CNN 주요 모델을 활용한 COVID-19 X선 이미지 분류

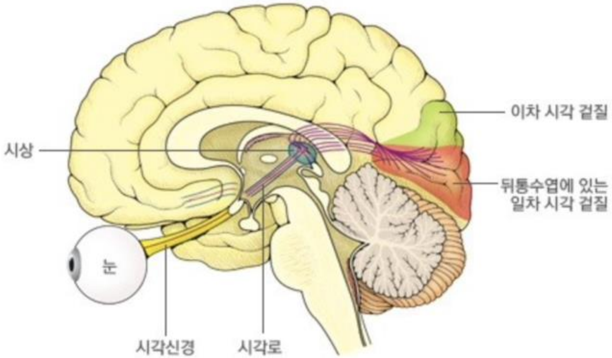
212STG13 박지원
212STG18 예지혜
218STG01 김지민

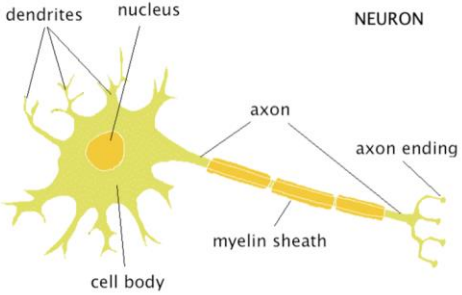
목차

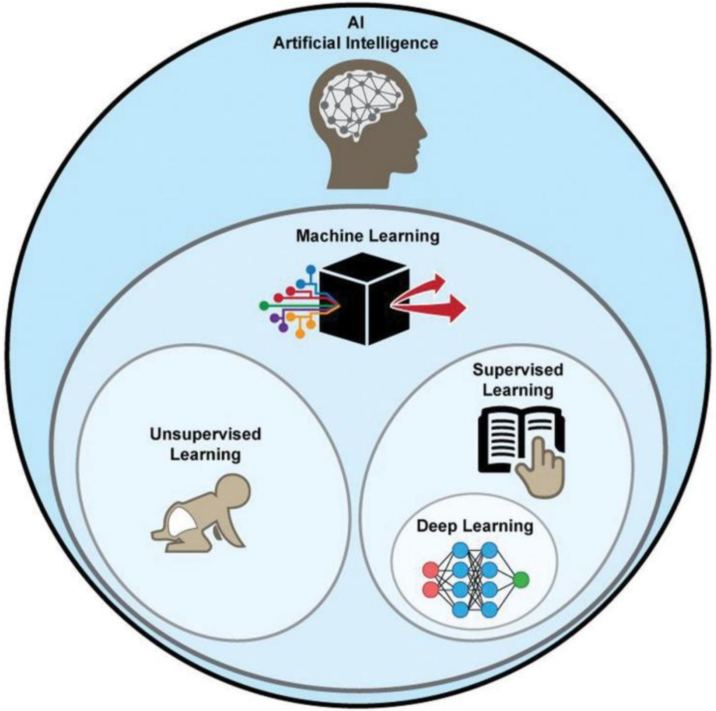
CNN 주요 개념 소개

Part 1. 연구 배경

이미지 분류와 기계학습 (machine learning)

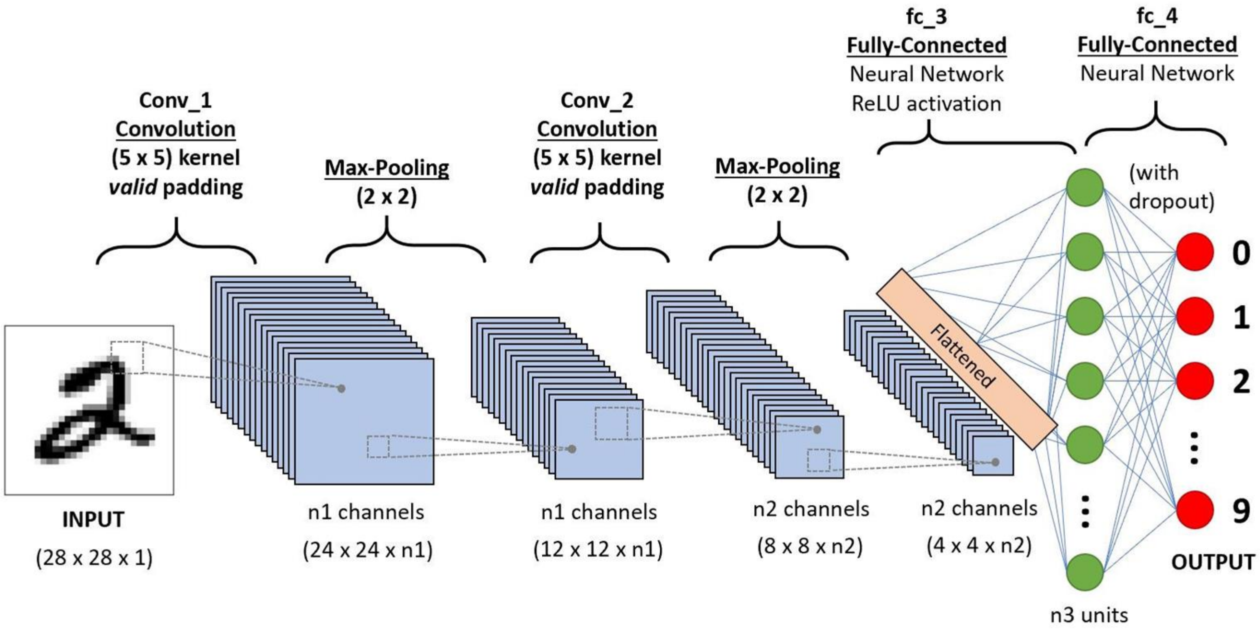






CNN이란?

convolutional neural network : 합성곱 신경망

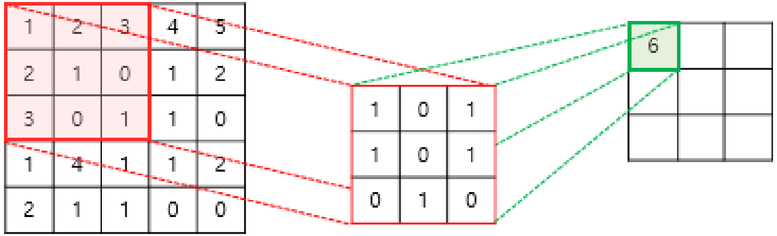


CNN의 특징

✓ Locality
(Local Connectivity)

✓ Shard Weights

CNN이란?

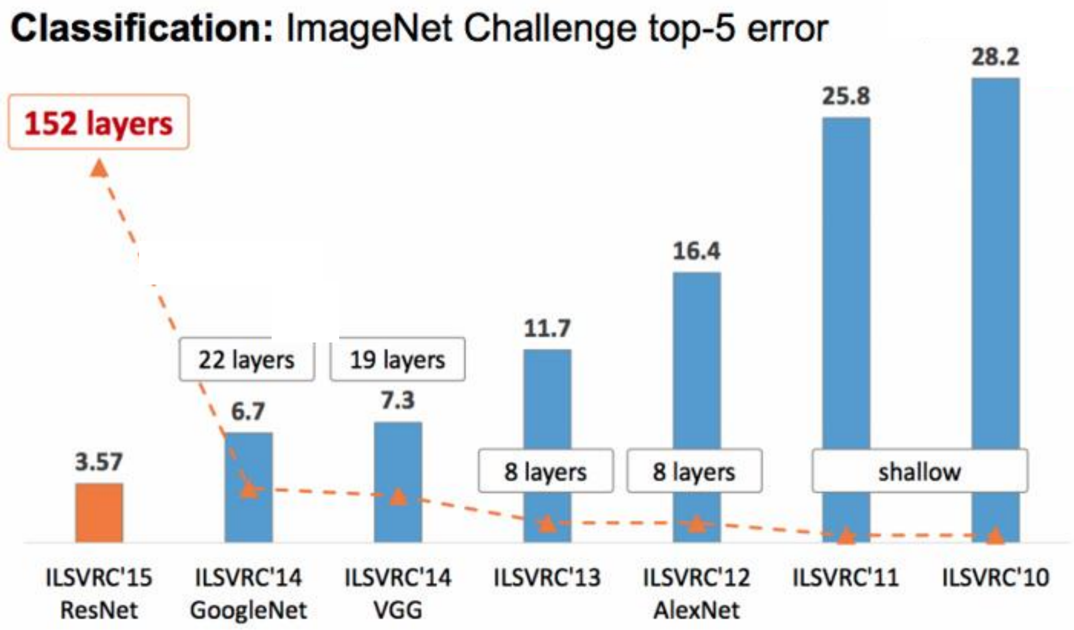


입력 커널 출력

Original 3x3 low-pass 3x3 Laplace 3x3 high-pass

ILSVRC (Imagenet Large Scale Visual Recognition Challenge)

