COSE474 Deep Learning

Project #2: CNN Architecture Implementation

2017320122 김정규

Code1 – Residual Block

```
#####################################################################
# Question 1 : Implement the "bottle neck building block" part.
# Hint : Think about difference between downsample True and False. How we make the difference by code?
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        if self.downsample:
            self.layer = nn.Sequential(
                ########################################
                ############ fill in here (20 points)
                conv1x1(in_channels, middle_channels, 2, 0),
                conv3x3(middle_channels, middle_channels, 1, 1),
                conv1x1(middle_channels, out_channels, 1, 0)
                ########################################
            )
            self.downsize = conv1x1(in_channels, out_channels, 2,0)

        else:
            self.layer = nn.Sequential(
                ########################################
                ############ fill in here (20 points)
                conv1x1(in_channels, middle_channels, 1, 0),
                conv3x3(middle_channels, middle_channels, 1, 1),
                conv1x1(middle_channels, out_channels, 1, 0)
                ########################################
            )
            self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)


    def forward(self, x):
        if self.downsample:
            out = self.layer(x)
            x = self.downsize(x)
            return out + x
        else:
            out = self.layer(x)
            if x.size() is not out.size():
                x = self.make_equal_channel(x)
            return out + x
#####################################################################
```

ResidualBlock() consists of a structure in which 3x3 conversion is performed after bottleneck, which reduces the channel depth to 1x1 conversion, and 1x1 conversion is performed again. When data with the number of in_channel channels enters and passes through the conv1x1 function, the number of out_channel channels is obtained. conv3x3 proceeds in the same flow as conv1x1 above, but the filter size changes from 1 to 3.

Code2 – ResNet50_layer4

```
#####################################################################
# Question 2 : Implement the "class, ResNet50_layer4" part.
# Understand ResNet architecture and fill in the blanks below. (25 points)
# (blank : #blank#, 1 points per blank )
# Implement the code.
class ResNet50_layer4(nn.Module):
    def __init__(self, num_classes= 10): # Hint : How many classes in Cifar-10 dataset?
        super(ResNet50_layer4, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7, stride=2, padding=3),
                # Hint : Through this conv-layer, the input image size is halved.
                #        Consider stride, kernel size, padding and input & output channel sizes.
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=0)
        )
        self.layer2 = nn.Sequential(
            ResidualBlock(in_channels=64, middle_channels=64, out_channels=256, downsample=False),
            ResidualBlock(in_channels=256, middle_channels=64, out_channels=256, downsample=False),
            ResidualBlock(in_channels=256, middle_channels=64,out_channels=256, downsample=True)
            )
        self.layer3 = nn.Sequential(
            ########################################
            ############ fill in here (20 points)
            ####### you can refer to the 'layer2' above
            ResidualBlock(in_channels=256, middle_channels=128, out_channels=512, downsample=False),
            ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=False),
            ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=False),
            ResidualBlock(in_channels=512, middle_channels=128, out_channels=512, downsample=True)

            ########################################
        )
        self.layer4 = nn.Sequential(
            ########################################
            ############ fill in here (20 points)
            ###### you can refer to the 'layer2' above
            ResidualBlock(in_channels=512, middle_channels=256, out_channels=1024, downsample=False),
            ResidualBlock(in_channels=1024, middle_channels=256, out_channels=1024, downsample=False),
            ResidualBlock(in_channels=1024, middle_channels=256, out_channels=1024, downsample=False),
            ResidualBlock(in_channels=1024, middle_channels=256, out_channels=1024, downsample=False),
            ResidualBlock(in_channels=1024, middle_channels=256, out_channels=1024, downsample=False),
            ResidualBlock(in_channels=1024, middle_channels=256, out_channels=1024, downsample=False)
            ########################################
        )

        self.fc = nn.Linear(1024, 10) # Hint : Think about the reason why fc layer is needed
        self.avgpool = nn.AvgPool2d((1,1), stride=2)
```

This is a class which combines blocks into one. ResNet50 learns repeatedly through a different residual block for each layer. Below is a reference for my code creation which is the basic structure of ResNet 50-layer.

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	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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

Results

```
(PyTorch_env) C:₩Users₩Administrator₩Project2₩Project2>python main.py
Epoch [1/1], Step [100/500] Loss: 0.5897
Epoch [1/1], Step [200/500] Loss: 0.5206
Epoch [1/1], Step [300/500] Loss: 0.5436
Epoch [1/1], Step [400/500] Loss: 0.5150
Epoch [1/1], Step [500/500] Loss: 0.4874
Accuracy of the model on the test images: 81.29 %
```

In this project, the data was classified into 10 classes. According to the analysis of the results, as a result of classification using 500 train image data in Epoch 1, it shows 81.29% performance in test data. This means that in the test image data, about 81 out of 100 scores were scored in the task of determining where one picture is classified among the 10 classes. Also, as the train progresses, the loss from each step decreases.