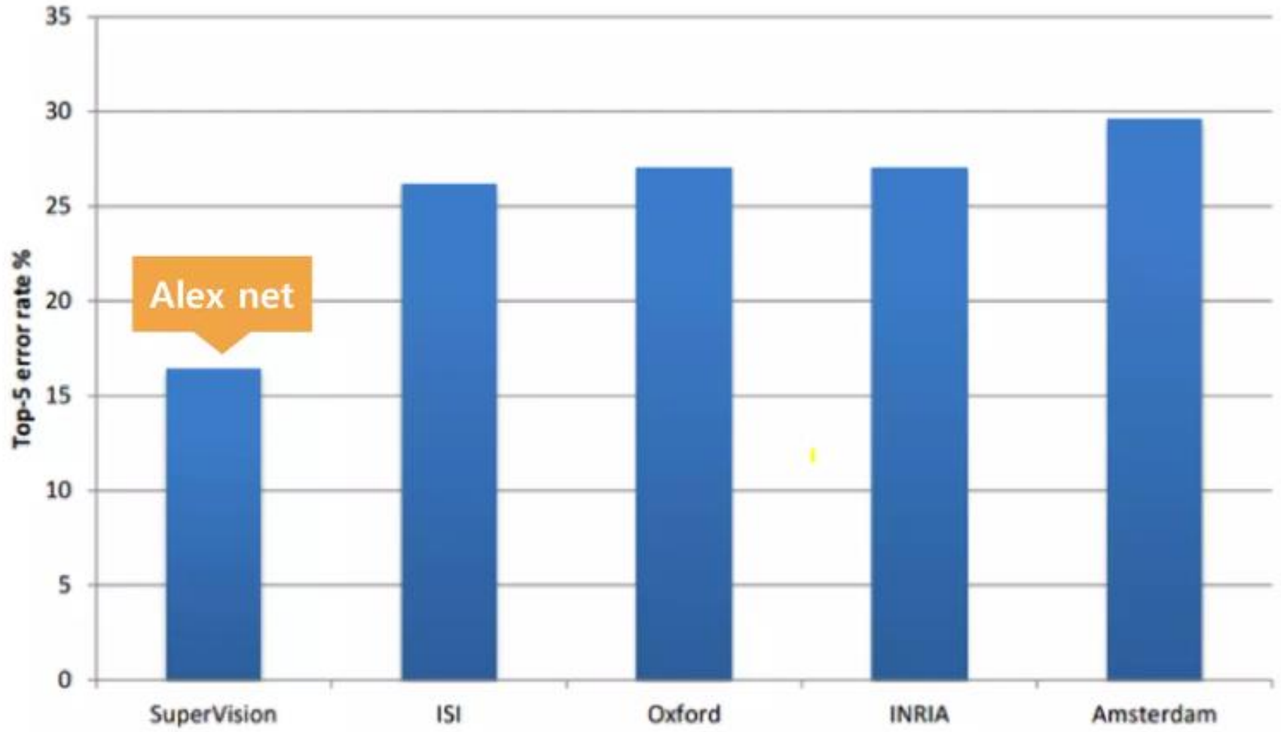
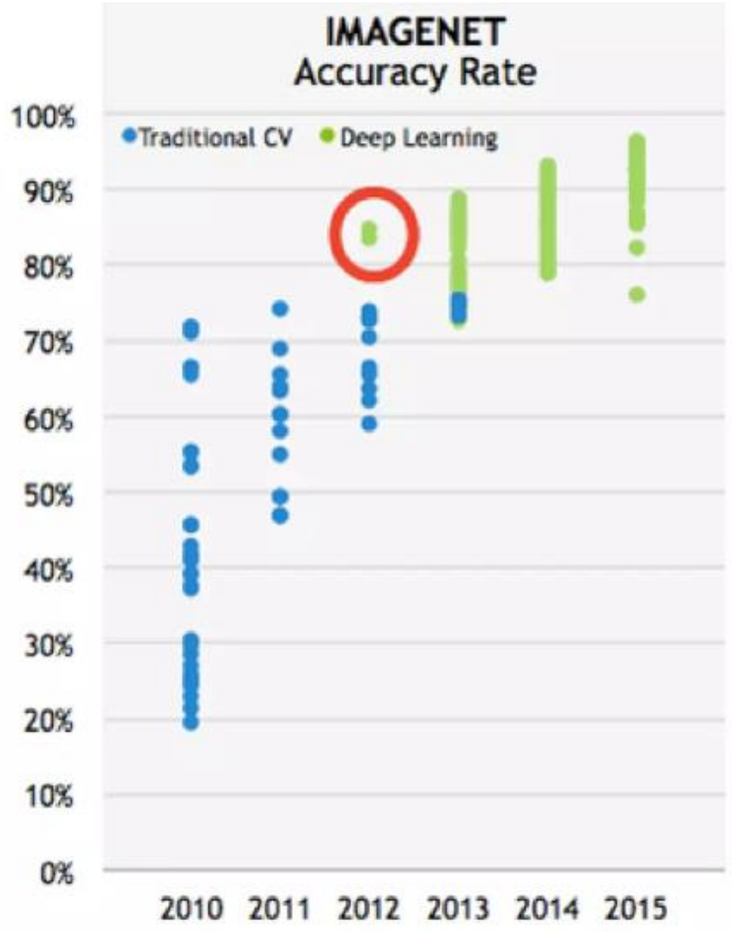
AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

