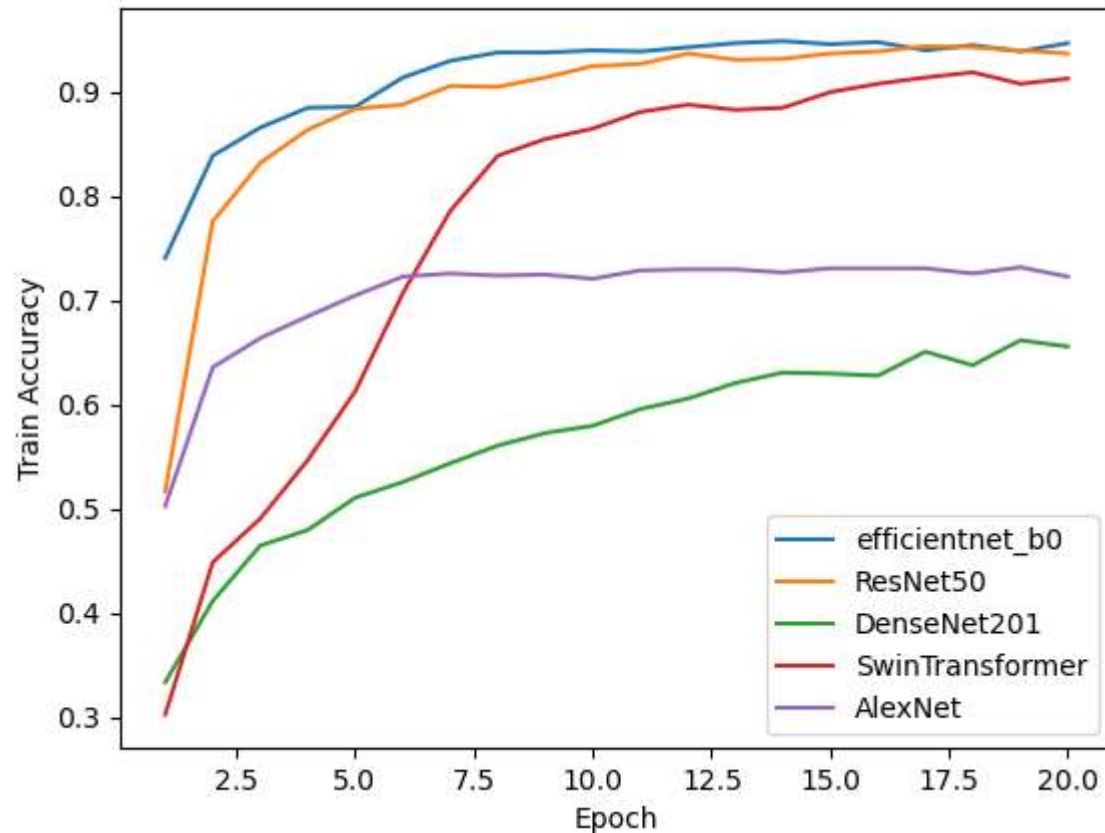
1.

저희 실험에서도 Loss 그래프와 Acc 그래프를 보면, ResNet50 보다 더 빠르게 성능이 향상되는 efficientnet을 보실 수 있습니다.

Train Loss Graph



Train Accuracy Graph



- 결과 분석

금속 표면

1.

3. AlexNet 보다 안 좋은 DenseNet201 ????

AlexNet은 5 convolutional layers + 3 dense layers 로 이루어진 거의 최초의 CNN 모델입니다.

그에 반해 DenseNet은 ResNet에서 컨볼루션 방법만 바꾼 발전된 모델로, AlexNet과는 비교가 안될 정도로 깊은 네트워크와 보다 좋은 성능을 나타냅니다.

