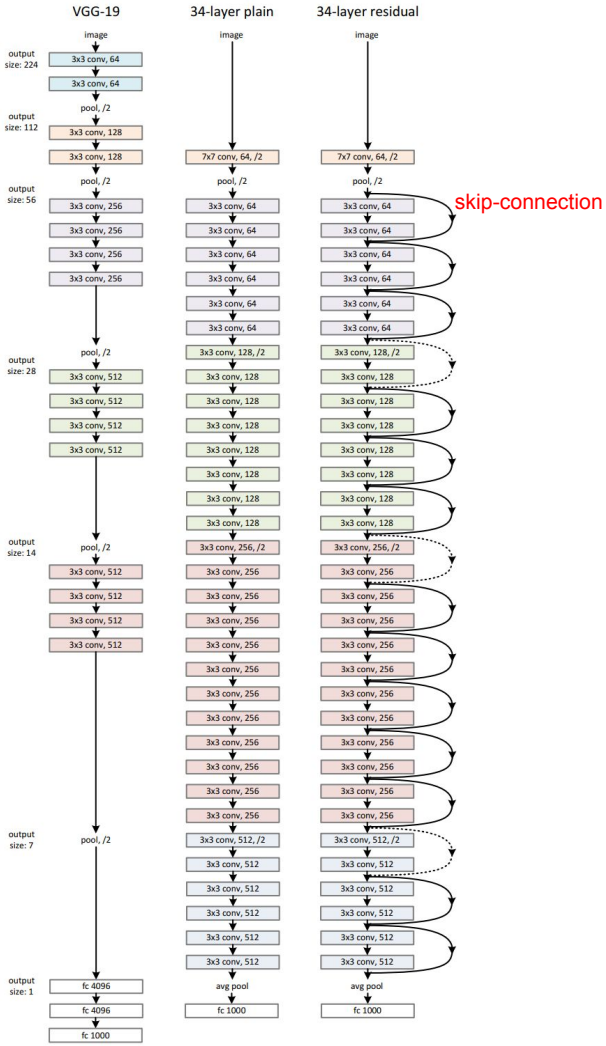
ResNet 디자인


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

pad=3

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer	
conv1	112×112	7×7, 64, stride 2					
		3×3 max pool, stride 2					
conv2_x	56×56	[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	stride=1
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	stride=2
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	stride=2
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] ×3	[1×1, 512; 3×3, 512] ×3	stride=2
	1×1	average pool, 1000-d f					
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9			

64 * 4 = 256
128*4 = 512
512*4 = 2048

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see a
sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

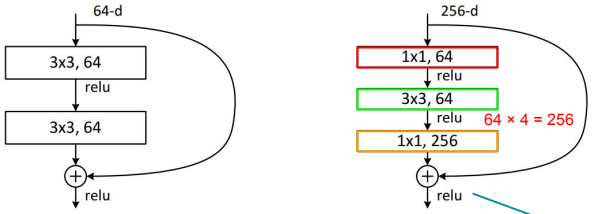


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

```python
class block(nn.Module):
    def __init__(
        self, in_channels, out_channels, identity_downsample=None, stride=1):
        super(block, self).__init__() # 상속받은 부모 클래스의 모든 attributes을 그대로 받아옴
        self.expansion = 4

        self.conv1 = nn.Conv2d( in_channels=in_channels, out_channels=out_channels, kernel_size=1, stride=1, padding=0 )
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d( out_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d( out_channels, out_channels * self.expansion, kernel_size=1, stride=1, padding=0 )
        self.bn3 = nn.BatchNorm2d(out_channels * self.expansion)
        self.relu = nn.ReLU()
        self.identity_downsample = identity_downsample


    def forward(self, x):
        identity = x.clone() # identity_value 저장

        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
```

out_channels * 4

pad=3

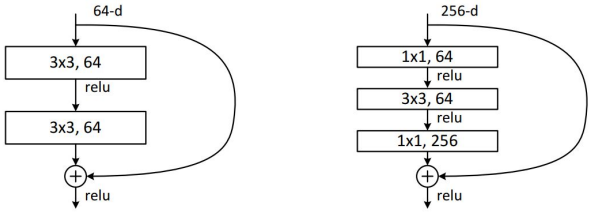
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[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	[1×1, 128
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		
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	[3×3, 512; 1×1, 2048] ×3	[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

stride=1

stride=2

```python
class ResNet(nn.Module):
    """ layers := residual_block 의 개수를 리스트 형태로 담고 있음
    (e.g) layers = [3, 4, 6, 3]\
    """
    def __init__(self, block, layers, image_channels, num_classes):
        super(ResNet, self).__init__() # 상속받은 부모 클래스의 모든 attributes을 그대로 받아옴
```

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.




```python
#%% 모델 타입
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)
```

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

pad=3

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

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

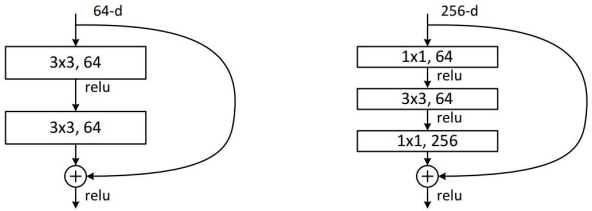


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

```python
class ResNet(nn.Module):
    """ layers := residual_block 의 개수를 리스트 형태로 담고 있음
    (e.g) layers = [3, 4, 6, 3]
    """
    def __init__(self, block, layers, image_channels, num_classes):
        super(ResNet, self).__init__() # 상속받은 부모 클래스의 모든 attributes을 그대로 받아옴


        """ conv1 정의
        """
        self.in_channels = 64
        self.conv1 = nn.Conv2d(image_channels, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)


        """ ResNet layers 정의
        나머지 conv2_x, conv3_x, conv4_x, conv5_x  정의
        """
        # Essentially the entire ResNet architecture are in these 4 lines below
        self.layer1 = self._make_layer( block, layers[0], out_channels=64, stride=1 )
        self.layer2 = self._make_layer( block, layers[1], out_channels=128, stride=2 )
        self.layer3 = self._make_layer( block, layers[2], out_channels=256, stride=2 )
        self.layer4 = self._make_layer( block, layers[3], out_channels=512, stride=2 )

        """ 마지막 레이어
        """
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * 4, num_classes)

    def forward(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 = x.reshape(x.shape[0], -1)
        x = self.fc(x)

        return x

    def _make_layer(self, block, num_residual_blocks, out_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 != out_channels * 4):
            identity_downsample = nn.Sequential(    nn.Conv2d(self.in_channels, out_channels * 4, kernel_size=1, stride=stride),
                                                    nn.BatchNorm2d(out_channels * 4),
                                                )


        """ Residual_block 추가
        """
        layers.append( block(self.in_channels, out_channels, identity_downsample, stride) )

        # The expansion size is always 4 for ResNet 50,101,152
        self.in_channels = out_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, out_channels))

        return nn.Sequential(*layers)
```

ex) ResNet50 일 때

layers = [3, 4, 6, 3]
layers[0] := 3

3 , 64, 1

64, 64, None, 1