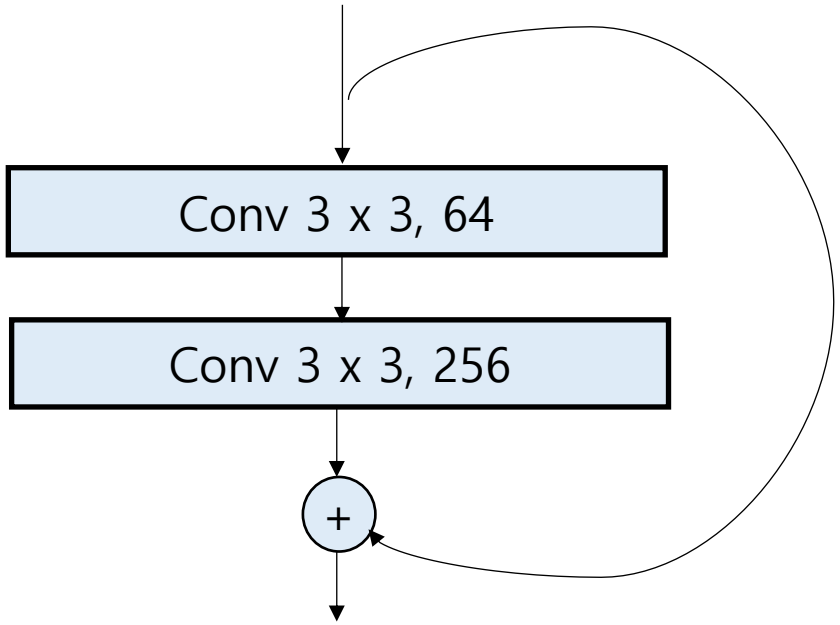
Resnet Architecture

M2019170 허성실

Basic Block

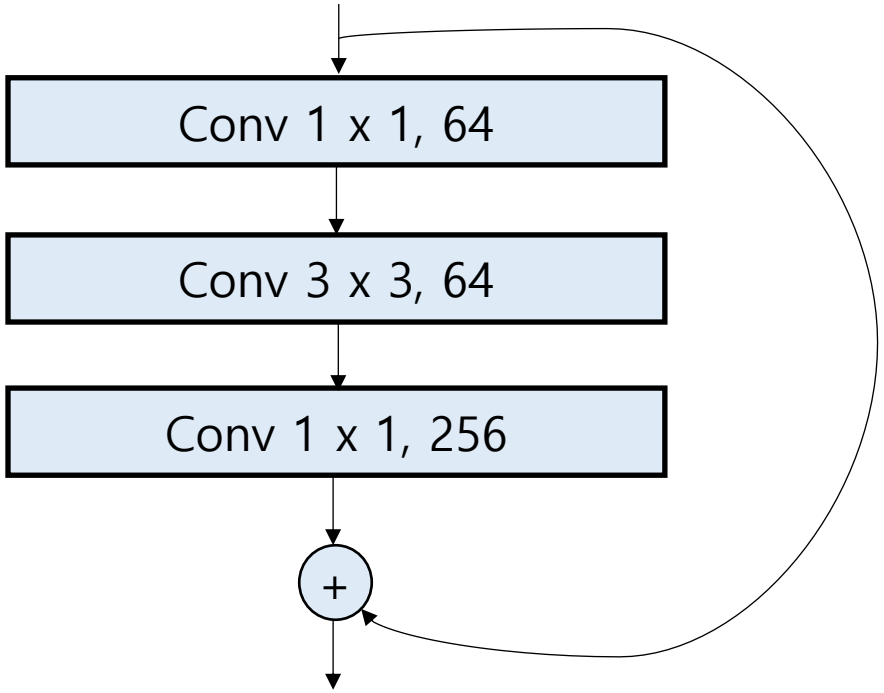
- 18/34 layer




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

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):

        identity = x
        #x.shape= 3,64,64라고 가정
        out = self.conv1(x) # 3x3 stride = 2
        #outd1 3,32,32가 될 것임.
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out) # 3x3 stride = 1
        out = self.bn2(out)

        if self.downsample is not None:  #downsample은 stride가 2가 들어오는 경우
            identity = self.downsample(x)

