

생활폐기물 및 재활용품 이미지 분류

RESNET-34를 이용한 전이학습을 중심으로

TEAM 7



목차

1 주제

주제 선정 배경
프로젝트 목표 및 기대효과
프로젝트 흐름

2 데이터

데이터 출처 및 선별
데이터 전처리

3 모델 학습 및 결과

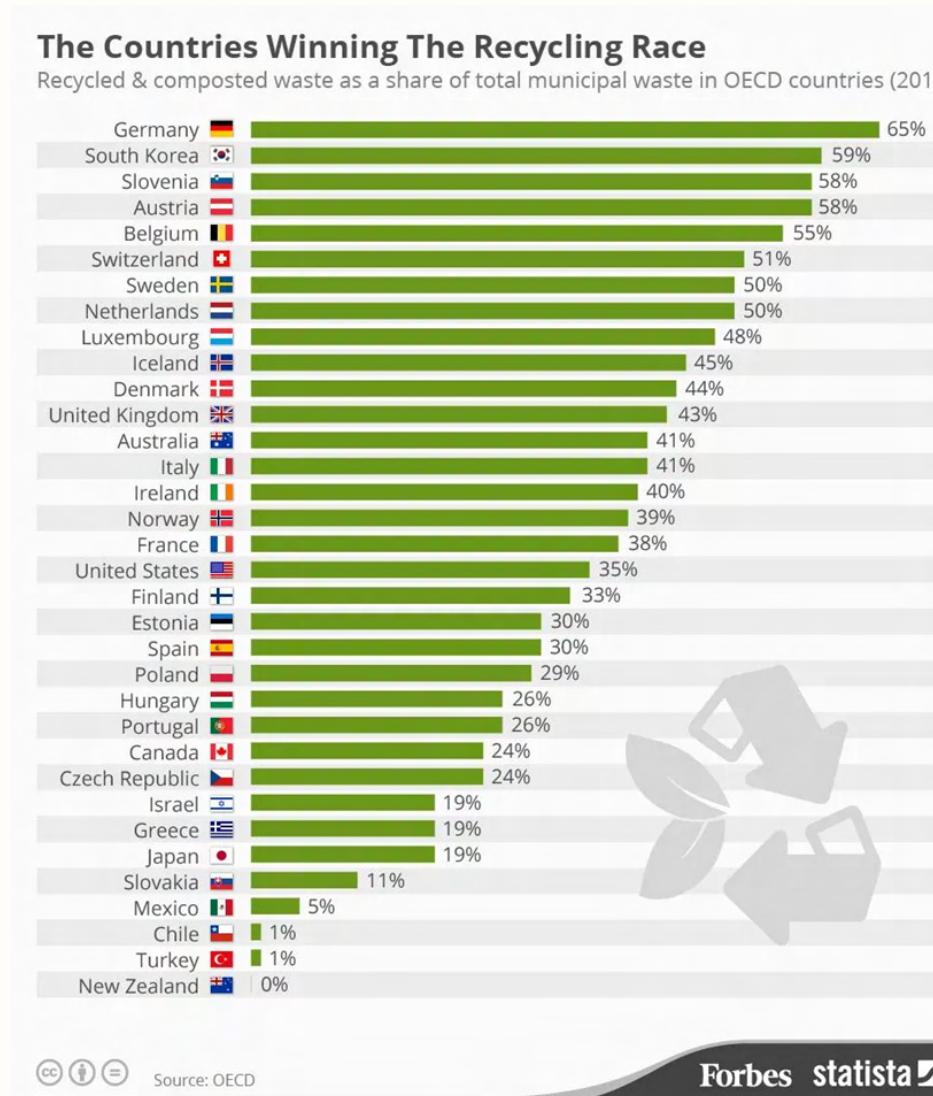
ResNet-34 미세조정

4 결론

적용 결과
개선점



1 주제 - 선정 배경



1. 전세계적으로 재활용 분류가 잘 이루어지지 않고 있음

2. 작업자들이 봉지에 담긴 쓰레기 더미를 해체하여 컨베이어 벨트에 옮겨 다른 클래스에 해당하는 것들을 수작업으로 분류하고 있음

1 주제 - 프로젝트 목표 및 기대효과

목표: 컨베이어벨트 위를 움직이는 생활폐기물 및 재활용품 이미지를 분류하는 모델 구현



<기대효과 I>



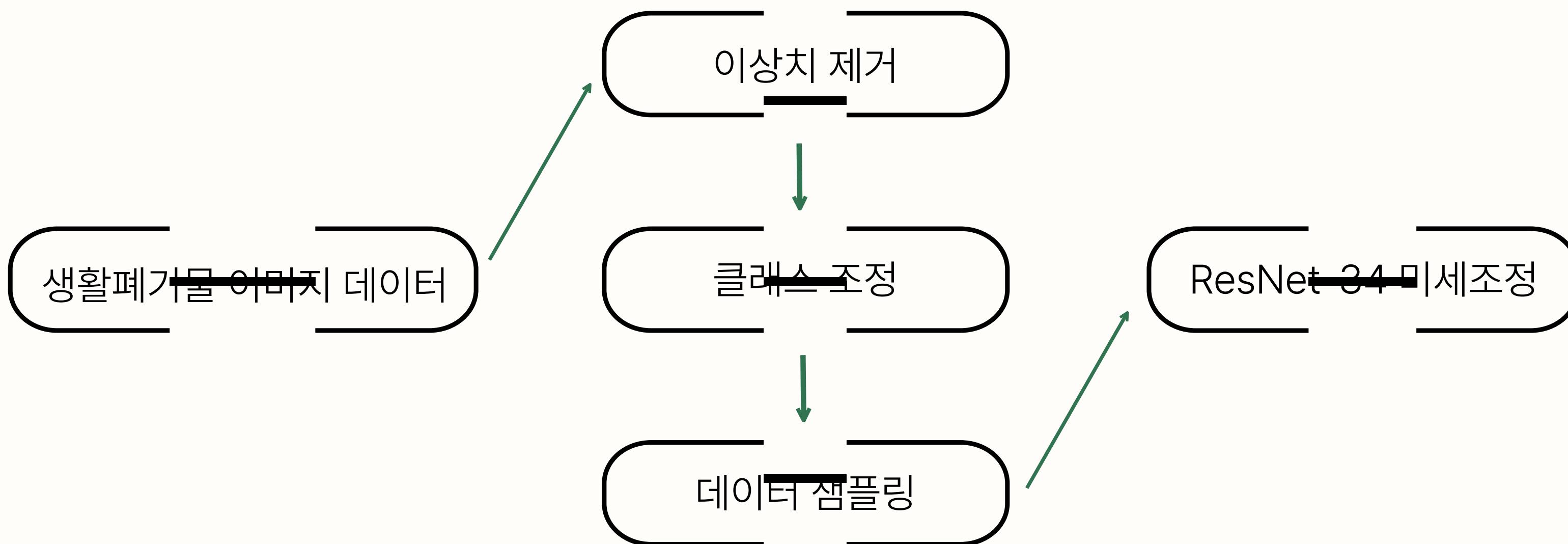
<기대효과 II>

1 주제 - 프로젝트 흐름

데이터 소스

데이터 전처리

모델링



2 데이터 - 출처 및 선별

데이터 내용	품목	데이터 형식	데이터 수량	데이터 크기
개별 재활용품 이미지	대분류 9종 (세부분류 15종)	jpg 이미지 파일	706,101건	약 2,700GB
재활용품 선별영상 추출 이미지	대분류 7종 (세부분류 13종)	jpg 이미지 파일	299,764건	약 170GB
라벨링 데이터	대분류 9종 (세부분류 15종)	json 라벨링 파일	1,005,865건	약 6GB

<원본 데이터>

- 구축 규모: 2022년에 구축된 100만장의 이미지 데이터
- 데이터 종류: 개별 재활용품 이미지, 재활용품 선별 영상 추출 이미지, 라벨링 데이터

⇒ 학습시간 및 기술적인 제약과 프로젝트 목표를 고려하여 영상에서 추출한 이미지 데이터 중 원본데이터의 클래스 별 비율을 고려하여 클래스 별로 900-1300장의 데이터를 랜덤하게 추출하여 사용

