Initialization

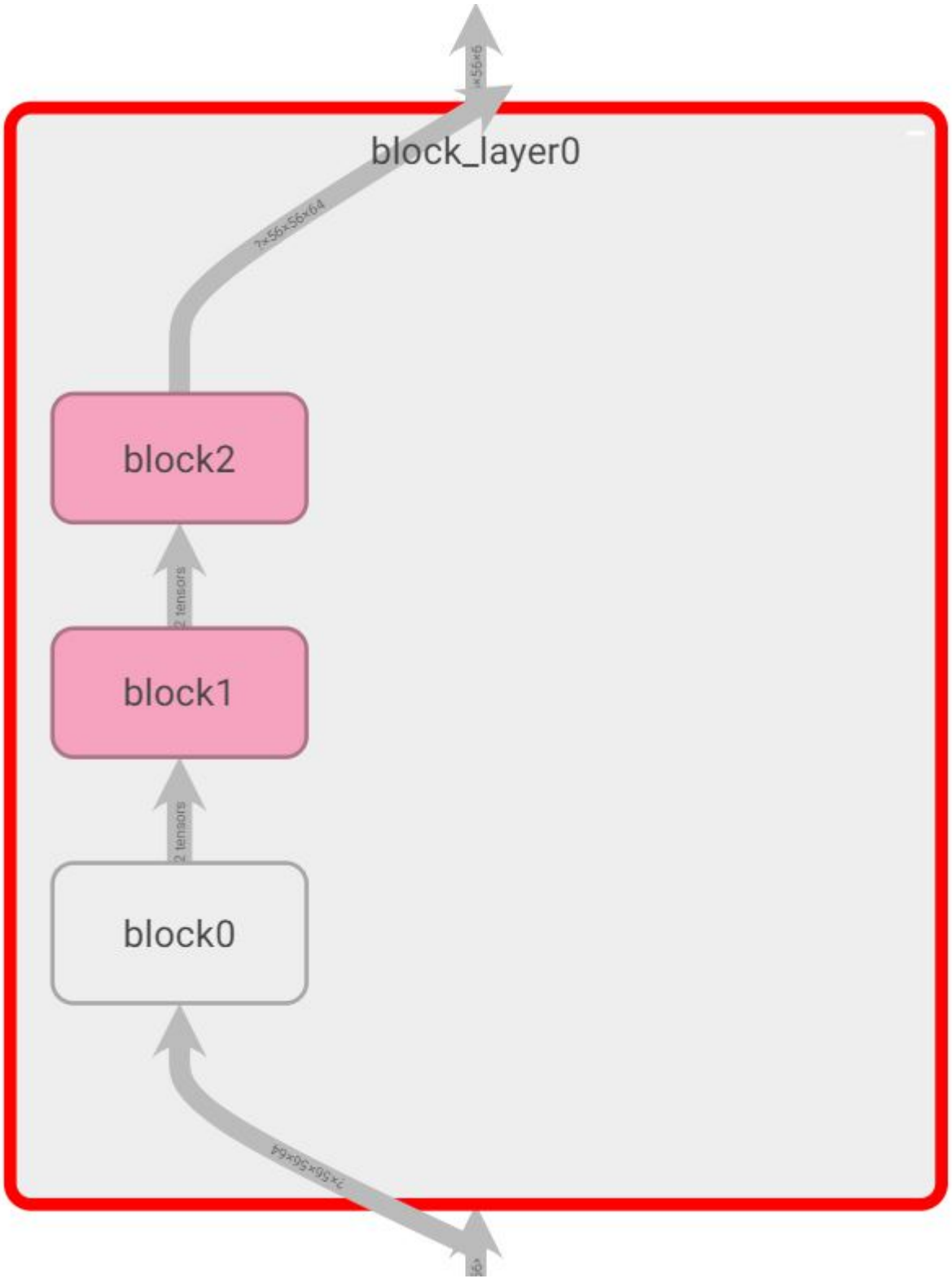
Discover how the ResNet model is structured for each of its configurations.

Chapter Goals:

- Learn about the ResNet model structure
- Understand the difference between channels-first and channels-last

A. Model overview

The ResNet model follows a repetitive structure built around its main building block. The overall model contains four layers of blocks, where the number of blocks per layer depends on how large we want our ResNet model to be. In our code, the `block_layer_sizes` dictionary gives the number of blocks for each of the four block layers, based on the total number of weight layers in the ResNet model.



Structure of a block layer with 3 blocks.

There are six options for how many weight layers (convolution + one fully-connected layer) are in the ResNet model: 18, 34, 50, 101, 152, 200. When the number of layers is less than 50, the model uses a regular building block. However, when the number of layers is 50 or more, the model uses a *bottleneck* block. We'll discuss what a bottleneck block is in a later chapter.

B. Filters and strides

The number of filters used in the model's convolution layers starts off at 64. In each subsequent block layer, we double the number of filters used. This follows the practice of increasing the number of filters for convolution layers deeper in the model.

Most of the blocks in ResNet use a stride size of 1 for their convolution layers. Only the first block of each block layer may use a different stride size. The stride sizes for the first block in each layer are defined in `self.block_strides`.

C. Channel placement

So far, all our model data has been in the NHWC format. This is referred to as *channels-last* placement because the channels are the last dimension. However, TensorFlow also supports *channels-first* placement, i.e. NCHW format. Since the NCHW format is optimized for GPU training and the NHWC format is a bit better when using CPUs, we create our model to allow for both formats.

When we initialize our model, the channel format (`'channels_first'` or `'channels_last'`) is specified through the `data_format` argument. This matches the `data_format` keyword argument that can be used in `tf.layers.conv2d`, `tf.layers.max_pooling2d`, and `tf.layers.average_pooling2d`.

Time to Code!

In this section of the course, you'll be creating the `ResNetModel` class, which represents a ResNet model.

Specifically in this chapter, you'll be completing the initialization of the `ResNetModel`. The `__init__` function is already completed with some variable initialization. Your task will be to set up the parts specific to the ResNet model: the blocks per layer and bottleneck usage.

We'll first set up the number of blocks to use per block layer.

Set `self.block_layer_sizes` equal to `block_layer_sizes[num_layers]`.

Now we set `self.bottleneck` to a boolean, which is `True` if the model uses bottleneck blocks and `False` if it uses regular blocks.

Set `self.bottleneck` equal to `num_layers >= 50`.

```
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
        # CODE HERE
```