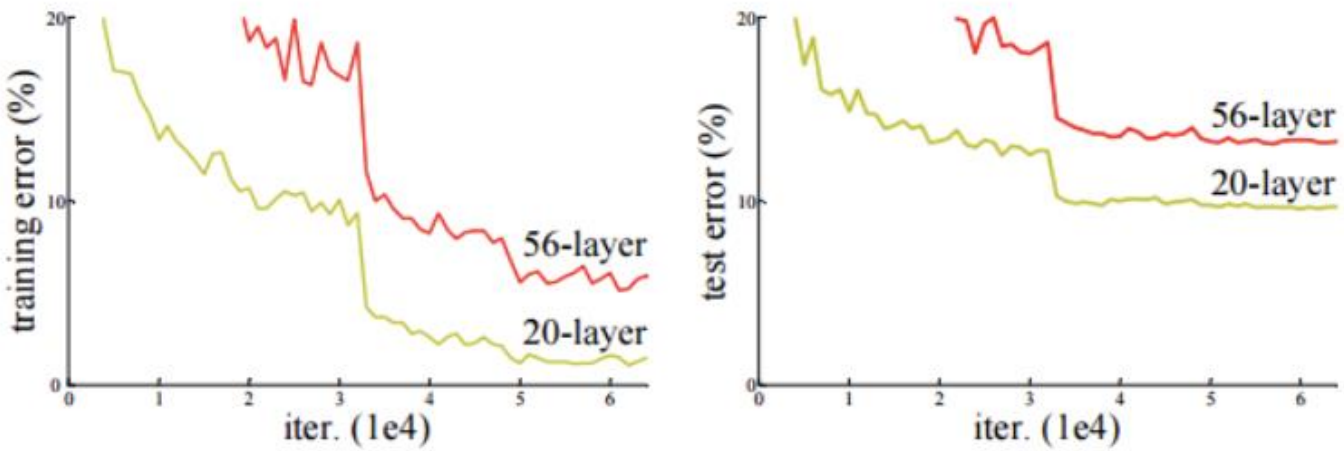
ResNet 구현

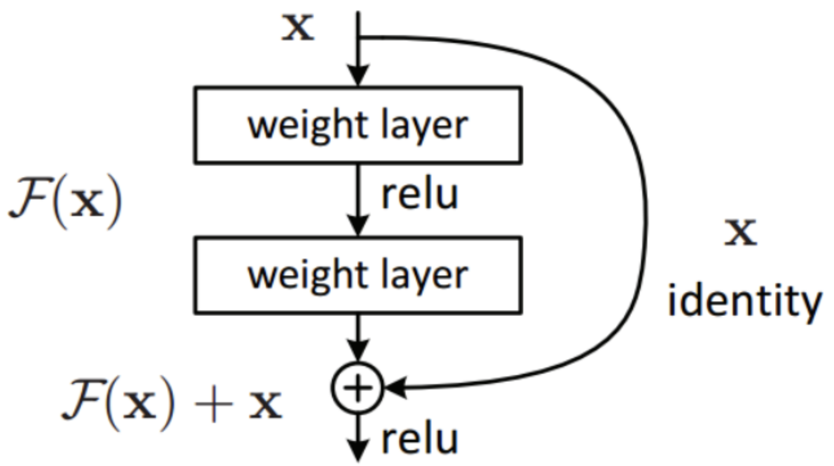
1. ResNet 설명

- 기존 Network의 문제점
  - 층이 깊어질수록 Gradient vanishing/exploding 문제가 발생



