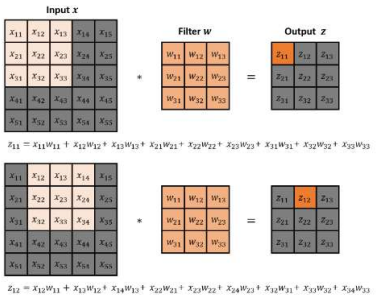
Summary Report [Week 11]. Convolutional Neural Networks (CNN)

20190106 KimByungjun (김병준)

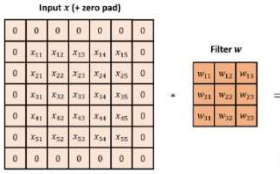
● Exercises

이번 Exercise 에선 CNN model 을 구현 후, CIFAR-10 dataset 에 대한 classification 을 진행하는 것이 목표이다. 단일 Filter 및 단일 channel input 에 대해 먼저 알아보자.

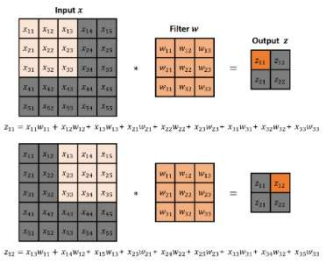
CNN 내 convolutional layer 는 fully-connected layer 와 달리 input 의 local 한 일부 부분만에 대하여 연산(합성곱)을 진행한다**([Fig 1])**. 이 때, output data size 의 변화없이 여러 합성곱 연산을 진행할 수 있도록, input data edge 에 0 을 채워 넣는 zero-padding**([Fig 2-1])**, 반대로 filter(kernel)의 적용 간격을 늘려 downsampled convolution 을 진행하는 stride(이 때, output size 는 감소)를 option 으로 적용할 수 있다**([Fig 2-2])**. 2D convolutional layer 는 pytorch 상에서 nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)으로 구현할 수 있으며, stride 의 기본값은 1, padding 은 0 이다. 만일, output size 와 input size 가 동일하도록 padding 을 넣고 싶다면 padding='same'으로 설정하면 된다.



[Fig 1] 2D Convolutional layer



[Fig 2-1] (Above) Zero-padding Applied



[Fig 2-2] (Right) Stride Applied

1) Implement a convolutional layer with a 7x7 kernel and stride 1 so that the output shape matches the input shape.

<Explanation>

Input 의 크기가 N x N, Kernel_size 를 K(즉, kernel 의 형태는 K x K), Stride 의 값을 S, Padding 의 값을 P 라고 할 때, K Padding 이 적용된 Input 은 (N + 2P) x (N + 2P) 의 형태를 가지며 그 안에 S 마다 간격을 둔 길이 K 의 Kernel 한 변은 $\frac{N+2P-K}{S}$ 가 정수 일 때, $\frac{N+2P-K}{S}+1$ 번, 정수가 아니라면 $\frac{N+2P-}{S}$ 번 들어간다. 정수인 경우를 가정한다면, output size 는 $(\frac{N+2P-K}{S}+1)\times(\frac{N+2P-}{S}+1)$ 이다.