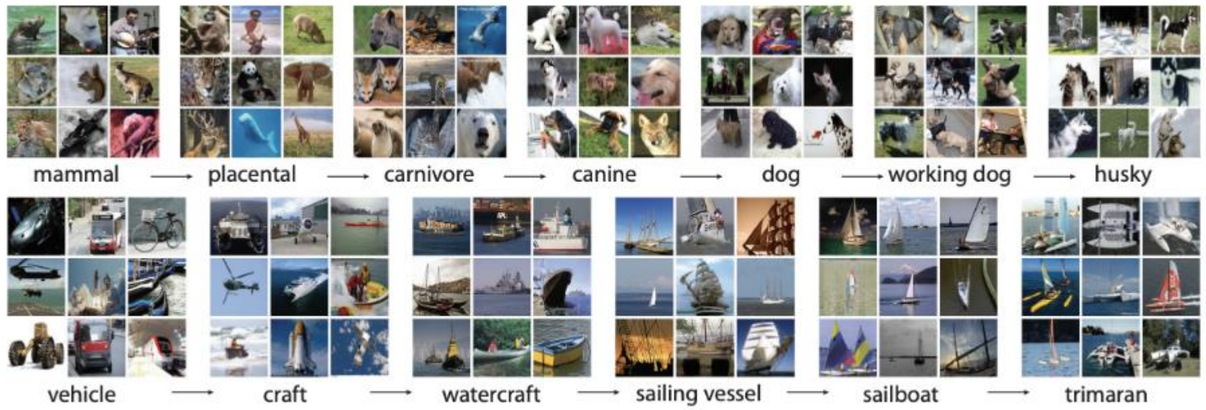
송용호

AlexNet(ILSVRC-2012)





Dataset

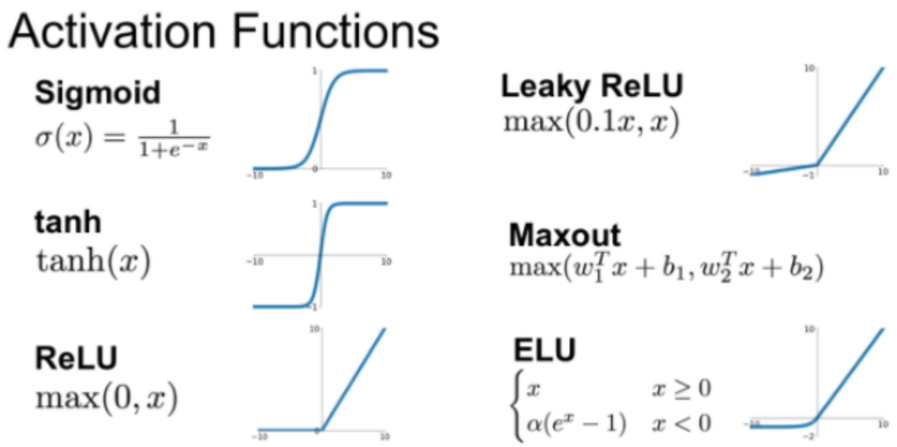


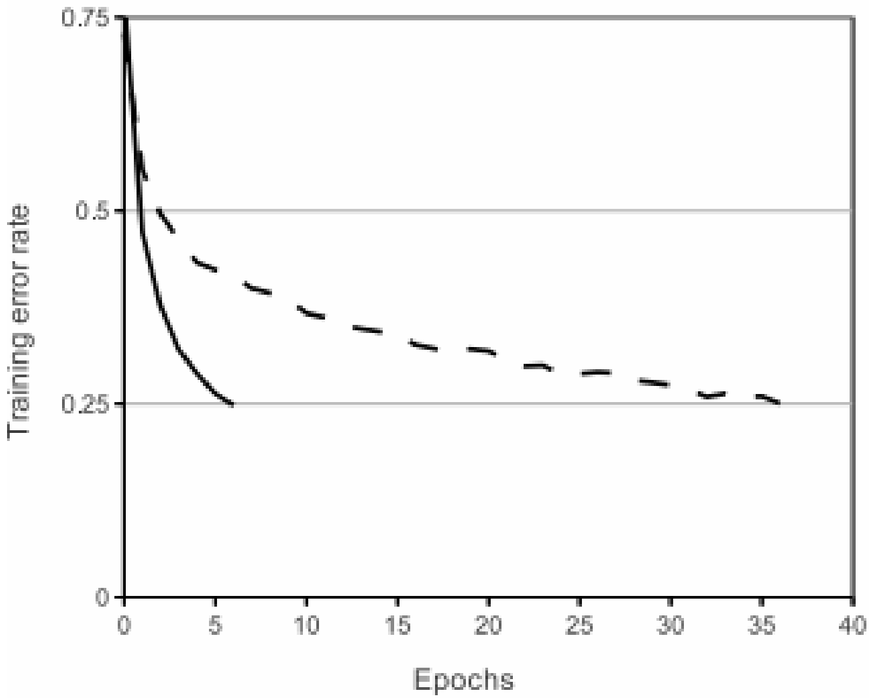
- 1500만 개 이상의 고해상도 이미지
- 2만 2천 개 클래스
- 1000개의 클래스, 각 클래스 당 1000개의 이미지
- 약 120만 개의 training 이미지, 5만 개의 validation, 15만 개의 test 이미지로 구성

- 이미지 크기를 256*256으로 고정.
- -> Fully-connected layer로 입력 크기 고정.
- Resize는 width height중에 짧은 곳을 256으로 변환, 그리고 center부분을 256 으로 cropped
- image의 pixel에서 traning set의 mean을 빼줌.

ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, we down-sampled the images to a fixed resolution of 256×256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image. We did not pre-process the images in any other way, except for subtracting the mean activity over the training set from each pixel. So we trained our network on the (centered) raw RGB values of the pixels.

