📖

[논문 리뷰] ResNet [2015]

Introduction

🌟 해당 논문은 **Residual Learning을 활용**함으로서, Optimizer되기 쉽고, increased depth에서 상당한 Accuracy를 얻을 수 있는 ResNet을 소개합니다.

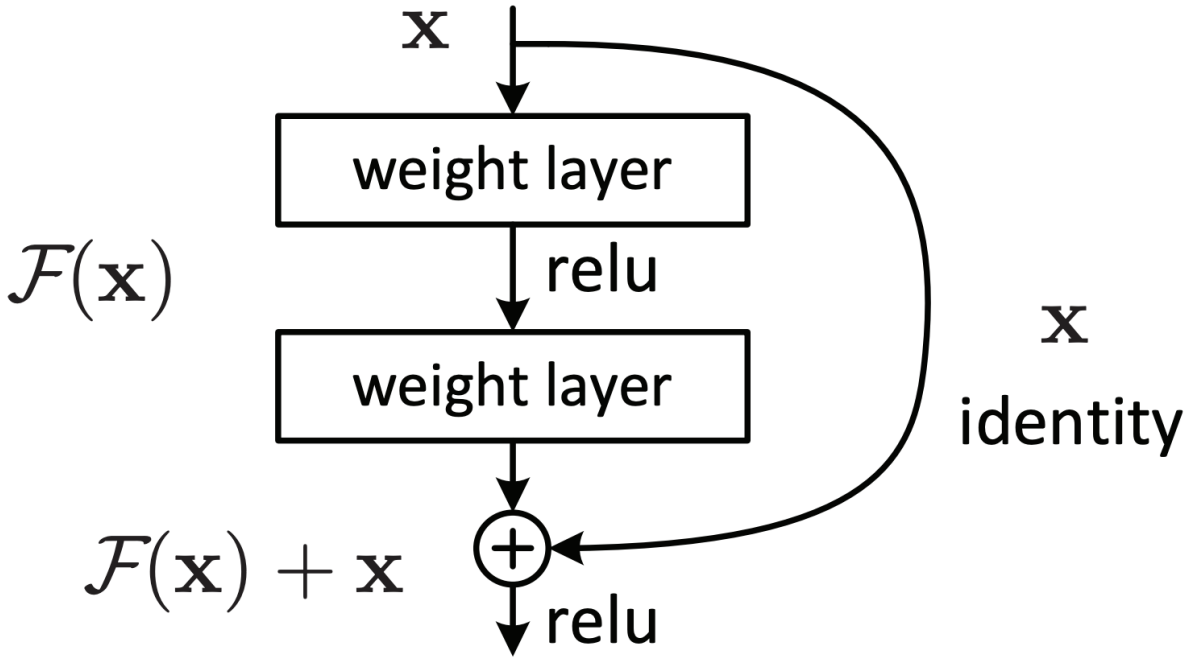
Residual Learning이란, 이전 **layer의 결과를 재활용**하는 것이라고 보면 됩니다.

Deep networks는 naturally하게 추상화에 대한 low / mid / high level의 특징을 classifier과 함께 multi-layer 방식으로 통합합니다. 여기서 각 추상화 level은 쌓인 layer의 수에 따라 더욱 높아질 수 있습니다.

(즉, **High-Level Feature들은, layer**가 **Deep한 것을**, 알 수 있습니다.)

🌟 Deeper Networks에서도, 제한된 상황에서도 최적화될 수 있는 방법이 존재합니다.

이러한 방법은, 추가된 **layer가 identity mapping**을 하는 것이고, **other layer가 추가되지 않은 다른 레이어**들은 더 얕은 모델에서 학습된 layer를 사용하는 것이다.



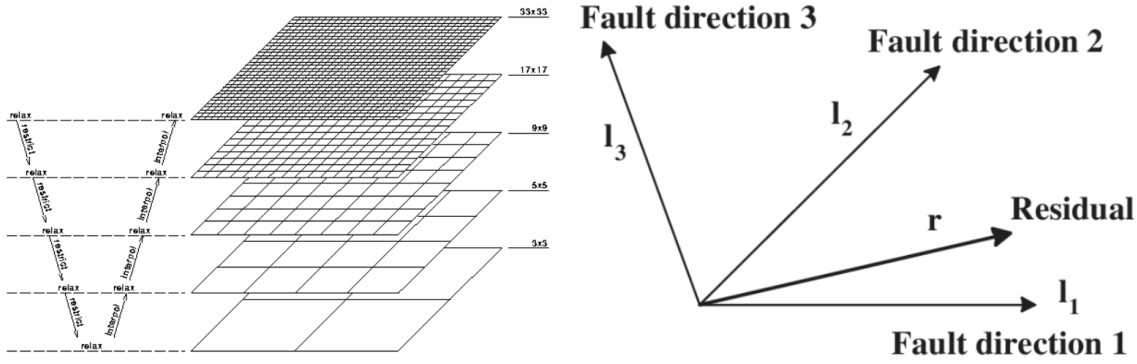
기존 네트워크는 입력을 받고, layer를 거쳐, 입력값 X를 겨처, 출력값 y를 하는 H(x)가 목적입니다.
하지만, **ResNet의 Residual Learning은 H(x)가 아닌 출력과 입력의 차인 H(x) - x를 얻는 것이 목표**입니다.
그러므로, Residual Function은 $F(x) = H(x) - x$ 를 최소화하는 것이 목적입니다.