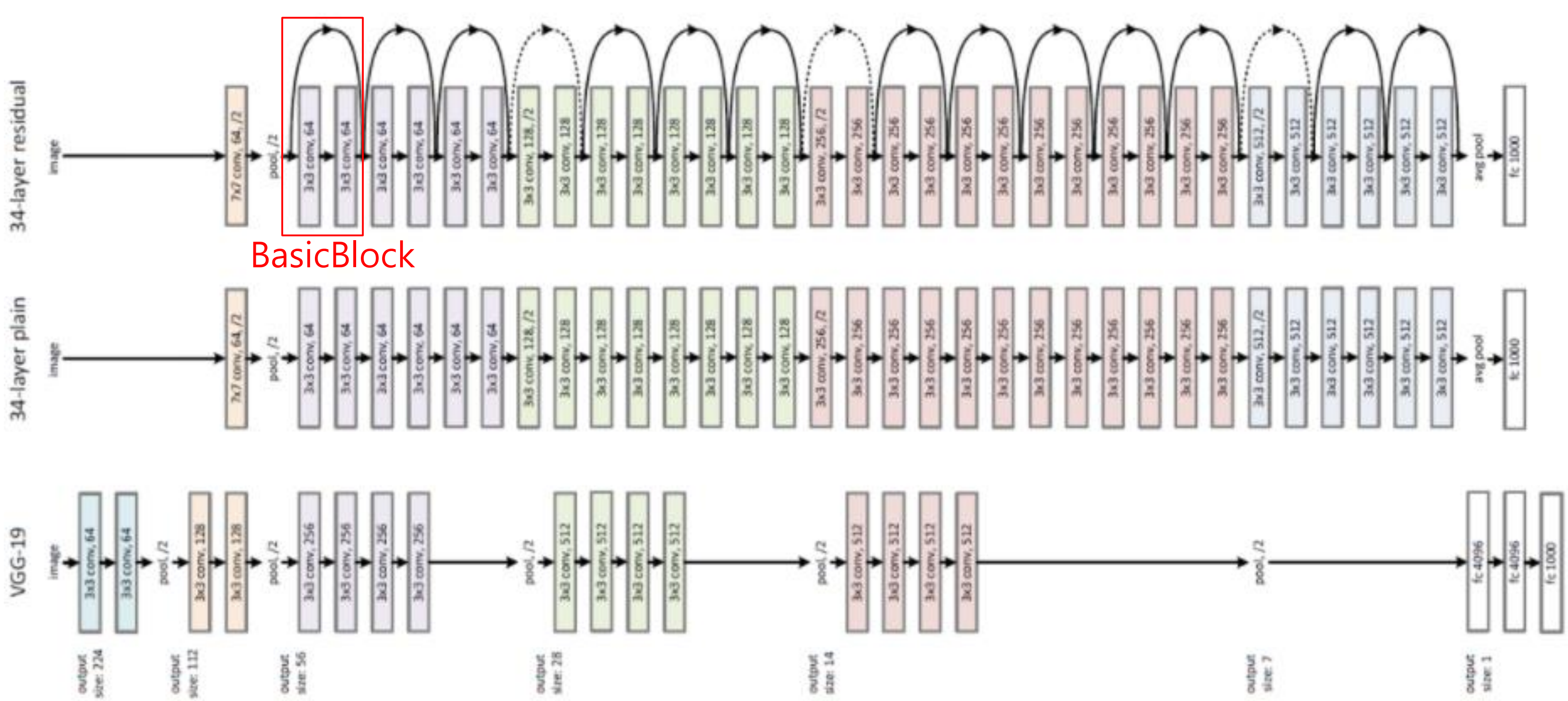
1. ResNet 설명

- ResNet의 핵심 아이디어
  - Skip connection (identity mapping)을 두어 gradient vanishing/exploding 문제를 효과적으로 해결



