

# 20.06.01

## 주제 ResNet

ResNet

Degradation

Skip connection 의 원리

Residual

ResNet의 이점

Lab-10-6-1 Advanced CNN(RESNET-1)

직접 해보기

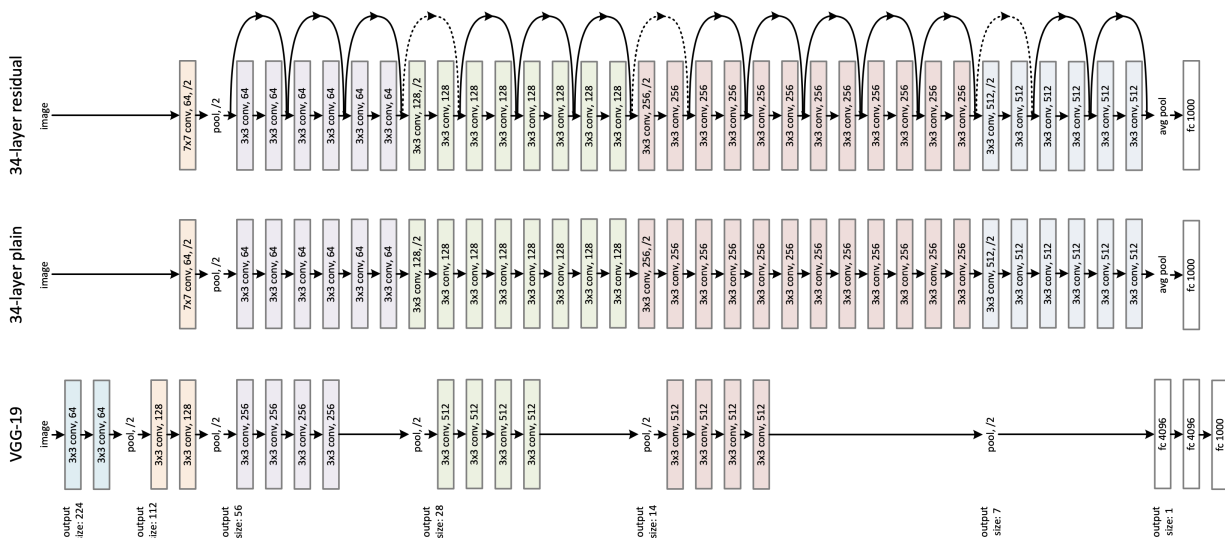
## ResNet

논문 : <https://arxiv.org/pdf/1512.03385.pdf>

논문 해설 : <https://leechamin.tistory.com/184>

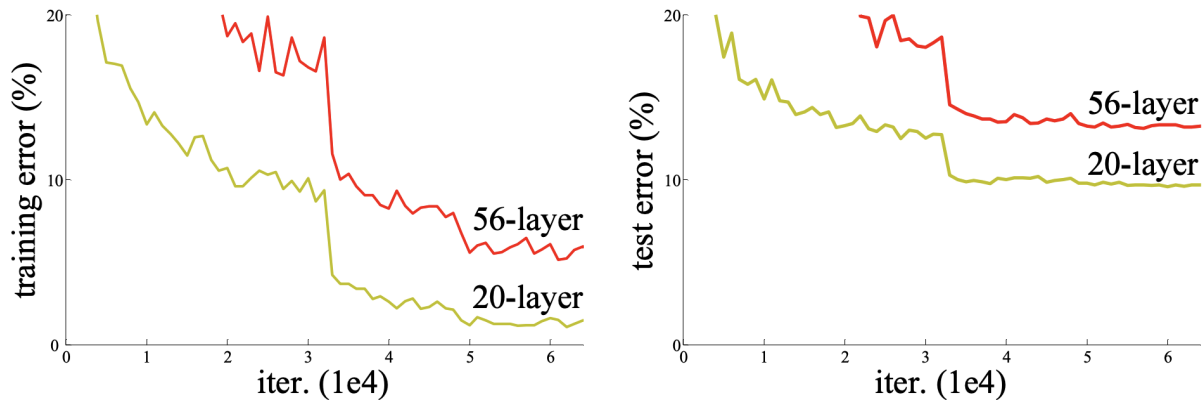
## Degradation

네트워크를 깊게 만들면, gradient vanishing/exploding 문제 때문에 학습이 잘 이루어지지 않고, 비용함수 값이 수렴하지 않는 문제가 있음이 밝혀짐.



이 그림에서 첫번째 네트워크 구조가 ResNet이고, 두번째는 plain layer. 차이점은 데이터가 스킵되는 구간이 resnet에서만 있다는 점인데, plain layer는 레이어가 깊어질 수록 gradient vanishing이나 exploding 문제가 발생하기 쉬워지는 문제가 있다.