ReLU



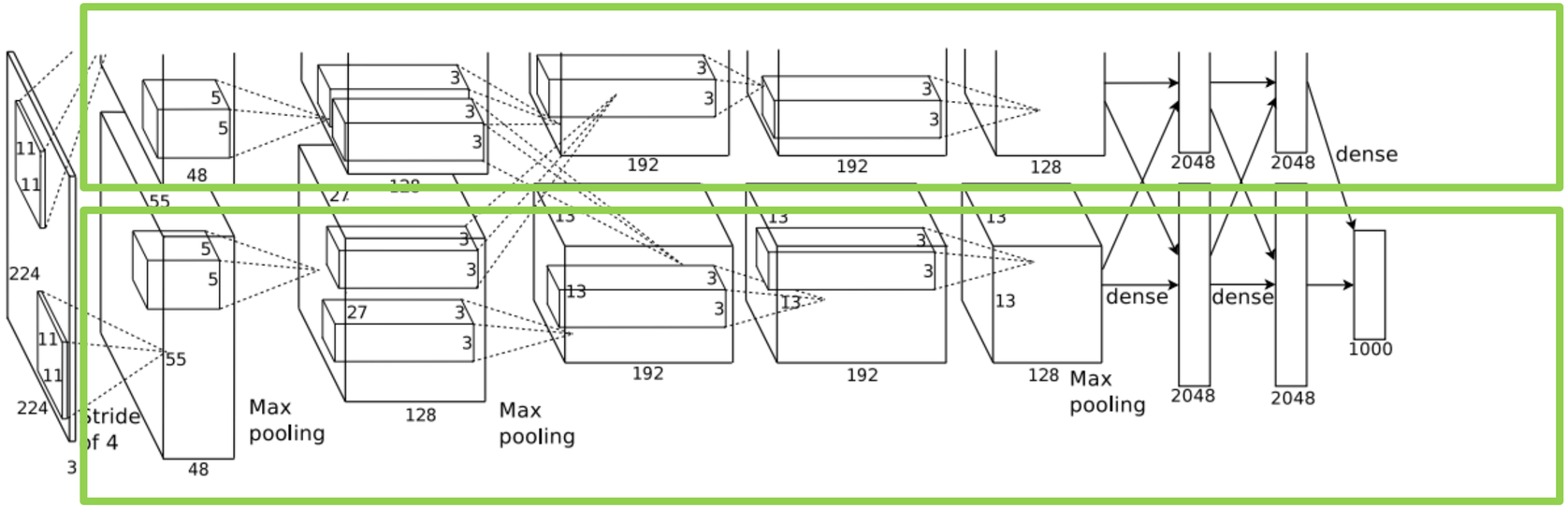


- 보통 Tanh, sigmoid 사용
- 이 모델에선 ReLU 사용
- 약 Quiz.1 배정도 빠르다

Training on Multiple GPUs

GTX 580		512:64:48 (4, 16)	772 (1544)	768	384		1002 (4008)	1.5 3	244	499

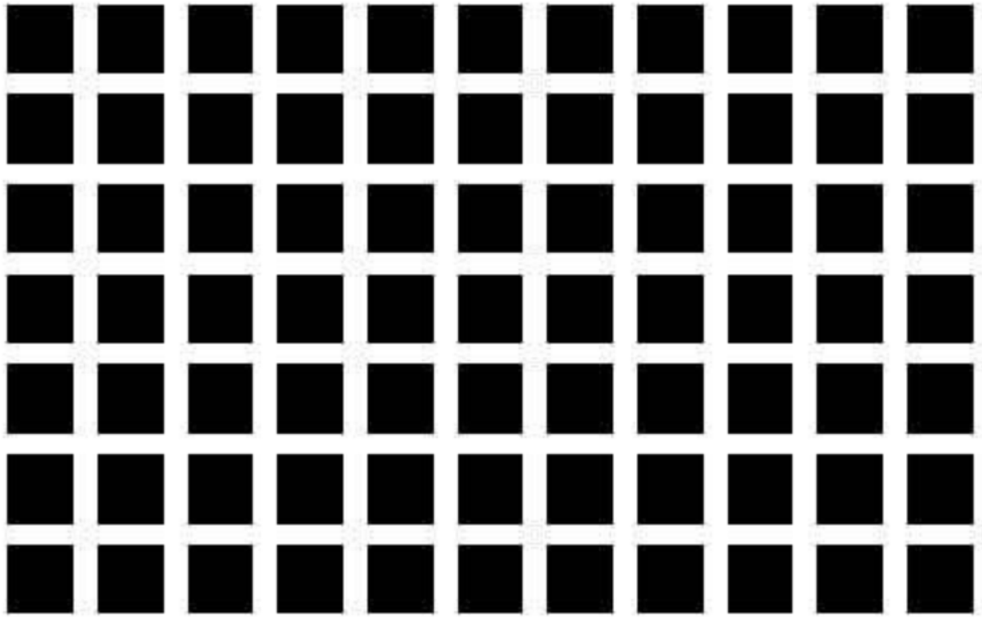
- 메모리 부족 Issue
- Gpu 2대로 병렬사용
- top5 2rror 1.2%, top 1 error 1.7% 절감



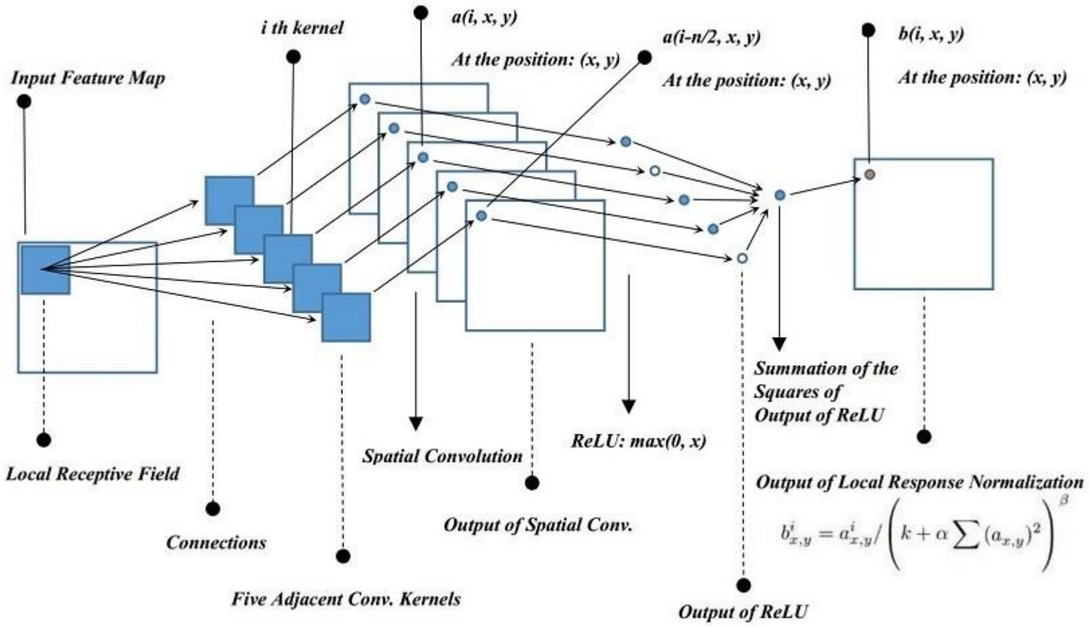
- GPU1
- GPU2

Local Response Normalization

Lateral inhibition
- 강한 뉴런의 활성화가 근처 뉴런의 활동을 억제



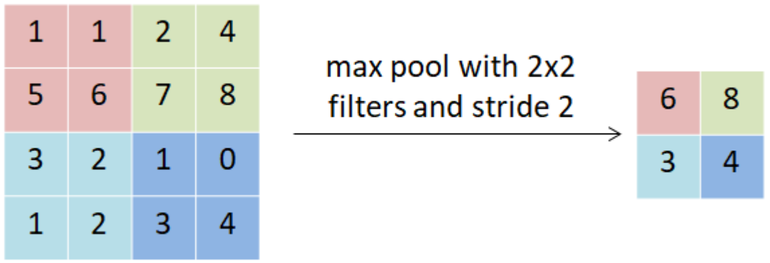
$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$