1. ResNet 설명

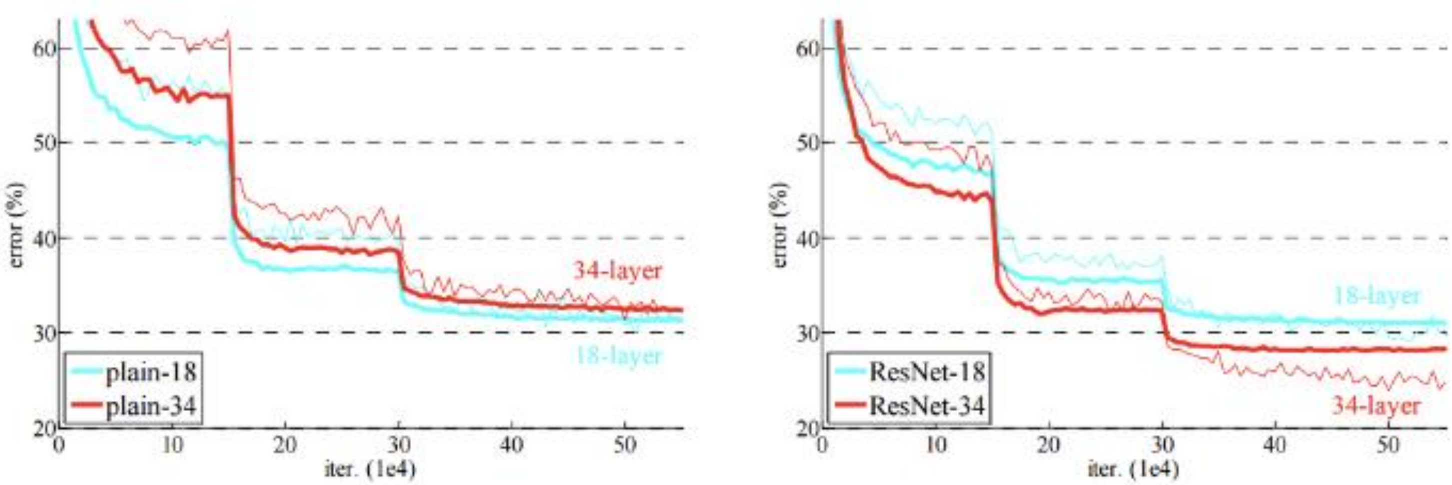
- ResNet의 핵심 아이디어
  - 모든 Convolution layer는 3x3 크기를 가짐
  - Residual connection (Convolution layer 2개) + Skip connection (identity mapping)으로 이루어진 BasicBlock을 만든다음, BasicBlcok을 깊게 쌓아서 높은 성능을 달성
  - Pooling을 제거하고, Stage 맨 앞 convolution layer의 stride=2로 다운샘플링
    - Stage를 거침에 따라 channel 길이는 2배, spatial size는 1/2x1/2이 됨.
  - Batch Normalization Layer(BN) 사용 **(Conv-BN-ReLU)**



1. ResNet 설명

- ResNet의 핵심 아이디어
  - 결과적으로 깊은 층을 쌓아 성능 향상

Plain Nerwork VS ResNet



1. ResNet 설명

- Batch Normalization Layer(BN)
  - 공변량 시프트(Covariate shift)문제를 해결
  - 일반적으로 Convolution Layer와 Activation Layer(e.g., ReLU) 사이에 사용

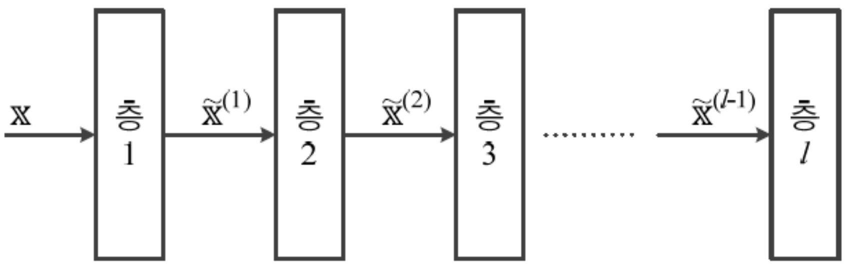
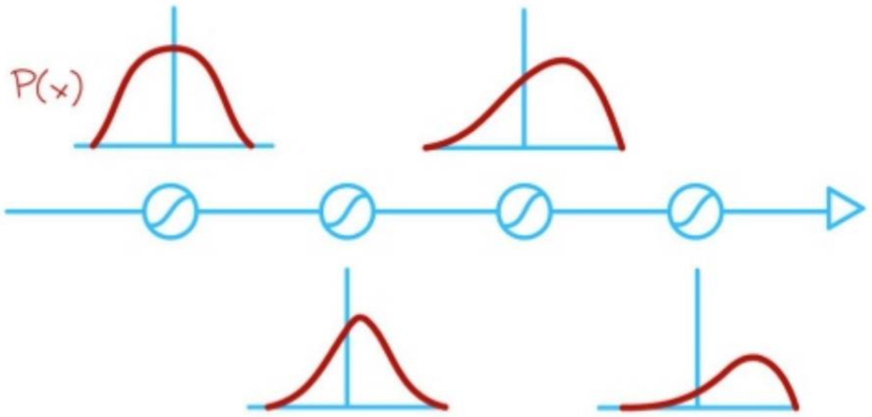


그림 5-17 공변량 시프트 현상



1. ResNet 설명

Batch Normalization Layer(BN)