그로 인해 오히려 레이어가 깊어질 수록 학습 효율이 떨어진다는 문제가 발생.



그림에서 보이듯 56 레이어 구조보다 20 레이어가 오히려 에러를 잘 감소시키는 것.

이에 대한 해결책으로 정규화 레이어 추가, 가중치 초기화 방법 등이 소개 되었고 꽤 깊은 모델이라도 학습을 수렴시킬 수 있게 되었지만, 학습이 수렴했다 하더라도 모델의 깊이가 깊어지다 보면 어느 순간 더 얇은 모델의 성능보다 더 나빠지는 현상이 발생했다. 이를 Degradation 문제라고 함.

이는 Overfitting(과적합) 문제와는 다른데 과적합 문제는 테스트 성능에 대한 문제이지만, Degradation은 훈련용 데이터에 대한 성능의 문제.

ResNet의 저자들은 이 Degradation 문제를 해결하기 위한 구조를 제시함.

## Skip connection 의 원리

기존의 뉴럴넷의 학습 목적은

입력값  $x$ 를 타겟값  $y$ 으로 맵핑하는 ( $x$ 를 넣으면  $y$ 가 나오는) 함수  $H(x)$  를 찾는 것.

이에 따라 뉴럴넷은  $H(x)-y$ 를 최소화 하는 방향으로 학습을 진행한다.

이 때  $x$ 와 짝지어진  $y$ 는 사실  $x$ 를 대변하는 것으로,

특히 이미지 분류 문제에서는 네트워크의 입 출력을 의미상 같게끔 맵핑해야한다.

그래서 ResNet에서는 관점을 바꿔 네트워크가  $H(x)-x$ 를 얻는 것으로 목표를 수정하였다.

입력과 출력의 잔차(Residual)를  $F(x)=H(x)-x$  라고 정의를 하고

네트워크는 이  $F(x)$ 를 찾는 것이다.

## Residual

여기서 Residual 이란 개념이 다소 낯설 수 있는데, 잔차는 오차와 유사한 개념으로,

모집단에서 회귀식을 얻었다면,

그 회귀식을 통해 얻은 예측값과 실제 관측값의 차이가 오차(error)이고,

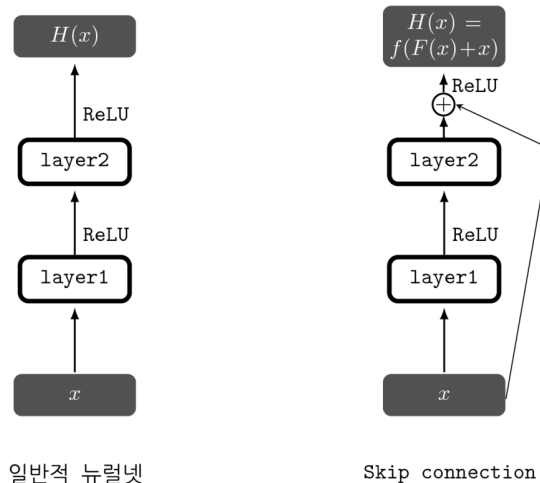
표본집단에서 회귀식을 얻었다면,

그 회귀식을 통해 얻은 예측값과 실제 관측값의 차이가 잔차(Residual)이다.

아무튼 이  $F(x)=H(x)-x$  는 잔차(Residual)이므로,  
 이렇게 잔차를 학습하는 것을 Residual learning, Residual mapping이라고 한다.  
 이를 줄여서 ResNet이라고 하는 것.  
 결과적으로 출력  $H(x)=F(x)+x$  가 된다.

이렇게 네트워크의 입력과 출력이 더해진 것을 다음 레이어의 입력으로 사용하는 것을  
 스킵연결(skip connection) 이라고 한다.

기존의 뉴럴넷은  $H(x)$ 가 어떻게든 정답과 같게 만드는 것이 목적이었다면,  
 이제 입력과 출력 사이의 잔차를 학습하는 것,  
 즉 최적의 경우  $F(x)=0$ 이 되어야하므로 학습의 목표가 이미 정해져 있기 때문에  
 학습 속도가 빨라질 것이고, 네트워크가 잔차를 학습하고 나면,  
 입력값의 작은 변화에도 민감하게 반응 할 것이라라는 것이 ResNet의 가설이다.



Skip connection을 구현 하는 것은 덧셈 연산의 추가 만으로 가능하다.  
 즉 extra parameter나 computational complexity가 추가되지 않는다.