IMAGENET



LeNet-5

↓

AlexNet

↓

VGG

↓

GoogLeNet

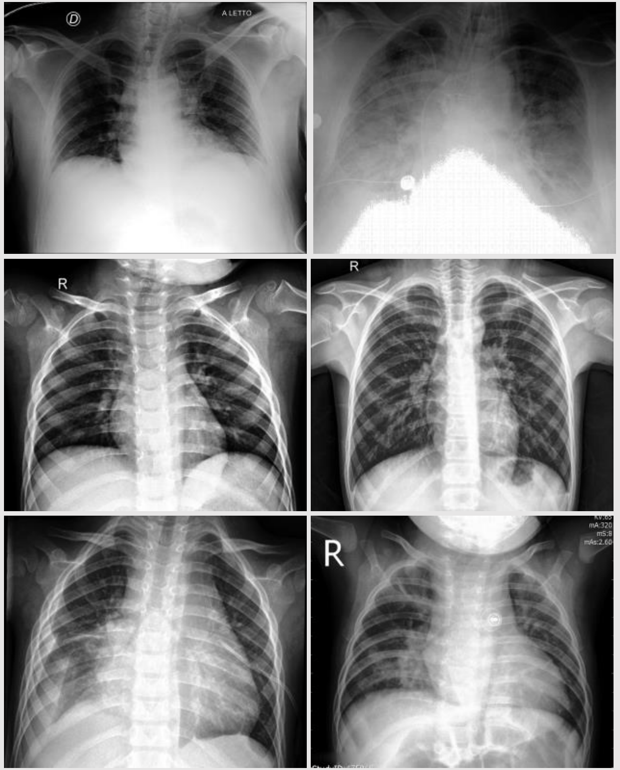
↓

ResNet

데이터 소개

Part 1. 연구 배경

Covid19 Image Dataset



코로나

정상

폐렴

데이터 소개

코로나 19 확진자, 정상, 폐렴 환자의
흉부 X선 사진 데이터

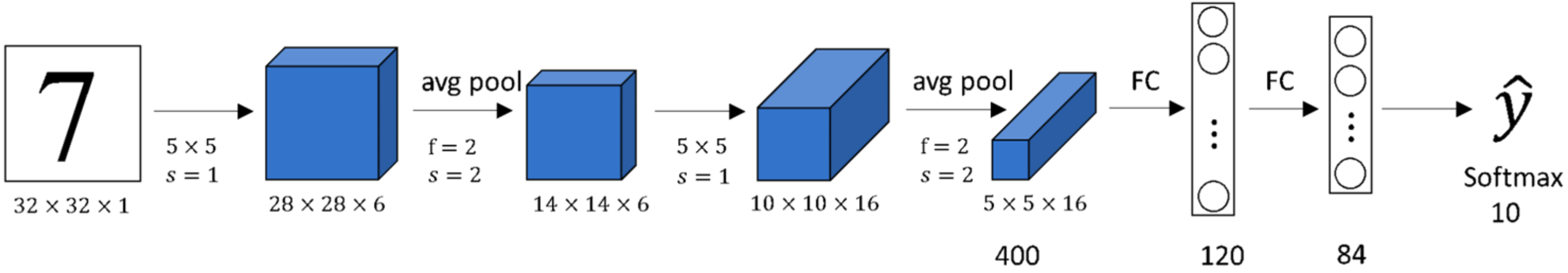
데이터 크기

	코로나	정상	폐렴	합계
train	111	70	70	251
test	26	20	20	66

AlexNet

Part 2. CNN 주요 모델 소개

LeNet

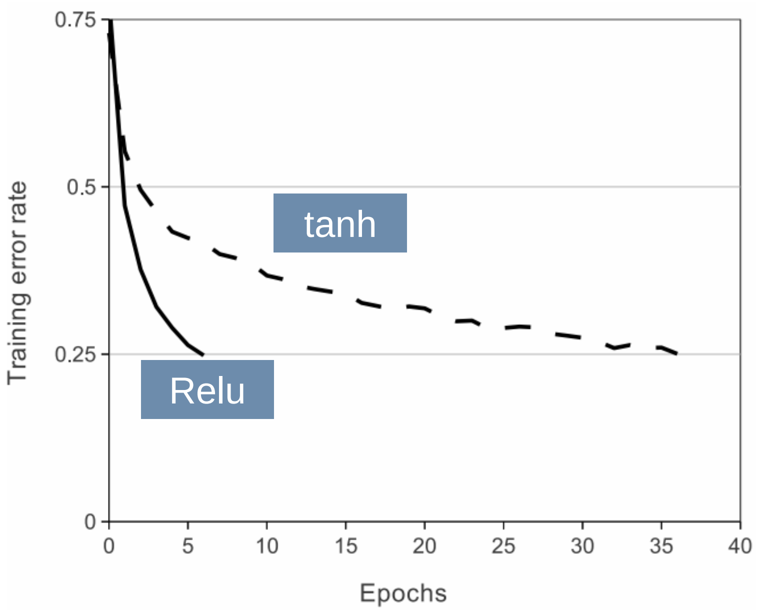


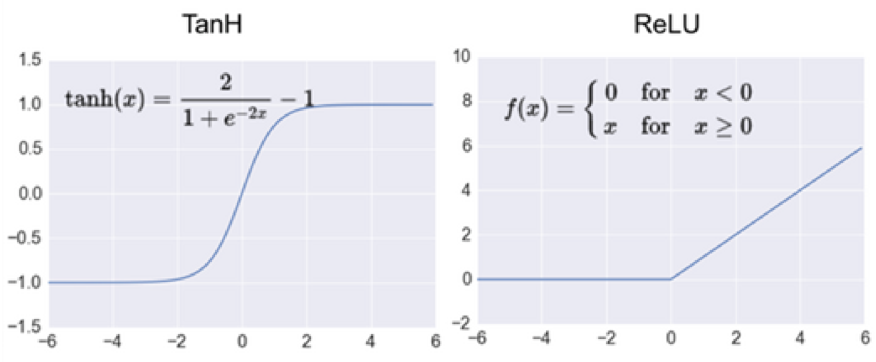
Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

AlexNet

논문	Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. **Imagenet classification with deep convolutional neural networks.** In *Advances in Neural Information Processing Systems 25*, p1106–1114, 2012.
주요 개념	- 활성화 함수로 ReLu 사용 - 두 개의 GPU를 사용하여 계산속도 향상