- 미니배치 $\mathbb{X}_B = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$에 식 (5.15)를 적용하여 $\tilde{\mathbb{X}}_B = \{z_1, z_2, \cdots, z_m\}$를 얻은 후, $\tilde{\mathbb{X}}_B$를 가지고 코드 1을 수행
  - 노드마다 독립적으로 코드 1을 수행(CNN에서는 노드 단위가 아닌 특징맵 단위로 수행)
- γ와 β: 노드마다 고유한 매개변수로서 학습으로 알아냄
- 규제효과(overfitting 방지), 수렴속도 향상, 초기 learning-rate에 둔감성 증대 효과

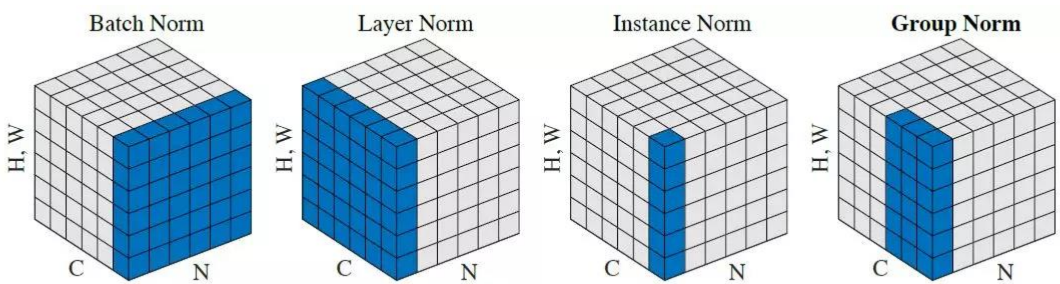
코드 1:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} z_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (z_i - \mu_B)^2$$

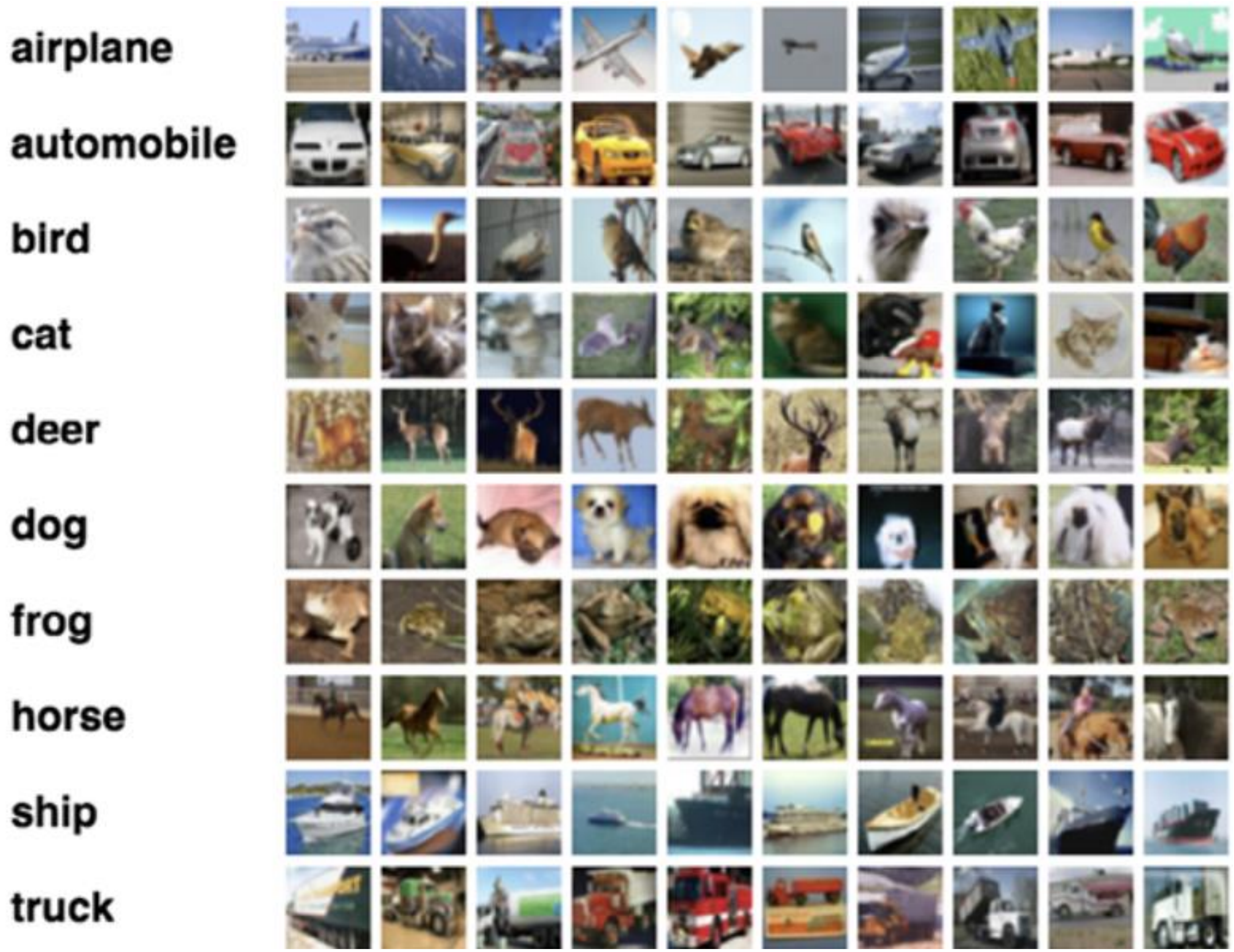
$$\tilde{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \qquad i = 1,2,\cdots,m$$