- 결과 분석

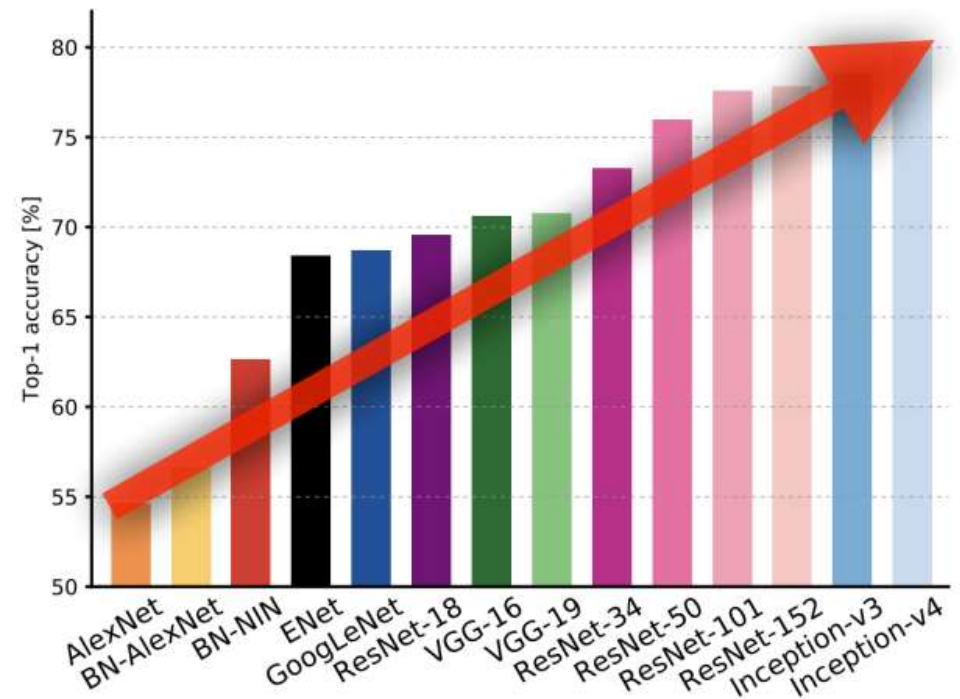
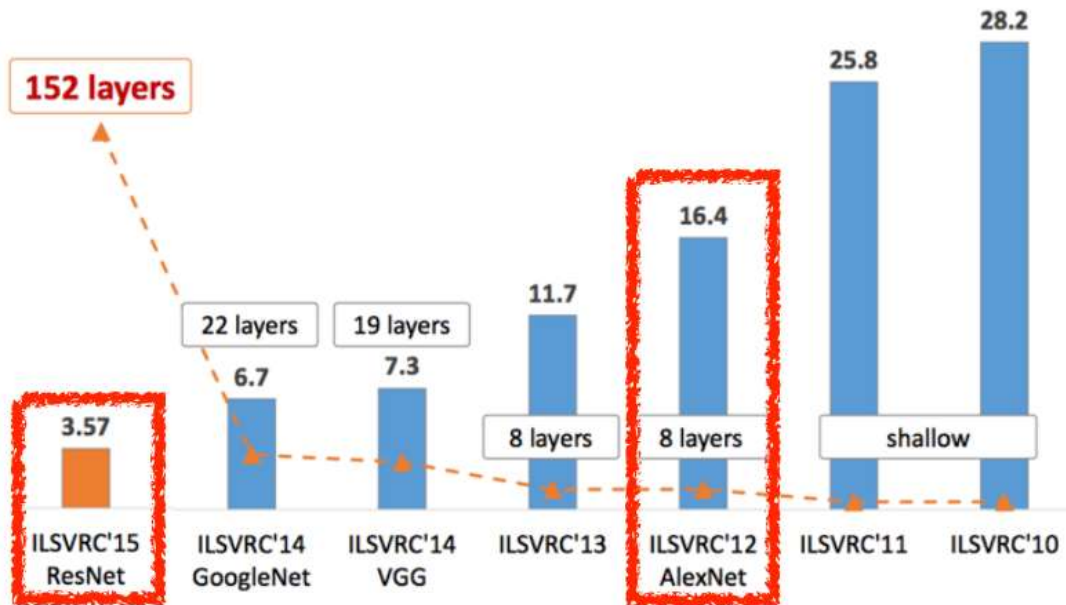
금속 표면

1.

3. AlexNet 보다 안 좋은 DenseNet201 ?????

AlexNet은 5 convolutional layers + 3 dense layers 로 이루어진 거의 최초의 CNN 모델입니다.

그에 반해 DenseNet은 ResNet에서 컨볼루션 방법만 바꾼 발전된 모델로, AlexNet과는 비교가 안될 정도로 깊은 네트워크와 보다 좋은 성능을 나타냅니다.



- 결과 분석

금속 표면

1.

3. AlexNet 보다 안 좋은 DenseNet201 ?????

어째서?

➤ 최적화 되지 않은 하이퍼 파라미터.