본 논문에서는, k=2, n=5, α=0.0001, β=0.75로 설정
Top-1 , top-5 error rates 각각 1.4%, 1.2% 감소

Overlapping Pooling

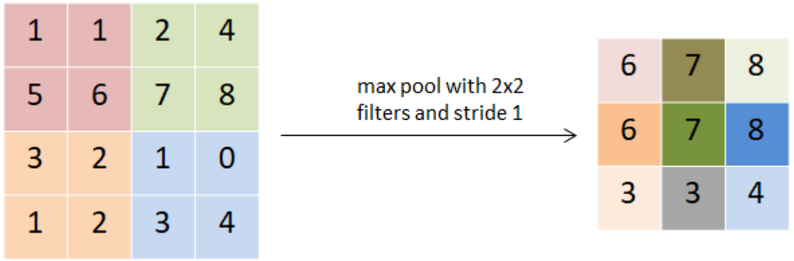
Max Pooling



Pooling window 크기 Quiz.2 stride 크기

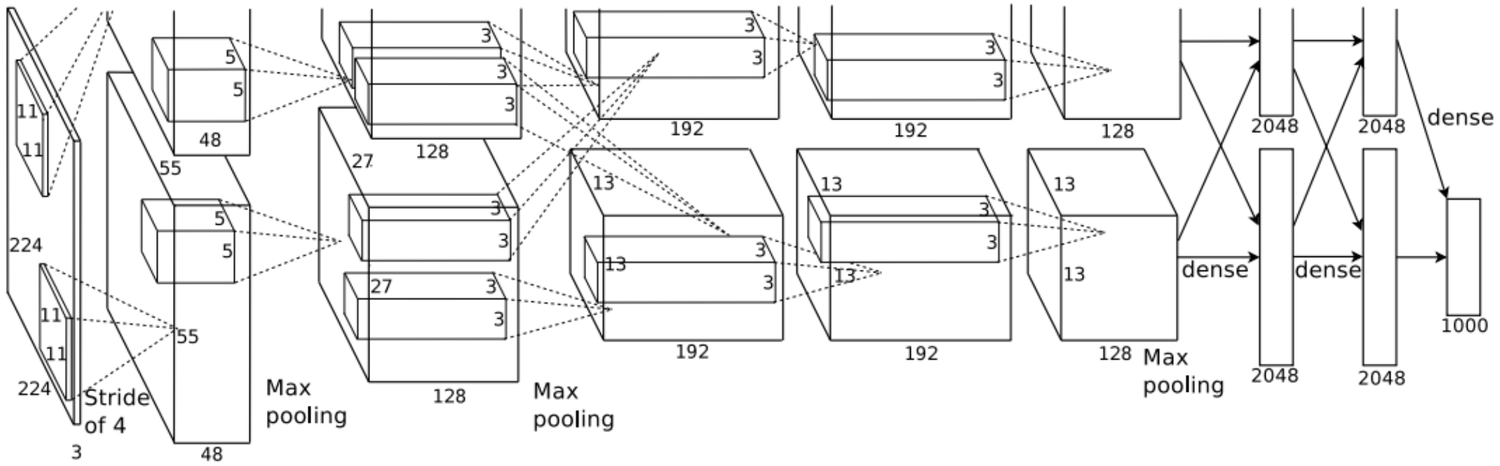
Top-1 , top-5 error rates 각각 0.4%, 0.3% 감소

