Solution

Here's how I did it. **NOTE**: there's more than 1 way to get the correct output shape. Your answer might differ from mine.

```
def conv2d(input):
    # Filter (weights and bias)
    F_W = tf.Variable(tf.truncated_normal((2, 2, 1, 3)))
    F_b = tf.Variable(tf.zeros(3))
    strides = [1, 2, 2, 1]
    padding = 'VALID'
    return tf.nn.conv2d(input, F_W, strides, padding) + F_b
```

I want to transform the input shape `(1, 4, 4, 1)` to `(1, 2, 2, 3)`. I choose `'VALID'` for the padding algorithm. I find it simpler to understand and it achieves the result I'm looking for.

```
out_height = ceil(float(in_height - filter_height + 1) / float(strides[1]))
out_width  = ceil(float(in_width - filter_width + 1) / float(strides[2]))
```

Plugging in the values:

```
out_height = ceil(float(4 - 2 + 1) / float(2)) = ceil(1.5) = 2
out_width  = ceil(float(4 - 2 + 1) / float(2)) = ceil(1.5) = 2
```

In order to change the depth from 1 to 3, I have to set the output depth of my filter appropriately:

```
F_W = tf.Variable(tf.truncated_normal((2, 2, 1, 3))) # (height, width, input_depth, out
F_b = tf.Variable(tf.zeros(3)) # (output_depth)
```

The input has a depth of 1, so I set that as the `input_depth` of the filter.

NEXT