HW4. CNN의 심화과정

> 아래의 모델은 'ResNet'의 기본 형태입니다.
>
> 이미지 분류를 위해 사용되는 신경망 모델로, 잔차블록을 이용하여 기울기 소실 문제를 완화한 모델입니다.

' '으로 표시된 빈 부분을 채워주시면 됩니다.

In [1]:
```
import torch
import torch.nn as nn

class BasicBlock(nn.Module):
    def __init__(self, in_put_channels, out_put_channels, kernel_size=3, stride=
        super(BasicBlock, self).__init__()

        # 1번째 conv layer
        self.conv_1 = nn.Conv2d(in_put_channels, out_put_channels, kernel_size=k
        self.bn_1 = nn.BatchNorm2d(out_put_channels)

        # 2번째 conv layer
        self.conv2 = nn.Conv2d(out_put_channels, out_put_channels, kernel_size=k
        self.bn2 = nn.BatchNorm2d(out_put_channels)

        self.downsample = downsample
        self.relu = nn.ReLU()

    def forward(self, x):
        identity = x

        # 1 layer
        out_put = self.conv_1(x)
        out_put = self.bn_1(out_put)
        out_put = self.relu(out_put)

        # 2 layer
        out_put = self.conv2(out_put)
        out_put = self.bn2(out_put)

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

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

        return out_put



class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=10):
        super(ResNet, self).__init__()

        # 1 conv layer
```

```
        self.in_put_channels = 64
        self.conv_1 = nn.Conv2d(3, self.in_put_channels, kernel_size=7, stride=2
        self.bn_1 = nn.BatchNorm2d(self.in_put_channels)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # layer 총 4개
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer1 = self._make_layer(block, 64, layers[0])



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


    def forward(self, x):
        x = self.conv_1(x)
        x = self.bn_1(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
```