AlexNet



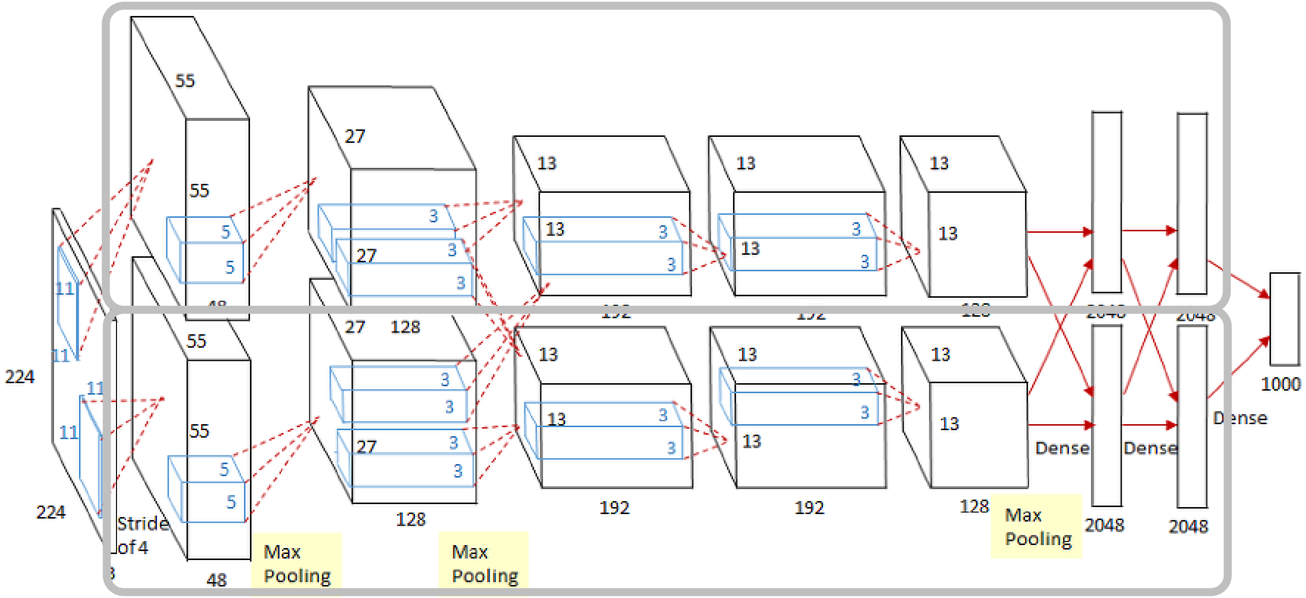


그래디언트 수렴 속도 빠름

AlexNet 구조

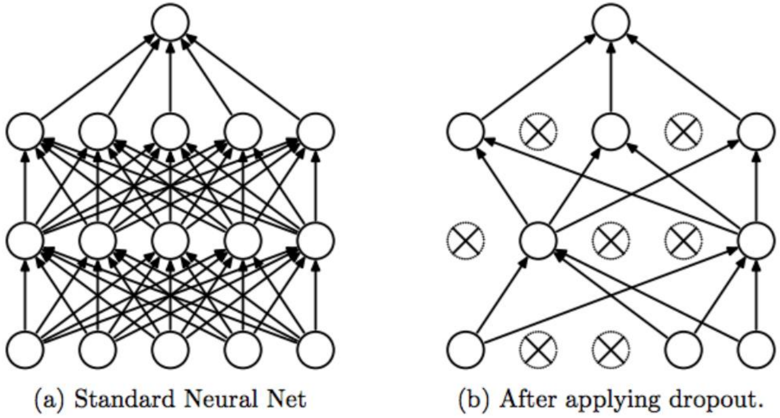
GPU 1

GPU 2

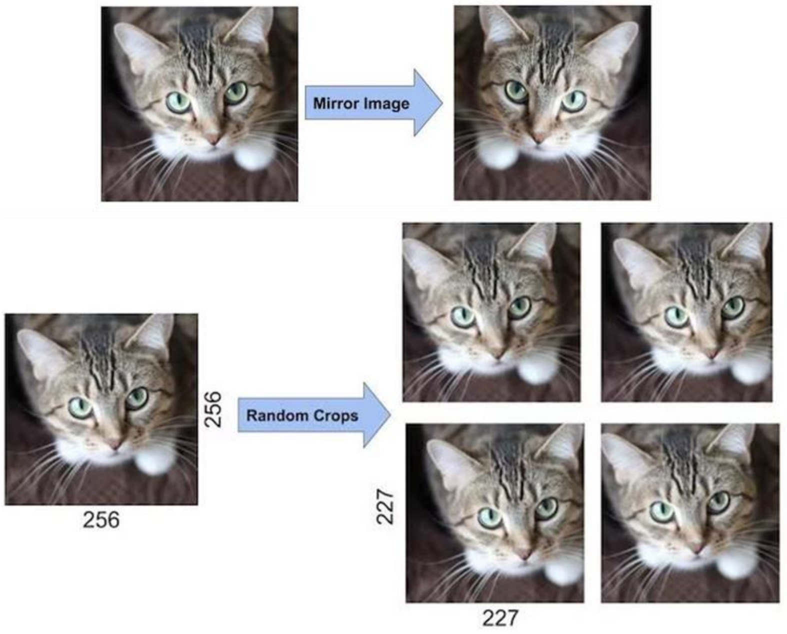


AlexNet - 과적합 방지

drop out



data augmentation



AlexNet - 코드

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=3):
        super(AlexNet,self).__init__()
        # input size : (b x 3 x 227 x 227)
        # 논문에는 image 크기가 224 pixel이라고 나와 있지만, 오타입니다.
        # 227x227을 사용합니다.

        # Conv layer
        self.net = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0), # (b x
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75, k=2),
            nn.MaxPool2d(kernel_size=3, stride=2), # (b x 96 x 27 x 27)

            nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2), # (b x
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(size=5, alpha=0.0001, beta=0.75, k=2),
            nn.MaxPool2d(kernel_size=3, stride=2), # (b x 256 x 13 x 13)

            nn.Conv2d(256, 384, 3, 1, 1), # (b x 384 x 13 x 13)
            nn.ReLU(inplace=True),

            nn.Conv2d(384, 384, 3, 1, 1), # (b x 384 x 13 x 13)
            nn.ReLU(inplace=True),

            nn.Conv2d(384, 256, 3, 1, 1), # (b x 256 x 13 x 13)
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, 2), # (b x 256 x 6 x 6)
        )
```

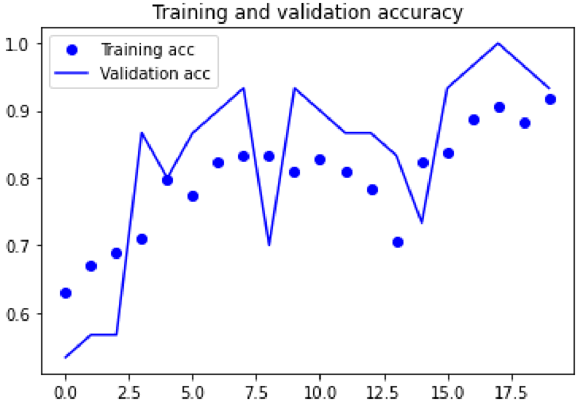
Relu

```
        # fc layer
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=(256 * 6 * 6), out_features=4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5, inplace=False),
            nn.Linear(in_features=4096, out_features=4096),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=4096, out_features=num_classes),
        )
```

drop out

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Alexnet 분석 결과

	AlexNet
train & valid accuracy	
학습 시간	4.71 min
test accuracy	80.3%

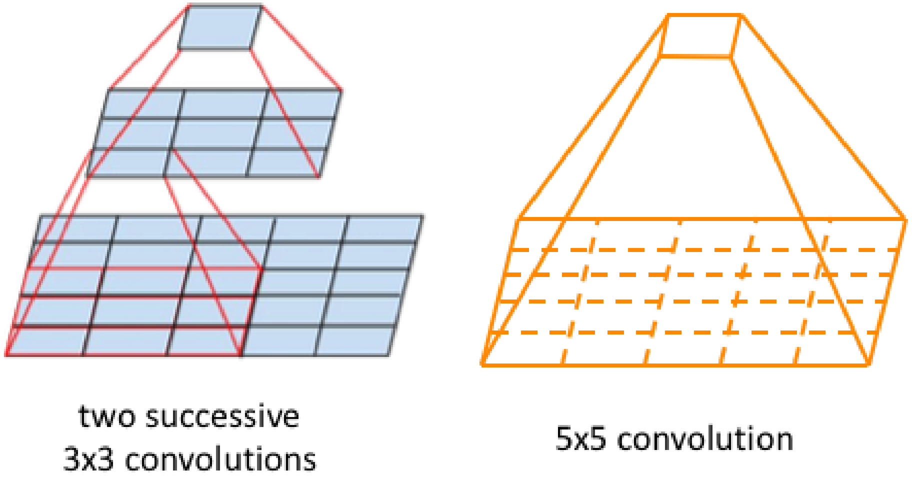
VGG

Part 2. CNN 주요 모델 소개

VGG (Visual Geometry Group Net)

논문	K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
배경	기존의 alexnet의 네트워크를 더 깊게 만들자.
주요 개념	conv - pooling 반복에서 conv의 개수를 늘림 very small (3x3) convolution filters

VGG



	3x3	5x5
파라미터	$2x(3^2C^2) = 18C^2$	$5^2C^2 = 25C^2$
층의 개수	2 layer	1 layer

- ✔ 층을 깊게하여 비선형성 추가
- ✔ 파라미터 감소

VGG

VGG11		VGG13		VGG16	VGG19
			onfiguration		
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

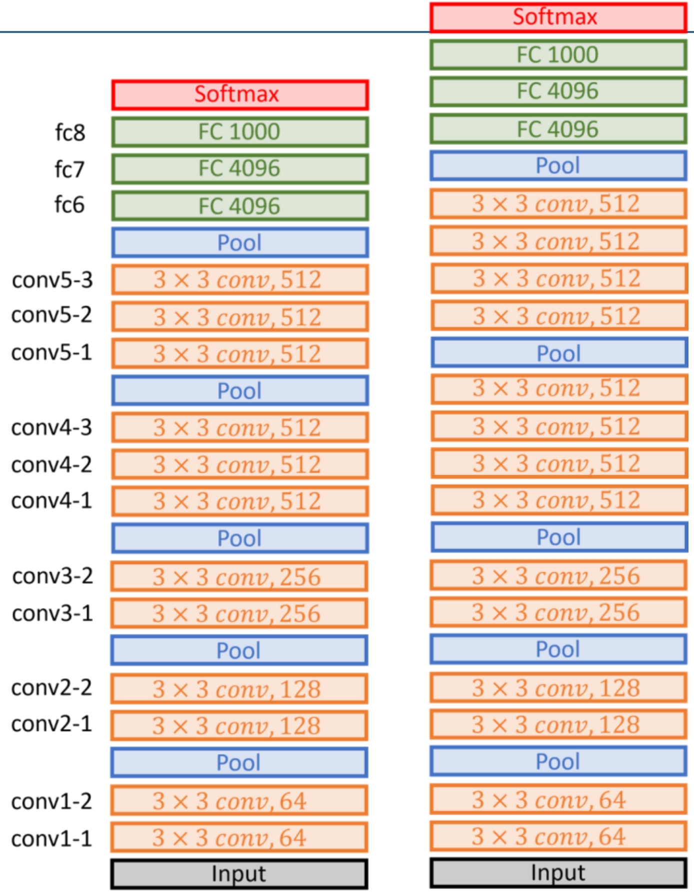
Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)		top-5 val. error (%)
	train (S)	test (Q)			
A	256	256	29.6		10.4
A-LRN	256	256	29.7		10.5
B	256	256	28.7	B	9.9
C	256	256	28.1	C	9.4
	384	384	28.1		9.3
	[256;512]	384	27.3		8.8
D	256	256	27.0	D	8.8
	384	384	26.8		8.7
	[256;512]	384	25.6		8.1
E	256	256	27.3		9.0
	384	384	26.9		8.7
	[256;512]	384	**25.5**		**8.0**

B → C : conv 1x1 추가 (28.7 → 28.1)
B → D : conv 3x3 추가 (28.7 → 27.0)

VGG 구조

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



VGG16 **VGG19**

VGG 코드

```python
# VGG type dict
# int : output chnnels after conv layer
# 'M' : max pooling layer
VGG_types = {
    'VGG11' : [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512,'M'],
    'VGG13' : [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512,'M'],
    'VGG16' : [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512,'M'],
    'VGG19' : [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512,'M']
}
```

숫자는 conv 층   **M은 pooling 층**

