Stacking Fire

Stack multiple fire modules together by creating a utility function.

Chapter Goals:

- Write a utility function to stack multiple fire modules

A. Utility function

While the SqueezeNet model uses very few parameters, it still has many layers. The original architecture, which was built for the larger ImageNet dataset, uses 8 fire modules. Our model is a condensed version, and only uses 4. However, that is still several fire modules, making it useful to write a utility function that can stack multiple layers.

In general, when we deal with more complex model architectures, there are going to be repetitions of the main building blocks in the model. Rather than writing layers upon layers of the same code, we can create a utility function that allows us to combine multiple building blocks. This is especially helpful for models that have dozens or even hundreds of layers, which is the case in the **ResNet** section.

Time to Code!

In this chapter, you'll be stacking fire modules by using the `fire_module` function from the previous chapter. The function that you'll complete which performs this task is `multi_fire_module` (line **41**).

The `params_list` argument is a list of tuples, where each tuple represents the arguments for a fire module, i.e. `(squeeze_depth, expand_depth, name)`. We'll loop through the `params_list` to create each of the fire modules.

Create a `for` loop that goes through each `params` tuple in `params_list`.

The `layer` argument is the initial input data for our multi-fire module. We'll use it as the `inputs` argument for each fire module, as well as set it to the output of each fire module. This way we can continuously reuse the same

variable.

Inside the `for` loop, set `layer` equal to `self.fire_module` applied with `layer` as the first argument and the remaining three arguments coming from `params`.

After the end of the `for` loop, `layer` represents the output of the final fire module.

Outside the `for` loop, return `layer`.

```
import tensorflow as tf

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

    # Convolution layer wrapper
    def custom_conv2d(self, inp
        return tf.layers.conv2
            inputs=inputs,
            filters=filters,
            kernel_size=kernel_
            activation=tf.nn.re
            padding='same',
            name=name)

    # SqueezeNet fire module
    def fire_module(self, input
        with tf.variable_scope(
            squeezed_inputs = s
                inputs,
                squeeze_depth,
                [1, 1],
                'squeeze')
            expand1x1 = self.cu
                squeezed_inputs
                expand_depth,
                [1, 1],
```