$$z_i' = \gamma \tilde{z}_i + \beta, \qquad i = 1,2,\cdots,m$$



2. ResNet 구현

- CIFAR-10에 동작하는 ResNet 구현



https://deep-learning-study.tistory.com/534

2. ResNet 구현

- Dataloader 설정

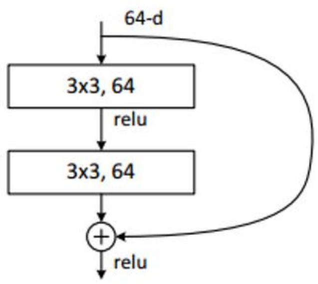
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

num_epochs = 80
learning_rate = 0.001

transform = transforms.Compose([
    transforms.Pad(4),
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32),
    transforms.ToTensor()])

# CIFAR-10 dataset
train_dataset = torchvision.datasets.CIFAR10(root='../../data/',
                                             train=True,
                                             transform=transform,
                                             download=True)

test_dataset = torchvision.datasets.CIFAR10(root='../../data/',
                                            train=False,
                                            transform=transforms.ToTensor())

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=100,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=100,
                                          shuffle=False)
```

https://deep-learning-study.tistory.com/534

2. ResNet 구현

BasicBlock (1/2)

```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride = 1):
        super().__init__()

        # BatchNorm에 bias가 포함되어 있으므로, conv2d는 bias=False로 설정합니다.
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        # identity mapping, input과 output의 feature map size, filter 수가 동일한 경우 사용.
        self.shortcut = nn.Sequential()

        # projection mapping using 1x1conv
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride= stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

        self.relu = nn.ReLU()
        self.maxPool = nn.MaxPool2d(4)  # added for BasicBlock verificaion
        self.fc = nn.Linear(out_channels * 8* 8, 10)  # added for BasicBlock verificaion (32/4 * 32/4)

```

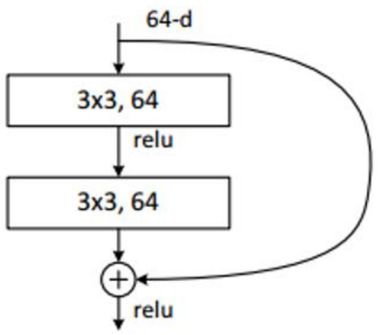


https://deep-learning-study.tistory.com/534

2. ResNet 구현

BasicBlock (2/2)

```
    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        x = self.maxPool(x)        # added for BasicBlock verificaion
        x = x.view(x.size(0), -1) # added for BasicBlock verificaionn
        x = self.fc(x)             # # added for BasicBlock verificaion
        return x

model = BasicBlock(3, 64)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```



https://deep-learning-study.tistory.com/534

2. ResNet 구현

- Training Phase

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)

        loss.backward()
        optimizer.step()

        if (batch_idx+1) % 100 ==0:
            print ("Epoch [{}/{}], step [{}/{}] Loss: {:4f}".format(epoch+1,
                        num_epochs, batch_idx+1, total_step, loss.item()))

    if (epoch+1) % 20 == 0:
        curr_lr /= 3
        update_lr(optimizer, curr_lr)
```

https://deep-learning-study.tistory.com/534

2. ResNet 구현

Test Phase