이전에는, Unreferenced Mapping인 H(x)를 학습시켜야 한다는 점으로 인하여, 문제가 있었는데,
이제는 **H(x) = x라는 최적의 목표 값이 사전에 pre-conditioning으로 제공되기에, $F(x)$ 학습이 쉬워집니다.**

Related Work

Residual Representations

- Vector quantization, Enoding Residual Vector는 Original Vector보다 훨씬 effective합니다.



multi grid 방식과 residual vector 방식

low-level 비전 및 컴퓨터 그래픽 문제에서 편미분 방정식을 풀기 위해 멀티 그리드 방식을 많이 사용해왔는데, 이 방식은 시스템을 여러 scale의 하위 문제로 재구성하는 것이다. 이때, 각 하위 문제는 더 큰 scale과 더 작은 scale 간의 residual을 담당한다.
여기서 멀티 그리드 방식 대신에 두 scale 간의 residual 벡터를 가리키는 변수에 의존하는 방식이 있는데, 이를 계층 기반 pre-conditioning이라고 한다. 이 방식은 해의 residual 특성에 대해 다루지 않는 기존 방식보다 훨씬 빨리 수렴하는 특징이 있다.
즉, 합리적인 문제 재구성과 전제 조건(pre-conditioning)은 최적화를 더 간단하게 수행해준다는 것을 의미한다.

Shortcut Connections

- ResNet's Shortcut Connections는 **parameter-free**이며, 0으로 수렴되지 않기에, **항상 모든 정보를 사용할 수 있습니다. 그러므로, 지속적인 Residual Learning**이 가능합니다.

Deep Residual Learning

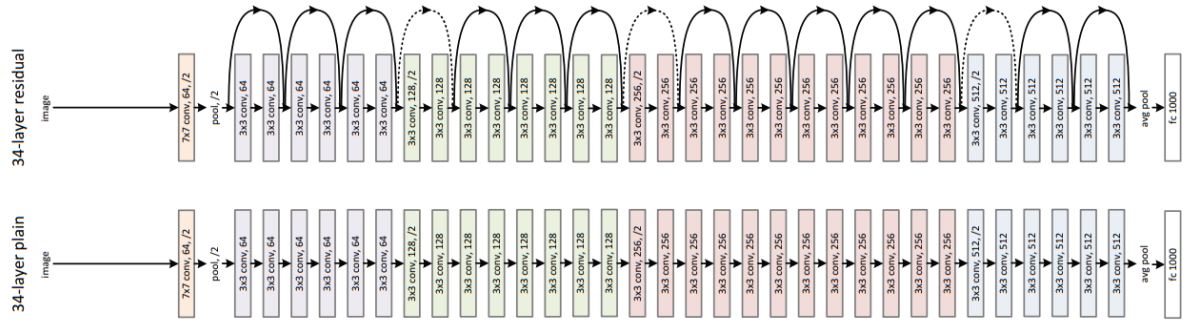
Residual Learning

- 실제로는 Identity Mapping이 Optimizer될 가능성은 낮습니다.
- ResNet's Reformulation은 아마, pre-conditiong을 추가하는데, 도움을 줍니다.
  pre-conditioning으로 인하여, Optimal function이 zero mapping보다 identity mapping에 더 가깝다면, solver가 identity mapping으로 참조하여, **작은 변화를 학습하는 것이 새로운 function을 만드는 것보다 더욱 선호**됩니다.

Identity Mapping by Shortcuts

- **Shortcut connection**은 parameter와 **Computational Complex를 추가**하지 않습니다.
- **F(x) + x 연산을 위하여, x와 F의 차원이 같아야 합니다.**