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

        return out
```

Bottleneck Block

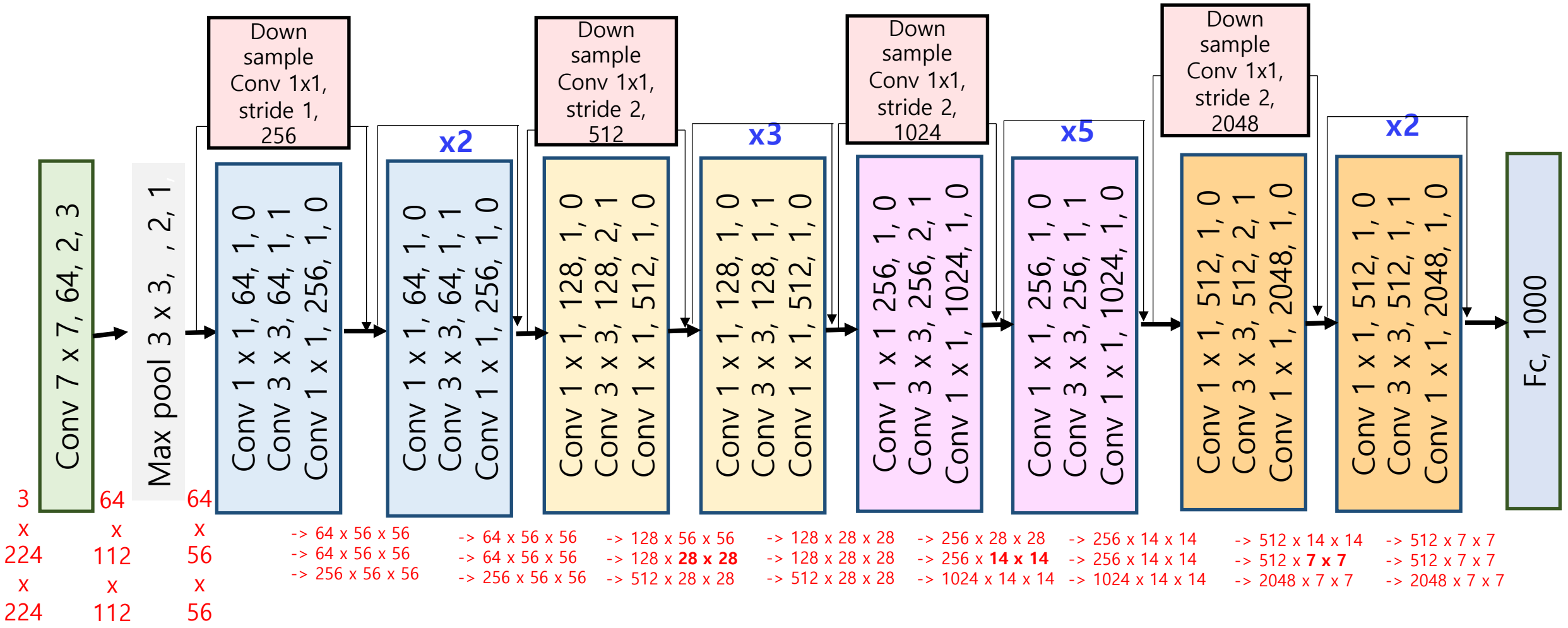
50/101/153 layer에 대해서는 bottleneck 구조를 사용하여 conv 1x1로 차원을 줄였다가 늘려서 연산량을 절감시킴




```python
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.conv1 = conv1x1(inplanes, planes) #conv1x1(64,64)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = conv3x3(planes, planes, stride)#conv3x3(64,64)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = conv1x1(planes, planes * self.expansion) #conv1x1(64,256)
        self.bn3 = nn.BatchNorm2d(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
        out = self.bn3(out)