Overlapping Pooling



Pooling window 크기 Quiz.2 stride 크기

Overall Architecture



$$\frac{I - K + 2P}{S} + 1$$

I : 입력 이미지의 너비(높이)
K : 커널의 너비(높이)
P: 패딩의 크기
S: stride 크기

[INPUT]
227*227*3 Size image

[CONV1]
96 11*11 Filter,
Stride 4, Pad 0
INPUT : 227*227*3
OUTPUT : 55*55*96

[MAX POOL1]
3*3 Filter, Stride 2
INPUT : 55*55*96
OUTPUT : 27*27*96

[NORM1]
LRN 을 이용
INPUT : 27*27*96
OUTPUT : 27*27*96

[CONV2]
256 5*5 Filter,
Stride 1, Pad 2
INPUT : 27*27*96
OUTPUT :
27*27*256

[MAX POOL2]
3*3 Filter, Stride 2
INPUT : 27*27*256
OUTPUT :
13*13*256

[NORM2]
LRN 을 이용
INPUT : 13*13*256
OUTPUT :
13*13*256

[CONV3]
384 3*3 Filter,
Stride 1, Pad 1
INPUT :
13*13*384
OUTPUT :
13*13*384

[CONV4]
384 3*3 Filter,
Stride 1, Pad 1
INPUT :
13*13*384
OUTPUT :
13*13*384

[CONV5]
256 3*3 Filter,
Stride 1, Pad 1
INPUT :
13*13*256
OUTPUT :
13*13*256

[MAX POOL3]
3*3 Filter, Stride 2
INPUT :
13*13*256
OUTPUT :
6*6*256

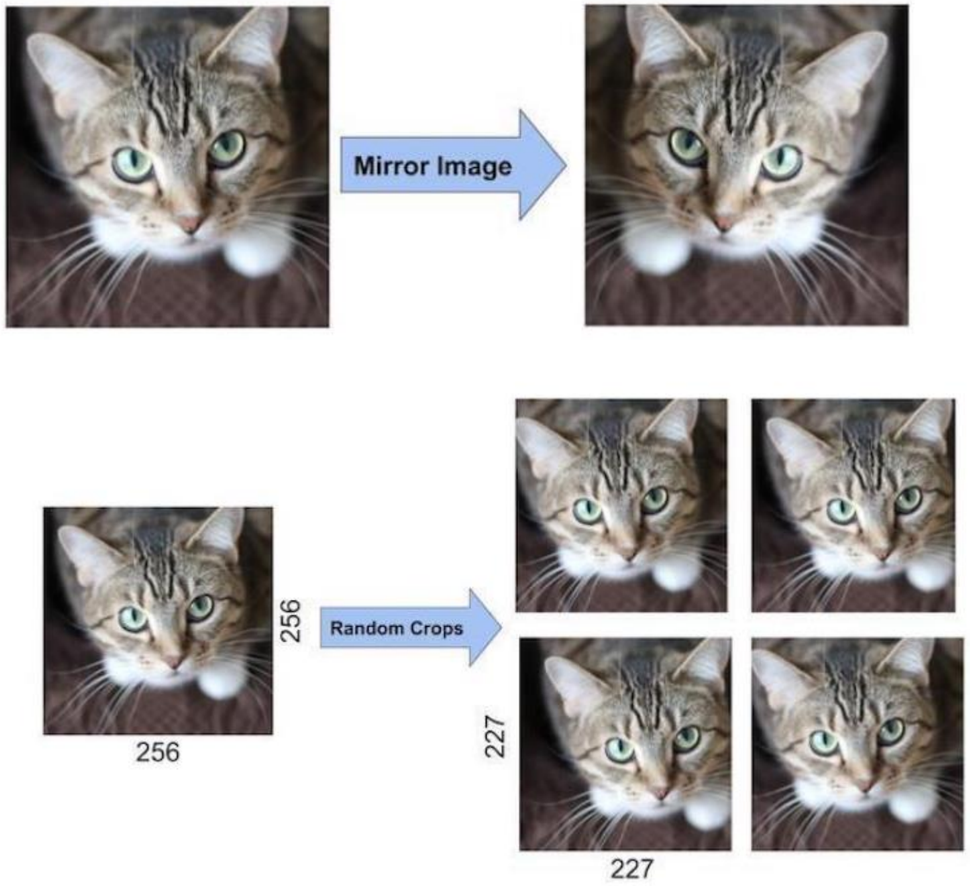
[FC1]
INPUT : 6*6*256
OUTPUT : 4096

[FC2]
INPUT : 4096
OUTPUT : 4096

[FC3]
softmax
INPUT : 4096
OUTPUT : 1000

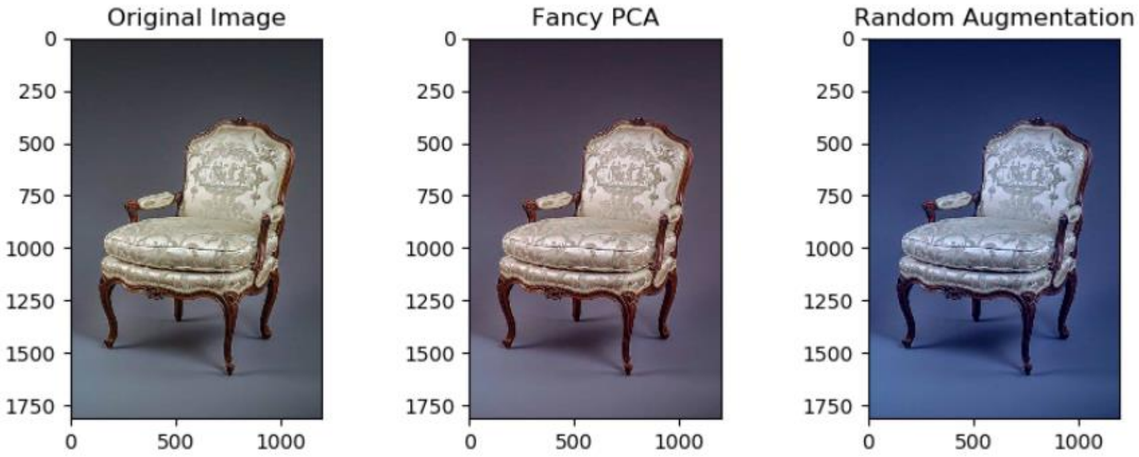
Data Augmentation

Image Translation / horizontal reflection



약 2000배 증가 32 X 32 X 2

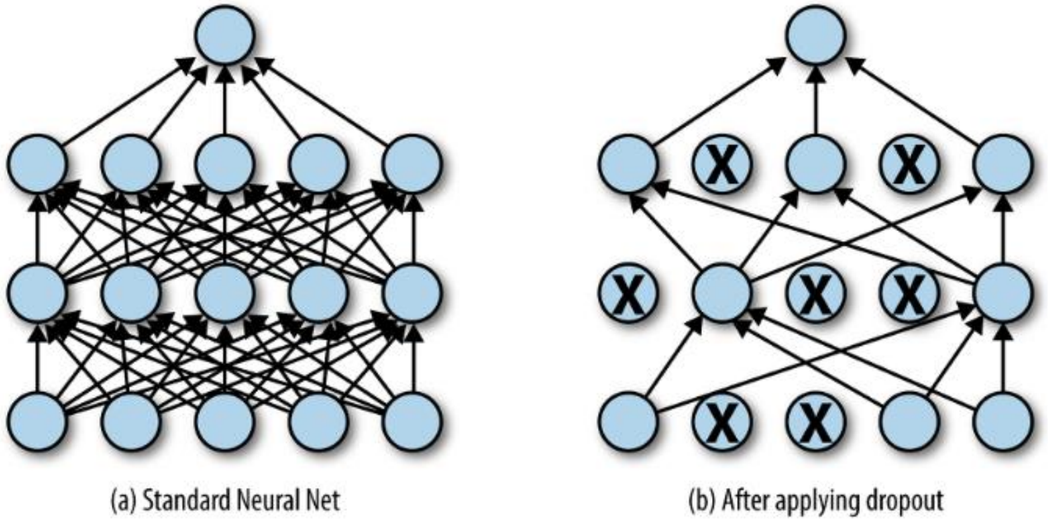
Jittering



RGB 공분산 고유벡터, 고유값 + RGB
픽셀 PCA를 통해 찾은 중요 성분
Top-1 error rates 1% 감소

$$[I_{xy}^R, I_{xy}^G, I_{xy}^B]^T + [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

