Assignment 3

Arturo Abril Martinez (2813498)

December 2024

1 Answers

question 1 The learnable weights of the conv layer are represented as a tensor of shape:

`(output_channels, input_channels, kernel_height, kernel_width)`

This is the set of *filters* of the convolutional layer. More explicitly (and excluding the bias terms) we will have `W0, W1, W2, ...` and so on up until the number given by `output_channels`. The total number of parameters will then be:

`kernel_width * kernel_height * input_channels * output_channels.`

The pseudocode to implement the convolution is:

```
Y = \
torch.zeros((batch_size, output_channels, output_height, \
    output_width))

for b in range(batch_size):
    for ch in range(output_channels)
        for row in range(output_height):
            for col in range(output_width):
                Y[b,ch,row,col] = \
                ( \
                    X[b,:,row*S:hk+row*S,col*S:wk+col*S] \
                    * W[ch,:,:,:]
                ).sum()
return Y
```

Where `S` represents the stride and `hk` and `wk` the kernel (or filter) height and width respectively. Note that the multiplication inside the `sum()` is element-wise, and in the pseudocode `W` is the set of all weight tensors, so that `W[0,:,:,:] = W0`. *Note: I assume the input images are already padded, if they need to be.*

question 2 Given that the size of the input is (C_{in}, H_{in}, W_{in}), we can calculate the size of the output as:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

H_{out} can be calculated using the same formula by symmetry, changing $W_{in} \to H_{in}$. The size of the output is then $(C_{out}, H_{out}, W_{out})$ where C_{out} can be chosen to be any number. K, S and P are the kernel size, stride and padding respectively[1].

question 3 Below is the pseudocode for the unfold operation. `p`, `k` and `S` represent the total number of patches, number of values per patch and stride respectively. `w` (`wk`) and `h` (`hk`) are the input (kernel) width and height respectively. *Note: I assume the input images are already padded, if needed to be.*

```
Y = torch.zeros((batch_size, p, k))
for b in range(batch_size):
    # iterate over rows in the output
    # each row will be a flattened image patch
    for row in range(p):
        # select patch
        i = row//w_out  # transition row every `w_out` patches
        j = row%(w-wk+1) # when `row` exceeds maximum, start over
        start_row, start_col = i*S, j*S
        patch = X[
            b, :, start_row : start_row+hk, \
            start_col : start_col+wk
        ]

        # flatten patch
        counter = 0
        for ch_patch in range(input_channels):
            for row_patch in range(hk):
                for col_patch in range(wk):
                    Y[b, row, counter] = \
                    patch[ch_patch, row_patch, col_patch]
                    counter += 1
# This way the output will have size (batch_size, p, k)
return Y
```

[1] I have ommited the batch size in the calculation. Of course, for every tensor in the input batch we would have an output tensor, which dimensions can be calculated with the formula provided.

A Appendix