## ResNet의 이점

ResNet을 ImageNet을 통해 검증한 결과


1. ResNet은 deep한 model에서도 쉽게 optimize가 가능했으며,  
 기존 뉴럴넷과는 달리 적은 training error를 보인다.
2. deep 한 ResNet은 depth가 깊어져도 이전의 뉴럴넷보다 더 쉽게  
 accuracy를 도출 할 수 있었다.

그러므로 매우 깊은 모델에서도 recognition tasks에서 훌륭한 퍼포먼스를 보여줌

## Lab-10-6-1 Advanced CNN(RESNET-1)

- BasicBlock
- BottleNeck Block

다양한 architecture가 시도되었고, 이를 똑같이 만들어볼 것이다.

 `torchvision.models.ResNet` 파일에 구현되어 있다. (두 Block 모두 만들어야 한다.)

- 3×224×224 입력을 기준으로 만들도록 되어 있음.
- input size가 다른 경우(CIFAR) ResNet 적용하는 방법을 알아본다.

## 직접 해보기

resnet은 torchvision.model에 포함되어 있어,

```
import torchvision.models.resnet as resnet
```

로 불러올 수 있습니다.

`torchvision.models` 문서로 들어가서 원하는 모델 옆에 **[SOURCE]**라고 되어있는 곳을 클릭하면 소스 코드를 볼 수 있습니다.

ResNet

```
torchvision.models.resnet18(pretrained=False, progress=True, **kwargs)
```

[\[SOURCE\]](#)

ResNet-18 model from "Deep Residual Learning for Image Recognition"

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

```
import torch
import torch.nn as nn
from .utils import load_state_dict_from_url

__all__ = ['ResNet', 'resnet18', 'resnet34', 'resnet50', 'resnet101',
           'resnet152', 'resnext50_32x4d', 'resnext101_32x8d',
           'wide_resnet50_2', 'wide_resnet101_2']

model_urls = {
    'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',
    'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',
    'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',
    'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth',
    'resnet152': 'https://download.pytorch.org/models/resnet152-b121ed2d.pth',
    'resnext50_32x4d': 'https://download.pytorch.org/models/resnext50_32x4d-7cdf4587.pth',
    'resnext101_32x8d': 'https://download.pytorch.org/models/resnext101_32x8d-8ba56ff5.pth',
    'wide_resnet50_2': 'https://download.pytorch.org/models/wide_resnet50_2-95faca4d.pth',
    'wide_resnet101_2': 'https://download.pytorch.org/models/wide_resnet101_2-32ee1156.pth',
}
```

- 생략

```
def conv3x3(in_planes, out_planes, stride=1, groups=1, dilation=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                      padding=dilation, groups=groups, bias=False, dilation=dilation)

def conv1x1(in_planes, out_planes, stride=1):
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False)
```

- 3 by 3이기 때문에 padding이 1이다..?
- 1 by 1 은 padding 0임을 알 수 있다.

```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None, groups=1,
                  base_width=64, dilation=1, norm_layer=None):
        super(BasicBlock, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        if groups != 1 or base_width != 64:
            raise ValueError('BasicBlock only supports groups=1 and base_width=64')
        if dilation > 1:
            raise NotImplementedError("Dilation > 1 not supported in BasicBlock")
        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = norm_layer(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = norm_layer(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x) # 1 by 1
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

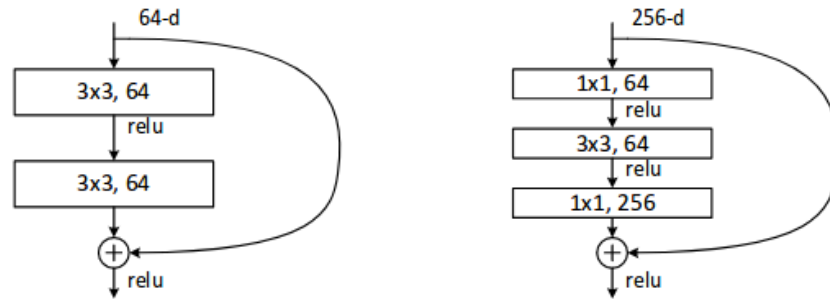
        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out
```

- BasicBlock

- expansion=1
- 오른쪽으로 빠지는 identity가 코드로 구현되어 있다.
- Conv - BatchNorm - ReLU, 더한 후 ReLU



- conv2는 conv1과 달리 stride가 주어지지 않음을 확인할 수 있다.(default 1)

