Max Pooling

Understand how padding layers can improve a CNN's performance.

Chapter Goals:

- Learn about max pooling and its purpose in CNNs
- Apply max pooling to the output of the convolution layer

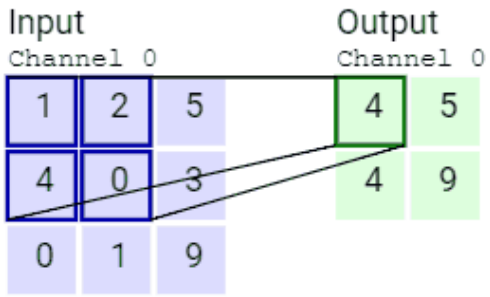
A. Purpose of pooling

While the convolution layer extracts important hidden features, the number of features can still be pretty large. We can use *pooling* to reduce the size of the data in the height and width dimensions. This allows the model to perform fewer computations and ultimately train faster. It also prevents overfitting, by extracting only the most salient features and ignoring potential distortions or uncommon features found in only a few examples.

B. How pooling works

Similar to a convolution, we use filter matrices in pooling. However, the pooling filter doesn't have any weights, nor does it perform matrix dot products. Instead, it applies a reduction operation to subsections of the input data.

The type of pooling that is usually used in CNNs is referred to as *max pooling*. The filters of max pooling use the *max* operation to obtain the maximum number in each submatrix of the input data. An example of max pooling is shown below:



Max pooling with a 2x2 filter and stride size of 1. Note that the output data has reduced dimensions compared to the input data.

Other types of pooling include min pooling and average pooling, which use the *min* and *average* operations, respectively, over subsections of the input data.

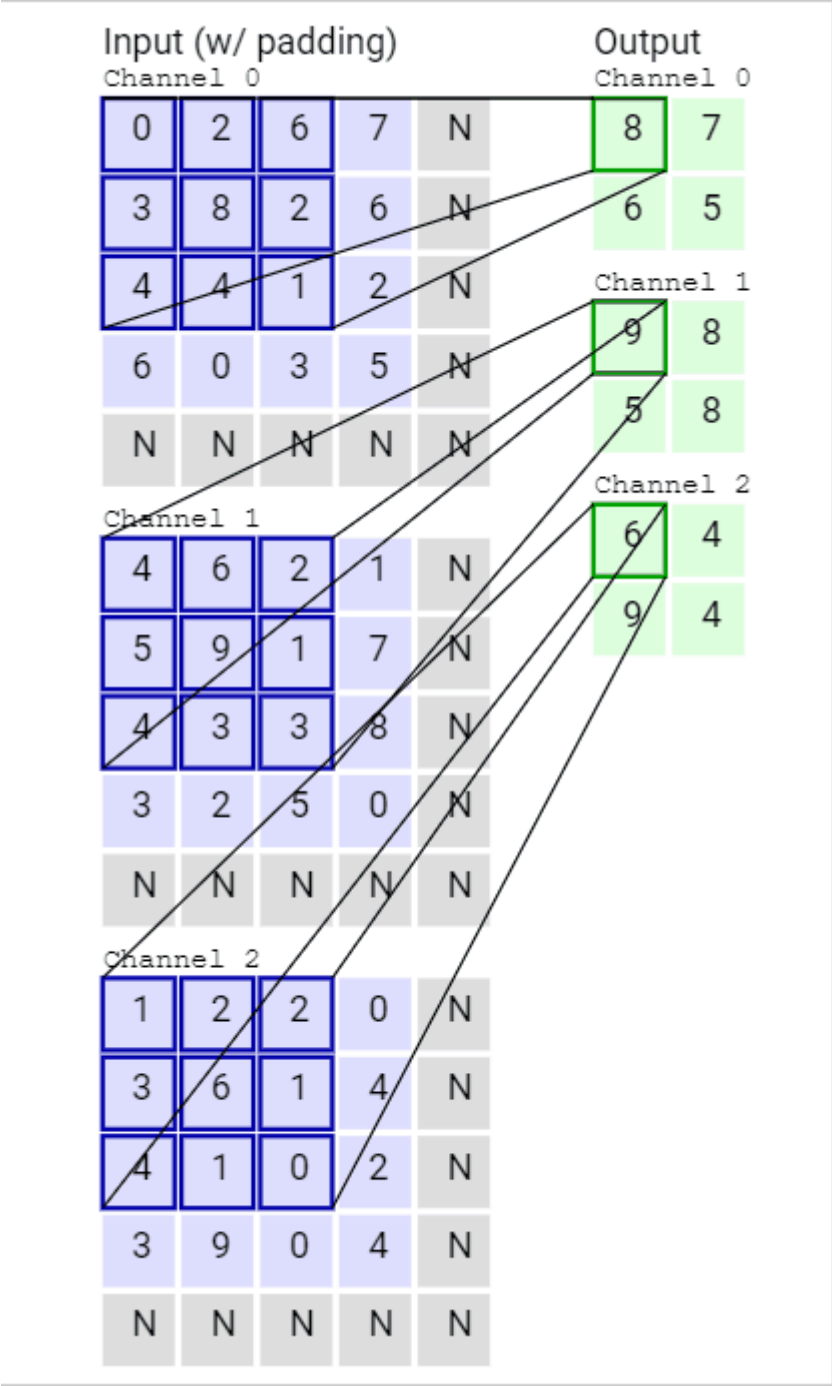
For input data with dimensions $H_{in} \times W_{in}$ the output of pooling with filter dimensions $H_F \times W_F$ and stride size S has the following height and width:

$$H_{out} = \left\lceil \frac{H_{in} - H_F + 1}{S} \right\rceil$$

$$W_{out} = \left\lceil \frac{W_{in} - W_F + 1}{S} \right\rceil$$

C. Padding

Similar to convolutions, we may want to pad our input data prior to pooling. We pad our data with a value dependent on the pooling operation. For example, in max pooling we pad each matrix with -∞. Since -∞ is smaller than every number, it allows us to resize the input data without adding any distortions when pooling. An example of max pooling with padding is shown below:



Max pooling with padding for input data with dimension 4x4. The "N" character represents -∞.

When we use padding, the output dimensions do not depend on the filter dimensions anymore. Specifically, for input data with dimensions $H_{in} \times W_{in}$, the output of padded pooling with a stride size of S has dimensions

$$H_{out} = \left\lceil \frac{H_{in}}{S} \right\rceil$$

$$W_{out} = \left\lceil \frac{W_{in}}{S} \right\rceil$$

Time to Code!

We'll apply max pooling to the output of our first convolution layer from the previous chapter. In TensorFlow, a max pooling layer is implemented with the

`tf.layers.max_pooling2d` function. The function takes in the following required arguments:

- `inputs`: The input data tensor.
- `pool_size`: The height and width dimensions of the pooling filter.
- `strides`: The stride size for the kernel matrix. Can be a single integer (same stride size for vertical/horizontal) or a tuple of 2 integers (manually specified vertical/horizontal stride sizes). The first element of the tuple is the vertical stride, the second is the horizontal stride.

The `tf.layers.max_pooling2d` function also takes in a keyword argument for padding. The values are either `'valid'` or `'same'`, with the default value being `'valid'`.

Set `pool1` equal to `tf.layers.max_pooling2d` applied with `conv1` as the inputs. The function will use a pooling size of `[2, 2]` and a stride size of 2, both horizontally and vertically. We'll also set the `name` argument to `'pool1'`.

```
import tensorflow as tf

class MNISTModel(object):
    # Model Initialization
    def __init__(self, input_dim, output_size):
        self.input_dim = input_dim
        self.output_size = output_size

    # CNN Layers
    def model_layers(self, inputs, is_training):
        reshaped_inputs = tf.reshape(
            inputs, [-1, self.input_dim, self.input_dim, 1])
        # Convolutional Layer #1
        conv1 = tf.layers.conv2d(
            inputs=reshaped_inputs,
            filters=32,
            kernel_size=[5, 5],
            padding='same',
            activation=tf.nn.relu,
            name='conv1')
        # CODE HERE
```