2021-2
일반화선형모형

Garbage Classification

통계학과 212STG19 윤성원

통계학과 212STG21 이선경

통계학과 212STG23 이소영

통계학과 212STG26 이현지

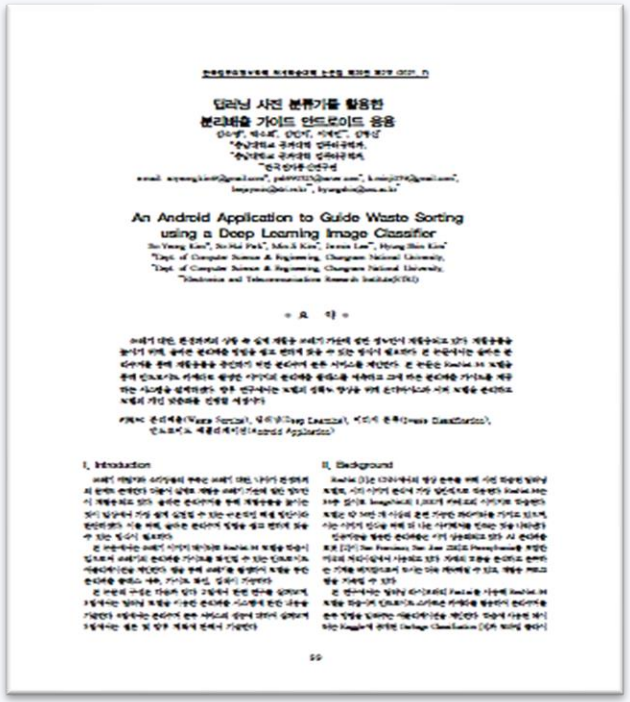
Index

목 차

서 론

이전 연구 고찰

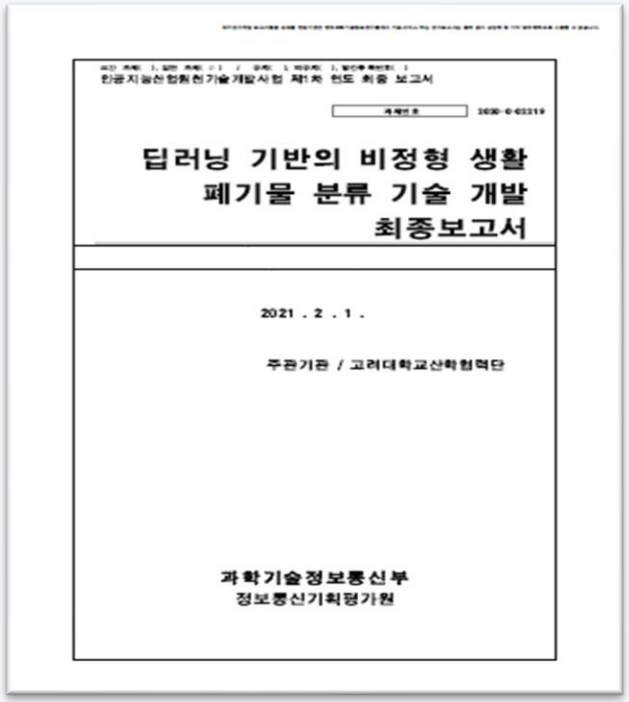
2021.07



딥러닝 사전 분류기를 활용한
분리배출 가이드 안드로이드 응용

An Android Application to Guide Waste Sorting
using a Deep Learning Image Classifier

2021.02



인공지능산업원천기술개발사업 제1차 연도 최종 보고서

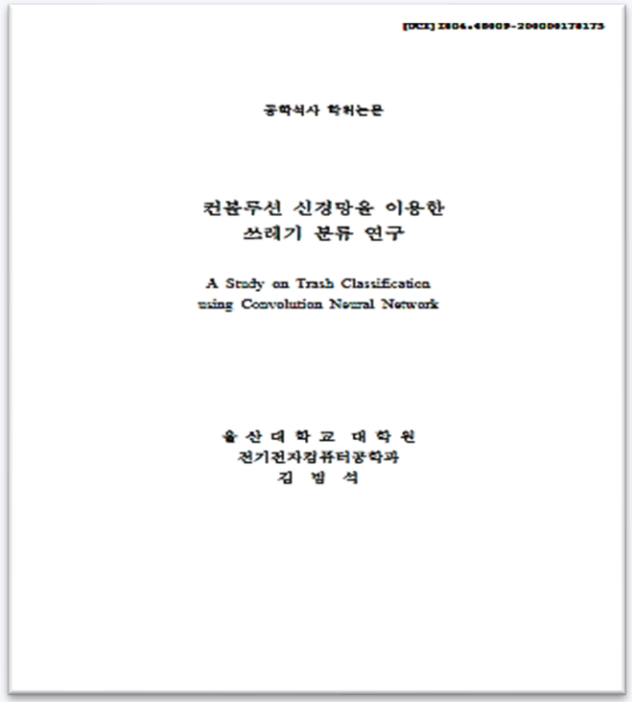
딥러닝 기반의 비정형 생활
폐기물 분류 기술 개발
최종보고서

2021 . 2 . 1 .

주관기관 / 고려대학교산학협력단

과학기술정보통신부
정보통신기획평가원

2018.12



공학석사 학위논문

컨퓨두선 신경망을 이용한
쓰레기 분류 연구

A Study on Trash Classification
using Convolution Neural Network

울 산 대 학 교 대 학 원
전기전자컴퓨터공학과
김 범 석

이전 연구 고찰

한국컴퓨터정보학회 하계학술대회 논문집 제29권 제2호 (2021. 7)

딥러닝 사진 분류기를 활용한 분리배출 가이드 안드로이드 응용

쓰레기 대란, 환경파괴의 상황 속 실제 재활용 쓰레기 가운데 절반 정도만이 재활용되고 있다. 재활용률을 높이기 위해, 올바른 분리배출 방법을 쉽고 편하게 찾을 수 있는 방식이 필요하다.

본 논문에서는 올바른 분리수거를 통해 재활용률을 증진하기 위한 분리수거 분류 서비스를 제안한다. 본 논문은 ResNet-34 모델을 통해 안드로이드 카메라로 촬영한 이미지의 분리배출 클래스를 예측하고 그에 따른 분리배출 가이드를 제공하는 시스템을 설계하였다.

향후 연구에서는 모델의 정확도 향상을 위해 온디바이스와 서버 모델을 분리하고 모델의 개인 맞춤화를 진행할 예정이다.

키워드: 분리배출(Waste Sorting), 딥러닝(Deep Learning), 이미지 분류(Image Classification),안드로이드 애플리케이션(Android Application)

1. introduction

쓰레기 이미지 데이터로 ResNet-34 모델을 학습시킴으로써 쓰레기의 분리배출 가이드를 확인할 수 있는 안드로이드 애플리케이션을 제안. 앱을 통해 쓰레기를 촬영하여 모델을 통한 분리배출 클래스 예측, 가이드 확인, 검색이 가능.