- 실제로 이번 실험에서 AlexNet과 DenseNet은 거의 비슷한 옵티마이저 SGD, LR, 스케줄러, 손실함수를 사용하였습니다.
- 실제로 초반부 DenseNet의 성능이 너무 좋지 않게 나와, 인터넷을 참고하여 손실함수를 CrossEntropyLoss로 바꾸고 LR도 AlexNet에 비해 10배 이상으로 바꾸자 성능이 3배 가까이 달라지는 것을 확인하였습니다.

Accuracy: 18.64406779661017
PS C:\Users\labadmin\Desktop\1



Accuracy : 38.050677966101696



(최종)

DenseNet201 = 63.05

1. 금속 표면

- Data 전 처리
- 사용 모델
- 모델 성능 비교
- 결과 분석

2. 얼굴 분류

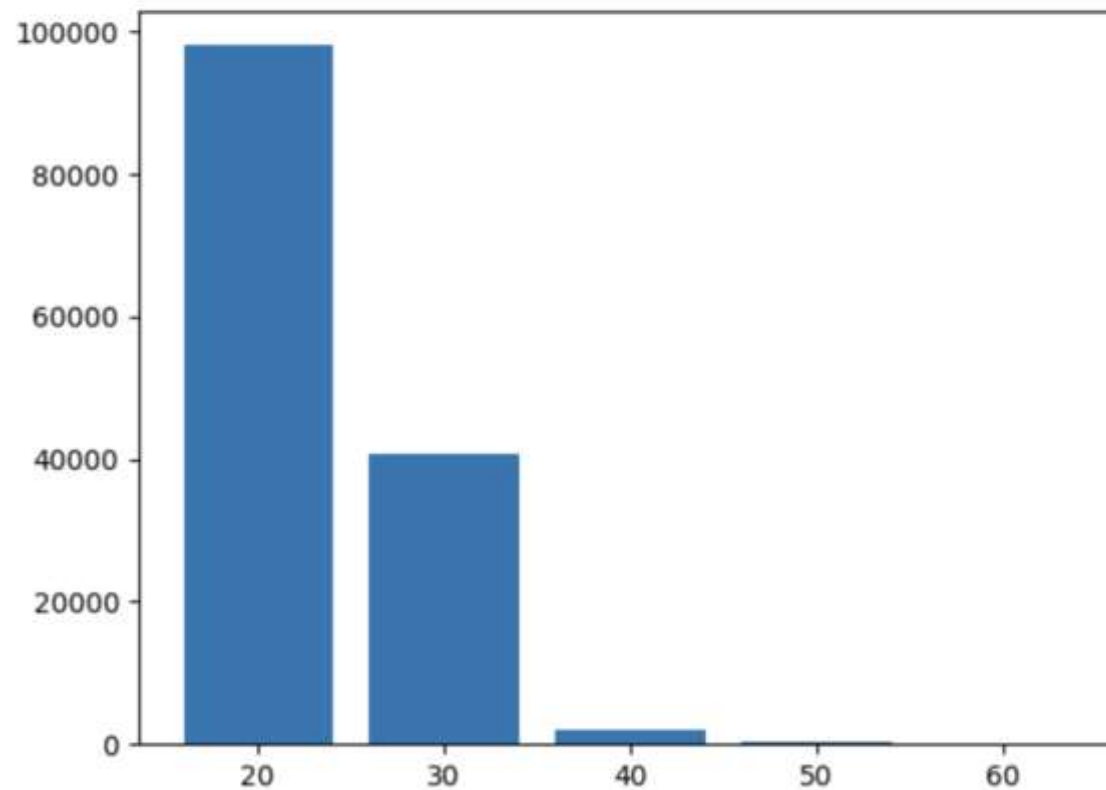
- Data 전 처리
- Test 이미지 수집
- 사용 모델
- 모델 성능 비교
- 결과 분석

- Data 전 처리 - 추가수집 (불균형)

얼굴 분류

2.

다른 나이대에 비해 20, 30대에 많은 데이터가 모여있어 이에 대한 해결이 필요함



- Data 전 처리 - 추가수집 (불균형)

얼굴 분류

2.