```python
def create_conv_laters(self, architecture):
    layers = []
    in_channels = self.in_channels # 3

    for x in architecture:
        if type(x) == int: # int means conv layer
            out_channels = x

            layers += [nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                                 kernel_size=(3,3), stride=(1,1), padding=(1,1)),
                       nn.BatchNorm2d(x),
                       nn.ReLU()]
            in_channels = x
        elif x == 'M':
            layers += [nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))]

    return nn.Sequential(*layers)
```

숫자 → conv 3x3

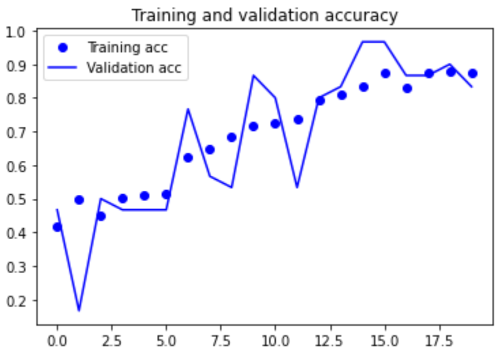
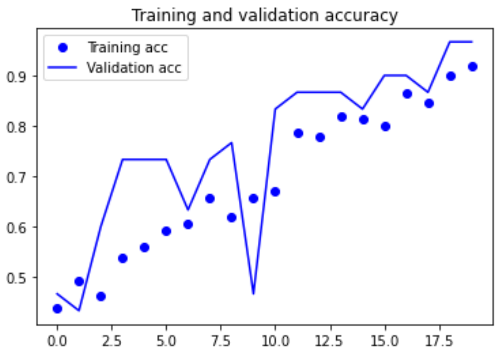
'M' → pooling

```python
:
model, in_channels=3, num_classes=3, init_weights=True):
lf).__init__()
s = in_channels

    # create conv_layers corresponding to VGG type
    self.conv_layers = self.create_conv_laters(VGG_types[model])

    self.fcs = nn.Sequential(
        nn.Linear(512 * 7 * 7, 4096),
        nn.ReLU(),
        nn.Dropout(),
        nn.Linear(4096, 4096),
        nn.ReLU(),
        nn.Dropout(),
        nn.Linear(4096, num_classes),
    )