- downsample
  - 만약 stride 값이 1이 아닌 값. 예를 들어 2라고 가정한다면 X로  $3 \times 64 \times 64$ 가 주어질 때, identity(X)가 out(conv 레이어를 통과한)과의 shape이 맞지 않아 덧셈이 불가능한 상황이 생기게 된다.

```
class Bottleneck(nn.Module):
    # Bottleneck in torchvision places the stride for downsampling at 3x3 convolution(self.conv2)
    # while original implementation places the stride at the first 1x1 convolution(self.conv1)
    # according to "Deep residual learning for image recognition" https://arxiv.org/abs/1512.03385.
    # This variant is also known as ResNet V1.5 and improves accuracy according to
    # https://ngc.nvidia.com/catalog/model-scripts/nvidia:resnet_50_v1_5_for_pytorch.

    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None, groups=1,
                 base_width=64, dilation=1, norm_layer=None):
        super(Bottleneck, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        width = int(planes * (base_width / 64.)) * groups
        # Both self.conv2 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv1x1(inplanes, width)
        self.bn1 = norm_layer(width)
        self.conv2 = conv3x3(width, width, stride, groups, dilation)
        self.bn2 = norm_layer(width)
        self.conv3 = conv1x1(width, planes * self.expansion)
        self.bn3 = norm_layer(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x) # 1x1 stride = 1
        out = self.bn1(out)
        out = self.relu(out)
```

```

out = self.conv2(out) # 3x3 stride = stride
out = self.bn2(out)
out = self.relu(out)

out = self.conv3(out) # 1x1 stride = 1, *planes
out = self.bn3(out)

if self.downsample is not None:
    identity = self.downsample(x)

out += identity
out = self.relu(out)

return out

```

- BottleNeck

- planes
- 위의 사진을 보면 3번째 레이어에서 channel의 수가 4배 증가한다.
  - resnet class에 구현되어 있다.
- downsample
  - out과 identity를 덧셈을 진행 후 ReLU가 진행되고 return 된다.

```

class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1000, zero_init_residual=False,
                  groups=1, width_per_group=64, replace_stride_with_dilation=None,
                  norm_layer=None):
        super(ResNet, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        self._norm_layer = norm_layer

        self.inplanes = 64 # 1
        self.dilation = 1
        if replace_stride_with_dilation is None:
            # each element in the tuple indicates if we should replace
            # the 2x2 stride with a dilated convolution instead
            replace_stride_with_dilation = [False, False, False]
        if len(replace_stride_with_dilation) != 3:
            raise ValueError("replace_stride_with_dilation should be None "
                             "or a 3-element tuple, got {}".format(replace_stride_with_dilation))

        self.groups = groups
        self.base_width = width_per_group
        self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, padding=3,
                                bias=False)
        self.bn1 = norm_layer(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2,
                                        dilate=replace_stride_with_dilation[0])
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2,
                                        dilate=replace_stride_with_dilation[1])
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2,
                                        dilate=replace_stride_with_dilation[2])

```

```

self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(512 * block.expansion, num_classes)

for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
    elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

# Zero-initialize the last BN in each residual branch,
# so that the residual branch starts with zeros, and each residual block behaves like an identity.
# This improves the model by 0.2-0.3% according to https://arxiv.org/abs/1706.02677
if zero_init_residual:
    for m in self.modules():
        if isinstance(m, Bottleneck):
            nn.init.constant_(m.bn3.weight, 0)
        elif isinstance(m, BasicBlock):
            nn.init.constant_(m.bn2.weight, 0)

def _make_layer(self, block, planes, blocks, stride=1, dilate=False):
    norm_layer = self._norm_layer
    downsample = None
    previous_dilation = self.dilation
    if dilate:
        self.dilation *= stride
        stride = 1
    if stride != 1 or self.inplanes != planes * block.expansion: #2
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride),
            norm_layer(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample, self.groups,
                        self.base_width, previous_dilation, norm_layer))
    self.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(block(self.inplanes, planes, groups=self.groups,
                            base_width=self.base_width, dilation=self.dilation,
                            norm_layer=norm_layer))

    return nn.Sequential(*layers)

def _forward_impl(self, x):
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1) # 일렬로 펼쳐줌.
    x = self.fc(x)

    return x

def forward(self, x):
    return self._forward_impl(x)

```



- 설명을 위한 resnet18, resnet50

```
def resnet18(pretrained=False, progress=True, **kwargs):  
    return _resnet('resnet18', BasicBlock, [2, 2, 2, 2], pretrained, progress,  
                    **kwargs)
```

```
def resnet50(pretrained=False, progress=True, **kwargs):  
    return _resnet('resnet50', Bottleneck, [3, 4, 6, 3], pretrained, progress,  
                    **kwargs)
```

- #2
  - channel을 맞추기 위해서도 downsample을 이용한다.