2. Background

ResNet은 CNN에서의 영상 분류를 위해 사전 학습된 딥러닝 모델로, 시각 이미지 분석에 가장 일반적으로 적용된다. ResNet-34는 34층 깊이로 ImageNet의 1000개 카테고리 이미지로 학습됨. 모델은 약 54만 개 이상의 훈련 가능한 파라미터를 가지고 있으며, 이는 이미지 인식을 위해 더 나은 아키텍처를 만드는 것을 나타냄. 딥러닝 라이브러리 Fast.ai를 사용해 ResNet-34모델을 학습시킴.

서 론

문제 제기

사용 데이터

- Kaggle의 Garbage Classification[3]과 모바일 환경에서 촬영한 이미지 파일을 사용
- 학습 데이터는 원본 데이터 셋의 50% 확률로 vertical flip 수행, brightness와 contrast jitter를 95%의 확률로 조정하는 데이터 증강을 통해 데이터를 변형, 모델 학습에 사용
- Resize = 384x512 / Batch size = 16 / Train : Test = 8 : 2

Table 1. ResNet34, ResNet152, MobileNet.v2의 성능

	ResNet-34	ResNet-152	Mobilenet.v2
test acc (%)	**94.7**	94.5	94.3
pred time (s)	**12.7**	18.4	16.4

목 표

다양한 깊이의 ResNet 모델을 훈련시켜 쓰레기 이미지 분류 최적 모델 찾기

해당 논문은 사전훈련학습망만 사용했으며 다양한 깊이를 시행해보지 않았다는 점에서 추가 분석 수행

데이터 소개

Garbage Image Data

2527개의 이미지 파일 (43.4 MB)을 6가지 Label에 맞추어 6개의 폴더 제공

Trash(127)	Plastic (472)	Paper(584)	Metal (400)	Glass (491)	Cardboard (393)






출처 : https://www.kaggle.com/asdasdasasdas/garbage-classification

데이터 전처리

1. Resize (224, 224)

쓰레기 형태는 다 나오면서 배경 부분은 최대한 자를 수 있는 224에서 가장 높은 성능을 보임

2. Batch size= 32

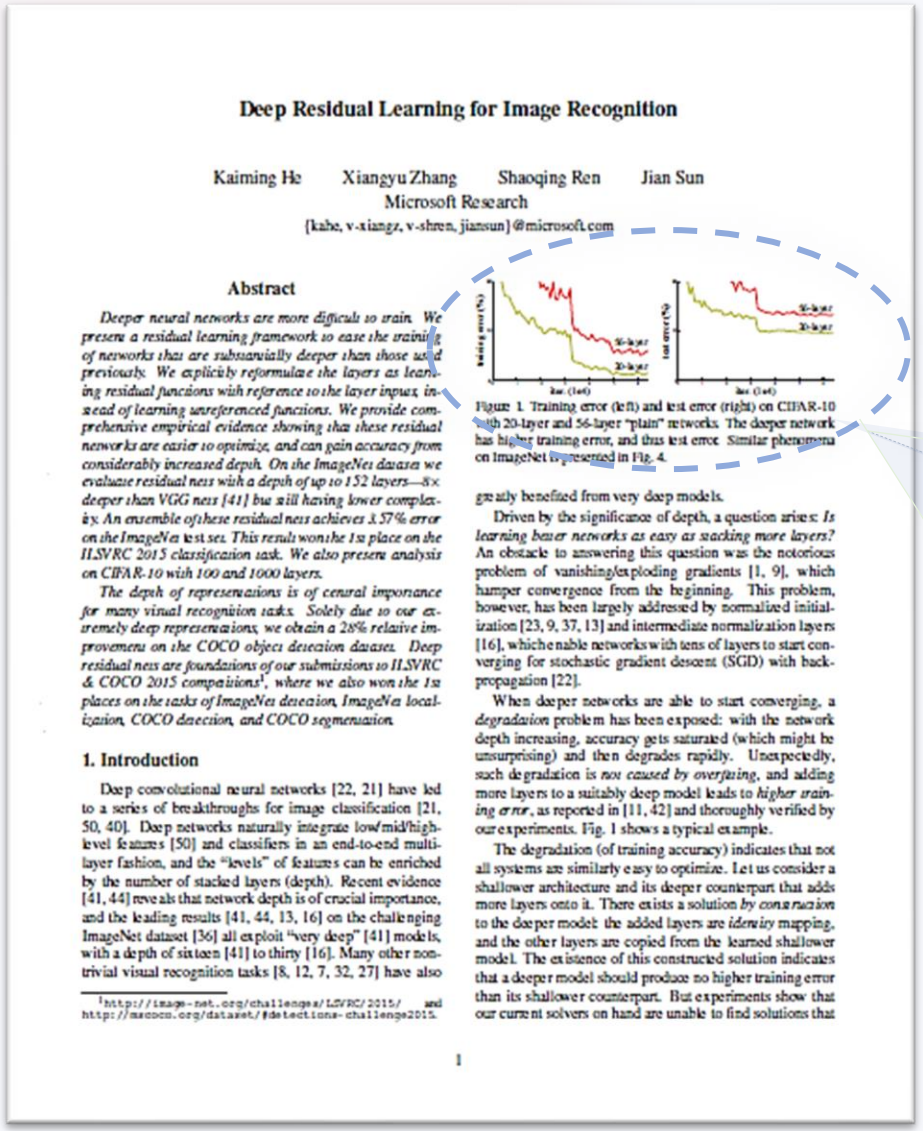
3. Train/Test Split

훈련 속도를 위해 valid set은 train set batch size의 2배로 지정

Train Set	1617 장	64%	DataLoader(train_ds, batch_size, shuffle = True, num_workers = 2, pin_memory = True)
Valid Set	404 장	16%	DataLoader(val_ds, batch_size*2, num_workers = 2, pin_memory = True)
Test Set	506 장	20%	DataLoader(test_ds, batch_size*2, num_workers = 2, pin_memory = True)

Resnet

ResNet



Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1], where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

1. Introduction

Deep convolutional neural networks [22, 21] have led to a series of breakthroughs for image classification [21, 50, 40]. Deep networks naturally integrate low/mid/high-level features [50] and classifiers in an end-to-end multi-layer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth). Recent evidence [41, 44] reveals that network depth is of crucial importance, and the leading results [41, 44, 13, 16] on the challenging ImageNet dataset [36] all exploit "very deep" [41] models, with a depth of sixteen [41] to thirty [16]. Many other non-trivial visual recognition tasks [8, 12, 7, 32, 27] have also

[1] http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.

Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [1, 9], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [11, 42] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

1

Deep Residual Learning for Image Recognition

2015년 ILSVRC(이미지인식 경진대회)에서
우승을 차지한 이미지 분류용 CNN 모델
(마이크로소프트사 개발)

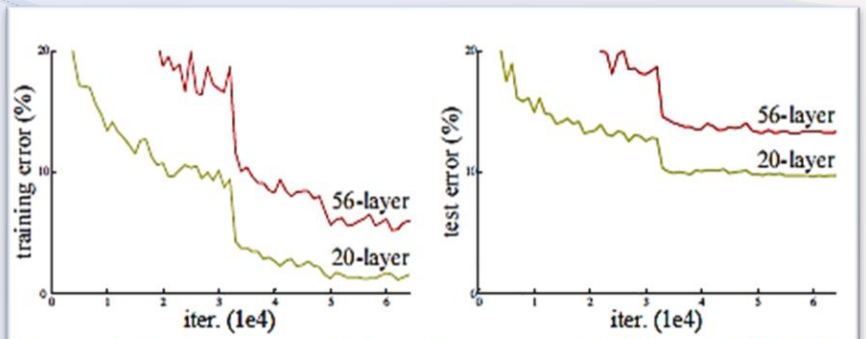
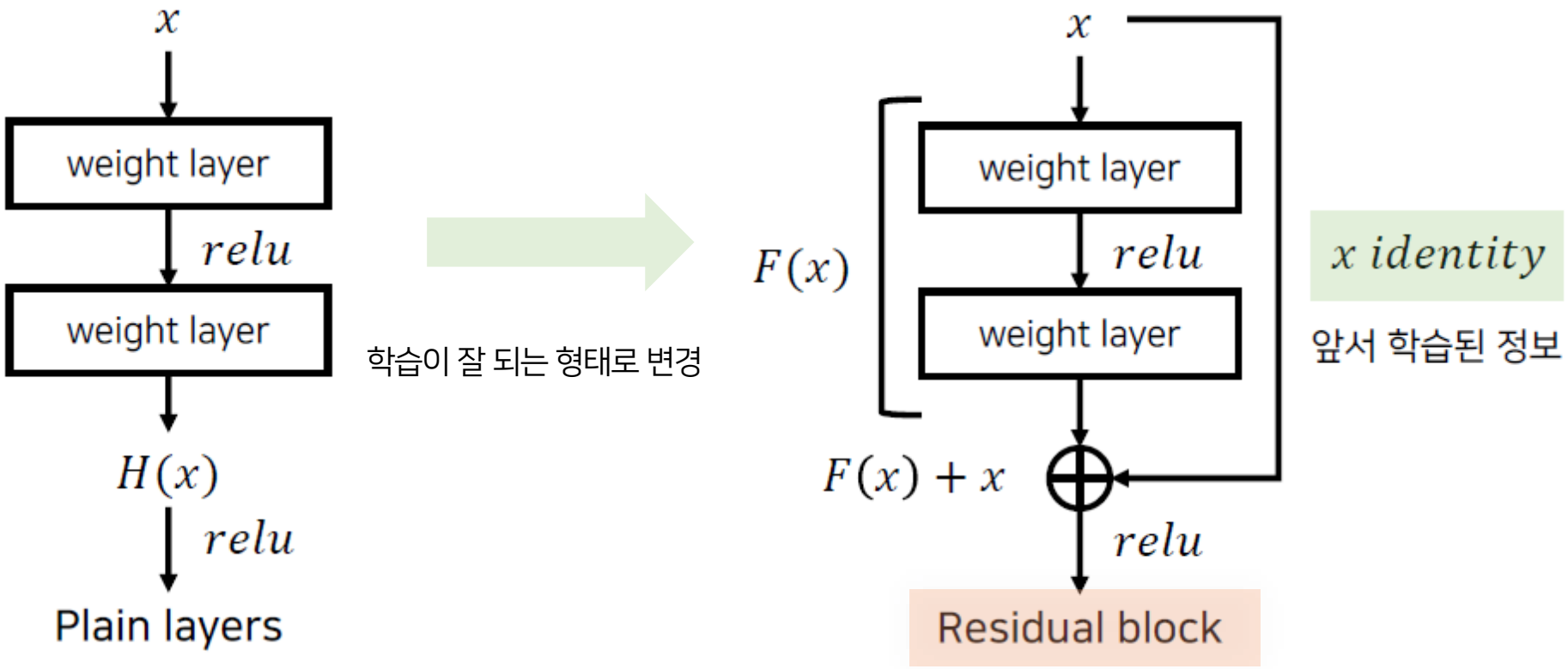


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

본 론

ResNet _ Residual Block

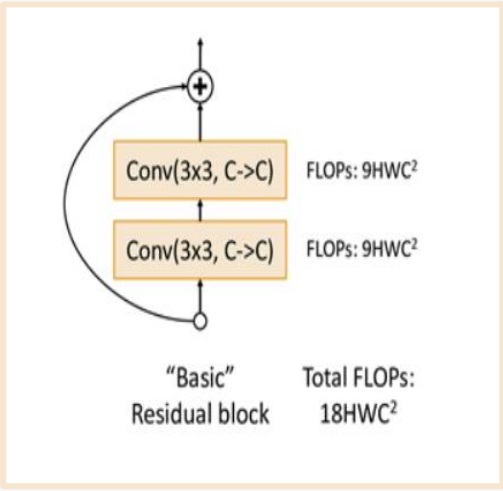


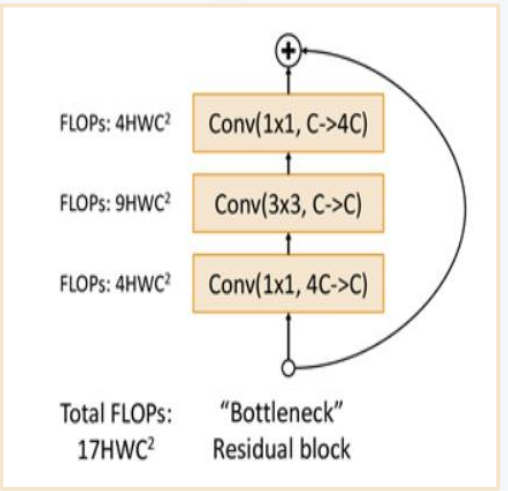
잔여블록(residualblock)을 이용해 네트워크의 **최적화(optimization) 난이도**를 낮추는 방법

실제로 내재한 mapping인 $H(x)$를 곧바로 학습하는 것은 어려우므로 대신 $F(x) = H(x)-x$ 를 학습함

ResNet _ Bottleneck Block

layer name	output size	-layer	34-layer	50- yer	101-layer	152-layer
conv1	112×112	7×7, 4, stride 2				
conv2_x	56×56	3×pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9





본 론

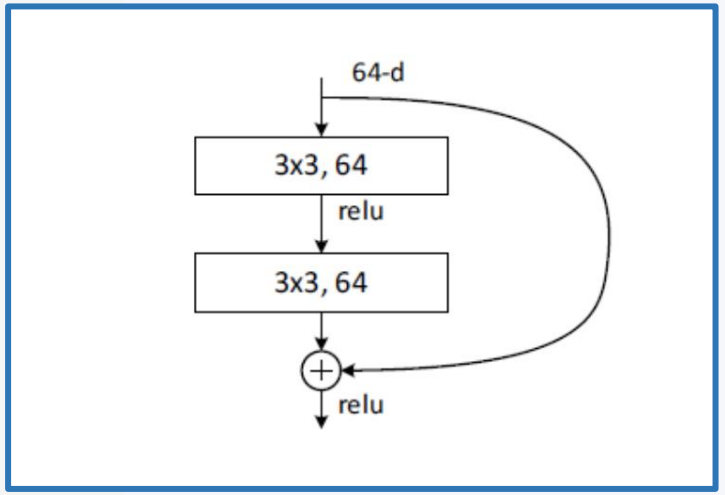
ResNet _ 논문 구현

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet _ 논문 구현

Basic block

```python
# ResNet18을 위해 최대한 간단히 수정한 BasicBlock 클래스 정의
class BasicBlock(nn.Module):
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()

        # 3x3 필터를 사용 (너비와 높이를 줄일 때는 stride 값 조절)
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes) # 배치 정규화(batch normalization)

        # 3x3 필터를 사용 (패딩을 1만큼 주기 때문에 너비와 높이가 동일)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes) # 배치 정규화(batch normalization)

        self.shortcut = nn.Sequential() # identity인 경우
        if stride != 1: # stride가 1이 아니라면, Identity mapping이 아닌 경우
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x) # (핵심) skip connection
        out = F.relu(out)
        return out
```