```
def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        # sum up batch loss
        test_loss +=  criterion(output, target)
        # get the index of the max log-probability
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader.dataset)
    print('₩nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)₩n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


for epoch in range(0, num_epochs):
    train(epoch)
    test()
```

```
Files already downloaded and verified
Epoch [1/80], step [100/500] Loss: 1.752413
Epoch [1/80], step [200/500] Loss: 1.534544
Epoch [1/80], step [300/500] Loss: 1.463914
Epoch [1/80], step [400/500] Loss: 1.280996
Epoch [1/80], step [500/500] Loss: 1.088101
```

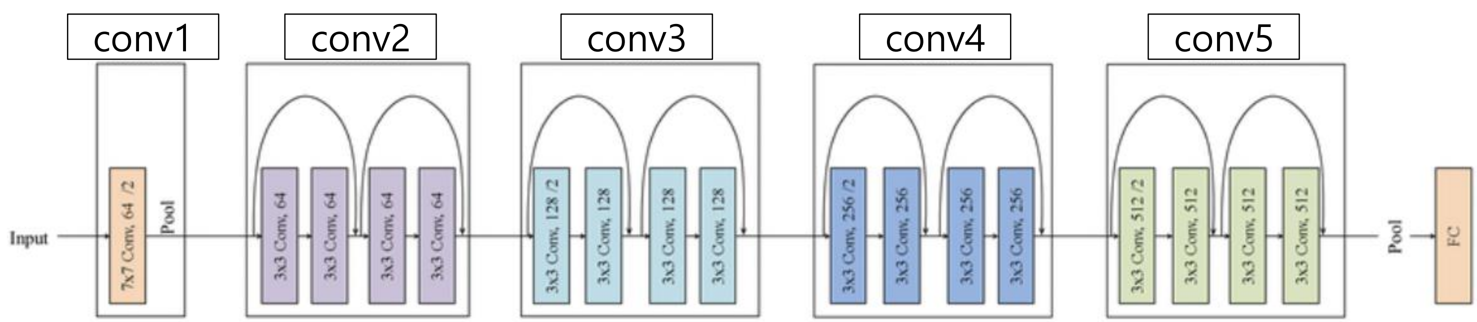
https://deep-learning-study.tistory.com/534

2. ResNet 구현

- ResNet18 구현(1/2)

```python
class ResNet(nn.Module):
    def __init__(self, block, num_block, num_classes=10, init_weights=True):
        super().__init__()

        self.in_channels=64

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )

        self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
        self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
        self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
        self.conv5_x = self._make_layer(block, 512, num_block[3], 2)

        self.avg_pool = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(512, num_classes)

```