나이대별 동양인 얼굴 이미지를 수집하는데 많은 어려움 있었음

AI HUB 가족 관계가 알려진 얼굴 이미지 데이터에 나이가 명시되어 있어
이 데이터를 가져왔고 나이 성별로 분류 진행

분류된 이미지에서 얼굴부분만 CROP
(MediaPipe Face Detection 활용)

```
def face_crop(file_path):  
    with mp_face_detection.FaceDetection(model_selection=1, min_detection_confidence=0.5) as face_detection:  
        image = cv2.imread(file_path)  
  
        results = face_detection.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  
  
        if not results.detections:  
            print(f'{file_path} >> 얼굴을 찾지 못함!')  
            os.remove(file_path)  
            return  
  
        h, w, c = image.shape  
        for detection in results.detections:  
            bbox = detection.location_data.relative_bounding_box  
            if bbox.xmin < 0:  
                bbox.xmin = 0  
            if bbox.ymin < 0:  
                bbox.ymin = 0  
            crop_img = image[int(bbox.ymin * h): int((bbox.ymin + bbox.height) * h),  
                             int(bbox.xmin * w): int((bbox.xmin + bbox.width) * w)]  
            print(bbox)  
            return crop_img
```



- Data 전 처리

얼굴 분류

2.

형체를 알아보기 힘들거나, 해당 나이와 맞지 않아 보이는 이미지는 최대한 제거



11644-5.jpg



11644-6.jpg



11644-9.jpg



224,224 크기로 변환과 작은 크기의 이미지에는 padding을 주었고 동시에 스플릿 진행

```
def make_square_image(image_path):
    origin_image = cv2.imread(image_path)

    try:
        height, width, channels = origin_image.shape
    except:
        print(image_path, 'x' * 10)
        return

    x = height if height > width else width
    y = height if height > width else width

    if 224 <= x and 224 <= y:
        x, y = 224, 224

    square_image = np.zeros((x, y, channels), np.uint8)
    square_image[int((y - height) / 2):int((y + height) / 2),
                  int((x - width) / 2):int((x + width) / 2)] = origin_image

    return square_image
```

```
for i in range(2, 83):
    globes1[str(i)] = "H." + str(i) + "%H_train_list", globes1[
        str(i) + "D." + str(i) + "%D_val_list"] = train_test_split(
            globes1["%H_image." + str(i) + "%." + str(i) + "%_path"], test_size=0.2, random_state=7)
    globes1[str(i) + "H." + str(i) + "%H_train_list"], globes1[
        str(i) + "D." + str(i) + "%D_val_list"] = train_test_split(
            globes1["%H_image." + str(i) + "%." + str(i) + "%_path"], test_size=0.2, random_state=7)

for i in range(2, 83):
    # save_img(data, gender, age, mode)
    save_img(globes1[str(i) + "H." + str(i) + "%H_train_list"], "H", str(i) + "D." + str(i) + "%D_val")
    save_img(globes1[str(i) + "D." + str(i) + "%D_val_list"], "D", str(i) + "H." + str(i) + "%H_train")
    save_img(globes1[str(i) + "H." + str(i) + "%H_train_list"], "H", str(i) + "D." + str(i) + "%D_val")
    save_img(globes1[str(i) + "D." + str(i) + "%D_val_list"], "D", str(i) + "H." + str(i) + "%H_train")
```

train

- F+20_29
- F+30_39
- F+40_49
- F+50_59
- F+60_69
- M+20_29
- M+30_39
- M+40_49
- M+50_59
- M+60_69



- Data 전 처리

얼굴 분류

2.