Basic block output = (# params , output_channels , input_width, input_height) → stride = 1
Basic block output = (# params , output_channels , input_width/2, input_height/2) → stride = 2

ResNet _ 논문 구현

Resnet 18

```python
# ResNet 클래스 정의
class ResNet(ImageClassificationBase):
    def __init__(self, block, num_blocks, num_classes=6):
        super(ResNet, self).__init__()
        self.in_planes = 64

        # 64개의 3x3 필터(filter)를 사용
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes # 다음 레이어를 위해 채널 수 변경
        return nn.Sequential(*layers)
```

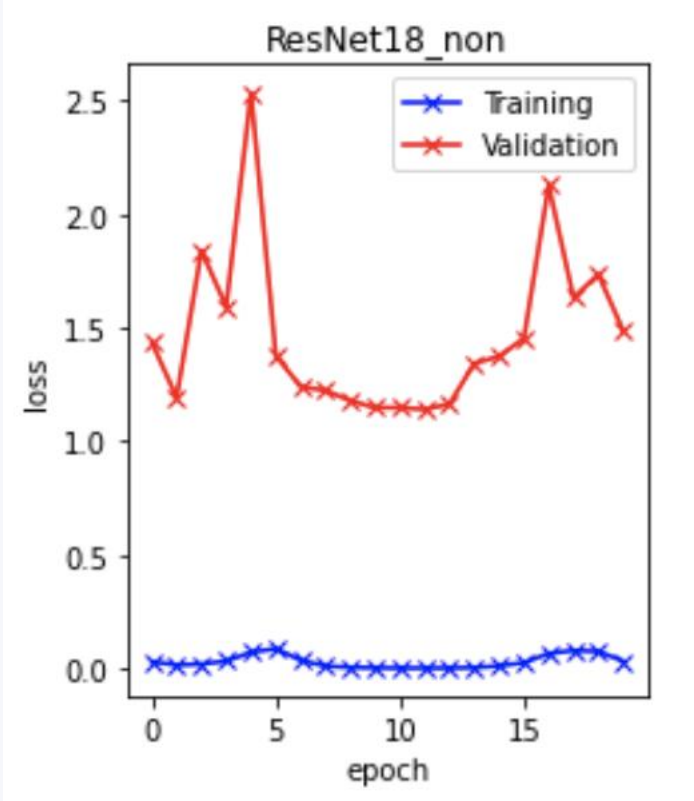
```python
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.maxpool(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out


# ResNet18 함수 정의
def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])

model_res18_non= ResNet18()
```

ResNet _ 논문 구현

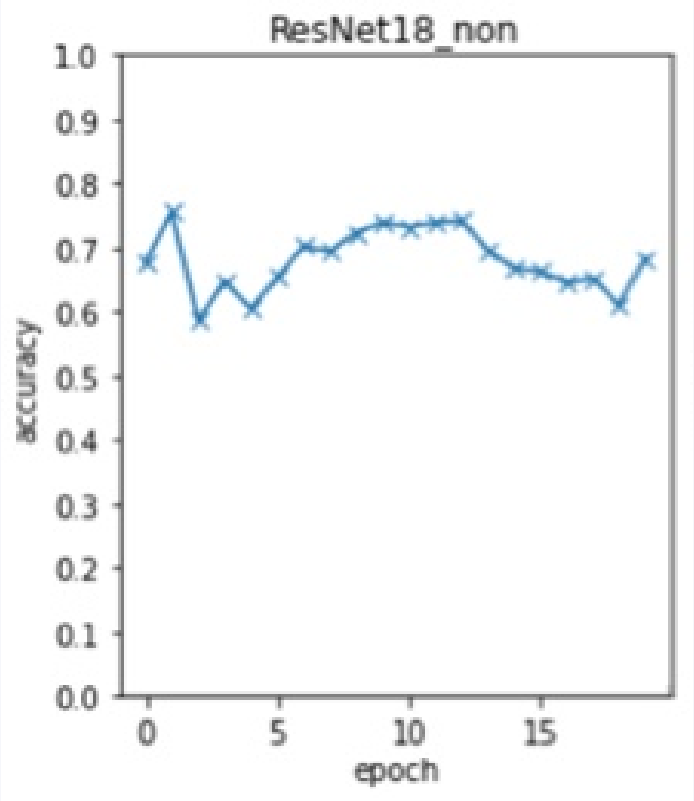
Resnet 18

논문 구현 세팅

- Optimizer : Adam
- Learning rate : 5.5e-5
- Batch Normalization BN 사용
- No dropout



Train loss/ Validation loss



Validation accuracy

ResNet _ 사전학습망

Torch에서 제공하는 사전 학습망 모형

```
import torchvision.models as models

Resnet 18
self.network = models.resnet18(pretrained=True

Resnet 34
self.network = models.resnet34(pretrained=True)

Resnet 50
self.network = models.resnet50(pretrained=True)

Resnet 101
self.network = models.resnet101(pretrained=True)

Resnet 152
self.network = models.resnet152(pretrained=True)
```

사전 학습망 + fully connected layer

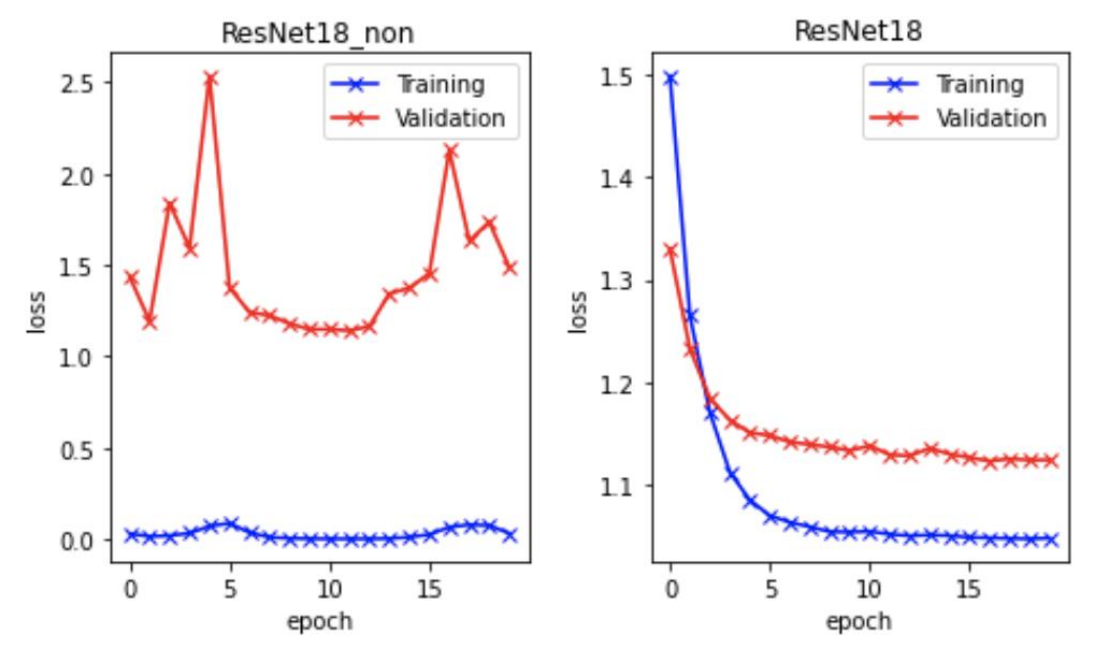
```
class ResNet18(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        # Use a pretrained model
        self.network = models.resnet18(pretrained=True)
        # Replace last layer
        num_ftrs = self.network.fc.in_features
        self.network.fc = nn.Linear(num_ftrs, len(dataset.classes))

    def forward(self, xb):
        return torch.sigmoid(self.network(xb))

model_res18 = ResNet18()
```