    # weight initialization
    if init_weights:
        self._initialize_weights()

def forward(self, x):
    x = self.conv_layers(x)
    x = x.view(-1, 512 * 7 * 7)
    x = self.fcs(x)
    return x
```

필요한 층을 계속 쌓음

Fully Connected 층 (FC layer)

VGG 분석 결과

	VGG 16	VGG 19
train & valid accuracy		
학습 시간	5.51 min	5.61 min
test accuracy	78.8%	83.3%

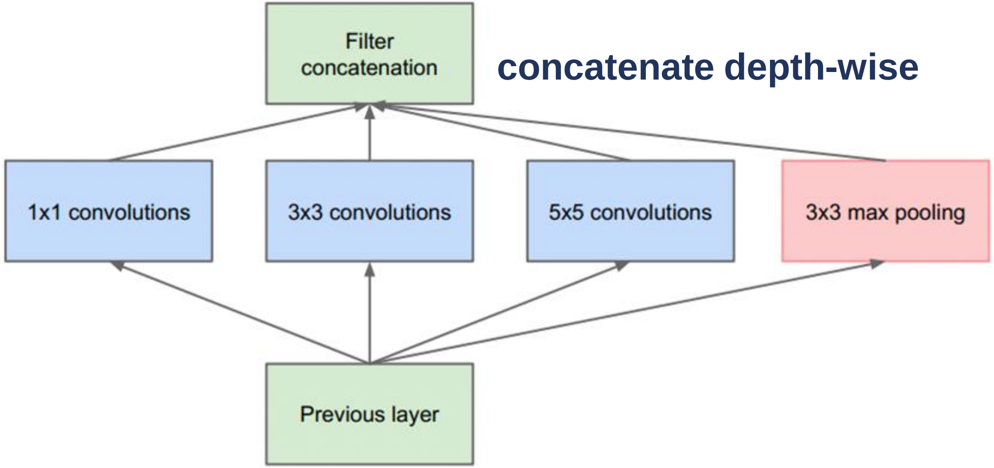
GoogLeNet

Part 2. CNN 주요 모델 소개

GoogLeNet

논문	C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. **Going deeper with convolutions**. In *CVPR*, 2014
배경	깊고(deep) 넓은(wide) 네트워크 → 성능 향상 but computationally expensive => Deeper networks, **with computational efficiency**
주요 개념	- Inception Module - Auxiliary classifier

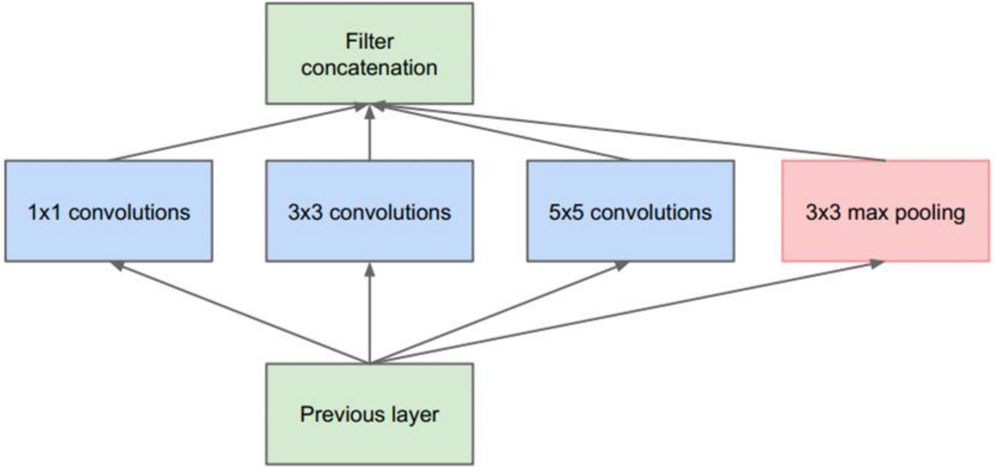
GoogLeNet - Inception Module



Inception Module

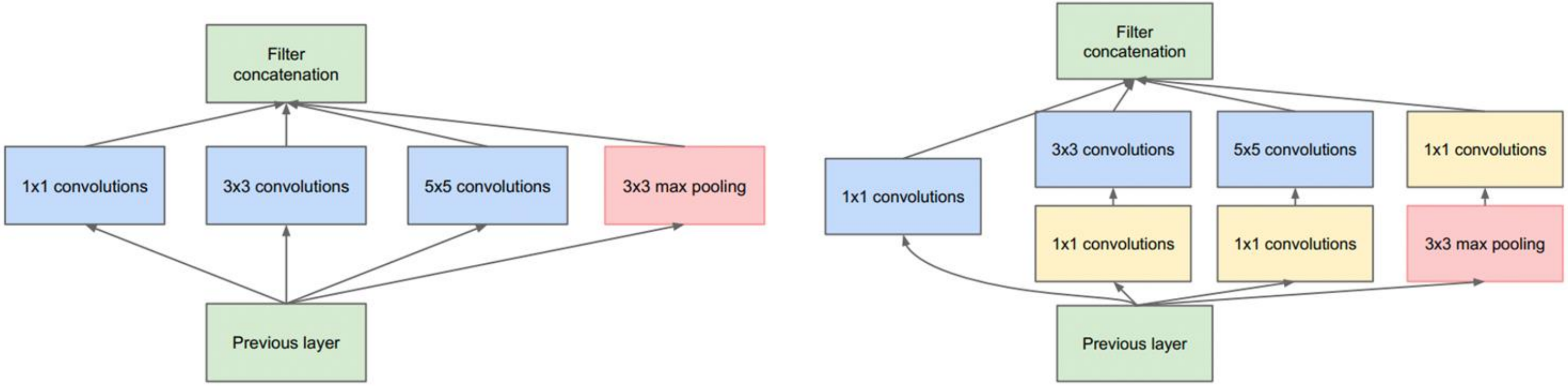
- 여러 size의 conv layer를 병렬적으로 연결한 형태
- 시각적 정보를 다양한 규모 (1x1, 3x3, 5x5)로 처리 => 다양한 특징 추출

GoogLeNet - Inception Module



Problem? computational complexity

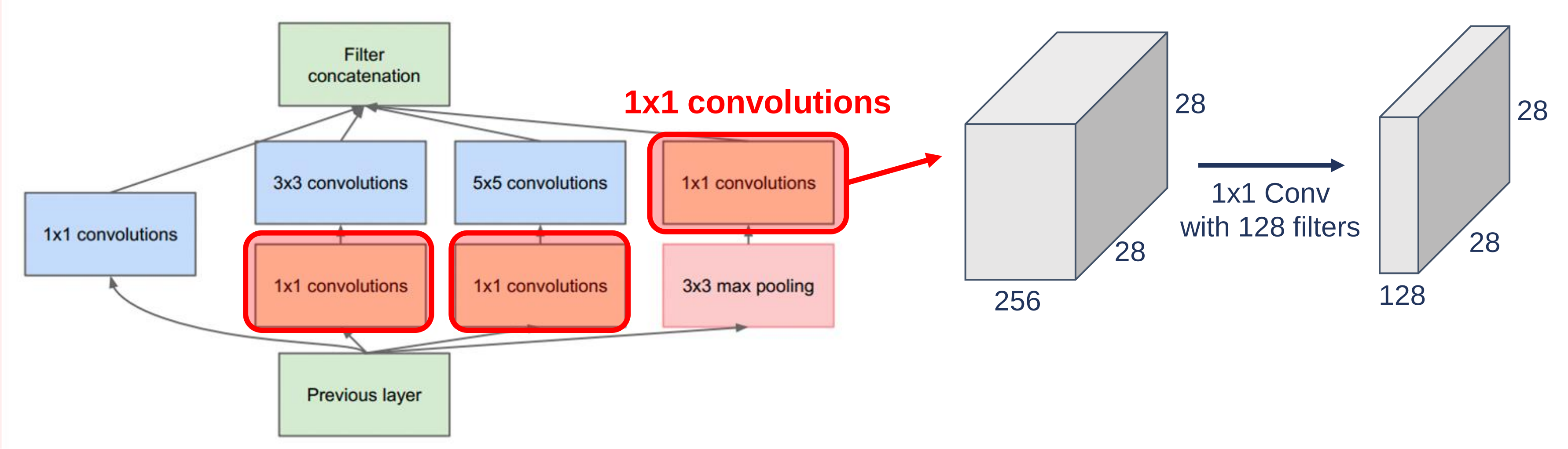
GoogLeNet - Inception Module



(a) Inception module, naïve version (b) Inception module with dimension reductions

Solution: “bottleneck” layers
=> dimension reduction by 1x1 conv

GoogLeNet - Inception Module



Inception Module with dimension reduction

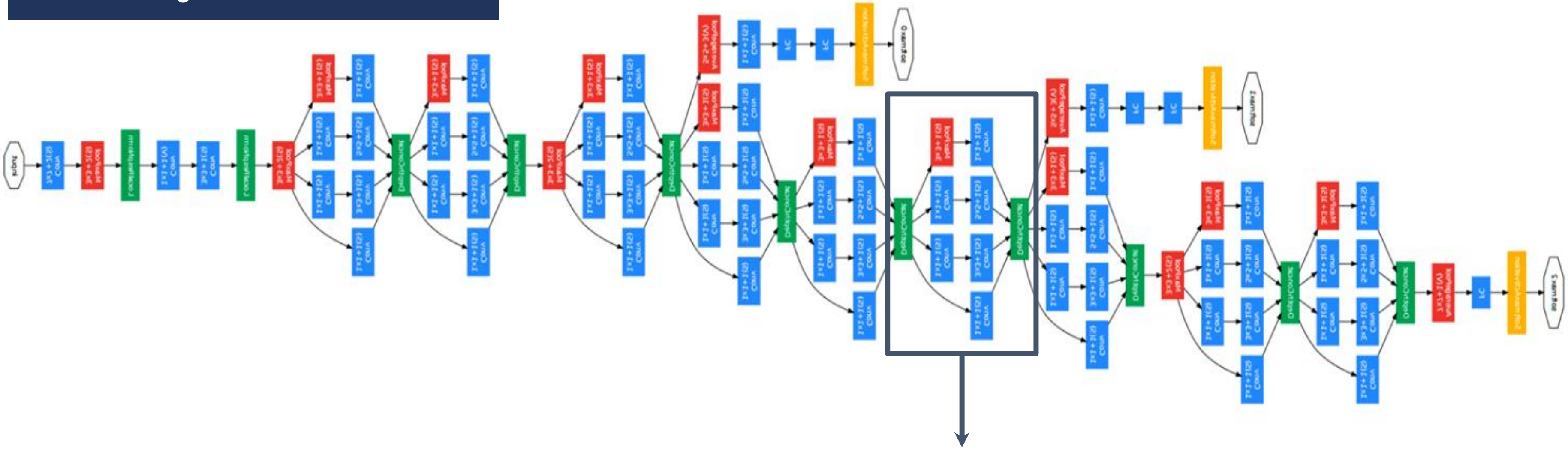
1x1 convolutions:

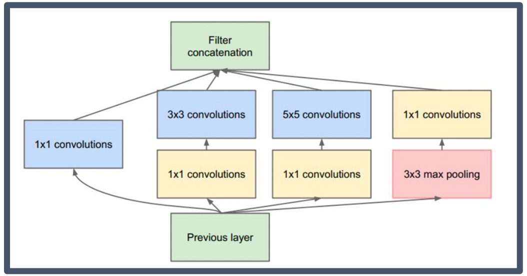
=> preserves spatial dimensions, reduces depth!

=> Projects depth to lower dimension

GoogLeNet - Full Architecture

Full GoogLeNet architecture

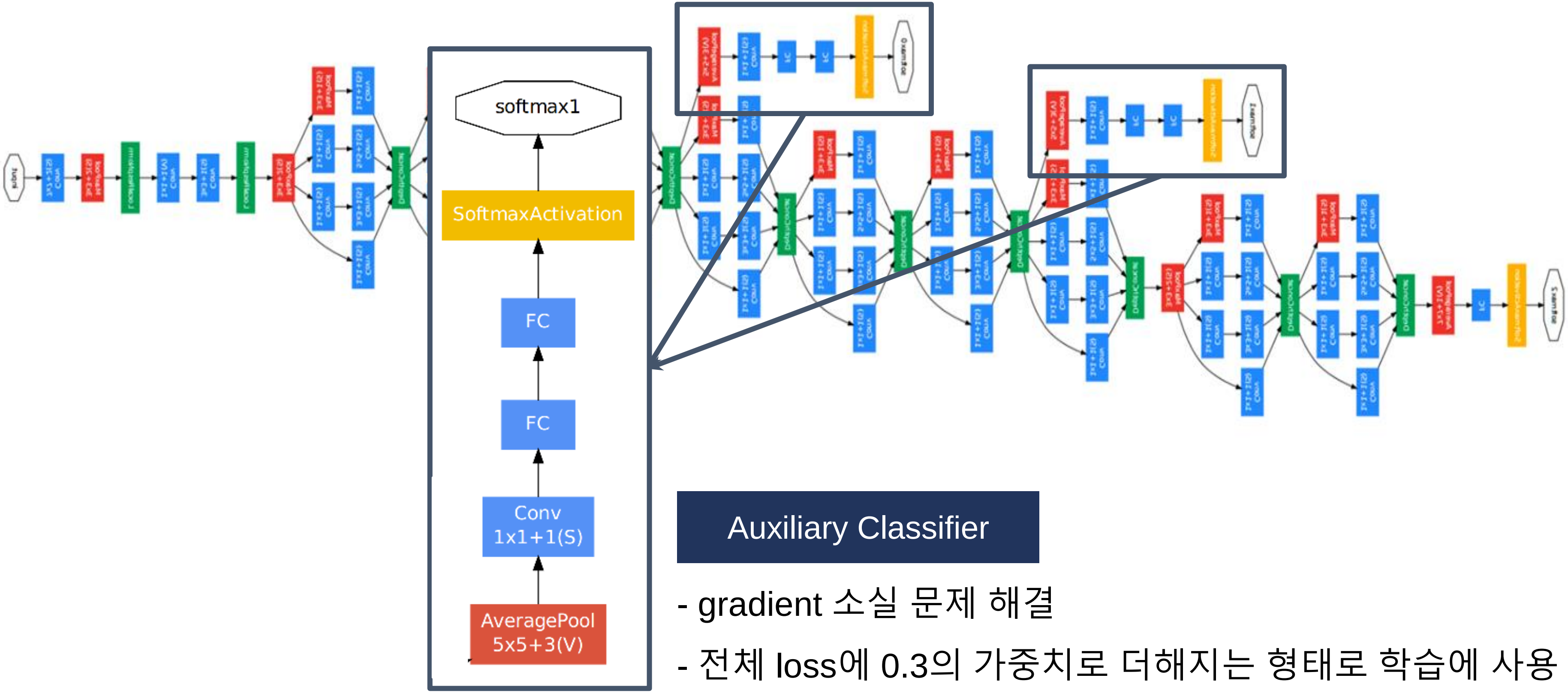




inception module의 효과

1. network size↑ 계산 복잡도↓
2. 다양한 scale의 특징 추출

GoogLeNet - Auxiliary classifier



Auxiliary Classifier

- gradient 소실 문제 해결
- 전체 loss에 0.3의 가중치로 더해지는 형태로 학습에 사용

GoogLeNet - 코드

Inception Block

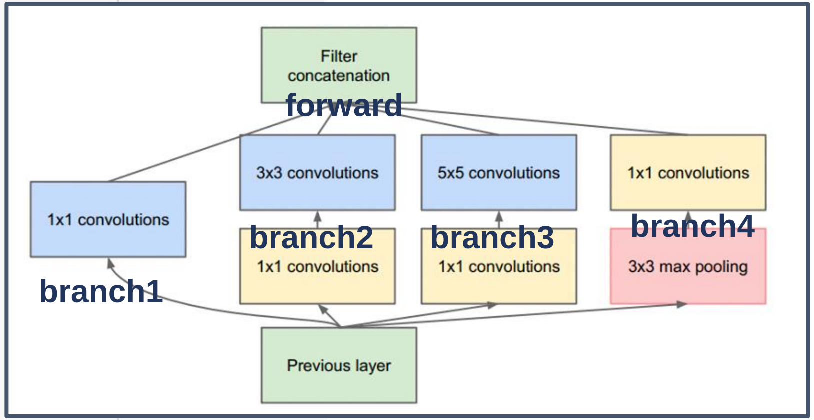
```
class Inception_block(nn.Module):
    def __init__(self, in_channels, out_1x1, red_3x3, out_3x3, red_5x5, out_5x5, out_1x1pool):
        super(Inception_block, self).__init__()

        self.branch1 = conv_block(in_channels, out_1x1, k

        self.branch2 = nn.Sequential(
            conv_block(in_channels, red_3x3, kernel_size=
            conv_block(red_3x3, out_3x3, kernel_size=3, p
        )

        self.branch3 = nn.Sequential(
            conv_block(in_channels, red_5x5, kernel_size=
            conv_block(red_5x5, out_5x5, kernel_size=5, p
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            conv_block(in_channels, out_1x1pool, kernel_size=1)
        )

    def forward(self, x):
        x = torch.cat([self.branch1(x), self.branch2(x), self.branch3(x), self.branch4(x)], 1)
        return x
```