```python
model = ResNet(BasicBlock, [2,2,2,2])
```

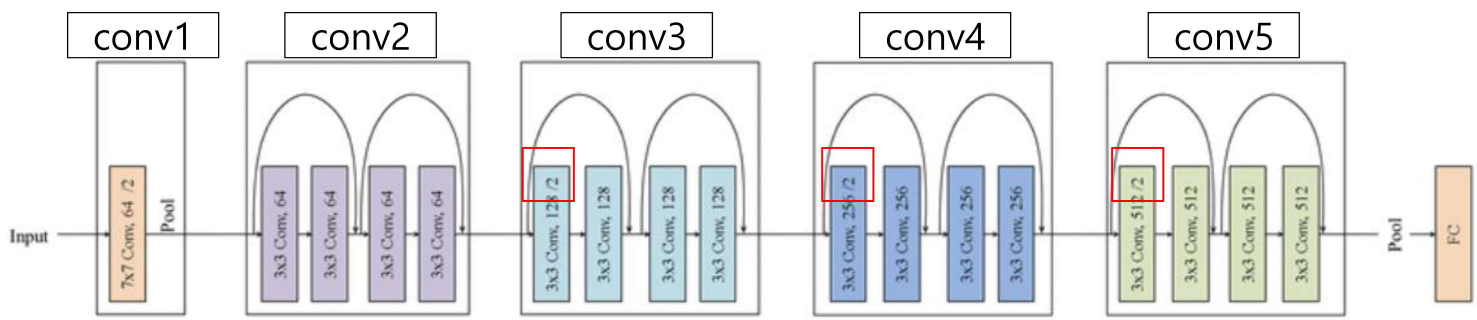


https://deep-learning-study.tistory.com/534

2. ResNet 구현

- ResNet18 구현 (2/2)

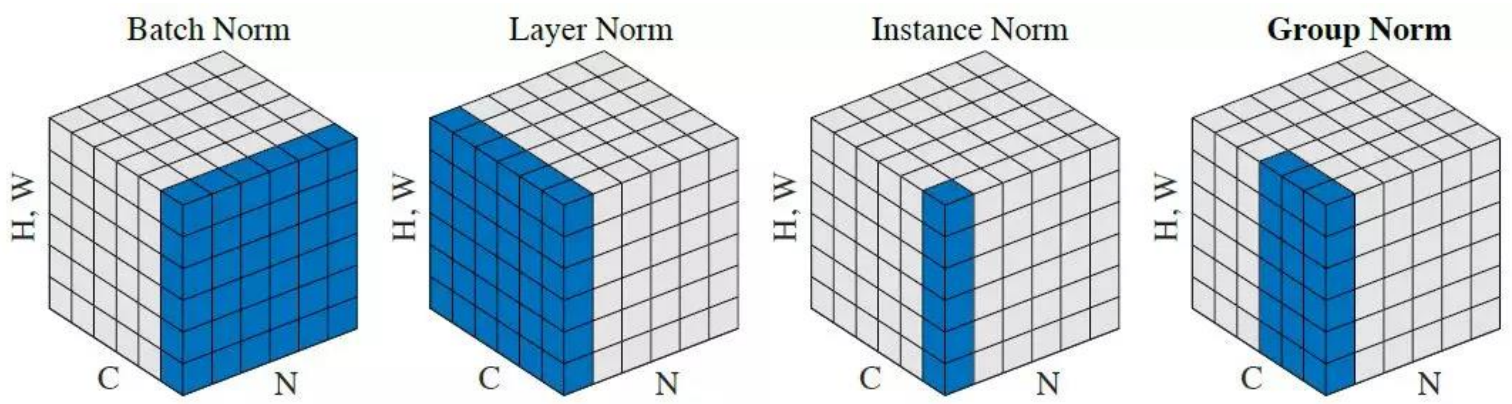
```python
    def _make_layer(self, block, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels

        return nn.Sequential(*layers)

    def forward(self,x):
        output = self.conv1(x)
        output = self.conv2_x(output)
        x = self.conv3_x(output)
        x = self.conv4_x(x)
        x = self.conv5_x(x)
        x = self.avg_pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