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

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

        return out
```

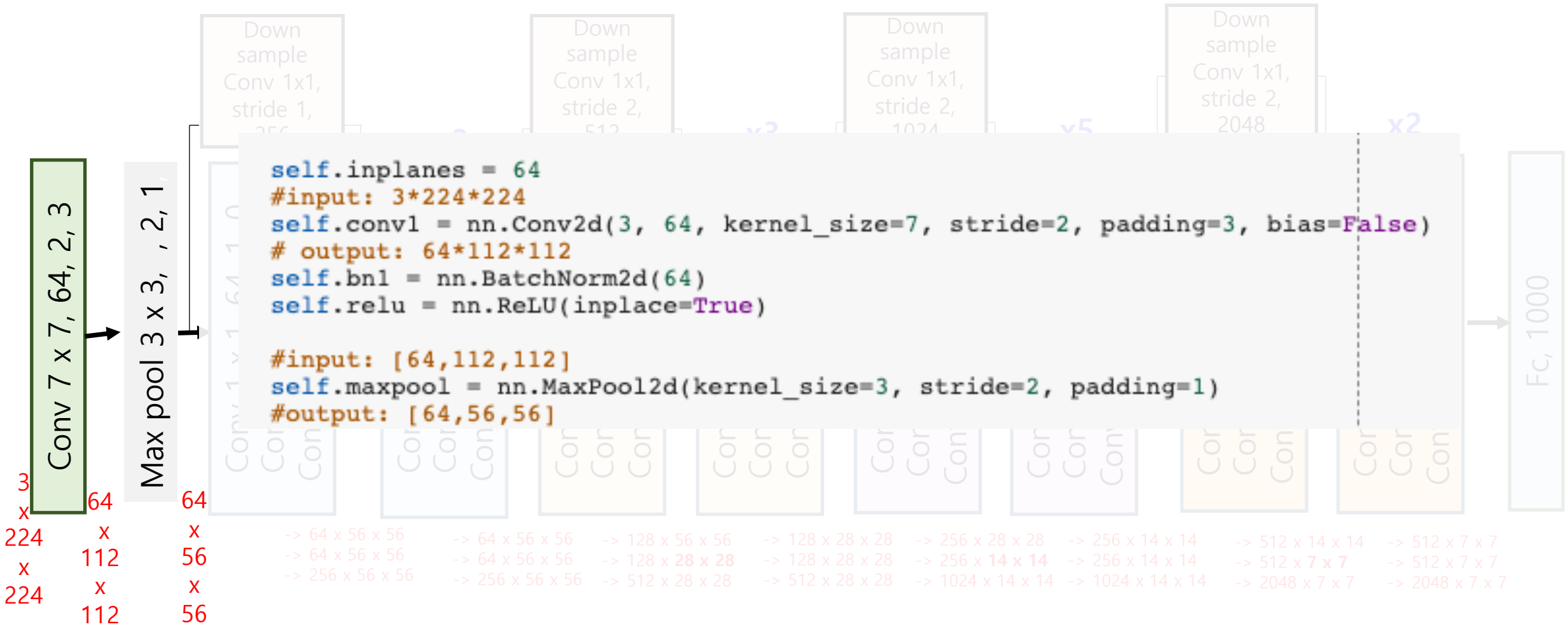
Resnet 50 architecture

Kernel, out_size, kernel, stride, padding