```
class conv_block(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(conv_block, self).__init__()

        self.conv_layer = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, **kwargs),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )

    def forward(self, x):
        return self.conv_layer(x)
```

GoogLeNet - 코드

Inception Block

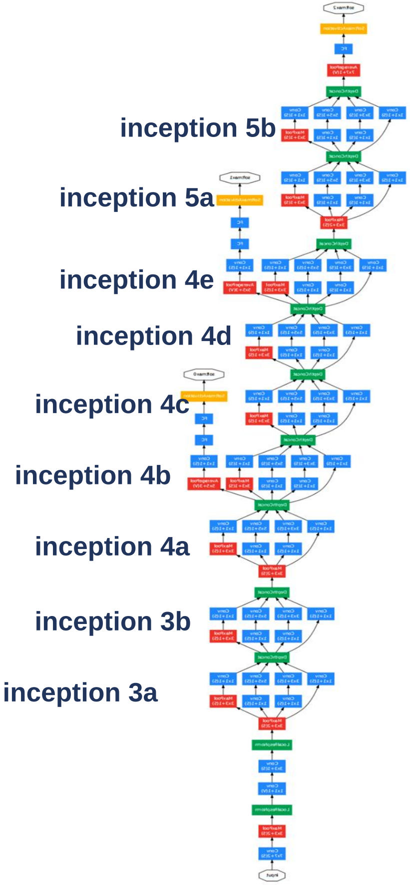
```
class Inception_block(nn.Module):
    def __init__(self, in_channels, out_1x1, red_3x3, out_3x3, red_5x5, out_5x5, out_1x1pool):
        super(Inception_block, self).__init__()

        self.branch1 = conv_block(in_channels, out_1x1, kernel_size=1)

        self.branch2 = nn.Sequential(
            conv_block(in_channels, red_3x3, kernel_size=1),
            conv_block(red_3x3, out_3x3, kernel_size=3, padding=1),
        )

        self.branch3 = nn.Sequential(
            conv_block(in_channels, red_5x5, kernel_size=1),
            conv_block(red_5x5, out_5x5, kernel_size=5, padding=2),
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            conv_block(in_channels, out_1x1pool, kernel_size=1)
        )

    def forward(self, x):
        x = torch.cat([self.branch1(x), self.branch2(x), self.branch3(x), self.branch4(x)], 1)
        return x
```



concatenate all branches

GoogLeNet - 코드

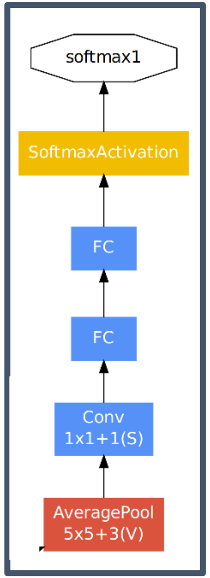
```python
class GoogLeNet(nn.Module):
    def __init__(self,aux_logits=True, num_classes=3, init_weights=True):
        super(GoogLeNet, self).__init__()
        assert aux_logits == True or aux_logits == False
        self.aux_logits = aux_logits

        self.conv1 = conv_block(3, 64, kernel_size=7, stride=2, padding=3)
        self.maxpool1 = nn.MaxPool2d(3, 2, 1)
        self.conv2 = conv_block(64, 192, kernel_size=3, stride=1, padding=1)
        self.maxpool2 = nn.MaxPool2d(3, 2, 1)
        self.inception3a = Inception_block(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception_block(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(3, 2, 1)
        self.inception4a = Inception_block(480, 192, 96, 208, 16, 48, 64)
        # auxiliary classifier
        self.inception4b = Inception_block(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception_block(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception_block(512, 112, 144, 288, 32, 64, 64)
        # auxiliary classifier
        self.inception4e = Inception_block(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(3, 2, 1)
        self.inception5a = Inception_block(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception_block(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(7, 1)
        self.dropout = nn.Dropout(p=0.4)
        self.fc1 = nn.Linear(1024, num_classes)
```

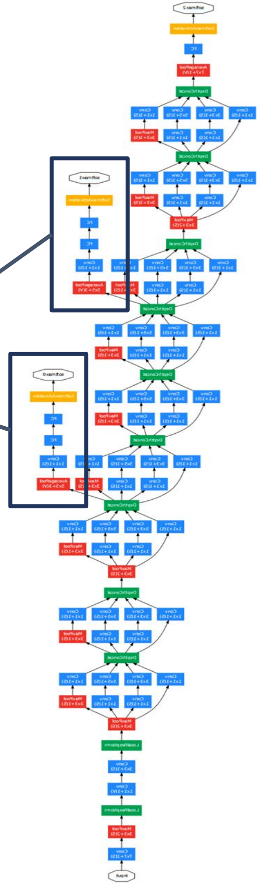


GoogLeNet - 코드

```python
def forward(self, x):
    x = self.conv1(x)
    x = self.maxpool1(x)
    x = self.conv2(x)
    x = self.maxpool2(x)
    x = self.inception3a(x)
    x = self.inception3b(x)
    x = self.maxpool3(x)
    x = self.inception4a(x)

    if self.aux_logits and self.training:
        aux1 = self.aux1(x)

    x = self.inception4b(x)
    x = self.inception4c(x)
    x = self.inception4d(x)

    if self.aux_logits and self.training:
        aux2 = self.aux2(x)

    x = self.inception4e(x)
    x = self.maxpool4(x)
    x = self.inception5a(x)
    x = self.inception5b(x)
    x = self.avgpool(x)

    x = x.view(x.shape[0], -1)

    x = self.dropout(x)
    x = self.fc1(x)
```

```python
class InceptionAux(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(InceptionAux, self).__init__()

        self.conv = nn.Sequential(
            nn.AvgPool2d(kernel_size=5, stride=3),
            conv_block(in_channels, 128, kernel_size=1),
        )

        self.fc = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(1024, num_classes),
        )