Dropout



- 0.5 의 확률로 hidden neuron의 값을 0으로 바꿔줌
- Dropout된 hidden neuron은 foward, backpropagation시 영향 X
- 3개 FC 중 앞 2개에만 적용
- Test시 dropout 적용 X, 0.5만 곱해줌

참고자료

[블로그]

https://killerwhale0917.tistory.com/14

https://jjuon.tistory.com/22

https://inha-kim.tistory.com/41

https://medium.com/ai-research-lab-kampala/alexnet-a-brief-review-14979ce7cc84

[유튜브 리뷰]

https://www.youtube.com/watch?v=40Gdctb55BY&pp=ygUHYWxleG5ldA%3D%3D

https://www.youtube.com/watch?v=vRtM4K8e_Q4&pp=ygUHYWxleG5ldA%3D%3D

https://www.youtube.com/watch?v=Nq3auVtvd9Q&t=340s&pp=ugMICgJrbxABGAHKBQdhbGV4bmV0

RESNET

김채연

Deep Residual Learning for Image Recognition 논문

딥러닝에서 neural networks가 깊어질수록 성능은 더 좋지만
train 과정이 어려워짐.
이 논문에서는 잔차를 이용한 잔차학습을 이용해서
깊은 신경망에서도 training이 쉽게 이뤄질 수 있다는 것을 보이고 방법론을 제시함

함수를 새로 만드는 방법 대신에 residual function,
즉 잔차 함수를 learing에 사용하는 것으로 layer를 재구성

결과적으로 152개의 layer를 쌓아서
기존의 모델들보다 좋은 성능을 내면서 복잡성은 줄이는 결과를 보여줌

Degradation Problem

network가 깊어질수록 accuracy가 떨어지는 (성능이 저하되는) 문제

과적합과는 다른 문제

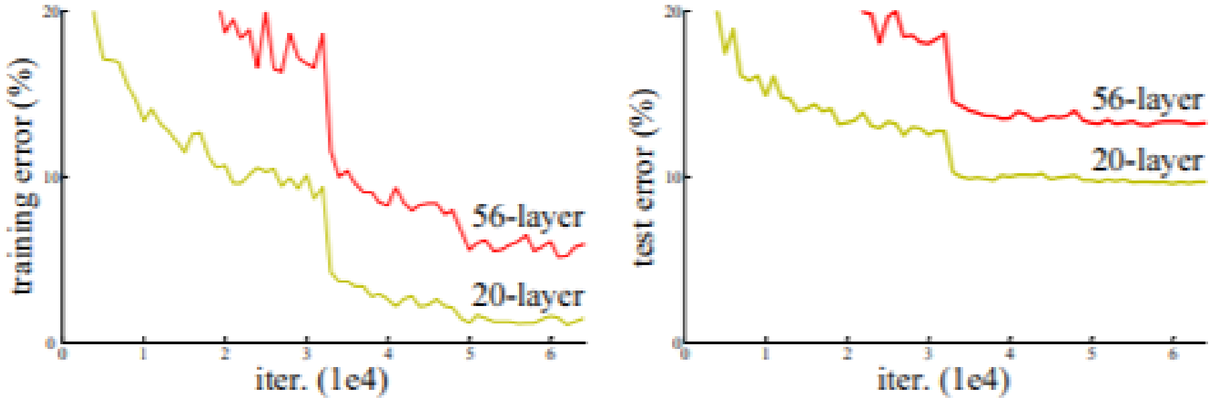


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

quiz: 왜 과적합과는 다른 문제일까?

Degradation Problem

그래서 이 문제를 해결하고자 논문에서는
Deep residual learing framework 라는 개념을 도입.

쌓여진 레이어가 그 다음 레이어에 바로 적합되는 것이 아니라
잔차의 mapping에 적합되도록 함.

Deep Residual Learning

input 을 x,
input이 통과하는 Function 을 F(x),
그리고 Output 을 H(x) 이라고 가정

Residual Learning 에서는 F(x) = H(x) - x 라고 정의함

원래 Output H(x) 에서 자기 자신인 x 를 빼주기 때문에 'Residual Learning' 이라는 이름을 가지게 된다

shortcut connection

ResNet에서는 원래의 목표 함수(H(x))를 직접 학습하는 대신,
입력(x)과 출력(H(x)) 사이의 차이, 즉 잔차(F(x) = H(x) - x)를 학습

이렇게 하면, 네트워크가 학습해야 하는 목표가 "잔차를 0에 가깝게 만드는 것"으로 단순화되며, 이는 학습 과정을 훨씬 용이하게 만들어줌

실제로 많은 경우에서 원본 입력(x)과 최종 출력(H(x))은 크게 다르지 않기 때문에, 이런 접근 방식이 효과적으로 작동함

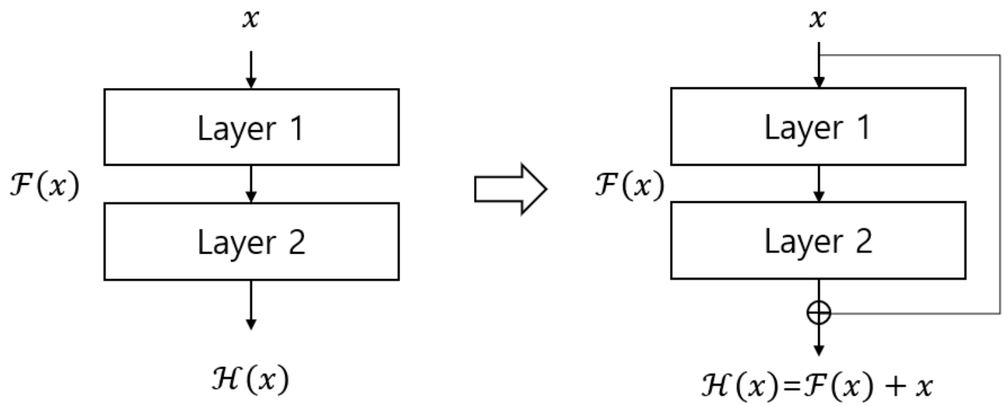
shortcut connection

"Shortcut Connection"이란

입력(x)을 네트워크의 여러 층을 건너뛰어 출력층 근처에 직접 더해주는 구조

즉, F(x) + x의 형태로, 신경망이 잔차 F(x)를 학습한 다음,
입력 x를 더해 최종 목표 함수 H(x)의 근사값을 출력한다.