데이터 불균형을 해소하고자 최대한 이미지를 수집했지만 데이터 수 부족
데이터 불균형을 고려하여 아래 표와 같이 데이터 수 조정

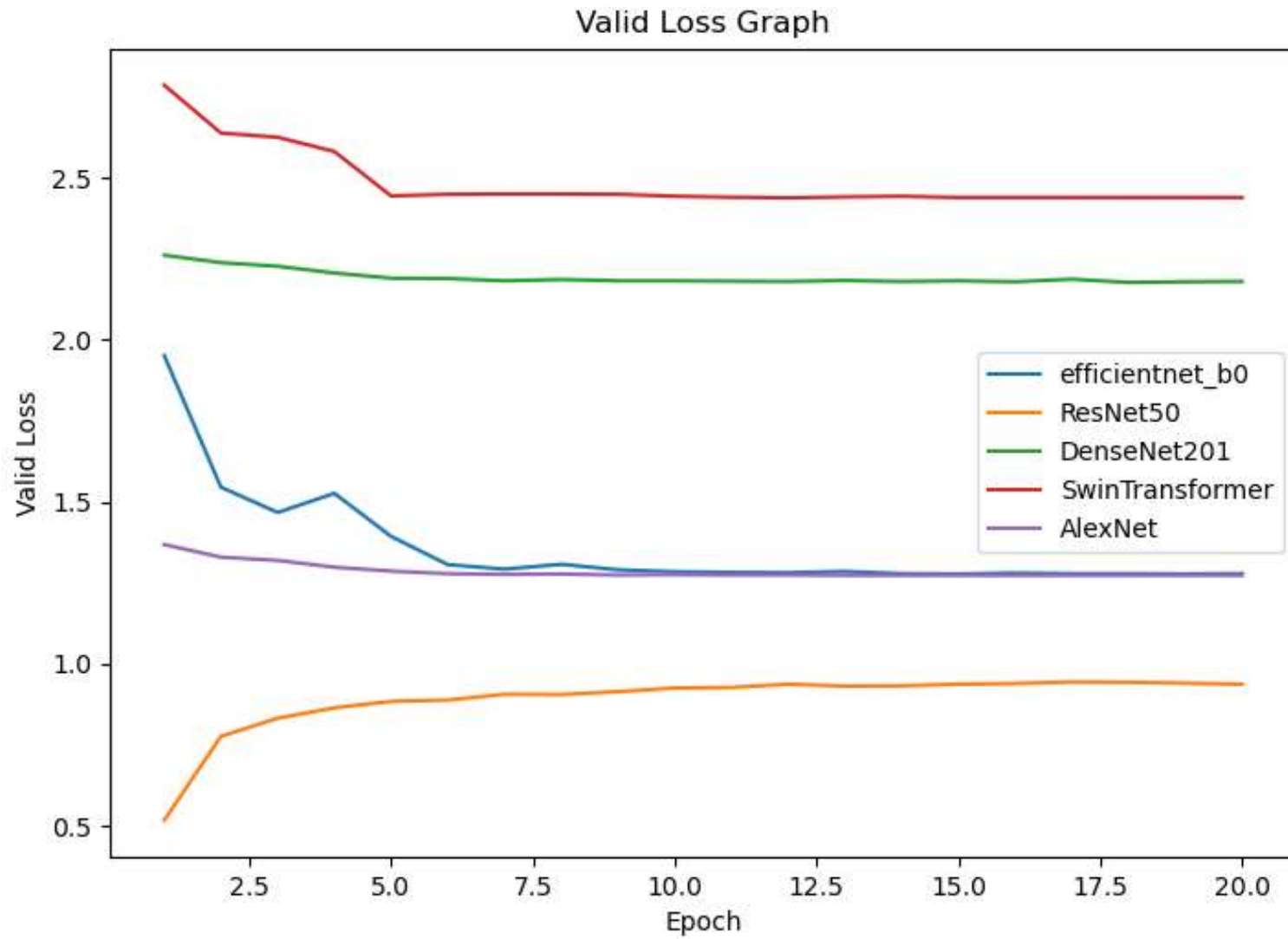
	20 남	20 여	30 남	30 여	40 남	40 여	50 남	50 여	60 남	60여
train	1514	1500	1590	1500	1524	1500	1671	1508	1112	1668
valid	400	400	400	400	426	381	418	378	279	419

2.