    def forward(self,x):
        x = self.conv(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x
```

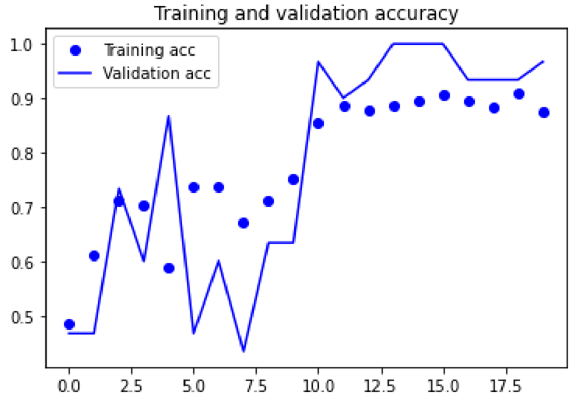




Auxiliary classifier

GoogLeNet - 분석 결과

환경 : colab gpu, epoch 20

	GoogLeNet
train & valid accuracy	
학습 시간	4.6092 min
test accuracy	89.4%

ResNet

Part 2. CNN 주요 모델 소개

Resnet (Residual Network)

논문	K. He, X. Zhang, S. Ren, and J. Sun. (Microsoft Research) Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.
배경	이론적으로, 네트워크를 깊게 쌓을수록 좋은 성능을 낸다고 밝혀져 있다. 그러나 현실에서는, 네트워크가 깊어질수록 성능이 나빠지는 문제가 발생한다. → Resnet은 이를 해결하고자 만든 방법
주요 개념	identity mapping과 shortcut connection을 이용한 Residual learning → 최적화하기 쉬워지고, 더 깊은 층을 쌓아 높은 정확도를 얻을 수 있다. 파라미터 증가가 없어 계산이 빠르다.

Resnet - deep layer의 문제점

기울기 소실/폭주 문제

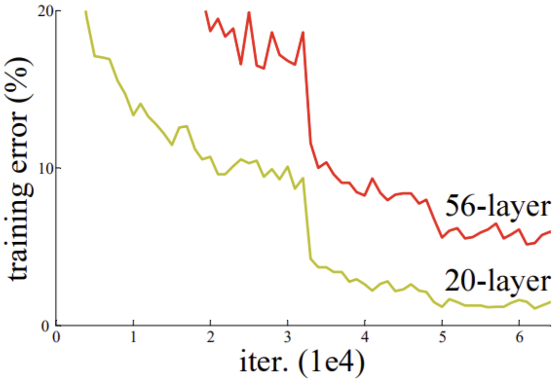
- 역전파 과정에서 기울기가 수렴하지 않는 것
- 층이 깊어질수록 더 쉽게 발생한다.

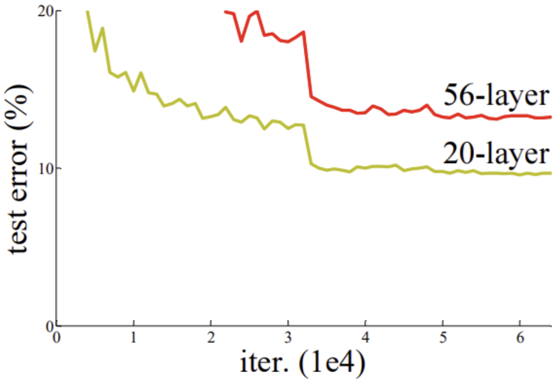
해결

- normalized initialization
- intermediate normalization layers

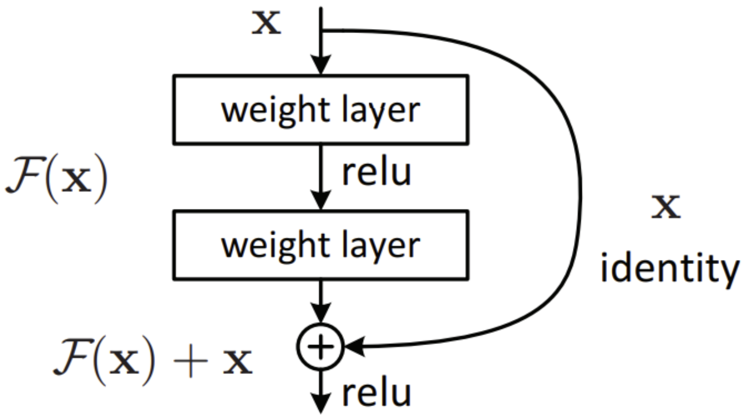
degradation 문제

- 층이 깊어질 때 오히려 성능이 떨어지는 문제
- train error와 test error 모두 높기 때문에 과적합에 의한 것이 아니다.
- 층이 깊어지면서 최적화하기 힘든 부분들이 발생하는 것





Resnet - Residual learning



building block

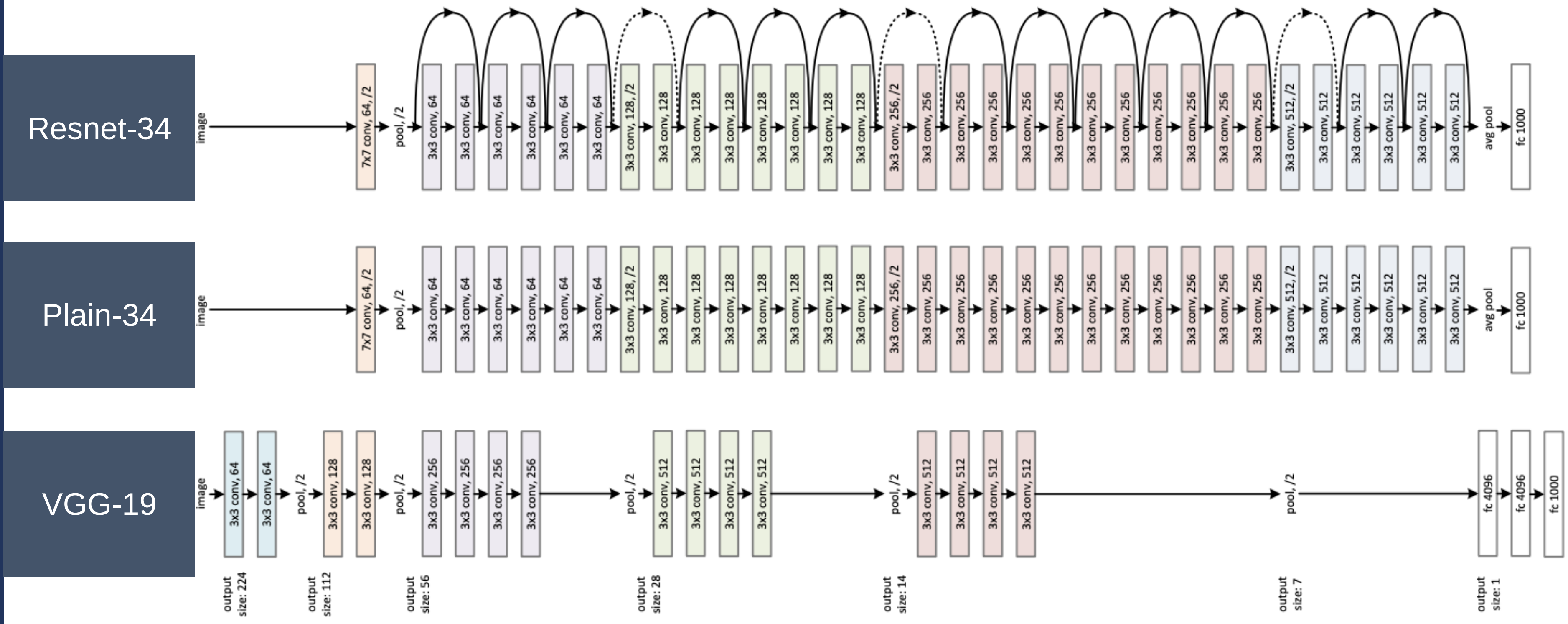
Residual Learning

H(x) : desired underlying mapping
F(x) = H(x) - x : residual mapping

각 층의 타겟함수를 추정하는 대신 이전에 학습된 모델의 출력과 추가된 레이어의 출력의 차이값인 residual만 학습한다. 연산이 간단해지고, error값 크기의 측면에서 학습이 더 쉽다는 것

✓ identity mapping : 인풋을 그대로 가져오는 것
✓ shortcut connection : 하나 이상의 layer를 뛰어 넘는 것

Resnet - 구조

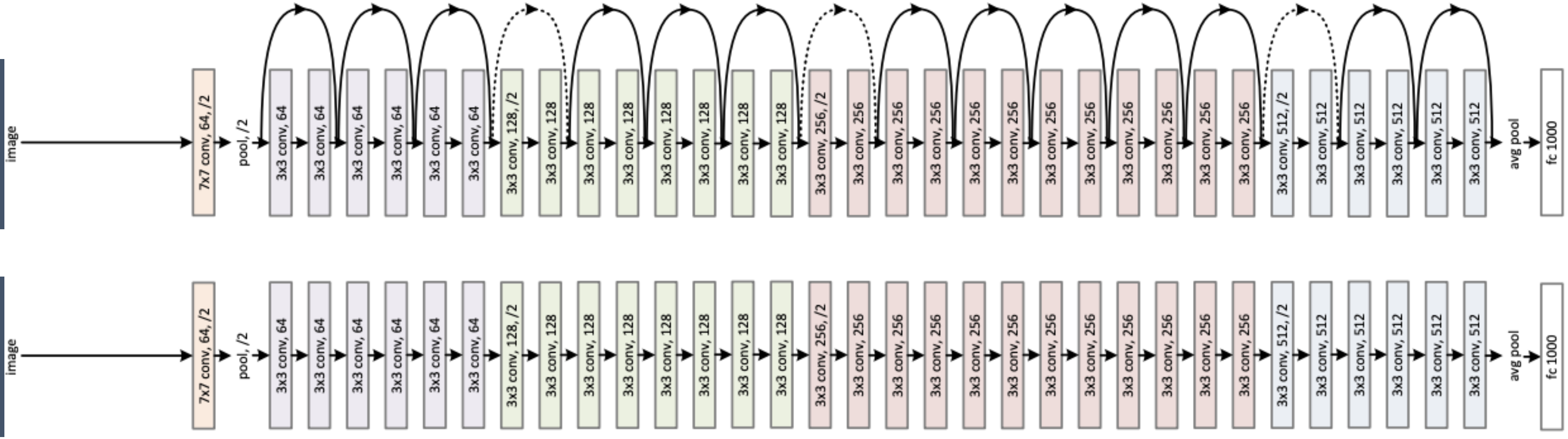


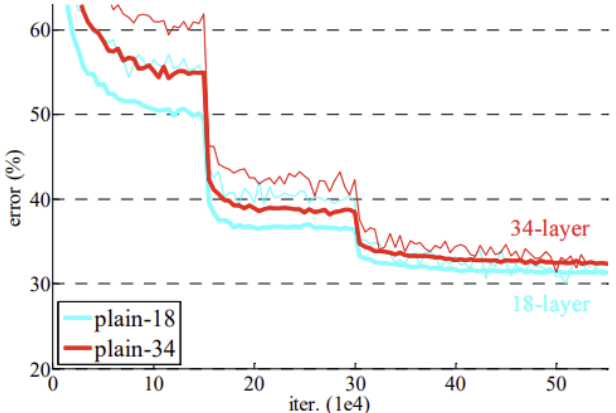
Resnet-34
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
Plain-34
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

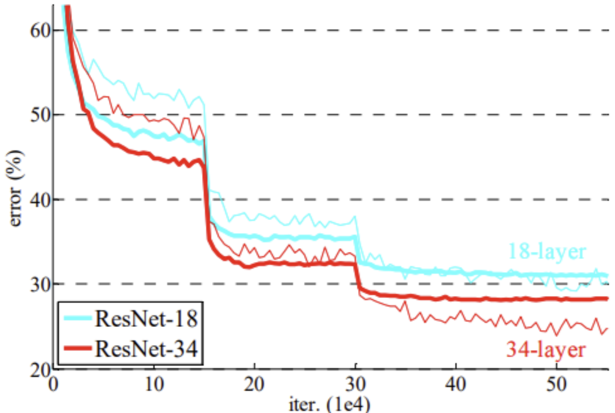
Resnet - 구조

Resnet-34

Plain-34



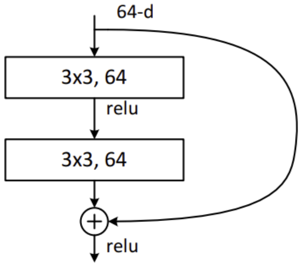




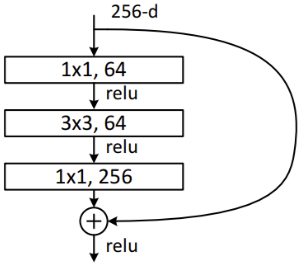
Top-1 error

	plain	resnet
18층	27.94	27.88
34층	28.54	25.03

Resnet - 구조 (residual block)



building block



bottlenet block

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

Resnet - 코드

F(x)

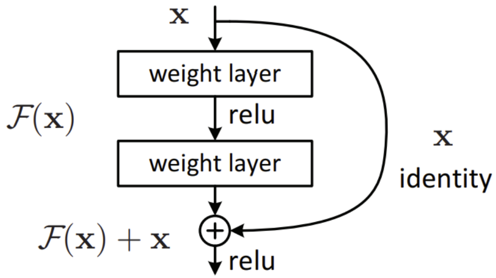
1층

2층

x

F(x) + x