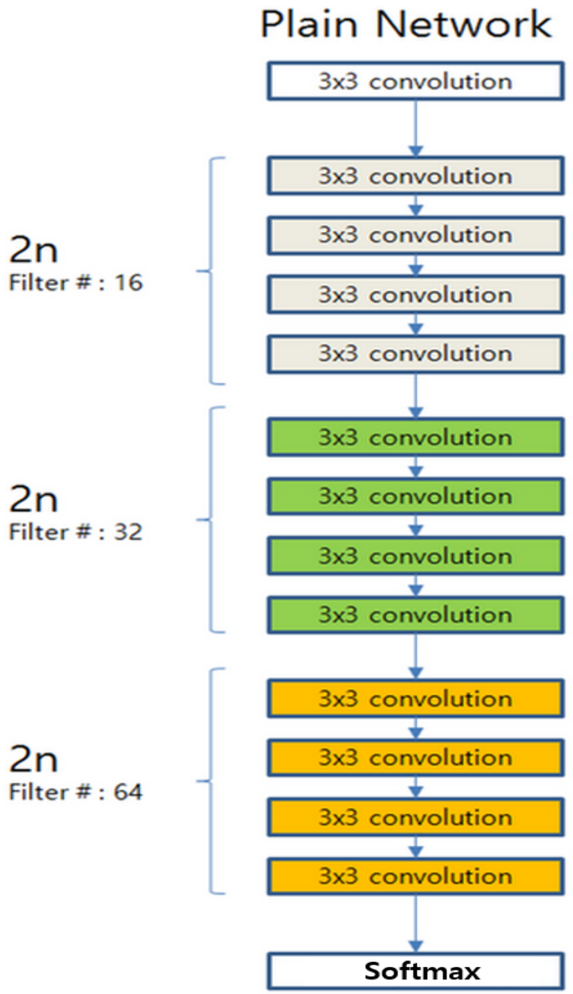
이 구조 덕분에, 네트워크는 더 깊어지더라도 학습이 쉬워지며,
깊은 신경망의 성능을 대폭 향상시킬 수 있다.



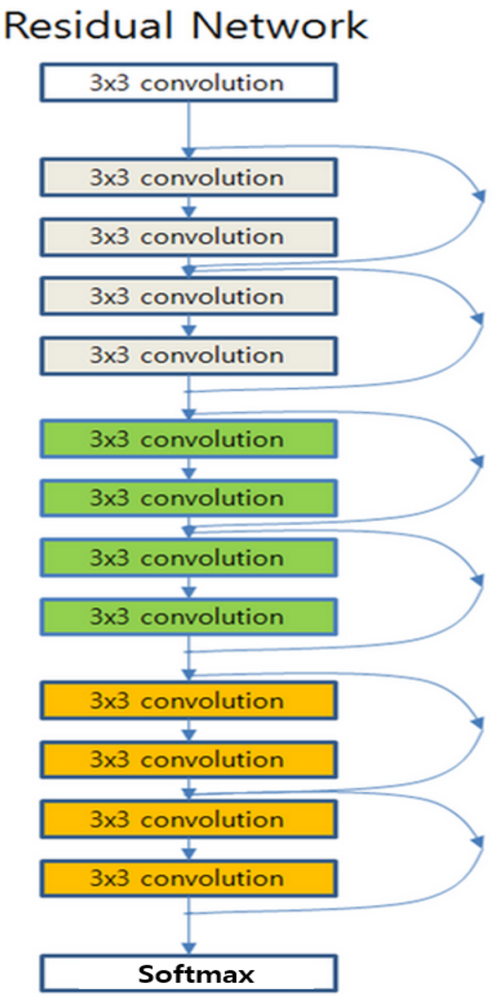
ResNet과 그의 핵심 구성 요소인 Skip Connection은 신경망이 복잡한 함수를 직접 학습하기보다는,
입력과 출력 사이의 작은 "차이" 또는 "잔차"를 학습함으로써 깊은 네트워크의 학습 문제를 해결하는 혁신적인 방법

Architectures (Plain Network , Residual Network)