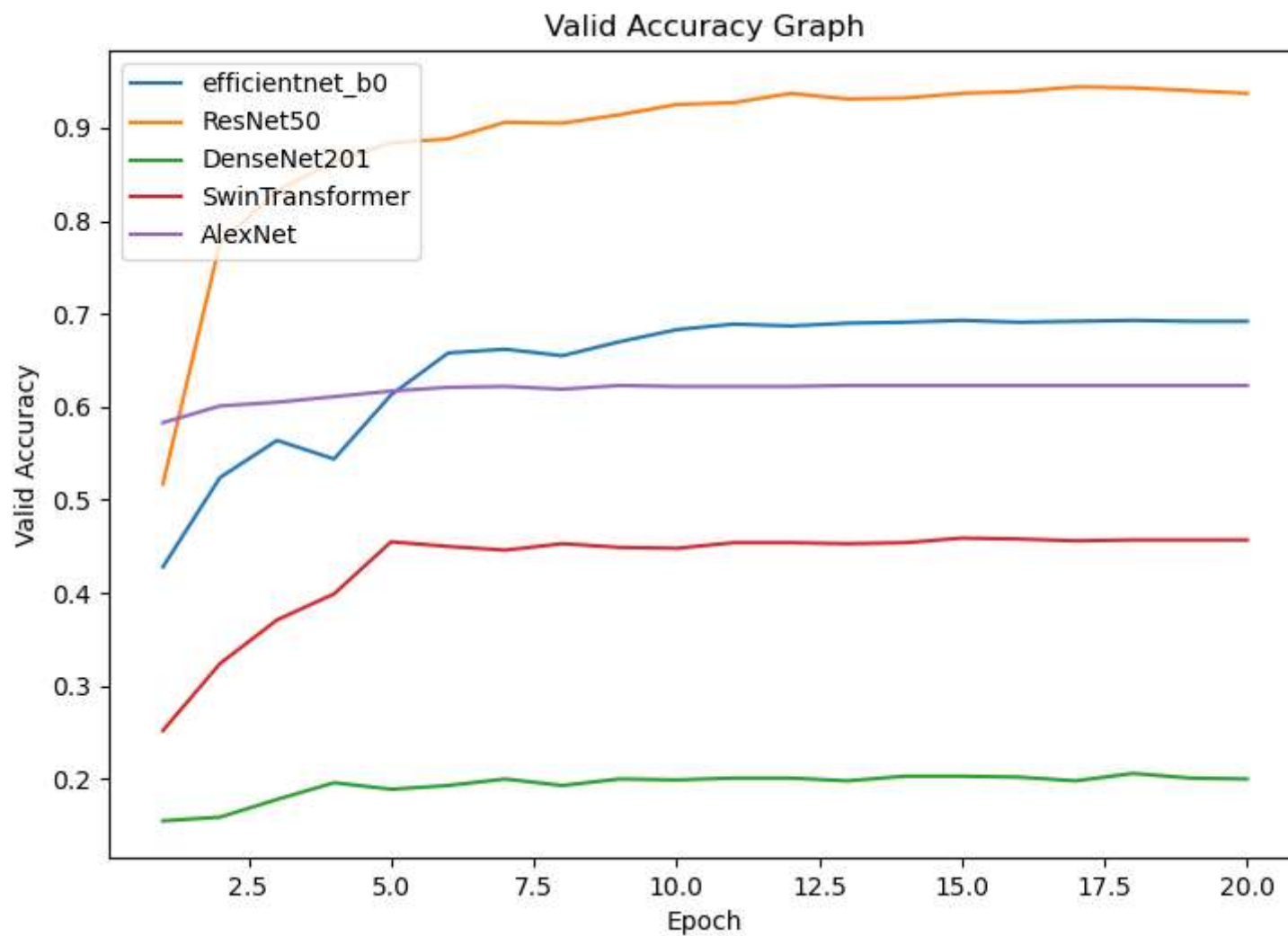
- 사용 모델

1. efficientnet_b0
2. ResNet50
3. DenseNet201
4. SwinTransformer
5. AlexNet

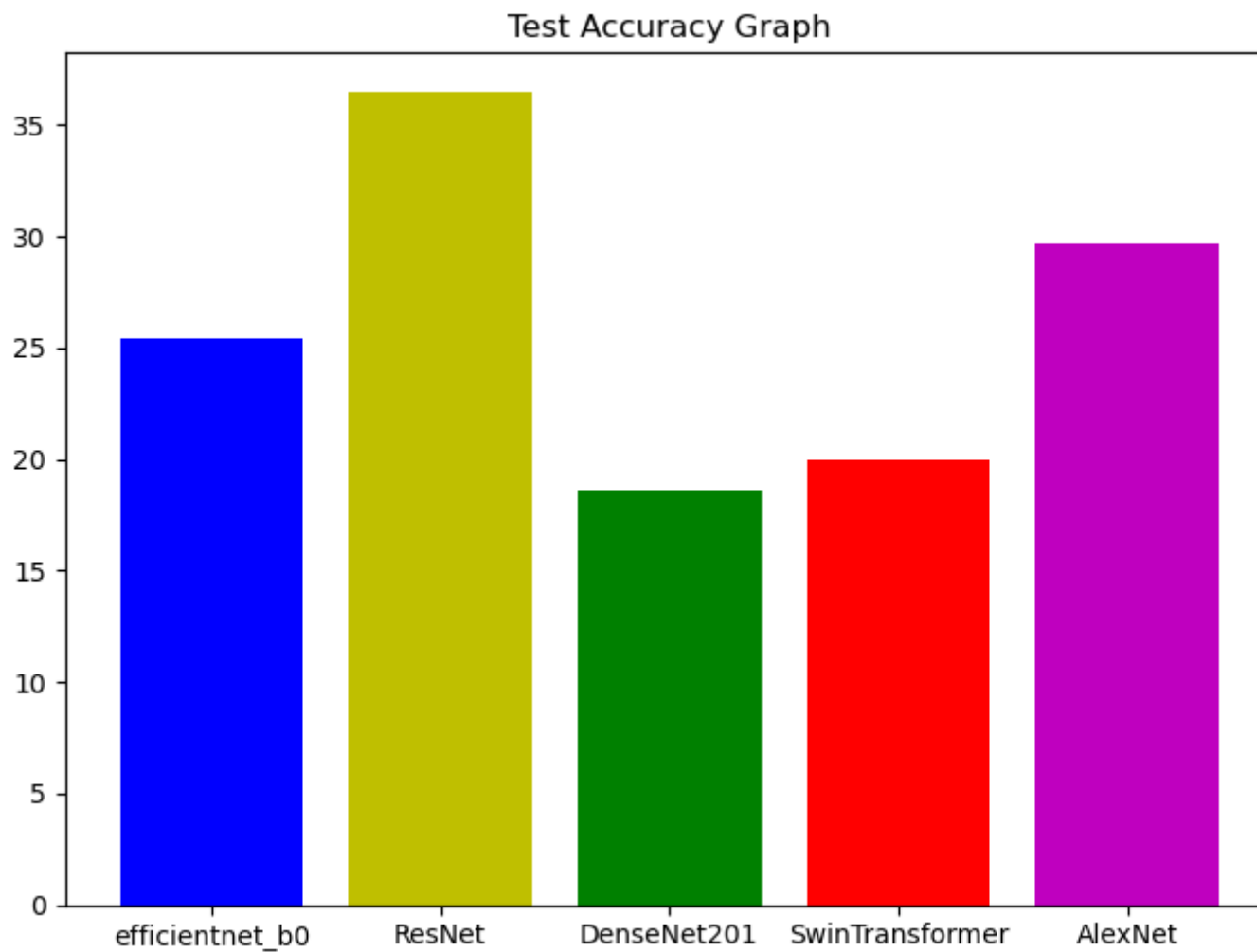
- 모델 성능 비교



- 모델 성능 비교



- 모델 성능 비교



efficientnet_b0 = 25.42

ResNet50 = 36.44

DenseNet201 = 18.64

SwinTransformer = 20.0

AlexNet = 29.66

- 결과 분석

얼굴 분류

2.

이전 문제인 금속탐지에 비해 너무 불균형적인 모델의 결과값 도출

이유 분석.

어째서?

- 결과 분석

얼굴 분류

2.

이전 문제인 금속탐지에 비해 너무 불균형적인 모델의 결과값 도출

1. 최대한 학습 데이터를 정제한다고 하였으나 발견하지 못한 학습에 방해되는 데이터가 있었을 수 있음.

ex)



24살 여성,

담배피는 16살 남성

(비록 학습에 쓰이지 않았으나
데이터 신뢰성 하락)



- 결과 분석

얼굴 분류

2.

큰 차이가 나는 Val 과 Test의 성능차이.

1. Train 및 Val 데이터와 괴리감 있는 Test 모델.
2. Test로 쓰인 이미지가 일반인이 아닌, 연예인 기준이므로 보다 더 나이 판별에 어려울 수 있음.

Ex) Test에 쓰인 데이터

모두 40대 연예인의 이미지.

(최대한 나이에 맞게 보이는 데이터를 수집하고자 최근 사진 위주로 수집하였으나 연예인들은 너무나 동안인 경우가 많았음. 이로 인해 Acc가 떨어졌을 가능성 多)



- 결과 분석

얼굴 분류

2.

데이터 외 하이퍼 파라미터 요인.

데이터 문제 외 요인으로 앞서 1번 문제에서도 이야기 했듯,
최적화된 하이퍼 파라미터값을 찾지 못해서 성능이 낮은 것일 수도 있음.

Ex)

Epoch의 부족, 러닝 레이트의 적절성 등..