```
class BasicBlock(nn.Module):
    expansion = 1
    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 * BasicBlock.expansion, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels * BasicBlock.expansion),
        )

        # identity mapping
        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

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

    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```



x와 F(x)의 차원이 같아야 더할 수 있으므로, 채널 개수의 변화로 차원이 다른 경우 1x1 conv 층으로 맞춰준다.

Resnet - 코드

F(x)

1층

2층

3층

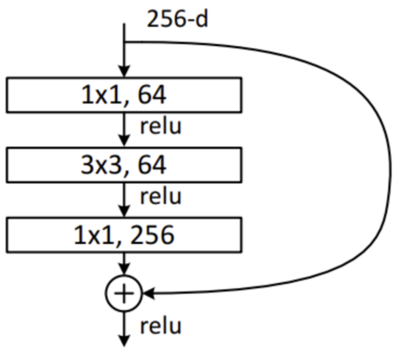
```
class BottleNeck(nn.Module):
    expansion = 4
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()

        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels * BottleNeck.expansion, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(out_channels * BottleNeck.expansion),
        )
```

x

```
        self.shortcut = nn.Sequential()

        self.relu = nn.ReLU()

        if stride != 1 or in_channels != out_channels * BottleNeck.expansion:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels*BottleNeck.expansion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels*BottleNeck.expansion)
```

F(x) + x

```
    def forward(self, x):
        x = self.residual_function(x) + self.shortcut(x)
        x = self.relu(x)
        return x
```



x와 F(x)의 차원이 같아야 더할 수 있으므로, 채널 개수의 변화로 차원이 다른 경우 1x1 conv 층으로 맞춰준다.

Resnet - 코드

```python
class ResNet(nn.Module):
    def __init__(self, block, num_block, num_classes=3, 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 * block.expansion, num_classes)

        # weights inittialization
```

residual block을 지정한 개수만큼 쌓는 함수

```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 * block.expansion

        return nn.Sequential(*layers)
```

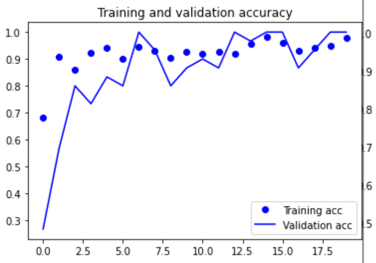
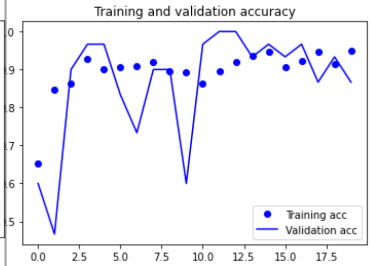
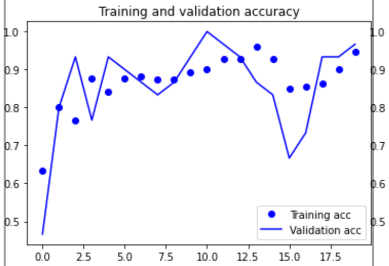
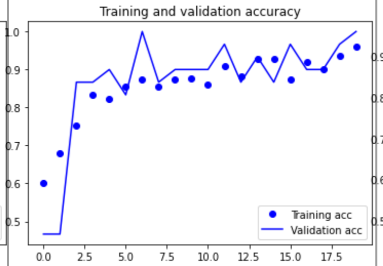
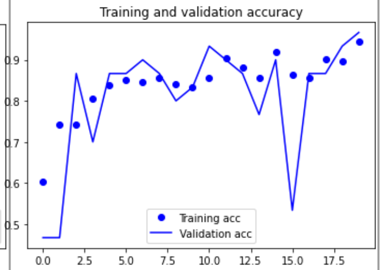
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[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

```python
def resnet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])

def resnet34():
    return ResNet(BasicBlock, [3, 4, 6, 3])

def resnet50():
    return ResNet(BottleNeck, [3, 4, 6, 3])

def resnet101():
    return ResNet(BottleNeck, [3, 4, 23, 3])

def resnet152():
    return ResNet(BottleNeck, [3, 8, 36, 3])
```

각 layer에 넣을 block의 개수

Resnet - 분석 결과

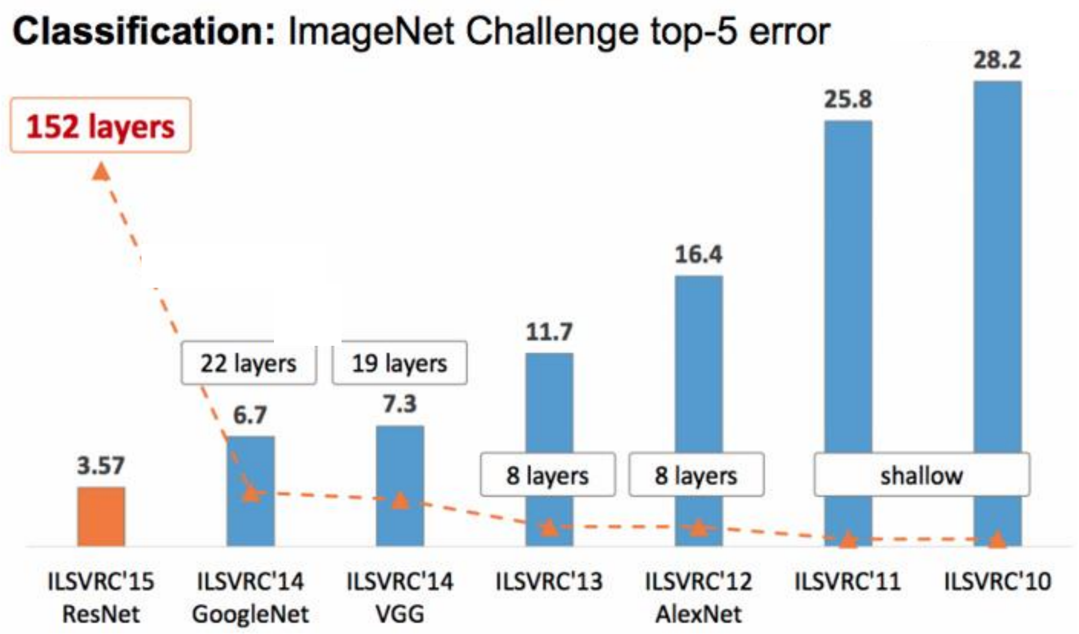
환경 : colab gpu, epoch 20

Resnet 18	Resnet 34	Resnet 50	Resnet 101	Resnet 152
				
4.79 min	4.99 min	5.36 min	6.12 min	7.32 min
90.9%	83.3%	89.4%	86.4%	84.8%

모델 비교

Part 3. 분석 결과

모델 성능


Classification: ImageNet Challenge top-5 error
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

모델 성능

모델	Alexnet	VGG-19	GoogLeNet	ResNet-18
깊이	8 layer	19 layer	22 layer	18 layer
학습 시간	4.71 min	5.61 min	4.61 min	4.79 min
테스트 정확도	80.3%	83.3%	89.4%	90.9%

한계점 및 제언

Part 3. 분석 결과

한계점 및 제언

1. 비교적 간단하고 적은 데이터
2. 부족한 gpu
3. 이미지 분류 프레임 워크에 대한 전반적인 이해도

참고문헌 및 출처

데이터 출처	https://www.kaggle.com/pranavraikokte/covid19-image-dataset
Alexnet	https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
VGG	https://arxiv.org/abs/1409.1556v6
GoogLeNet	https://arxiv.org/abs/1409.4842v1
ResNet	https://arxiv.org/abs/1512.03385v1
코드 구현 참고	https://github.com/Seonghoon-Yu/Paper_Review_and_Implementation_in_PyTorch

감사합니다