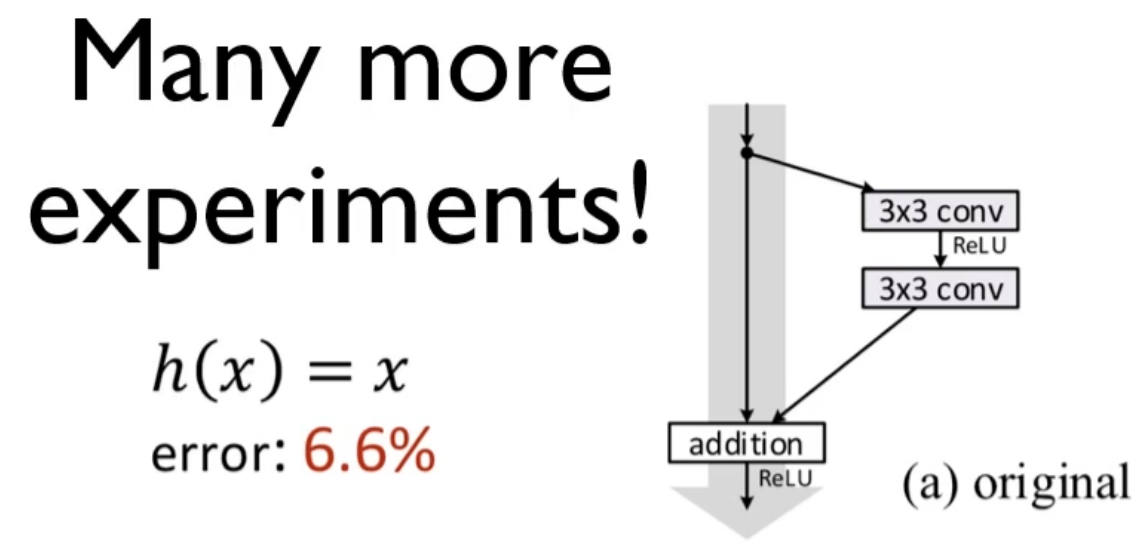
CNN Case Study_ResNet

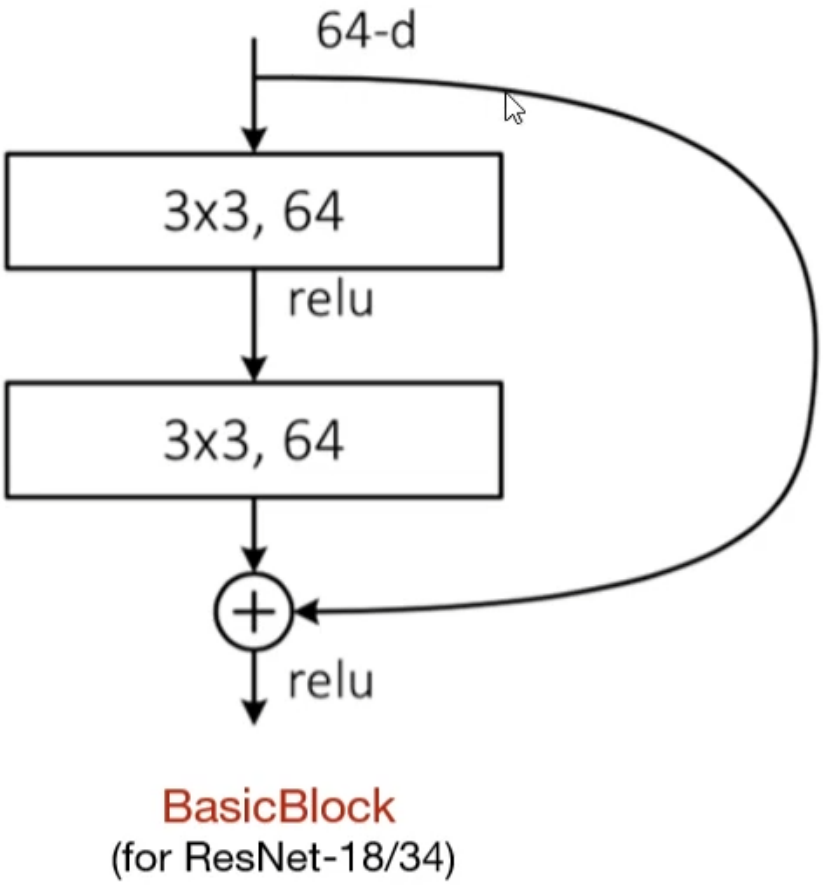
상태	CNN
담당자	

Architecture



`torchvision.models.resnet`을 사용해서 ResNet을 만들어 보자. (`resnet(18, 34, 50, 101, 152)`) 단, 여기에서는 3x224x224 input을 기준으로 만들도록 되어 있는데, CIFAR-10처럼 input size가 다른 경우에 ResNet을 적용하려 한다.

