Adding Fire

Add fire modules to the model and learn about delayed downsampling.

Chapter Goals:

- Add a multi-fire module block to the model
- Learn about delayed downsampling

A. Delayed downsampling

While reducing the data dimensions via max pooling can help our model training, strategic placement of max pooling layers can improve the SqueezeNet model's accuracy. Rather than applying max pooling within the first fire module (e.g. after the squeeze layer), we instead apply it after the first two fire modules.

Placing the max pooling layers later in the model structure is referred to as delayed downsampling. We wait until later in the model structure to downsample the data (i.e. reduce height/width dimensions through pooling), so the earlier convolution layers can have a larger dimension input. The creators of SqueezeNet showed that having larger dimension inputs for more layers in the model can help improve its accuracy.

Time to Code!

In this chapter, we'll continue building the model_layers function by adding the first multi-fire module.

We'll start by setting the list of parameter tuples to describe our fire modules. The list will contain two tuples representing the fire_module arguments. For both fire modules, we use 32 as the squeeze depth and 64 as the expand depth. The first fire module will be named 'fire1' while the second will be named 'fire2'.

Set fire_params1 equal to a list of two tuples with the specified values.

Now we apply the multi-fire module block with our list of tuple parameters.

Set multi_fire1 equal to self.multi_fire_module with pool1 as the first argument and fire_params1 as the second argument.

We perform max pooling on the output of the multi-fire module block.

Set pool2 equal to self.custom_max_pooling2d applied with multi_fire1 as the first argument and 'pool2' 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)
   # SqueezeNet fire module
   def fire_module(self, inputs, squeeze_depth, expand_depth, name):
        with tf.variable_scope(name):
            squeezed_inputs = self.custom_conv2d(
                inputs,
                squeeze_depth,
                [1, 1],
                'squeeze')
            expand1x1 = self.custom_conv2d(
                squeezed_inputs,
                expand_depth,
                [1, 1],
                'expand1x1')
            expand3x3 = self.custom_conv2d(
                squeezed_inputs,
                expand_depth,
                [3, 3],
                'expand3x3')
            return tf.concat([expand1x1, expand3x3], axis=-1)
```

```
# Utility function for multiple fire modules
def multi_fire_module(self, layer, params_list):
    for params in params_list:
        layer = self.fire_module(
            layer,
            params[0],
            params[1],
            params[2])
    return layer
# Model Layers
# inputs: [batch_size, resize_dim, resize_dim, 3]
def model_layers(self, inputs, is_training):
    conv1 = self.custom_conv2d(
        inputs,
        64,
        [3, 3],
        'conv1')
    pool1 = self.custom_max_pooling2d(
        conv1,
        'pool1')
    # CODE HERE
```









ני