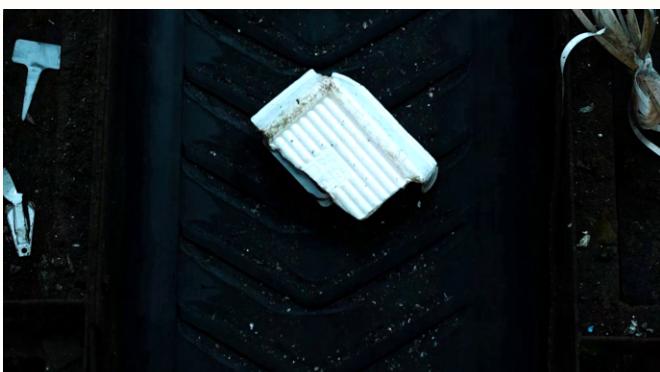
이미지 종류	종분류	세부분류	파일 포맷	수량
01. 금속캔	001. 철캔	.jpg / .json	30,129	
	002. 알류미늄캔	.jpg / .json	21,171	
02. 종이	001. 종이	.jpg / .json	21,176	
	001. 무색단일	.jpg / .json	39,538	
03. 페트병	002. 유색단일	.jpg / .json	33,170	
	001. PE	.jpg / .json	25,273	
04. 플라스틱	002. PP	.jpg / .json	39,088	
	003. PS	.jpg / .json	39,009	
05. 스티로폼	001. 스티로폼	.jpg / .json	15,039	
	001. 비닐	.jpg / .json	9,015	
06. 비닐	001. 갈색	.jpg / .json	9,052	
	002. 녹색	.jpg / .json	9,036	
	003. 투명	.jpg / .json	9,068	
07. 유리병				