ResNet Architectures

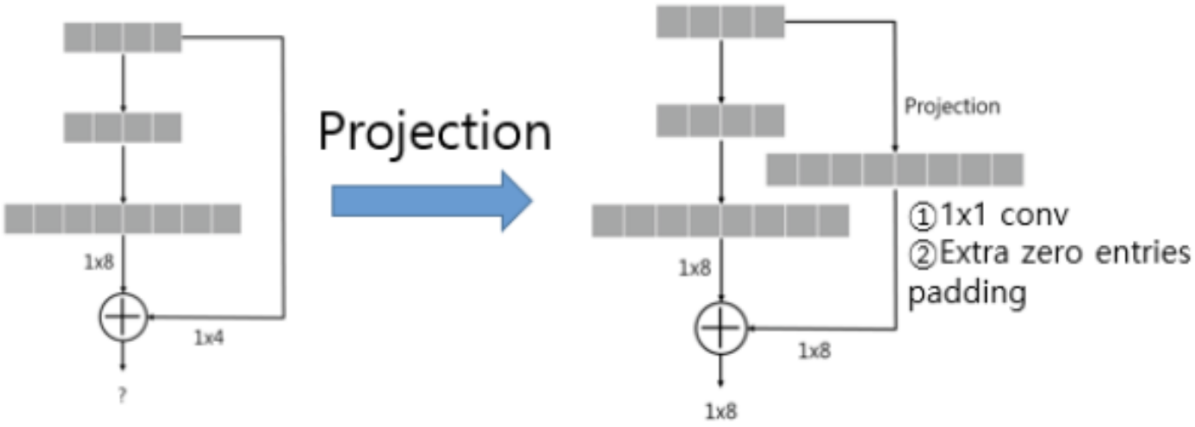


VGGNet에서 영감을 받아 baseline으로 사용하였습니다.
Conv2d의 사이즈가 3x3이고, 2가지 규칙에 기반하여, 설계되었습니다.

- Output Feature map size를 같게 하기 위해서, 같은 수의 Conv filter를 사용해야 합니다.
- Feature map size가 halved 된다면, Time Complexity를 동일하게 유지하게 위해,
  number of filter is Doubled. (**2배 늘려줍니다.**)

Plain Network에 기반하여, Shorcut Connections을 사용합니다.
Network에 상응하는 Counterpart Residual Version으로 사용합니다.
Dimension이 줄어들었을 때, 2가지 방법을 활용합니다.

- **Zero Padding을 활용하여, 차원을 increasing**합니다. (not add parameter)
- **Projection shortcut은 (1 x 1 Convolution)을 사용**합니다.



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

Implementation

- Image의 짧은 부분이 [260, 480] 사이가 되도록 random하게 Resize를 수행합니다.
- Horizontap Flip을 적용하며, [224, 224] 사이즈로 random하게 Image를 Crop 수행합니다.
- Standard Color Augmentation 과 BatchNorm2d를 적용합니다.
- He initialization을 활용하여, 가중치를 초기화하며,

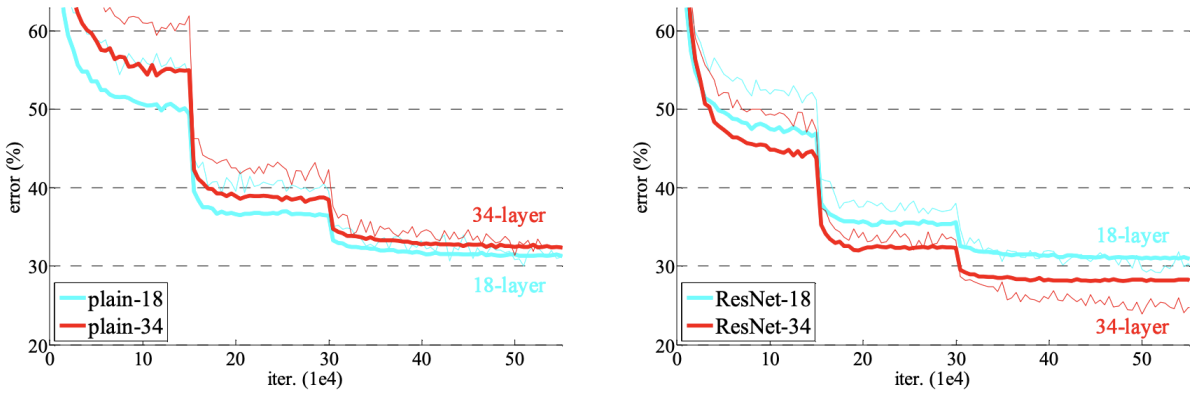
  Optimizer는 SGD(Batch Size : 256)을 적용합니다.
- Learning Rate은 0.1로 시작하며, Weight Decay를 0.0001로 적용하고,

  Momentum은 0.9를 적용합니다.
- $60 * 10^4$만큼 반복 작업을 수행합니다. Dropout은 사용하지 않습니다.

Experiments

🌟 **plain Model**은 **Exponentially low convergence rate**을 가지기에 training error의 감소에 좋지 못한 영향을 끼쳤을 것이라고 추축합니다.

ResNet에서는 Residual Learning으로 인하여, **Degradation문제**가 잘 해결되었으며, Depth가 증가하였더라도, 좋은 정확도를 가질 수 있음을 의미합니다.