model = ResNet(BasicBlock, [2,2,2,2])
```



https://deep-learning-study.tistory.com/534

실습 1

- Batch Normalization 외에 아래의 4개 이상의 Normalization 방법에 대해 찾아보고, 구현한 결과를 보이시오
  - (예시) LayerNorm: https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html

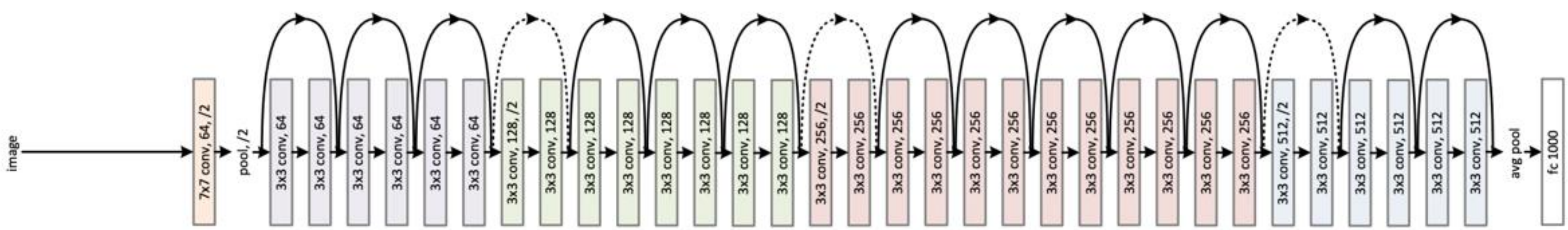


실습 2

■ ResNet34를 구현하시오.

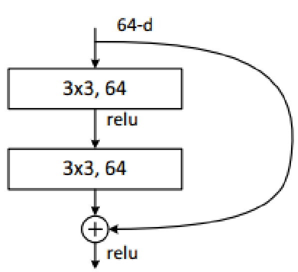
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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

34-layer residual

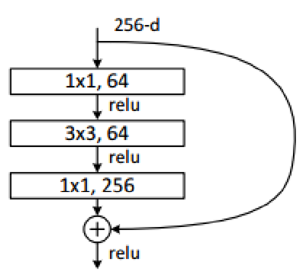


실습 3

- 50층 이상의 ResNet은 매개변수 사용을 최소화 하기 위해 BottleNeck 을 이용한다. BottleNeck을 구현하고, 이를 이용하여 ResNet50도 구현하시오(구현 결과를 CIFAR10에 학습/검증하시오)
  - BottleNeck: Conv1x1을 이용하여 채널 차원 축소 후 Conv3x3을 사용 – 다시 Conv1x1로 채널 차원을 복원하는 방식으로 동작하는 Block
  - (참고) https://deep-learning-study.tistory.com/534



BasicBlock



BottleNeck

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max 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

실습 4

- 모델의 학습 시 가중치 초기화는 성능에 큰 영향을 미친다.
  - Xavier_normal방법, Kaiming_ normal 방법이 무엇인지 인터넷을 통해 알아보고, 이 방법들로 가중치를 초기화 하여 학습 후 결과를 비교하시오.
  - (참고1) https://pytorch.org/docs/stable/nn.init.html
  - (참고2) https://deep-learning-study.tistory.com/534

실습 5

- 아래 ResNet 논문을 읽고, 논문과 동일한 실험 조건에서 ResNet50을 ImageNet 데이터에 대해 학습하고 실험 결과를 보이시오.
  - (논문) https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

CVF
This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

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 [40] 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, 49, 39]. Deep networks naturally integrate low/mid/high-level features [49] 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 [40, 43] reveals that network depth is of crucial importance, and the leading results [40, 43, 12, 16] on the challenging ImageNet dataset [35] all exploit "very deep" [40] models, with a depth of sixteen [40] to thirty [16]. Many other non-trivial visual recognition tasks [7, 11, 6, 32, 27] have also greatly benefited from very deep models.

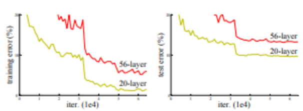


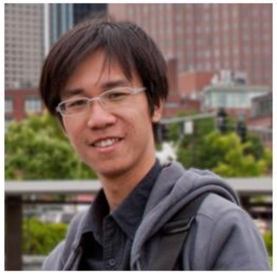
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

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 [14, 1, 8], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 8, 36, 12] 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 [10, 41] 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] http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.



Kaiming He 何恺明

Research Scientist

Facebook AI Research (FAIR), Menlo Park, CA

kaiminghe@fb.com

I am a Research Scientist at Facebook AI Research (FAIR). My research areas include computer vision and deep learning.

I have published some highly influential papers in computer vision and deep learning. My paper on Deep Residual Networks (ResNets) is the most cited paper in all research areas in Google Scholar Metrics 2019, 2020, 2021. The residual connection is a fundamental component in modern deep learning models (e.g., Transformers, AlphaGo Zero). My works on object detection and segmentation, including Faster R-CNN and Mask R-CNN, have made significant impact and are among the most cited papers in these areas. In recent years, I have been interested in unsupervised learning and my explorative works are the top cited papers published in CVPR 2020, 2021, 2022. I have a couple of papers that are among the top-10 most cited papers published in top-tier conferences for each year. My publications have 400,000 citations (as of March 2022) with an increase of over 100,000 per year.

I am a recipient of several prestigious awards in computer vision, including the PAMI Young Researcher Award in 2018, the Best Paper Award in CVPR 2009, CVPR 2016, ICCV 2017, the Best Student Paper Award in ICCV 2017, the Best Paper Honorable Mention in ECCV 2018, CVPR 2021, and the Everingham Prize in ICCV 2021.

Before joining FAIR in 2016, I was with Microsoft Research Asia from 2011 to 2016. I received my PhD degree from the Chinese University of Hong Kong in 2011, and my BS degree from Tsinghua University in 2007.