선별영상 추출 이미지

<선택한 데이터>

2 데이터 전처리 - 이상치 제거

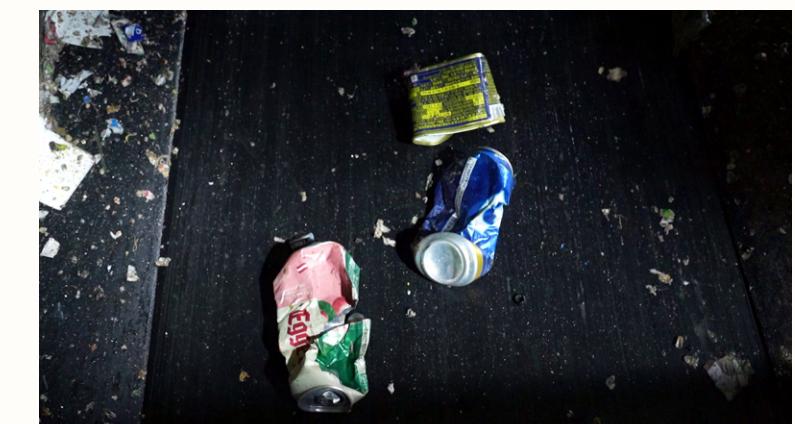
정상 데이터



이상치



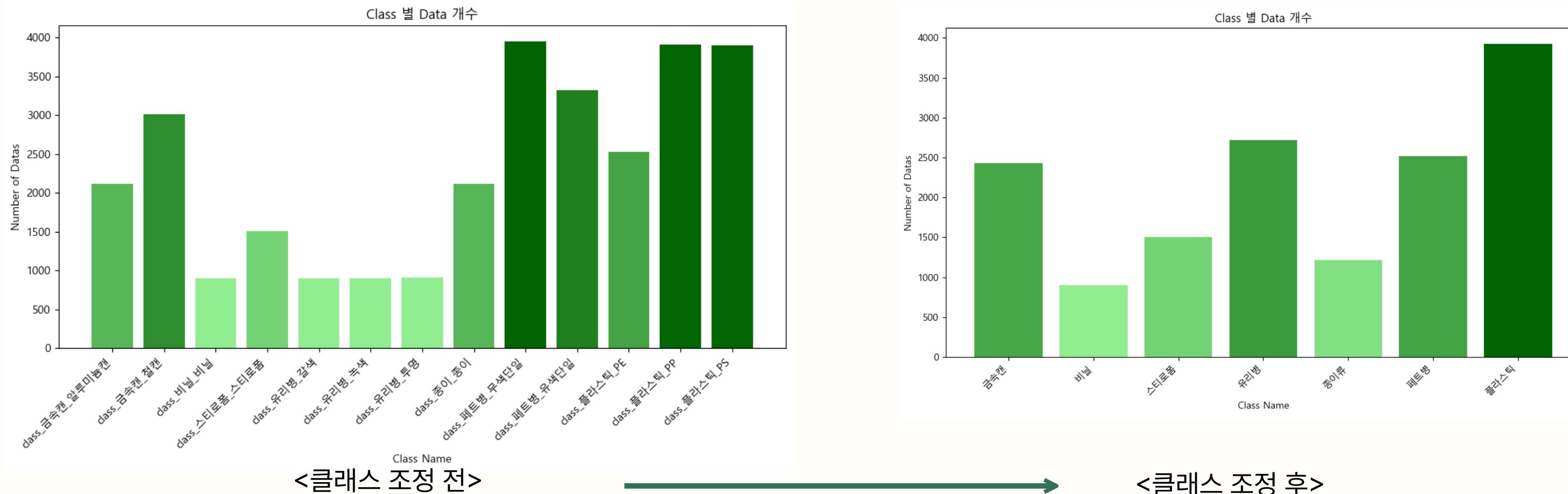
<데이터가 없는 경우>



<잘못된 클래스로 분류된 경우>

<다양한 클래스가 섞여있는 경우>

2 데이터 전처리 - 클래스 조정



- 프로젝트 목표가 재활용품 분류 모델을 구축하는 것임을 고려하여 재활용품 분류 기준에 적합한 중분류를 적용함
- 그 결과 13개의 클래스(세부분류)에서 7개의 클래스(중분류)로 클래스를 조정함

* 막대 그래프의 색은 클래스의 데이터가 많을수록 진하게 표현됨

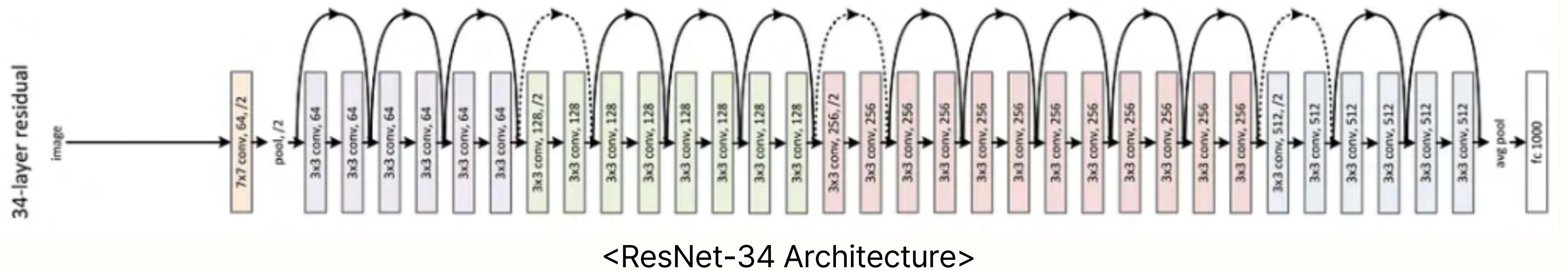
2 데이터 전처리 - 샘플링

```
# 클래스별 데이터 확인
print(f"Train class distribution: {[train_dataset.targets.count(i) for i in range(len(class_merge_map))]}")
print(f"Validation class distribution: {[val_dataset.targets.count(i) for i in range(len(class_merge_map))]}")
print(f"Test class distribution: {[test_dataset.targets.count(i) for i in range(len(class_merge_map))]}")

Train set size: 4410
Validation set size: 945
Test set size: 945
Train class distribution: [630, 630, 630, 630, 630, 630, 630]
Validation class distribution: [135, 135, 135, 135, 135, 135, 135]
Test class distribution: [135, 135, 135, 135, 135, 135, 135]
```

- 클래스 데이터 불균형을 해소하기 위해 각 클래스의 비율을 고려하여 데이터 샘플링
- Train set, Validation set, Test set에 모두 적용

3 모델 학습 및 결과 - 'ResNet-34' 전이학습



ResNet-34를 선택한 이유

- ResNet은 ImageNet으로 학습한 모델로 이미지 분류에 적합한 모델임
 - 학습하는 데이터의 양이 많지 않기 때문에 ResNet-34로 충분히 학습 가능하다고 판단함

미세조정 및 기본 세팅

- ResNet은 1000개 클래스를 구분하는 모델이므로 7개의 클래스임을 고려하여 final fully connected layer의 개수를 미세조정

3 모델 학습 및 결과 - 미세조정

```

Epoch 48/50: 100%|██████████| 256/256 [02:49<00:00, 1.51it/s]
Epoch [48/50], Loss: 1.0410, Accuracy: 99.99%
Validation Loss: 106.0523, Accuracy: 62.39%
Epoch 49/50: 100%|██████████| 256/256 [02:49<00:00, 1.51it/s]
Epoch [49/50], Loss: 0.9332, Accuracy: 99.99%
Validation Loss: 105.7165, Accuracy: 62.62%
Epoch 50/50: 100%|██████████| 256/256 [02:49<00:00, 1.51it/s]
Epoch [50/50], Loss: 0.9233, Accuracy: 100.00%
Validation Loss: 106.4916, Accuracy: 62.62%