(conv1) + 3 x (3 + 4 + 6+ 3) + (fc) = 48 + 2 = 50



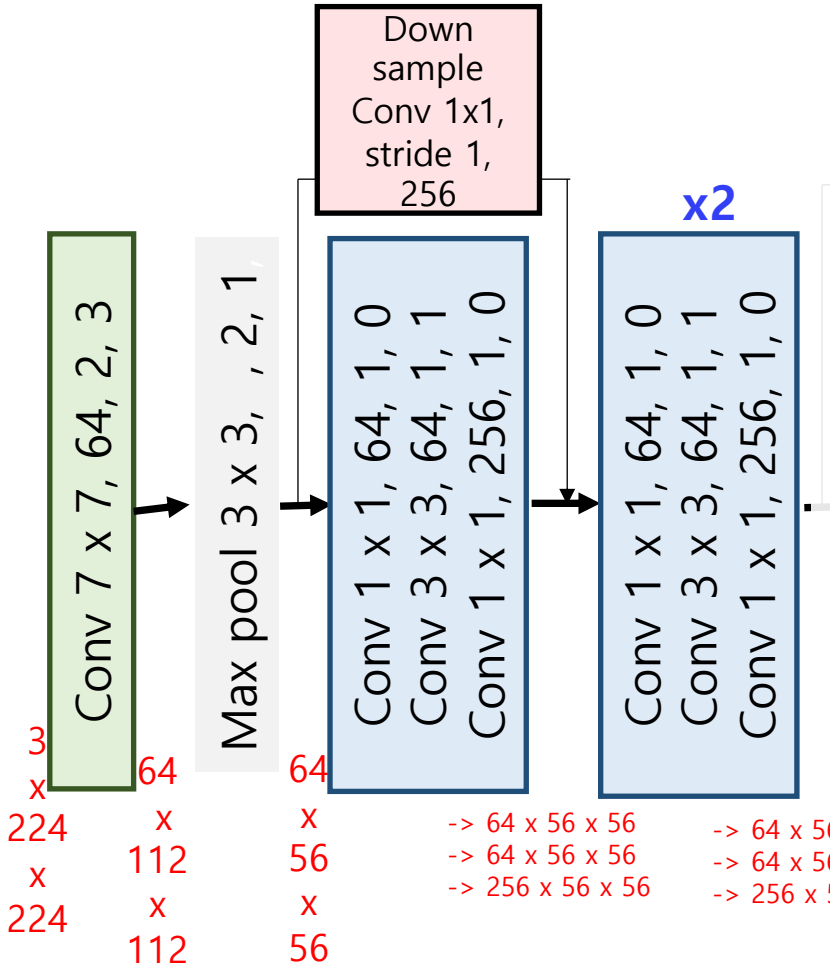
Resnet architecture

Kernel, out_size, kernel, stride, padding



Resnet architecture

Kernel, out_size, kernel, stride, padding



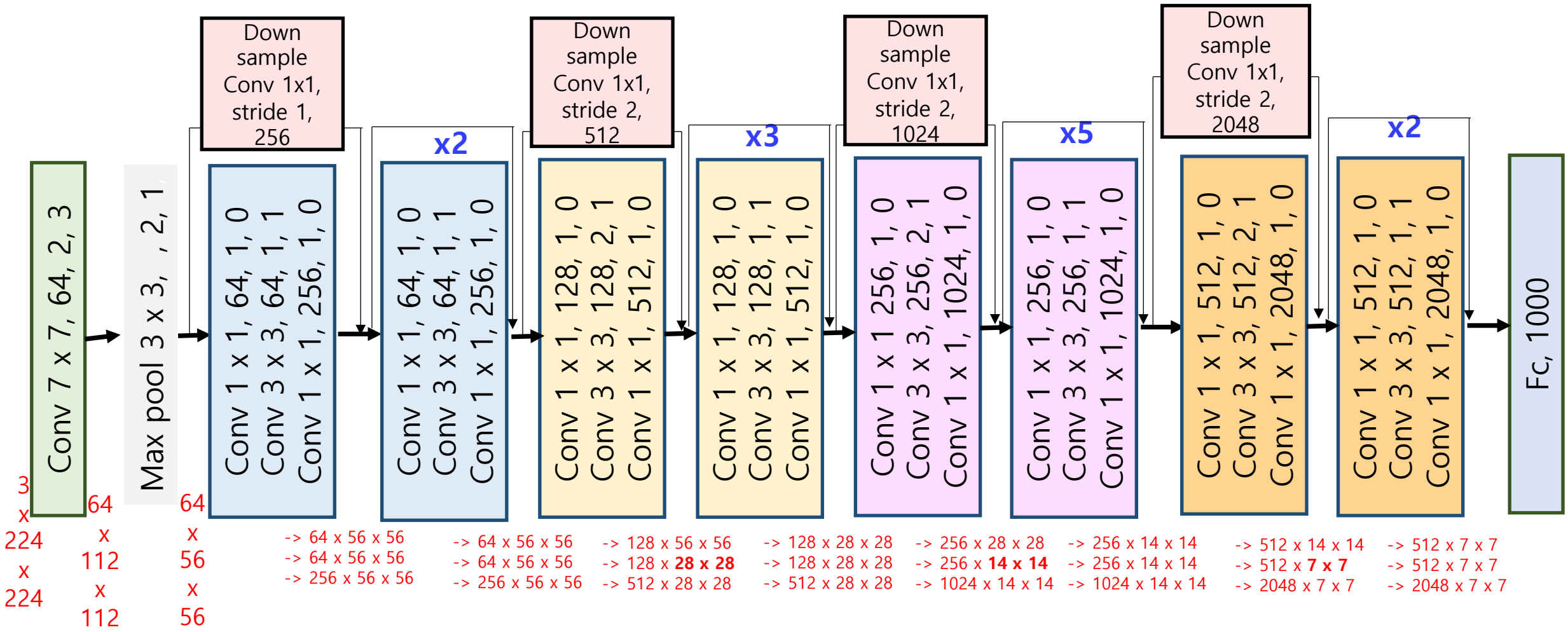

