Model Layers

Learn about the high-level architecture of SqueezeNet.

Chapter Goals:

- Learn the high-level architecture of SqueezeNet
- Understand the use of non-fire module convolution layers

A. Overview

As mentioned in the previous chapter, we'll be building a condensed version of the SqueezeNet model. The differences between our model and the original are:

- Our model uses only the first 4 fire modules from the original SqueezeNet (rather than all 8)
- The initial convolution layer for our model uses fewer filters (64 vs. 96) and a smaller kernel size (3x3 vs. 7x7)
- The max pooling layers in our model use smaller filters (2x2 vs. 3x3)

The changes are mainly due to the fact that our dataset is CIFAR-10, which has significantly smaller images and fewer categories than ImageNet, the dataset used to train the original SqueezeNet. We go in-depth on the ImageNet dataset in the **ResNet** section.

B. Initial layers

We start off with a regular convolution layer, rather than a fire module, to apply the initial filters to our image data. This is because we don't want to start off with squeezing our image data and potentially losing out on important general features. It's usually a good idea for any CNN to start off with a regular convolution layer.

Likewise, it's a good idea to apply max pooling after the initial convolution layer. Although it's less of a need with small images like in the CIFAR-10 dataset, reducing the dimension of the initial data helps our model train faster dataset, reducing the dimension of the initial data helps our model train faster and pick up on important features. In our code, each of the max pooling layers

has a stride size of 2 and a filter size of 2x2, so the data's width and height are halved after each max pooling layer.

Time to Code!

In this chapter you'll be building the first convolution+pooling layer in the `model_layers` function (line **30**).

We start our model off with a regular convolution layer.

Set `conv1` equal to `self.custom_conv2d` applied with `inputs` as the first argument, `64` as the number of `filters`, `[3, 3]` as the `kernel_size`, and `'conv1'` as the `name`.

Next we apply max pooling to the convolution layer's output. We provide a wrapper function for max pooling called `custom_max_pooling2d`. This automatically sets the `strides` argument to `2` and the `pool_size` (i.e. filter size) to `[2, 2]`.

Set `pool1` equal to `self.custom_max_pooling2d` applied with `conv1` as the first argument and `'pool1'` as the `name`.

```
import tensorflow as tf

class SqueezeNetModel(object):
    # Model Initialization
    def __init__(self, original_dim, resize_dim, output_size):
        self.original_dim = original_dim
        self.resize_dim = resize_dim
        self.output_size = output_size

    # Convolution layer wrapper
    def custom_conv2d(self, inputs, filters, kernel_size, name):
        return tf.layers.conv2d(
            inputs=inputs,
            filters=filters,
            kernel_size=kernel_size,
            activation=tf.nn.relu,
            padding='same',
            name=name)

    # Max pooling layer wrapper
    def custom_max_pooling2d(self, inputs, name):
        return tf.layers.max_pooling2d(
            inputs=inputs,
            pool_size=[2, 2],
            strides=2,
            name=name)

    # Model Layers

    # inputs: [batch_size, resize_dim, resize_dim, 3]
    def model_layers(self, inputs, is_training):
        # CODE HERE
        pass
```