Osp assignment 9 1716009 백지희

The explanation of the code

[VGG-16]

주어진 코드를 수정없이 사용하였기 때문에 작성 코드 설명은 생략한다.

[ResNet-50]

ResNet은 residual block이 있는 것이 특징이다. 이 코드에서는 bottleneck building block을 residual block으로 사용한다. Bottleneck building block은 1x1, 3x3, 1x1 block, 3개로 구성되어 있다.

```
########################################
############## fill in here (20 points)
# Hint : use these functions (conv1x1, conv3x3)

conv1x1(in_channels, middle_channels, 2, 0),
conv3x3(middle_channels, middle_channels, 1, 1),
conv1x1(middle_channels, out_channels, 1, 0)

########################################
```

```
########################################
############## 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)

########################################
```

앞에서 정의한 함수, conv1x1, conv3x3이 1x1과 3x3 block을 의미한다. 따라서 함수를 호출하여 layer를 쌓았고, 앞에 layer의 output은 이어지는 layer의 input이 된다. 따라서 함수의 parameter가 앞에 함수 output size가 이어지는 함수 input size가 되도록 함수를 사용하였다.

두개 코드 모두 bottleneck building block이지만 stride 값에 차이가 있다. Resnet-50의 layer2와 layer3에서 activation map의 크기가 절반이 되어야 한다. 이를 strided convolution을 통해 구현할 수 있고 stride값이 2가 되면 stride값이 1일 때의 activation map의 크기보다 절반 작아지게 된다.

```
def __init__(self, num_classes=10): # Hint : How many classes in Cifar-10 dataset?
```

Cifar-10의 class수는 총 10개이다.

Layer number	Network	Output Image size
Layer 1	7x7 conv, channel = 64, stride = 2 3x3 max pool, stride = 2	8 x 8
Layer 2	[1x1 conv, channel = 64, 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 2 [1x1 conv, channel = 64, stride = 2 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 1	4 x 4
Layer 3	[1x1 conv, channel = 128, 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 3 [1x1 conv, channel = 128, stride = 2 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 1	2 x 2
Layer 4	[1x1 conv, channel = 256, 3x3 conv, channel = 256, 1x1 conv, channel = 1024] x 6	2 x 2
	AvgPool	1 x 1
	Fully connected layer	?

Network구성은 위의 표를 따랐다.

```
self.layer1 = nn.Sequential(
    nn.Conv2d(3, 64, stride=2, kernel_size=7, padding=3), #blank#, #blank#, #blank#, #blank#, #blank# ),
        # 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(3, 2, 1)#blank#, #blank#, #blank#)
)
```

Layer1이다. Kernel size가 7인 filter로 convolutional filtering이 이루어져야 하고, stride값은 2로 activation map의 사이즈가 절반으로 줄어야 한다. Conv2d의 input channel size는 3이며 output channel size는 64가 되어야 한다. 이에 따라 parameter값을 지정해주었다. max pooling의 kernel size는 3이며, stride값은 2로 parameter값을 지정해주었다. 또한 layer1을 지나고 나면 output의 size가 8x8이 되어야 한다. 따라서 conv에서 padding 값으로 3을 주었다.

```
self.layer2 = nn.Sequential(
    ResidualBlock(64, 64, 256, False),#blank#, #blank#, #blank#, #blank#),
    ResidualBlock(256, 64, 256, False),#blank#, #blank#, #blank#, #blank#),
    ResidualBlock(256, 64, 256, True)#blank#, #blank#,#blank#, #blank#)
)
```

Layer2이다. 여기에는 총 3개의 residual block이 들어가며 마지막 residual block을 들어간 결과로 activation map의 크기가 절반 줄어들어야 한다. 따라서 마지막 세번째 ResidualBlock 함수 호출 시에 downsample bool값을 True로 하여 stride값을 2로 activation map의 size를 줄이도록 하였다. ResidualBlock 함수의 parameter는 순서대로, input_channel, middle_channel, output_channel size를 의미한다. 앞에 block의 output은 이어지는 block의 input size와 같아야 함에 유의하며 channel size parameter값을 넣어주었다.

```
########################################
############## fill in here (20 points)
####### you can refer to the 'layer2' above

ResidualBlock(256, 128, 512, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(512, 128, 512, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(512, 128, 512, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(512, 128, 512, True)  # blank#, #blank#,#blank#, #blank#)

########################################
```

Layer3이다. 여기에는 총 4개의 residual block이 들어가며 마지막 residual block을 들어간 결과로 activation map의 크기가 절반 줄어들어야 한다. 따라서 마지막 네번째 ResidualBlock 함수 호출 시에 downsample bool값을 True로 하여 stride값을 2로 activation map의 size를 줄이도록 하였다. ResidualBlock 함수의 parameter는 순서대로, input_channel, middle_channel, output_channel size를 의미한다. 앞에 block의 output은 이어지는 block의 input size와 같아야 함에 유의하며 channel size parameter값을 넣어주었다.

```
########################################
############# fill in here (20 points)
####### you can refer to the 'layer2' above

ResidualBlock(512, 256, 1024, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(1024, 256, 1024, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(1024, 256, 1024, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(1024, 256, 1024, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(1024, 256, 1024, False),  # blank#, #blank#, #blank#, #blank#),
ResidualBlock(1024, 256, 1024, False)  # blank#, #blank#,#blank#, #blank#)

########################################
```

Layer4이다. 여기에는 총 6개의 residual block이 들어가며 ResidualBlock 함수의 parameter는 순서대로, input_channel, middle_channel, output_channel size를 의미한다. 앞에 block의 output은 이어지는 block의 input size와 같아야 함에 유의하며 channel size parameter값을 넣어주었다.

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

Fully connected layer를 사용하면 channel size를 조절할 수 있다. Class의 수인 10으로 channel수를 조절하여 label과 매칭될 수 있도록 하였다. AvgPool2d의 parameter는 각각 channel 수, stride 수를 의미한다.

The results of the code

[VGG-16]

```
Epoch [1/1], Step [100/500] Loss: 0.1938
Epoch [1/1], Step [200/500] Loss: 0.1857
Epoch [1/1], Step [300/500] Loss: 0.1837
Epoch [1/1], Step [400/500] Loss: 0.1885
Epoch [1/1], Step [500/500] Loss: 0.1918
Accuracy of the model on the test images: 86.37 %

Process finished with exit code 0
```

[ResNet-50]

```
Epoch [1/1], Step [100/500] Loss: 0.2884
Epoch [1/1], Step [200/500] Loss: 0.2919
Epoch [1/1], Step [300/500] Loss: 0.2950
Epoch [1/1], Step [400/500] Loss: 0.2999
Epoch [1/1], Step [500/500] Loss: 0.3019
Accuracy of the model on the test images: 81.55 %

Process finished with exit code 0
```

Analysis

Vgg-16은 85퍼센트, ResNet은 80퍼센트 이상의 정확도가 나오도록 코드를 작성하라는 주석을 따라 각각의 정확도 86.37, 81.55로 목표로 하는 정확도가 나오도록 하였다.