```
self.layer1 = self._make_layer(bll 64, layers[0])  #3
```

```
# self.layer1 = self._make_layer(Bottleneck, 64, layers[0], 3)
def _make_layer(self, block, planes, blocks, stride=1):

    downsample = None

    if stride != 1 or self.inplanes != planes * block.expansion: # 64*4(이 코드에서는 채널 수 맞추기 위해서도 사용)
        print(self.inplanes)
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride), #conv1x1(64, 64*4, 1) #stride=1
            nn.BatchNorm2d(planes * block.expansion), #batchnrom2d(256)
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))

    self.inplanes = planes * block.expansion #self.inplanes = 64 * 4 = 256

    for _ in range(1, blocks):
        layers.append(block(self.inplanes, planes)) # * 2

    return nn.Sequential(*layers)
    #Bottleneck(64,64,1, downsample) / Bottlneck(256,64) / Bottlneck(256,64)
    #Bottleneck(256,128,2, downsample) / Bottlneck(512,128) / Bottlneck(512,128) / Bottlneck(512,128)
```

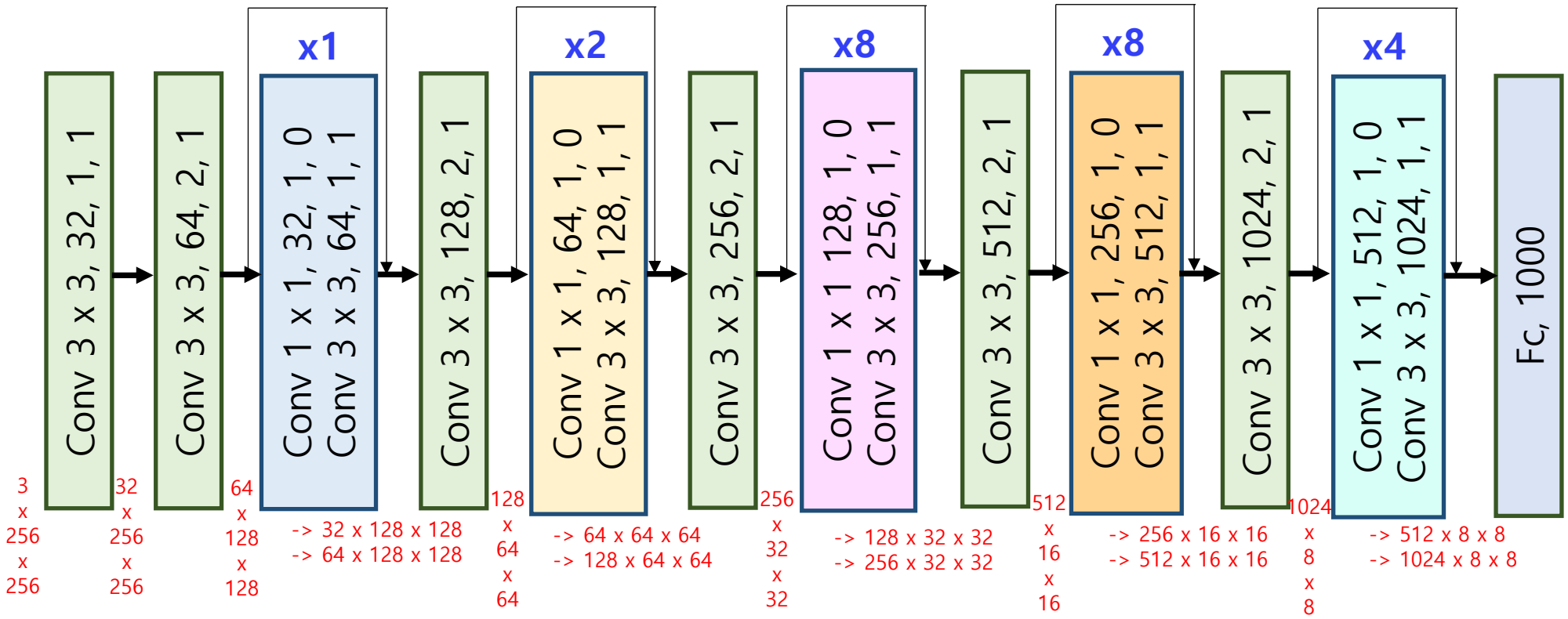
-> 128 x 56 x 56 -> 128 x 28 x 28 -> 256 x 28 x 28 -> 256 x 14 x 14 -> 512 x 14 x 14 -> 512 x 7 x 7
-> 128 x **28 x 28** -> 128 x 28 x 28 -> 256 x **14 x 14** -> 256 x 14 x 14 -> 512 x **7 x 7** -> 512 x 7 x 7
-> 512 x 28 x 28 -> 512 x 28 x 28 -> 1024 x 14 x 14 -> 1024 x 14 x 14 -> 2048 x 7 x 7 -> 2048 x 7 x 7

Resnet 50 architecture

Kernel, out_size, kernel, stride, padding



YOLOv3 Architecture



YOLOv3 Architecture