이 부분 변경

결 론

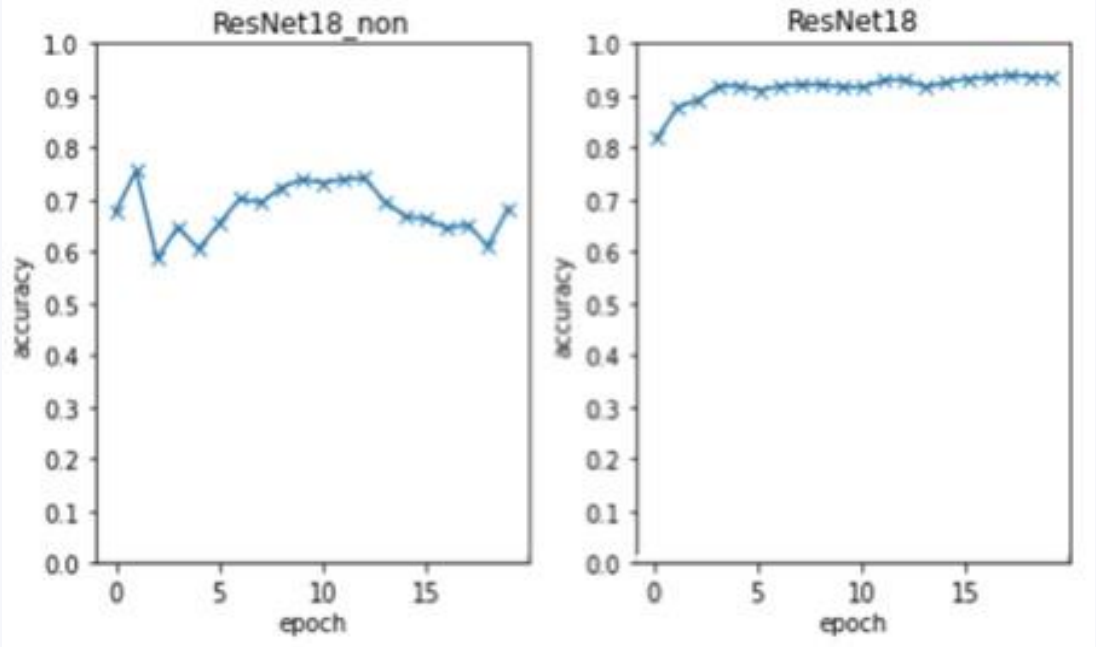
결과 _ ResNet 18 논문 구현 vs 사전 학습망

Train loss/ Validation loss



논문 구현 모형(왼) 사전학습모형(오)

Validation accuracy



논문 구현 모형(왼) 사전학습모형(오)

결과 _ 사전학습망

Model	Time	Validation accuracy
ResNet18	5분	0.9371
ResNet34	7분	0.9397
ResNet50	13분	0.9527
ResNet101	22분	0.9482
ResNet152	31분	0.9482

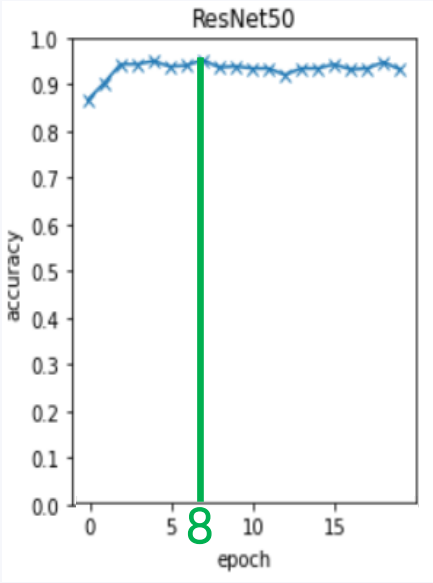
결 론

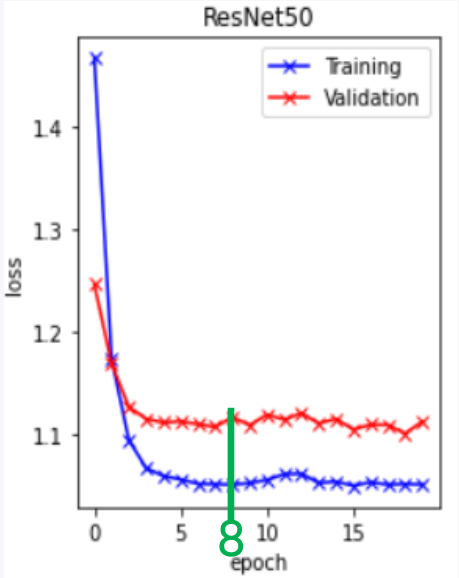
최적 모형 선택

Epoch이 8인 ResNet50모형을 최적 모형으로 선택

```
Epoch 1: train_loss: 1.4675, val_loss: 1.2462, val_acc: 0.8670, time : 0.6828
Epoch 2: train_loss: 1.1736, val_loss: 1.1689, val_acc: 0.9031, time : 0.6763
Epoch 3: train_loss: 1.0944, val_loss: 1.1267, val_acc: 0.9437, time : 0.6732
Epoch 4: train_loss: 1.0670, val_loss: 1.1153, val_acc: 0.9437, time : 0.6783
Epoch 5: train_loss: 1.0594, val_loss: 1.1114, val_acc: 0.9504, time : 0.6747
Epoch 6: train_loss: 1.0560, val_loss: 1.1127, val_acc: 0.9388, time : 0.6753
Epoch 7: train_loss: 1.0518, val_loss: 1.1099, val_acc: 0.9411, time : 0.6777
Epoch 8: train_loss: 1.0513, val_loss: 1.1073, val_acc: 0.9527, time : 0.6783
Epoch 9: train_loss: 1.0509, val_loss: 1.1167, val_acc: 0.9371, time : 0.6763
Epoch 10: train_loss: 1.0524, val_loss: 1.1090, val_acc: 0.9393, time : 0.6789
Epoch 11: train_loss: 1.0555, val_loss: 1.1193, val_acc: 0.9348, time : 0.6801
Epoch 12: train_loss: 1.0612, val_loss: 1.1144, val_acc: 0.9348, time : 0.6812
Epoch 13: train_loss: 1.0615, val_loss: 1.1208, val_acc: 0.9210, time : 0.6789
Epoch 14: train_loss: 1.0526, val_loss: 1.1108, val_acc: 0.9348, time : 0.6798
Epoch 15: train_loss: 1.0539, val_loss: 1.1152, val_acc: 0.9348, time : 0.6796
Epoch 16: train_loss: 1.0499, val_loss: 1.1048, val_acc: 0.9437, time : 0.6797
Epoch 17: train_loss: 1.0533, val_loss: 1.1095, val_acc: 0.9326, time : 0.6811
Epoch 18: train_loss: 1.0509, val_loss: 1.1093, val_acc: 0.9348, time : 0.6794
Epoch 19: train_loss: 1.0516, val_loss: 1.1008, val_acc: 0.9482, time : 0.6802
Epoch 20: train_loss: 1.0511, val_loss: 1.1125, val_acc: 0.9326, time : 0.6807
```