quiz 2 이 없는 Plain Network



quiz 2 이 있는 Network



experiment

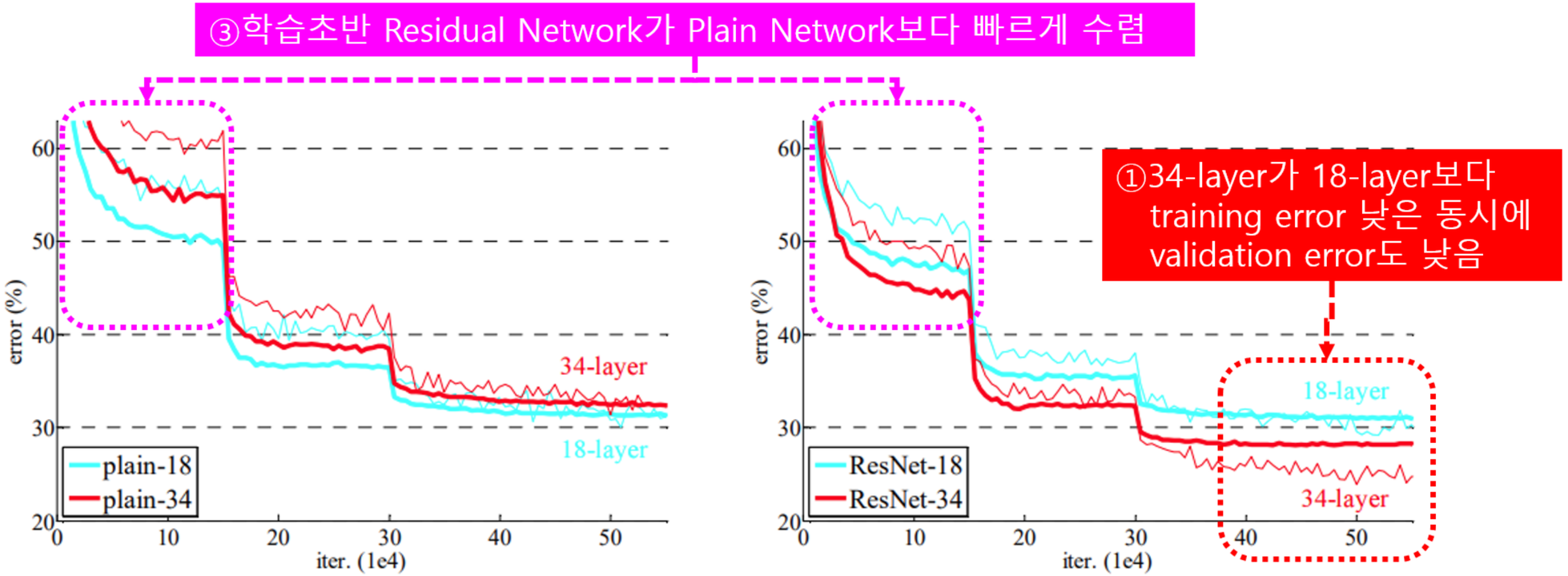


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

experiment

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

①34-layer가 18-layer보다 성능이 좋음

②Plain Network보다 ResNet이 더 성능 좋음

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

experiment

①ResNet에서 34-layer가 18-layer 성능이 좋으며,
18/34 layer인 plain network와 비교했을 때도 더 낮은 training error를 보여줌
이 결과로부터 제안하는 residual learning이 깊은 망에서 training error가 더 큰 degradation 문제를 해결함

②resnet이 같은 구조의 plain network과 비교했을 때
training error를 줄이는 동시에 더 작은 top-1 error를 보여주어 residual learning이 깊은 망에서 효율적임

③resnet에서 학습 시 더 빠르게 수렴
위의 학습그래프에서 학습 초반부를 보면 resnet이 plain network보다 더 가파르게 수렴

Resnet 코드 짜보기

```python
# convolution layer 정의
def conv3x3(in_planes, out_planes, stride=1, groups=1, dilation=1):
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride, padding=dilation,
    groups=groups, bias=False, dilation=dilation)
```

```python
# Basic Block
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

        # 에러문구
        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')

        # 모델
        self.conv1 = conv3x3(inplanes, planes, stride) # resnet은 차원이 바뀌는 블록의 첫번째 conv에서 stride를 이용해 다운샘플링 한다.
        self.bn1 = norm_layer(planes)
        self.relu = nn.ReLU(inplace=True)
        self.con2 = conv3x3(planes, planes)
        self.bn2 = norm_layer(planes)
        self.downsample = downsample
        self.stride = stride
```

Resnet 코드 짜보기

```
def forward(self, x):
    indentity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

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

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

    out += identity # 마지막 relu 전에 identity mapping 해주기!
    out = self.relu(out)

    return out
```

Resnet 코드 짜보기

```python
class Bottleneck(nn.Module):
    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

        # WideResNet, ResNeXt에서 사용
        width = int(planes * (base_width / 64.)) * groups

        # 모델
        # stride가 1이 아닌 경우, self.conv2, self.downsample 층 둘 다 input을 다운샘플
        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
```

```python
def forward(self, x):
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

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

    out = self.conv3(out)
    out = self.bn3(out)

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

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

    return out
```

Resnet 코드 짜보기

```
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__()

        # 파라미터 기본설정
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        self.norm_layer = norm_layer
        self.inplanes = 64 # input 기본값
        self.dilation = 1

        if replace_stride_with_dilation is None:
            replace_stride_with_dilation = [False, False, False]
        if len(replace_stride_with_dilation) != 3:
            raise ValueError(f'replace_stride_with_dilation should be None or a 3 element tuple, got {replace_stride_with_dilation}')

        self.groups = groups
        self.base_width = width_per_group
```

Resnet 코드 짜보기

```
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)

# layer1~4에서 block의 종류와 갯수로 resenet18, resnet50 등 모델의 차이가 생긴다.
# 파라미터 layers[0]등은 아래 _make_layer에서 blocks로 받은 값 : block의 개수
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])

# 모든 resnet 공통부분, 모든 block이 끝나면 수행
self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(512 * block.expansion, num_classes)
# ====================================

# 모델 초기화
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)

# 각 residual branch의 마지막 BN에 zero-initialize를 해주어 residual branch는 zero로 시작하여 각 residual block이 identity처럼 작동하도록 함
if zero_init_residual:
    for m in self.moduels():
        if isinstance(m, Bottleneck):
            nn.init.constant_(m.bn3.weight, 0)
        elif isinstance(m, BasicBlock):
            nn.init.constant_(m.bn2.weight, 0)
```

Resnet 코드 짜보기

```python
# block 내부에 layer를 만드는 함수
# block : 블록종류(basic, bottleneck)
# blocks : 블록 내 layer 개수 (이 파라미터는 resnet18의 경우 [2, 2, 2, 2]로 입력받음)
def _make_layer(self, block, planes, blocks, stride=1, dilate=False):
    norm_layer = self._norm_layer
    downsample = None
    previous_dilation = self.dilation

    if dilation:
        self.dilation *= stride
        stride = 1

    # strid가 1이 아니거나 입출력차원이 맞지 않는경우 downsample 진행
    if stride != 1 or self.inplanes != planes*block.expansion:
        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

    # 입력된 blocks만큼 block반복
    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)
```

Resnet 코드 짜보기

```
def _forward_impl(self, x):

    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)
    x = self.fc(x)
    return x

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

# resnet18, resnet50 만들기
def _resnet(arch, block, layers, pretrained, progress, **kwargs):
    model = ResNet(block, layers, **kwargs)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch], progress=progress)
        model.load_state_dict(state_dict)
    return model

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)
```

참고자료

https://velog.io/@woojinn8/CNN-Networks-4.-ResNet
https://jays0606.tistory.com/3
https://tobigs.gitbook.io/tobigs/deep-learning/computer-vision/resnet-deep-residual-learning-for-image-recognition

https://blog.kubwa.co.kr/%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%B7%B0-deep-residual-learning-for-image-recognition-2015-%EA%B0%84%EB%8B%A8%ED%95%9C-%EC%8B%A4%EC%8A%B5%EC%BD%94%EB%93%9C-a55cb68981b1