ResNet의 Single 모델의 경우 이전의 Ensemble모델을 뛰어넘었으며, **Ensemble을 ResNet에 적용하는 경우, Top-5 Error 3.57%를 달성**할 수 있었습니다.

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

▼ **MODEL [Pytorch]**

```
from torchsummary import summary
import torch
import torch.nn as nn

class block(nn.Module):
    def __init__(self, in_channels, intermediate_channels, identity_downsample = None, stride = 1):
        super(block, self).__init__()
        self.expansion = 4
        self.conv1 = nn.Conv2d(
            in_channels, intermediate_channels, kernel_size = 1, stride = 1, padding = 1, bias = False
        )
        self.bn1 = nn.BatchNorm2d(intermediate_channels)
        self.conv2 = nn.Conv2d(
            intermediate_channels,
            intermediate_channels,
            kernel_size = 3,
            stride = stride,
            padding = 1,
            bias = False
        )
        self.bn2 = nn.BatchNorm2d(intermediate_channels)
        self.conv3 = nn.Conv2d(
                intermediate_channels.
                intermediate_channels * self.expansion,
                kernel_size = 1,
                stride = 1,
                padding = 0,
                bias = False
        )
        self.bn3 = nn.BatchNorm2d(intermediate_channels * self.expansion)
        self.relu = nn.ReLU()
        self.identity_downsample = identity_downsample
        self.stride = stride

    def forward(self, x):
        idenetity = x.clone()         # Copy()

        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)
```

```
        x = self.conv3(x)
        x = self.bn3(x)

        if self.identity_downsample is not None:
                identity = self.identity_downsample(identity)

        x += identity
        x = self.relu(x)
        return x

class ResNet(nn.Module):
    def __init__(self,block, layers, image_channels, num_classes):
        super(ResNet, self).__init()__
        self.in_channels = 64
        self.conv1 = nn.Conv2d(image_channels, 64, kernel_size = 7, stride = 2, padding = 3, bias = False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)

        self.layer1 = self._make_layer(
            block, layers[0], intermediate_channels = 64, stride = 1
        )
        self.layer2 = self._make_layer(
            block, layers[1], intermediate_channels = 128, stride = 2
        )
        self.layer3 = self._make_layer(
            block, layers[2], intermediate_channels = 256, stride = 2
        )
        self.layer4 = self._make_layer(
            block, layers[3], intermediate_channels = 512, stride = 2
        )
        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(512 * 4, num_classes)

    def _make_layer(self, block, num_residual_blocks, intermediate_channels, stride):
        identity_downsample = None
        layers = []

        # Either if we half the input space for ex, 56x56 -> 28x28 (stride=2), or channels changes
        # we need to adapt the Identity (skip connection) so it will be able to be added
        # to the layer that's ahead
        if stride != 1 or self.in_channels != intermediate_channels * 4:
            identity_downsample = nn.Sequential(
                nn.Conv2d(
                    self.in_channels,
                    intermediate_channels * 4,
                    kernel_size=1,
                    stride=stride,
                    bias=False
                ),
                nn.BatchNorm2d(intermediate_channels * 4),
            )

        layers.append(
            block(self.in_channels, intermediate_channels, identity_downsample, stride)
        )

        # The expansion size is always 4 for ResNet 50,101,152
        self.in_channels = intermediate_channels * 4

        # For example for first resnet layer: 256 will be mapped to 64 as intermediate layer,
        # then finally back to 256. Hence no identity downsample is needed, since stride = 1,
        # and also same amount of channels.
        for i in range(num_residual_blocks - 1):
            layers.append(block(self.in_channels, intermediate_channels))

        return nn.Sequential(*layers)

def ResNet50(img_channel=3, num_classes=1000):
    return ResNet(block, [3, 4, 6, 3], img_channel, num_classes)
```

```
def ResNet101(img_channel=3, num_classes=1000):
    return ResNet(block, [3, 4, 23, 3], img_channel, num_classes)


def ResNet152(img_channel=3, num_classes=1000):
    return ResNet(block, [3, 8, 36, 3], img_channel, num_classes)


def test():
    net = ResNet101(img_channel=3, num_classes=1000)
    y = net(torch.randn(4, 3, 224, 224))
    print(y.size())


test()
```

Deep Residual Learning for Image Recognition

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

https://arxiv.org/abs/1512.03385

ResNet (Deep Residual Learning for Image Recognition) 논문 리뷰

논문 제목 : Deep Residual Learning for Image Recognition 오늘은 Deep Residual Learning for Image Recognition에서 마이크로소프트팀이 소개한 ResNet에 대해 다뤄보려 한다. ResNet은 수학적으로 어려운 개념이 적용되었다기보다는 방법론적으로 신박한 개념이

https://phil-baek.tistory.com/entry/ResNet-Deep-Residual-Learning-for-Image-Recognition-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0