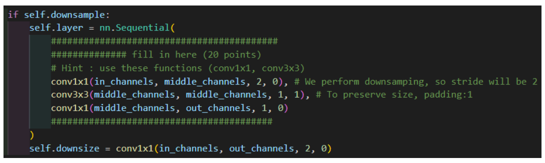
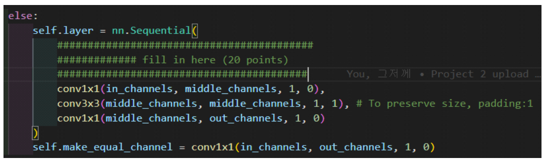
There are several ways to process image data efficiently, like CNN. In recent years, many models were developed, but they suffered commonly from vanishing(and exploding) gradient problem as weight sharing. Finally, this problem was resolved by He, who devised “skip connection”(often called residual blocks) by doing identically mapping additionally. In this report, we will explain code and result of resnet-50 which uses skip connection directly with 50 blocks.

In resnet-50, we use 1×1, 3×3, 1×1 cnn blocks sequentially, and then add input vector to result. In convolution, we process [Convolution, Batch normalization, ReLU]. Note that when we don’t perform downsampling, the result should be kept in the same size as the input. There is no problem doing 1×1 convolution, but when model passes 3×3 convolution with no padding, size reduces by two (output size will be $(H-3+1)\times(H-3+1)=(H-2)\times(H-2)$ with input $H\times H$.) In addition, there is a process of doubling the size of the kernel and doubling the size of the channel each layer. At this time, when proceeding with the first Residual, reduce the size twice by setting the stride to 2. Finally, since the number of channels in the input layer and result may be different, 1×1 convolution is performed for the input layer. Refer to above codes.

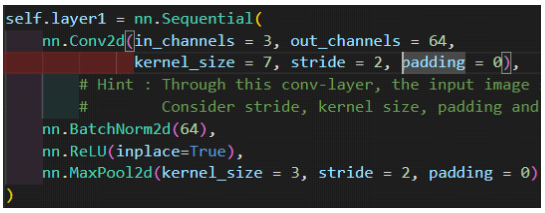


```
if self.downsample:
    self.layer = nn.Sequential(
        ##########################################
        ############## fill in here (20 points)
        # Hint : use these functions (conv1x1, conv3x3)
        conv1x1(in_channels, middle_channels, 2, 0), # We perform downsamping, so stride will be 2
        conv3x3(middle_channels, middle_channels, 1, 1), # To preserve size, padding:1
        conv1x1(middle_channels, out_channels, 1, 0)
        #########################################
    )
    self.downsize = conv1x1(in_channels, out_channels, 2, 0)
```

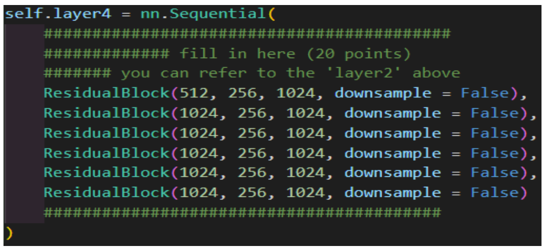


```
else:
    self.layer = nn.Sequential(
        ##########################################
        ############# fill in here (20 points)
        ##########################################        You, 그저께 • Project 2 upload …
        conv1x1(in_channels, middle_channels, 1, 0),
        conv3x3(middle_channels, middle_channels, 1, 1), # To preserve size, padding:1
        conv1x1(middle_channels, out_channels, 1, 0)
    )
    self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)
```

Using aboves, we construct resnet-50 with 4 layers, and each layer has several blocks. For example, at Layer 3, there are 3 blocks without downsampling and 1 block with downsampling at last. Note that 1. In early, channel of image are 3, so input channel is 3 at first layer, 2. there are no downsampling blocks on Layer 4(Final block). After that, we process global average pooling, which return only 1 value each channel. There are 1024 channels at layer 4, so we get 1024 values. Finally, these values pass last fully connected layer and return highest values among them by using softmax classifier. On the right is the code for what was explained above. In addition, several optimization techniques such as batch normalization have added some parameters, and the final model has approximately 14M trainable parameters.

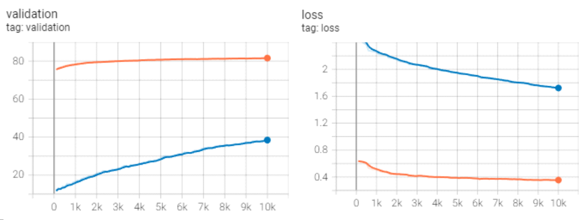


```
self.layer1 = nn.Sequential(
    nn.Conv2d(in_channels = 3, out_channels = 64,
              kernel_size = 7, stride = 2, padding = 0),
        # Hint : Through this conv-layer, the input image
        #        Consider stride, kernel size, padding and
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 0)
)
```

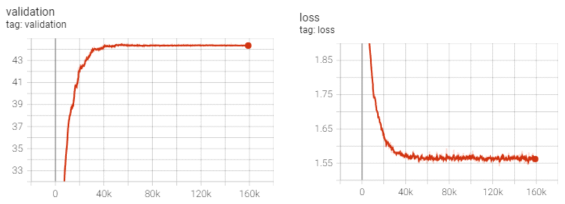


```
self.layer4 = nn.Sequential(
    ###########################################
    ############# fill in here (20 points)
    ####### you can refer to the 'layer2' above
    ResidualBlock(512, 256, 1024, downsample = False),
    ResidualBlock(1024, 256, 1024, downsample = False),
    ResidualBlock(1024,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

Now let's look at the results of running the model in practice. Note that we use pre-trained model, and fine-tuning only 1 epoch. Thanks to pre-trained model, test accuracy would be about 80%. For more detail, let’s compare the results of running 20 epochs after loading the checkpoint and without it. Blue is vanilla model, and orange is fine-tuning model. Accuracy is 82% and 40%, which is about twice the difference. Then, is the pre-trained model the resnet-50 we are currently learning? If right, accuracy of vanilla model will converge to 82%, or above. To check this, we learn vanilla resnet-50 enough, specifically for 320 epoch. Refer to graph below.



In practice, both loss and validation decreased sharply at the beginning, but it can be seen that it converges to a constant value as the epoch increases. In addition, we can see that it goes down very smoothly at the beginning and then both loss and validation vibrate as going back. It means model converges to optimal, and limitation of this model is performed about 45% accuracy. From these results, we can infer 3 facts. First, our pretrained model is not resnet-50, and more complicated, like resnet-180, se-net, or more recent model. Secondly, the process that we've done now is one of the most frequently used methods in modern deep learning. More specifically, the process of identifying the meaning of each word and optimizing it on our dataset often occurs in NLP. In this process, it is better to use the information of the existing pre-trained language model rather than new learning, which called **transfer learning**. Finally, if the actual model is simple, it can be seen that no matter how much data there is, it suffers from high bias. (at lecture note 3, learning theory)


validation
tag: validation
40k
80k
120k
160k
loss
tag: loss
40k
80k
120k
160k