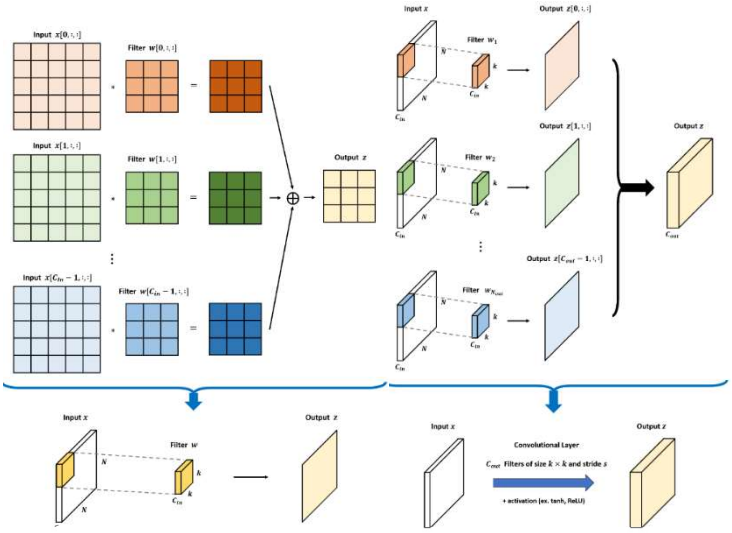
```
[12]  # Make the output have the same size as the input.
      # - Use a 7x7 kernel with stride 1.
      # - Use a scalar value instead of 'same' in the padding option.
      x = torch.randn([1, 1, 32, 32])  # Input (shape = (batch_size=1, n_channels=1, width=32, height=32))

      """ ### Implement the code yourself ### """
      # equivalent to nn.Conv2d(1, 1, kernel_size=7, padding='same')
      conv5 = nn.Conv2d(1,1, kernel_size=7, stride=1, padding=3)
      ## without padding : output size = 32-7+1=26
      ## To make the output size same as input (32),
      ## put padding with width 3 so that the size increase 26 to 26+3×2=32.
      """ ######################################## """

      z5 = conv5(x)
      print(z5.shape)     # This should be torch.Size([1, 1, 32, 32])

      torch.Size([1, 1, 32, 32])
```

[Fig 3] Code & Result for Exercise 1

Exercise 1 에서는 N=32, K=7, S=1 일 때 output size 도 32 x 32 가 나오도록 padding 옵션에 scalar 값을 넣는 게 목적이다. padding = 'same' 과 같은 결과를 얻기 위해, 위 식을 적용하면 $\frac{N+2P-K}{S}+1=\frac{32+2P-7}{1}+1=32 \Longrightarrow 26+2P=32, P=3$이다. 즉, input 의 edge 에 두께 3 의 0 을 추가해줘야 하며, 이를 구현한 것이 **[Fig 3]**이다.

Result. Output z5 의 shape 은 Input 과 같은 shape 인 torch.Size([1, 1, 32, 32])이 나타난다.

[Fig 4]와 같이 Input 이 multiple channel 을 가지게 되는 경우를 살펴보자. 예를 들면, RGB 의 3 개의 channel 을 가지는 input 에 대해 single convolutional filter 은 동일한 값의 depth 3 filter 가 되어야 한다. 즉, single filter 가 적용되는 경우, input 도 두께가, filter 도 두께가 있는 상황이며 그 두 두께가 일치해야 한다. 여기에 **[Fig 5]**와 같이 추가로 filter 의 개수를 늘리고자 한다. Convolutional filter 은 input 의 특징을 나타내며, 그 개수가 많을수록 input 이 가진 서로



[Fig 4] Figure for 3D input with single convolutional filter

[Fig 5] Figure for 3D input with multiple convolutional filters

다른 특징들을 더 담아낸다고 볼 수 있다. C_{in}의 channel 수를 가진 input 과 두께 C_{in}의 filter C_{out} 개가 합성곱 연산되면, output 의 channel 수는 C_{out}이 된다.

특정 Convolutional layer 통과 후, overfitting 방지를 위한 regularization 기법으로 Pooling, Dropout 이 있다. Pooling 은 여러 pixel 을 그 value 를 대표하는 대표값 하나로 줄이는 기법으로, Input data 를 작게 만들어 feature 수를 줄여 overfitting 을 방지한다. 만일, 그 대푯값이 최대값이라면 max-pooling, 평균이라면 mean-pooling 이라 부른다. Pooling 시, kernel_size (K 라 하자) 옵션을 부여하여 선택 pixel 의 수를 지정할 수 있고, 이 경우 Input size 가 N x N 이었다면, pooling 이후 size 는 N/K x N/K 가 된다 (이 때, batch 크기와 channel 수에는 변화 없다). Dropout 은 각 pixel 의 값을 dropout rate p 의 확률로 0 으로 만든다. 단, 이는 training 때만 적용되고, evaluation 시에는 dropout 이 적용되지 않는다.