Epoch = 20 loss/accuracy 그래프



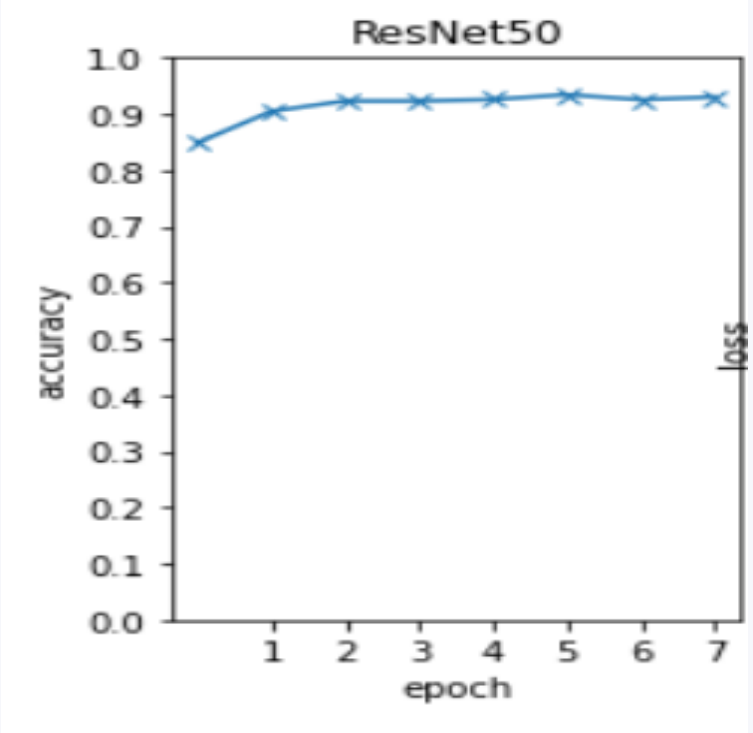


결 론

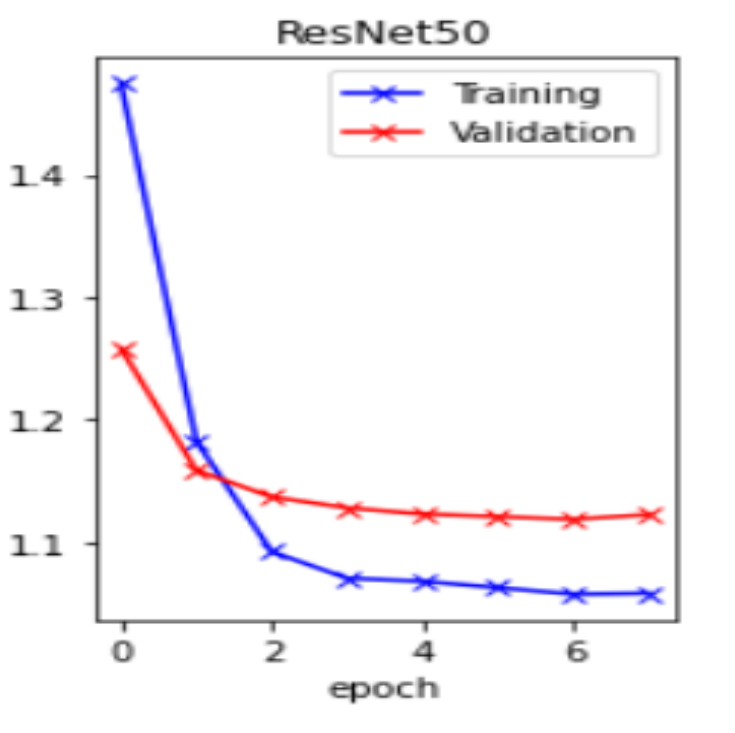
최적 모형 선택

Epoch = 8

Accuracy



Loss function

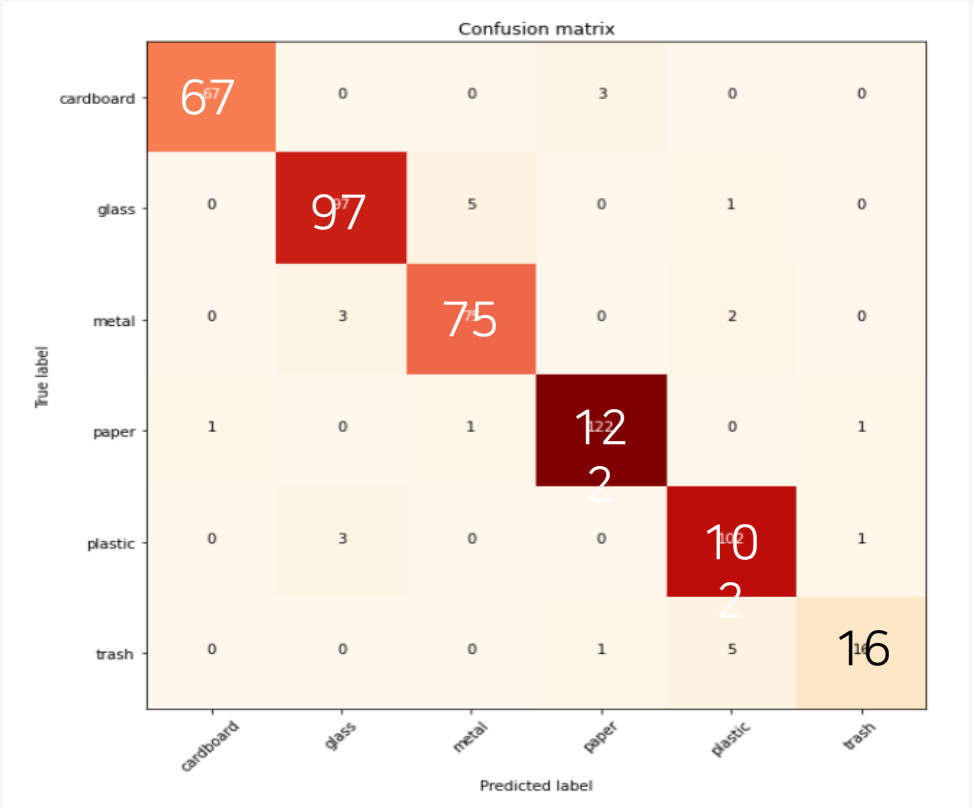


결 론

최적 모형 선택

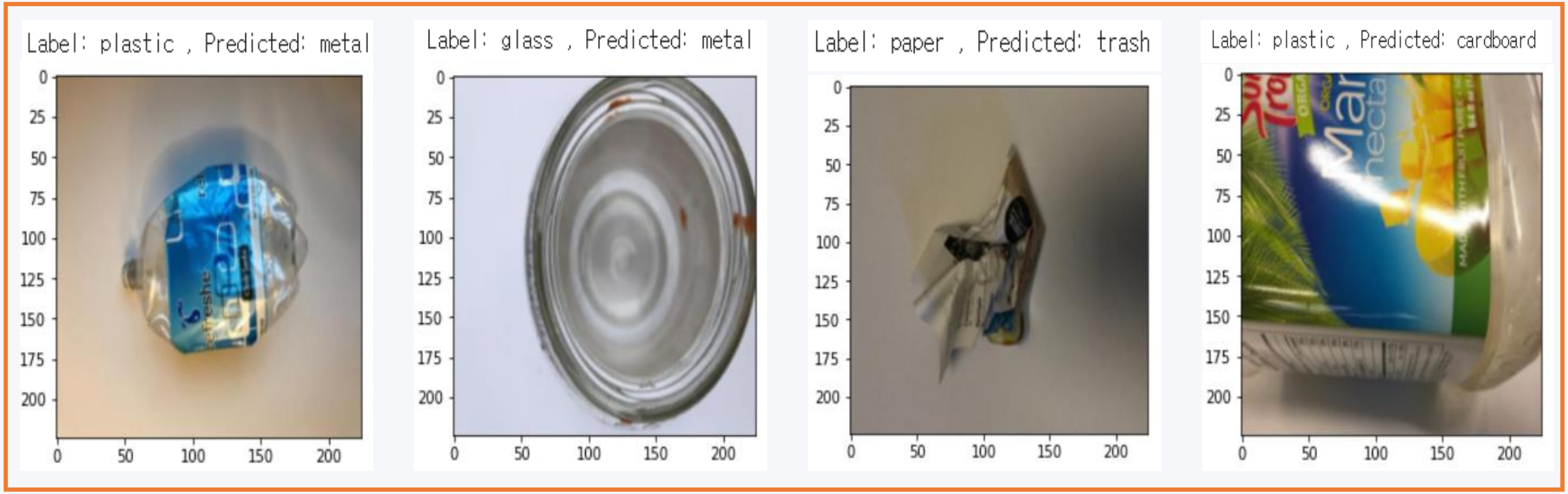
Label	Accuracy
Cardboard	96%
Glass	94%
Metal	94%
Paper	97%
Plastic	96%
Trash	72%

Confusion Matrix



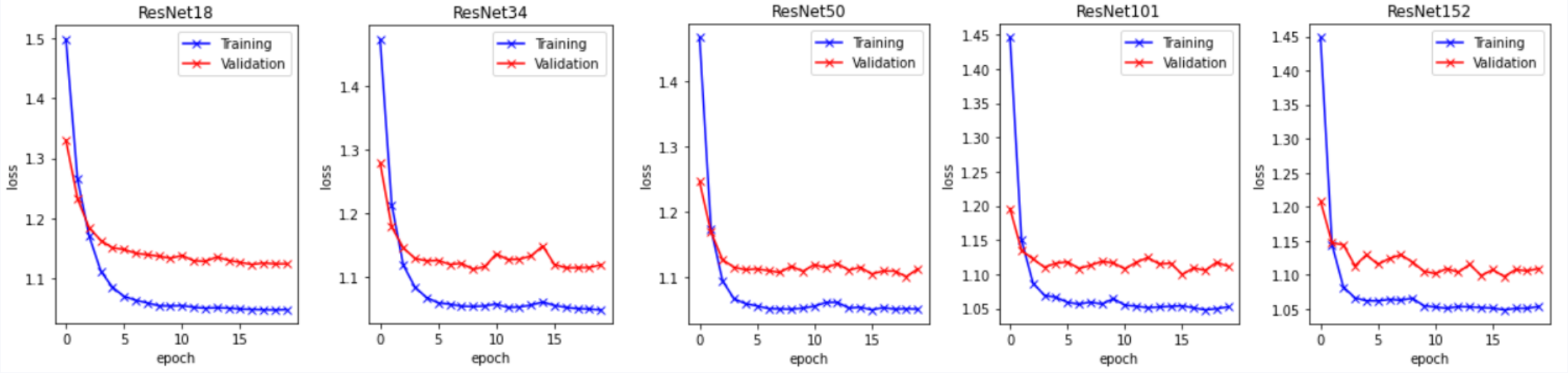
