Solution

The answer is **14x14x20**.

We can get the new height and width with the formula resulting in:

```
(32 - 8 + 2 * 1)/2 + 1 = 14
(32 - 8 + 2 * 1)/2 + 1 = 14
```

The new depth is equal to the number of filters, which is 20.

This would correspond to the following code:

```
input = tf.placeholder(tf.float32, (None, 32, 32, 3))
filter_weights = tf.Variable(tf.truncated_normal((8, 8, 3, 20))) # (height, width, inpu
filter_bias = tf.Variable(tf.zeros(20))
strides = [1, 2, 2, 1] # (batch, height, width, depth)
padding = 'SAME'
conv = tf.nn.conv2d(input, filter_weights, strides, padding) + filter_bias
```

Note the output shape of `conv` will be [1, 16, 16, 20]. It's 4D to account for batch size, but more importantly, it's not [1, 14, 14, 20]. This is because the padding algorithm TensorFlow uses is not exactly the same as the one above. An alternative algorithm is to switch `padding` from `'SAME'` to `'VALID'` which would result in an output shape of [1, 13, 13, 20]. If you're curious how padding works in TensorFlow, read this document.

In summary TensorFlow uses the following equation for 'SAME' vs 'PADDING'

SAME Padding, the output height and width are computed as:

`out_height` = ceil(float(in_height) / float(strides[1]))

`out_width` = ceil(float(in_width) / float(strides[2]))

VALID Padding, the output height and width are computed as:

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

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

NEXT