Batch normalization 은 Input batch 각각에 대한 convolutional layer 통과 output 을 normalization 진행 후, learnable parameter γ, β 로 scaling, shifting 하는 과정이다. 이는, 모든 feature 의 scale 을 맞추어 data 를 안정화시킨다.

이제 CIFAR-10 dataset 을 10 개의 label(숫자 0~9 로 표현)로 분류하는 classification model 을 만들어보자. Dataset 은 [Fig 6]와 같이 여러 대상으로 구성되어 있다. 해당 Exercise 에선 batch size 가 128, 그리고 RGB 3 channel, size 32 x 32 로 구성된 input 을 사용한다.

[Fig 6] CIFAR-10 Dataset Visualization

2) Implement *conv1, conv2, activ, max_pooling,* and *liner,* and implement the forward process of the model. (Below is the Specifications.)

- **conv1 : 16 filters of size 3x3, followed by ReLU, and max-pooling**
- **conv2 : 32 filters of size 3x3, followed by ReLU, and max-pooling**
- **Both conv1, conv2 have stride 1, zero-padding for size matching between input and output**
- **linear : no activation function**

<Explanation>

위 Specification 에 맞게 conv1 은 input channel 3, output channel 16 으로 설정하며, stride 는 1, input 과 output size 를 동일하게 만들기 위해 padding 은 'same'으로 설정한다. 이 때, output channel 인 16 이 다음 conv2 의 input channel 이 되고, conv2 의 output channel 은 32 로 설정한다(나머지 option 들은 동일하다).

Activation function 은 nn.ReLU()를 쓰며, max_pooling 은 kernel_size 를 2 로 두어 매 pooling 마다 data 를 가로도 절반, 세로도 절반으로 축소시킨다.

```
[16] # Define model
     class CNNClassificationModel(nn.Module):
         def __init__(self):
             super(CNNClassificationModel, self).__init__()

             """ ### Implement the code yourself ### """
             self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding='same')
             self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding='same')
             self.activ = nn.ReLU()                    # Activation (ReLU)
             self.max_pooling = nn.MaxPool2d(kernel_size = 2)        # Max-pooling
             self.linear = nn.Linear(32*8*8,10)                # Linear layer (use n

             """ ################################### """

         def forward(self, x):

             """ ### Implement the code yourself ### """

             out = self.conv1(x)          # conv. layer 1
             out = self.activ(out)        # activation
             out = self.max_pooling(out)          # max-pooling

             out = self.conv2(out)          # conv. layer 2
             out = self.activ(out)          # activation
             out = self.max_pooling(out)          # max-pooling

             out = out.reshape(out.size(0),-1)       # Reshape the output to shape (

             out = self.linear(out)          # Linear layer

             """ ################################### """

             return out

     model = CNNClassificationModel().to(device)
```

[Fig 7] Code for Exercise 2

Activation function 은 nn.ReLU()를 쓰며, max_pooling 은 kernel_size 를 2 로 두어 매 pooling 마다 data 를 가로도 절반, 세로도 절반으로 축소시킨다.

