Padding

Create custom padding layers to maintain consistency in padding.

Chapter Goals:

- Implement consistent padding based only on kernel size

A. Padding consistency

In the **CNN** section, we mentioned that TensorFlow uses the minimum amount of padding necessary when we set `padding='same'` for our convolution or pooling layers. However, the amount of padding used depends on the kernel size, stride size, and input height/width. This leads to inconsistent padding amounts at different layers of our model.

Since ResNet has so many layers and follows a repetitive structure, we want to maintain as much consistency as possible. Therefore, when we pad our data for a convolution layer, we want the padding to be solely based on the size of the kernel.

Time to Code!

In this chapter you'll be completing the custom padding function, `custom_padding`.

We pad the same number of rows as we do columns, which will be one less than the kernel dimension.

Set `pad_total` equal to one less than `kernel_size`.

We define padding "before" as padding to the left and top of the data, while padding "after" is padding to the right and bottom. Similar to TensorFlow, we'll distribute the padding as evenly as possible.

Set `pad_before` equal to half (rounded down) of `pad_total`.

Set `pad_after` equal to the remaining amount of padding.

The function we use to pad our data is `tf.pad`. The function takes in two required arguments:

- `tensor`: The tensor that we apply padding to.
- `paddings`: A tensor of integers with shape `(num_dims, 2)`, where `num_dims` represents the number of dimensions of the input tensor. Note that `paddings` can also be a list of lists, where the outer list has length `num_dims` and each inner list contains two integers.

Since our data is either NHWC or NCHW format, `num_dims = 4`. However, depending on the channel placement, the `paddings` argument to `tf.pad` can differ slightly. Therefore, we use an `if...else` code block to cover both data formats.

Create an `if...else` with `self.data_format == 'channels_first'` as the `if` condition.

The `paddings` argument will have shape `(4, 2)`, since `num_dims = 4`. We'll use a list of four smaller lists, each containing two integers. Each two integer list corresponds to a specific dimension in our data, with the first integer representing the amount of padding "before" and the second integer representing the amount of padding "after".

Since we only apply padding to the H and W dimensions of our data, we set the padding for the N and C dimensions to `[0, 0]`. The N and C dimensions correspond to indexes 0 and 1 of the `paddings` list in NCHW (channels-first) format. For the H and W dimensions, we set the padding to `[pad_before, pad_after]`.

Inside the `if` block, set `padded_inputs` equal to `tf.pad` with `inputs` as the first argument and the NCHW `paddings` list as the second argument.

For NHWC (channels-last) format, the C dimension is index 3 of the `paddings` list while the H and W dimensions are indexes 1 and 2, respectively. So indexes 0 and 3 of the NHWC `paddings` list will contain `[0, 0]` and indexes 1 and 2 will contain `[pad_before, pad_after]`.

Inside the `else` block, set `padded_inputs` equal to `tf.pad` with `inputs` as

the first argument and the NHWC `paddings` list as the second argument.

```
import tensorflow as tf

block_layer_sizes = {
    18: [2, 2, 2, 2],
    34: [3, 4, 6, 3],
    50: [3, 4, 6, 3],
    101: [3, 4, 23, 3],
    152: [3, 8, 36, 3],
    200: [3, 24, 36, 3]
}

class ResNetModel(object):
    # Model Initialization
    def __init__(self, min_aspect_dim, resize_dim, num_layers, output_size,
        data_format='channels_last'):
        self.min_aspect_dim = min_aspect_dim
        self.resize_dim = resize_dim
        self.filters_initial = 64
        self.block_strides = [1, 2, 2, 2]
        self.data_format = data_format
        self.output_size = output_size
        self.block_layer_sizes = block_layer_sizes[num_layers]
        self.bottleneck = num_layers >= 50

    # Custom padding function
    def custom_padding(self, inputs, kernel_size):
        # CODE HERE
        pass

    # Custom convolution function w/ consistent padding
    def custom_conv2d(self, inputs, filters, kernel_size, strides, name=None):
        if strides > 1:
            padding = 'valid'
            inputs = self.custom_padding(inputs, kernel_size)
        else:
            padding = 'same'
        return tf.layers.conv2d(
            inputs=inputs, filters=filters, kernel_size=kernel_size,
            strides=strides, padding=padding, data_format=self.data_format,
            name=name)
```