# Fully-Connected

Understand how fully-connected layers can be used to aggregate and flatten data.

Chapter Goals:

- Convert the NHWC format data into a batch of flattened vectors
- Apply a fully-connected layer to the flattened data

## A. Fully-connected layer

We apply a fully-connected layer of size 1024 (i.e. the number of neurons in the layer) to the output data of the second pooling layer. The number of units is somewhat arbitrary. Enough to be powerful, but not so much as to be too resource intensive. The purpose of the fully-connected layer is to aggregate the data features before we convert them to logits. This allows the model to make better predictions than if we had just converted the pooling output directly to logits. We'll use those logits in a later chapter

## B. Flattening

The data we have been using in our model is of the NHWC format. However, in order to use a fully-connected layer, we need the data to be a matrix, where the number of rows represents the batch size and the columns represent the data features. Similar to how we reshaped our data in the Reshaping chapter, we'll use `tf.reshape` again. This time, though, we'll be reshaping in the opposite direction and converting *from* NHWC to a 2-D matrix.

Since the first dimension remains the batch size, we'll still use -1 when reshaping. To calculate the size of the second dimension (i.e. the total number of data features in `pool2`), we'll use the inherent `shape` property of tensors in TensorFlow. The `shape` property gives us a `tf.TensorShape` object, which we can convert to a list of integers using its `as_list` function.

```
print(repr(data.shape))
print(repr(data.shape.as_list()))
```

Obtaining the shape of a batch data tensor. Note that "None" represents the (unknown) batch size.

The resulting list of integers contains the NHWC sizes of the tensor. The flattened data size is then just the product of the H, W, and C sizes.

## Time to Code!

In this chapter, we'll create a helper function for the `model_layers` function. The helper, `create_fc`, applies a fully-connected layer to the final max-pooling layer of the model.

We first need to get the height, width, and number of channels in `pool2` before we flatten it.

**Set `hwc` equal to `pool2.shape.as_list()`, sliced from index `1` to the end.**

Using the list of HWC sizes, we can get the total flattened size for the `pool2` data features.

**Set `flattened_size` equal to the product of the integers in `hwc`.**

We'll flatten the `pool2` tensor using `tf.reshape`. The new shape will have -1 in the first dimension, representing the batch size, and `flattened_size` in the second dimension.

**Set `pool2_flat` equal to `tf.reshape` with `pool2` as the first argument and the new shape as the second argument.**

We'll now apply a fully-connected layer to the flattened data.

**Set `dense` equal to `tf.layers.dense` applied with `pool2_flat` as the inputs, `1024` as the output size, `tf.nn.relu` as the `activation`, and `name` equal to `'dense'`.**
**Then return `dense`.**

```
import tensorflow as tf

class MNISTModel(object):
    # Model Initialization
    def __init__(self, input_dim, output_size):
        self.input_dim = input_dim
```

```
        self.output_size = output_size

    # Apply fully-connected layer
    def create_fc(self, pool2):
        # CODE HERE
        pass

    # CNN Layers
    def model_layers(self, inputs, is_training):
        reshaped_inputs = tf.reshape(
            inputs, [-1, self.input_dim, self.input_dim, 1])
        # Convolutional Layer #1
        conv1 = tf.layers.conv2d(
            inputs=reshaped_inputs,
            filters=32,
            kernel_size=[5, 5],
            padding='same',
            activation=tf.nn.relu,
            name='conv1')
        # Pooling Layer #1
        pool1 = tf.layers.max_pooling2d(
            inputs=conv1,
            pool_size=[2, 2],
            strides=2,
            name='pool1')
        # Convolutional Layer #2
        conv2 = tf.layers.conv2d(
            inputs=pool1,
            filters=64,
            kernel_size=[5, 5],
            padding='same',
            activation=tf.nn.relu,
            name='conv2')
        # Pooling Layer #2
        pool2 = tf.layers.max_pooling2d(
            inputs=conv2,
            pool_size=[2, 2],
            strides=2,
            name='pool2')
        dense = self.create_fc(pool2)
```