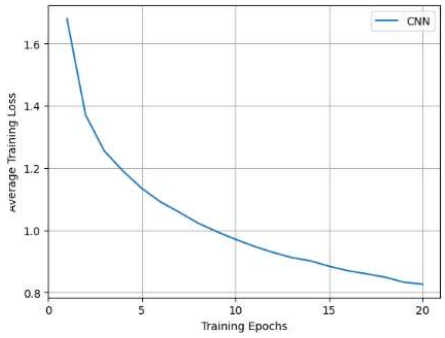
conv1 → activ → max_pooling → conv2 → activ → max_pooling → reshaping → linear 의 순서로 진행된다. 즉, linear layer 를 통과하기 직전의 output 은 channel 수는 32 가 되었고, max_pooling 을 2 번 적용하였으므로 32 x 32 size 가 8 x 8 이 되었다. 즉, batch 내 단일 data 의 shape 은 32 x 8 x 8 이며, linear layer 을 들어가기 전에 이를 flatten 시켜 1D 형태로 바꿔주어야 한다. 이에 맞게, __init__ 에서는 nn.Linear(32*8*8, 10) (label 이 10 개이므로), forward process 에서 reshaping 을 위해서 out = out.reshape(out.size(0),-1) 로 바꿔준다(out.size(0)은 batch 크기인 128, -1 자리에는 32*8*8 과 같다).

위 Specification 에 적힌 대로 순서에 맞게 forward process 를 구성하면 **[Fig 7]**과 같이 code implementation 을 하게 된다.

Model training 을 위해 loss function 은 Cross-Entropy 를, optimizer 는 Adam optimizer 를 사용하며 CIFAR-10 dataset 에 대한 classification CNN model 을 training 시키면 **[Fig 8-1]**과 같은 validation accuracy 와 **[Fig 8-2]**와 같은 average training loss 를 얻을 수 있다. 마지막 epoch 를 보면 model 은 약 66.57%의 validation accuracy 를 보임을 알 수 있다.

```
Epoch:   1 | Iter: 351 | Average Loss: 1.679 | Elapsed Time:   14.5 s
Epoch:   1 | Validation Accuracy: 48.200 %

Epoch:   2 | Iter: 351 | Average Loss: 1.369 | Elapsed Time:   22.7 s
Epoch:   2 | Validation Accuracy: 54.250 %

Epoch:  19 | Iter: 351 | Average Loss: 0.833 | Elapsed Time:  167.2 s
Epoch:  19 | Validation Accuracy: 66.230 %

Epoch:  20 | Iter: 351 | Average Loss: 0.826 | Elapsed Time:  175.9 s
Epoch:  20 | Validation Accuracy: 66.570 %
```

[Fig 8-1] (Above) CNN model validation accuracy



[Fig 8-2] (Right) Average Training Loss

```
# Print test accuracy
print('Test Accuracy: {:5.3f} %'.format(correct / total * 100))
```

```
Test Accuracy: 72.951 %
```

[Fig 9] Test accuracy for CNN model

Training 이 완료된 model 을 가지고, test 를 진행한 결과, [Fig 9]와 같이 **72.951%**의 test accuracy 를 보인다.

● Discussion

1) Using scalar scale for padding option

만일, padding 에 'same' 대신에 scalar 값을 넣을려면 어떻게 해야할까? 앞서 설명했듯이, Input 의 크기 : N x N, Kernel_size : K, Stride : S, Padding : P 라고 할 때, output size 는 $(\frac{N+2P-K}{S}+1)\times(\frac{N+2P-K}{S}+1)$ 이므로, $\frac{N+2P-}{S}+1=\frac{N+2P-}{1}+1=N+2P-2$ 로 정리할 수 있고, input size N 에 관계없이 P = 1 이면, input size 와 output size 는 같아지게 된다(물론 channel 은 별개이다). 따라서, padding 옵션에 'same' 대신 1 을 넣어도, 같은 structure 의 model 이 구현되며, 73.933 %의 test accuracy 를 보였다.

```
# Define model
class CNNClassificationModel(nn.Module):
    def __init__(self):
        super(CNNClassificationModel, self).__init__()

        """ ### Implement the code yourself ### """
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.activ = nn.ReLU()                  # Activation (ReLU)
        self.max_pooling = nn.MaxPool2d(kernel_size = 2)          # Max-pool
        self.linear = nn.Linear(32*8*8,10)                # Linear layer (u

        """ ################################### """
```

[Fig 10] Change padding option in scalar value