결 론

최적 모형 선택 _ 오분류 예시



결 론

가설 확인 및 정리



가 설 layer가 깊어질수록 모델의 성능이 개선될 것이다

ResNet50까지는 성능이 좋아졌지만 layer가 더 깊어지면서 성능이 떨어짐

논 의

문제점

1. 실제 쓰레기 종류에 비해 Kaggle 데이터는 category가 적음

2. 한 장에 하나의 쓰레기만 나와있으며 배경이 흰색으로 깔끔히 처리된 데이터이기 때문에
   높은 성능을 보였을 것으로 예상

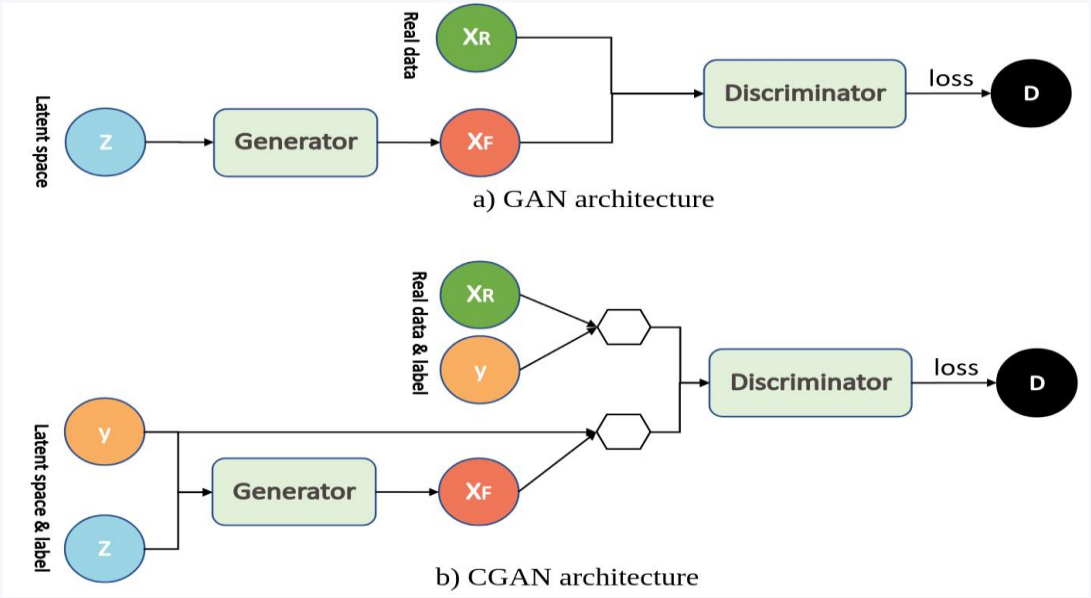
3. 서론에 제시했던 논문들은 kaggle에서 제공하는 데이터 외에 직접 사진을 찍어서 충당
   But 많은 쓰레기 이미지를 직접 찍기에는 시간 및 종류에 한계가 있음