BasicBlock



BasicBlock
(for ResNet-18/34)

```
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) # 만약 stride가 있다면 그 값으로
        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

        out = self.conv1(x) # 3x3 stride = stride
        out = self.bn1(out)
        out = self.relu(out)

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

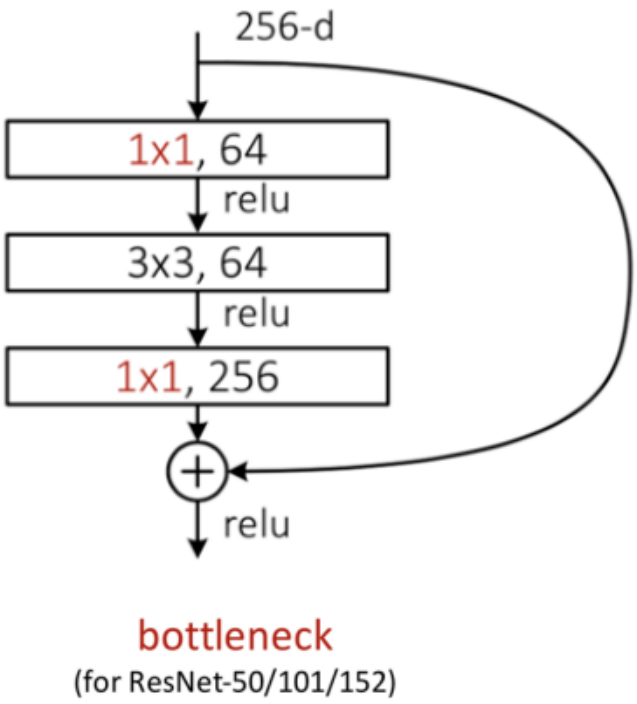
```
        if self.downsample is not None:
            identity = self.downsample(x)

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

        return out
```

`out.shape`은 3x32x32인데, `identiy.shape`은 3x64x64이므로 연산이 불가능하다. 따라서 `self.downsample`을 통해 연산이 가능하도록 조작해 주자.

Bottleneck



bottleneck
(for ResNet-50/101/152)

```
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
```

마지막 1x1 conv에서 채널의 수가 64에서 256으로 증가하는데, 이는 코드의 `planes * self.expansion` 에서 확인할 수 있다.

ResNet

```
class ResNet(nn.Module):
    # model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs) #resnet 50
    def __init__(self, block, layers, num_classes=1000, zero_init_residual=False):
        super(ResNet, self).__init__()

        self.inplanes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)

        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64, layers[0]'''3''')
        self.layer2 = self._make_layer(block, 128, layers[1]'''4''', stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2]'''6''', stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3]'''3''', stride=2)

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

        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.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
```

```
        # Zero-initialize the last BN in each residual branch,
        # so that the residual branch starts with zeros, and each residual block behaves like an identity.
        # This improves the model by 0.2~0.3% according to https://arxiv.org/abs/1706.02677
        if zero_init_residual:
            for m in self.modules():
                if isinstance(m, Bottleneck):
                    nn.init.constant_(m.bn3.weight, 0)
                elif isinstance(m, BasicBlock):
                    nn.init.constant_(m.bn2.weight, 0)

    def _make_layer(self, block, planes, blocks, stride=1):

        downsample = None

        if stride != 1 or self.inplanes != planes * block.expansion:

            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride), #conv1x1(256, 512, 2)
                nn.BatchNorm2d(planes * block.expansion), #batchnrom2d(512)
            )

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

        self.inplanes = planes * block.expansion #self.inplanes = 128 * 4

        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes)) # * 3

        return nn.Sequential(*layers)


    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.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

`resnet50`을 먼저 살펴보자. `ResNet`의 `block` 인자로 `Bottleneck`이, `layers` 인자로 `[3, 4, 6, 3]`이 들어간다.

```
def resnet50(pretrained=False, **kwargs):
    model = ResNet(Bottleneck, [3, 4, 6, 3], **kwargs) #=> 3*(3+4+6+3) +(conv1) +1(fc) = 48 +2 = 50
    return model
```

우선 다음부터 확인해 보자. 3x224x224의 input이 Conv1 - bn1 - ReLU - MaxPool을 지나면 어떤 shape가 될까?

```
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)

        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
```

3x224x224 → (conv1) → 64x112x112 → bn1 → relu → maxpool → 64x56x56 의 shape을 가진 텐서가 output으로 나오게 될 것이다.

그 다음 `layer1` , `layer2`, `layer3`, `layer4`을 지나기 전에 `make_layer` 함수를 다시 확인해 보자.

```
def _make_layer(self, block, planes, blocks, stride=1):

        downsample = None

        if stride != 1 or self.inplanes != planes * block.expansion:

            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride), #conv1x1(256, 512, 2)
                nn.BatchNorm2d(planes * block.expansion), #batchnrom2d(512)
            )

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

        self.inplanes = planes * block.expansion #self.inplanes = 128 * 4

        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes)) # * 3

        return nn.Sequential(*layers)
```

첫 번째 레이어인 `self.layer1 = self._make_layer(block, 64, layers[0]'''3''')`의 경우, `self.layer1 = self.make_layer(Bottleneck, 64, 3)`이다. (이 예시에서 model로 bottleneck을 사용하였기 때문)

self.layer1

`slef.inplanes` 가 64, `planes`가 64, `block.expansion`이 4(`Bottleneck` 클래스에서 expansion을 4라고 설정)이기 때문에 `self.inplanes != planes * block.expansion`이 `True`이다.

채널을 맞추기 위한 용도로 `downsample`을 사용하게 되는데, 이 `downsample`은 `conv1x1(64, 256, stride=1)`와 `nn.BatchNorm2d(256)`을 지나게 된다.

그 다음 빈 레이어를 만든 다음 `block`(여기서는 `Bottleneck`)을 통과한 값들을 append한다.

self.layer2

`self.inplanes`가 256으로 변경된 다음 시작된다.