```

Test Accuracy: 63.65%

```

Epoch 19/50: 100%|██████████| 256/256 [03:19<00:00, 1.28it/s]
Epoch [19/50], Loss: 221.8959, Accuracy: 70.74%
Validation Loss: 1.0279, Validation Accuracy: 63.48%
No improvement for 1 epochs.
Epoch 20/50: 100%|██████████| 256/256 [03:20<00:00, 1.28it/s]
Epoch [20/50], Loss: 212.6778, Accuracy: 71.32%
Validation Loss: 1.0680, Validation Accuracy: 63.65%
No improvement for 2 epochs.
Epoch 21/50: 100%|██████████| 256/256 [04:10<00:00, 1.02it/s]
Epoch [21/50], Loss: 197.5938, Accuracy: 74.15%
Validation Loss: 1.0366, Validation Accuracy: 64.56%
No improvement for 3 epochs.
Epoch 22/50: 100%|██████████| 256/256 [03:29<00:00, 1.22it/s]
Epoch [22/50], Loss: 192.7032, Accuracy: 74.96%
Validation Loss: 1.0332, Validation Accuracy: 64.10%
No improvement for 4 epochs.
Early stopping triggered. Training terminated.
Training complete.

```

```

# 테스트 모델 실행
test_model(model, test_loader, 13)
✓ 1m 11.3s

```

Overall Test Accuracy: 64.44%

Class-wise Accuracy:

Class 0:	67.41%	(91/135)
Class 1:	93.33%	(126/135)
Class 2:	65.93%	(89/135)
Class 3:	49.63%	(67/135)
Class 4:	86.67%	(117/135)
Class 5:	71.85%	(97/135)
Class 6:	85.93%	(116/135)

Overall Test Accuracy: 69.84%

Class-wise Accuracy:

Class 0:	99.26%	(134/135)
Class 1:	67.41%	(91/135)
Class 2:	54.07%	(73/135)
Class 3:	98.52%	(133/135)
Class 4:	58.52%	(79/135)
Class 5:	68.89%	(93/135)
Class 6:	42.22%	(57/135)

```

Epoch 20/50: 100%|██████████| 138/138 [01:31<00:00, 1.50it/s]
Epoch [20/50], Loss: 36.4146, Accuracy: 92.93%
Validation Loss: 1.0731, Validation Accuracy: 69.63%
No improvement for 1 epochs.

```

```

Epoch 21/50: 100%|██████████| 138/138 [01:31<00:00, 1.51it/s]
Epoch [21/50], Loss: 31.1518, Accuracy: 94.15%
Validation Loss: 0.9606, Validation Accuracy: 71.22%
Validation metric improved. Model saved.
Epoch 22/50: 100%|██████████| 138/138 [01:31<00:00, 1.51it/s]
Epoch [22/50], Loss: 27.5112, Accuracy: 95.03%
Validation Loss: 0.9782, Validation Accuracy: 70.79%
No improvement for 1 epochs.

```

```

Epoch 23/50: 100%|██████████| 138/138 [01:31<00:00, 1.51it/s]
Epoch [23/50], Loss: 25.6063, Accuracy: 95.51%
Validation Loss: 0.9931, Validation Accuracy: 70.26%
No improvement for 2 epochs.

```

```

Epoch 24/50: 100%|██████████| 138/138 [01:31<00:00, 1.51it/s]
Epoch [24/50], Loss: 25.2556, Accuracy: 95.60%
Validation Loss: 1.0103, Validation Accuracy: 71.11%
No improvement for 3 epochs.

```

```

Epoch 25/50: 100%|██████████| 138/138 [01:31<00:00, 1.51it/s]
Epoch [25/50], Loss: 22.7516, Accuracy: 96.35%
Validation Loss: 1.0219, Validation Accuracy: 70.26%
No improvement for 4 epochs.
Early stopping triggered. Training terminated.
Training complete.