이를 해결하고자 real data와 비슷한 사진을 CGAN을 통해 생성하기 시도

문제 해결 _ CGAN

[논문] Conditional Generative Adversarial Nets(CGAN) _ 2014

대표적인 비지도 학습으로 합성데이터 생성 모델 **GAN의 조건부 버전**



Min-Max Problem

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]$$

논 의

문제 해결 _ CGAN

```python
# discriminator
discriminator = keras.Sequential(
    [
        keras.layers.InputLayer((224, 224, discriminator_in_channels)),
        layers.Conv2D(64, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.GlobalMaxPooling2D(),
        layers.Dense(1),
    ],
    name="discriminator",
)
```

```python
# Create the generator.
generator = keras.Sequential(
    [
        keras.layers.InputLayer((generator_in_channels,)),
        # We want to generate 128 + num_classes coefficients to reshape into a
        # 7x7x(128 + num_classes) map.
        layers.Dense(7 * 7 * generator_in_channels),
        layers.LeakyReLU(alpha=0.2),
        layers.Reshape((7, 7, generator_in_channels)),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
       # 최종 (224,224)가 되도록 4*4 conv2d 레이어 배열
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(3, (7, 7), padding="same", activation="sigmoid"),
    ],
    name="generator",
)
```

```python
all_digits = X.astype("float32") / 255.0  # 0 ~ 1 사이로 픽셀 스케일링
all_labels = to_categorical(y, 6) # label을 one-hot encoding
```

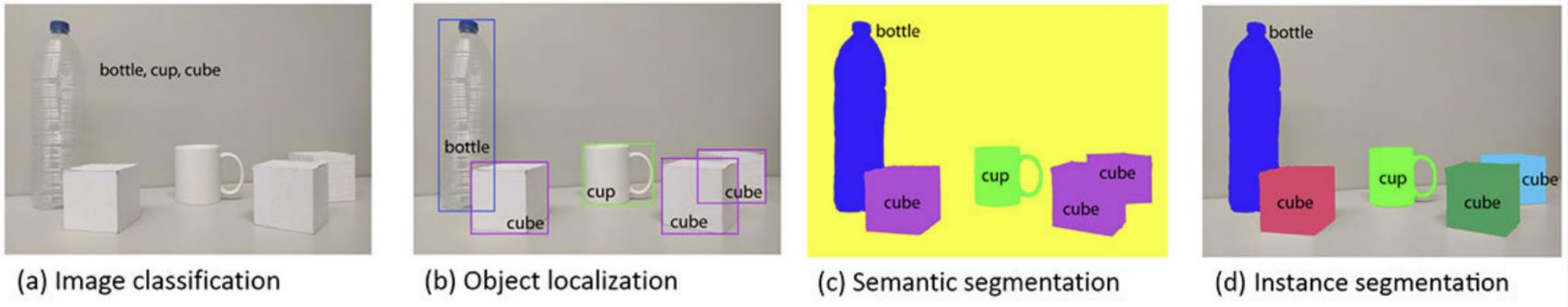
```python
class ConditionalGAN(keras.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(ConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim
        self.gen_loss_tracker = keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = keras.metrics.Mean(name="discriminator_loss")
```

Object Detection/Segmentation 개념을 활용한 재활용 쓰레기 자동 처리 장치

쓰레기가 벨트 컨베이어를 타고 이동하면서 AI 비전(이미지) 기반으로 사진을 찍어 물체를 확인

외형만 보고 판단하는 수준에서 색, 라벨, 병뚜껑 여부, 내부 이물질 여부 및 중류, 오염 구분

활용 가능 기술 – YOLO V3 , RetinaNet / U-Net , Mask R-CNN



(a) Image classification (b) Object localization (c) Semantic segmentation (d) Instance segmentation

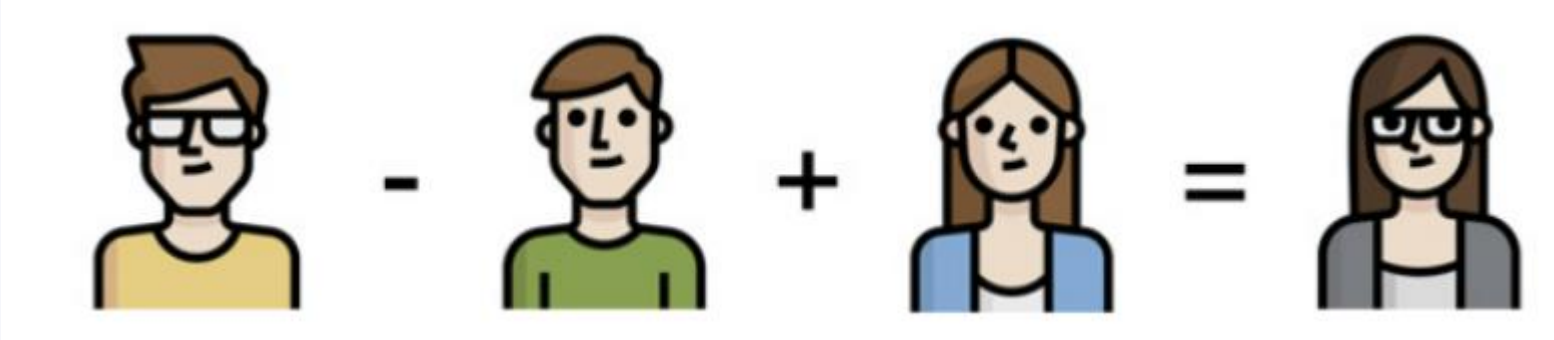
☹ 다만 생활 폐기물 가운데 재활용되는 것을 선별한다는 것은 재활용되지 않는 모든 폐기물 데이터를 갖고 있어야 한다는 의미

발전 방향

Object Detection/Segmentation 개념을 활용한 재활용 쓰레기 자동 처리 장치

😟 다만 생활 폐기물 가운데 재활용되는 것을 선별한다는 것은 재활용되지 않는 모든 폐기물 데이터를 갖고 있어야 한다는 의미

해결 방법 GAN을 활용해 디자인 및 특징이 변경된 쓰레기 이미지 생성



출 처

참고 논문

[논문] Deep Residual Learning for Image Recognition (ResNet) _ *MicroSoft (2015)*

출처 :https://arxiv.org/pdf/1512.03385.pdf

[논문] Conditional Generative Adversarial Nets(CGAN) _ *Mehdi Mirza(2014)*

출처 : https://arxiv.org/abs/1411.1784

[논문] 딥러닝 사진 분류기를 활용한 분리배출 가이드 안드로이드응용 _ *충남대 학교(2021)*

출처 : 한국컴퓨터정보학회 하계학술대회 논문집 제29권 제2호 (2021. 7)

[논문] 딥러닝 기반의 비정형 생활 폐기물 분류 기술 개발 _ *과학기술정보통신부, 고려대학교(2020)*

출처 : https://scienceon.kisti.re.kr/srch/selectPORSrchReport.do?cn=TRKO202100009105&dbt=TRKO

Q & A

감사합니다 :)