```

3 모델 학습 및 결과 - 미세조정

```
def forward(self, x):
    x = F.relu(self.top_model(x))
    x = nn.AdaptiveAvgPool2d((1,1))(x)
    x = x.view(x.shape[0], -1) # flattening
    x = self.bn1(x)
    x = F.relu(self.fc1(x))
    x = self.bn2(x)
    x = self.fc2(x)

    return x
```

```
def forward(self, x):

    # Forward through the backbone (pretrained ResNet-34)
    x = self.backbone(x) # Pass through ResNet-34 feature extractor
    x = nn.AdaptiveAvgPool2d((1, 1))(x) # Global average pooling
    x = x.view(x.size(0), -1) # Flatten features

    # Classifier part
    x = self.bn1(x)
    x = F.relu(self.fc1(x))
    x = self.dropout1(x)
    x = self.bn2(x)
    x = F.relu(self.fc2(x)) # Intermediate FC layer
    x = self.dropout2(x)
    x = self.bn3(x)
    x = F.relu(self.fc3(x)) # Intermediate FC layer
    x = self.dropout3(x)
    x = self.bn4(x)
    x = self.fc4(x) # Final output layer

    return x
```

```
def forward(self, x):

    # Forward through the backbone (pretrained ResNet-34)
    x = self.backbone(x) # Pass through ResNet-34 feature extractor
    x = nn.AdaptiveAvgPool2d((1, 1))(x) # Global average pooling
    x = x.view(x.size(0), -1) # Flatten features

    # Classifier part
    x = self.bn1(x)
    x = F.relu(self.fc1(x))
    x = self.dropout1(x)
    x = self.bn2(x)
    x = F.relu(self.fc2(x)) # Intermediate FC layer
    x = self.dropout2(x)
    x = self.bn3(x)
    x = F.relu(self.fc3(x)) # Intermediate FC layer
    x = self.dropout3(x)
    x = self.bn4(x)
    x = self.fc4(x) # Final output layer

    return x
```

Overall Test Accuracy: 78.20%

Class-wise Accuracy:

- Class 0: 100.00% (135/135)
- Class 1: 45.93% (62/135)
- Class 2: 42.96% (58/135)
- Class 3: 96.30% (130/135)
- Class 4: 72.59% (98/135)
- Class 5: 96.30% (130/135)
- Class 6: 93.33% (126/135)

<model 1>

Overall Test Accuracy: 80.74%

Class-wise Accuracy:

- Class 0: 100.00% (135/135)
- Class 1: 57.78% (78/135)
- Class 2: 44.44% (60/135)
- Class 3: 97.78% (132/135)
- Class 4: 75.56% (102/135)
- Class 5: 96.30% (130/135)
- Class 6: 93.33% (126/135)

<model 2>

Overall Test Accuracy: 82.86%

Class-wise Accuracy:

- Class 0: 100.00% (135/135)
- Class 1: 62.96% (85/135)
- Class 2: 47.41% (64/135)
- Class 3: 97.04% (131/135)
- Class 4: 81.48% (110/135)
- Class 5: 96.30% (130/135)
- Class 6: 94.81% (128/135)

<model 3>

4 결론 - 적용 결과

Class: Metal Can
Confidence: 0.90



[실제 클래스: 메탈 캔]

Class: Plastic
Confidence: 0.47



[실제 클래스: 플라스틱]

<컨베이어 벨트 환경을 연출하여 모델에 적용한 결과>

4 결론 - 개선점

기술적 제약

1. 데이터 용량이 너무 커서 일부 데이터만 선별해서 활용함
 2. 학습시간의 제한으로 ResNet-34보다 성능이 좋다고 평가되는 모델을 활용하지 못함
- > 기술적 제약이 극복되면 더 좋은 성능을 보이는 모델을 구현할 수 있을 것으로 기대됨

모델의 개선점

1. 특정 클래스만 잘 구분하는 경향이 있음
- > 메탈 캔, 유리병, 페트병 같은 이미지는 형태가 정해져있기 때문이라고 생각됨
- > YOLO 모델을 활용하여 추후 연구를 발전시킬 수 있다고 판단됨

Reference

- <https://www.khan.co.kr/article/202402250800001>
- <https://www.youtube.com/watch?v=T3I2aF0z6Bo>
- https://blog.lgchem.com/2023/01/26_geekble_collaboration/
- <https://www.donga.com/news/lt/article/all/20201231/104713889/1>
- <https://www.epa.gov/recycle/recycling-basics-and-benefits#:~:text=Recycling%20provides%20many%20benefits%20to,and%20minerals%20for%20new%20products>



감사합니다

TEAM7
2019147039 오동하
2020147030 홍승은
2022147047 김진웅
2022121011 김윤성