Solution

Here's how I did it:

```
def LeNet(x):
    # Reshape from 2D to 4D. This prepares the data for
    # convolutional and pooling layers.
    x = tf.reshape(x, (-1, 28, 28, 1))
    # Pad 0s to 32x32. Centers the digit further.
    # Add 2 rows/columns on each side for height and width dimensio
ns.
    x = tf.pad(x, [[0, 0], [2, 2], [2, 2], [0, 0]], mode="CONSTANT"
)

    # 28x28x6
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6)))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1 = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding=
'VALID') + conv1_b

    conv1 = tf.nn.relu(conv1)

    # 14x14x6
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2
, 2, 1], padding='VALID')

    # 10x10x16
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16)))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2 = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padd
ing='VALID') + conv2_b
```

```
    conv2 = tf.nn.relu(conv2)

    # 5x5x16
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2
, 2, 1], padding='VALID')

    # Flatten
    fc1 = flatten(conv2)
    # (5 * 5 * 16, 120)
    fc1_shape = (fc1.get_shape().as_list()[-1], 120)

    fc1_W = tf.Variable(tf.truncated_normal(shape=(fc1_shape)))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1 = tf.matmul(fc1, fc1_W) + fc1_b
    fc1 = tf.nn.relu(fc1)

    fc2_W = tf.Variable(tf.truncated_normal(shape=(120, 10)))
    fc2_b = tf.Variable(tf.zeros(10))
    return tf.matmul(fc1, fc2_W) + fc2_b
```

Walkthrough

Let's go through it layer by layer.

```
# 28x28x6
conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6)))
conv1_b = tf.Variable(tf.zeros(6))
conv1 = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VAL
ID') + conv1_b
```

I want to transform the shape from **32x32x1** to **28x28x6**. So I create a filter with the shape (5, 5, 1, 6). Recall this stands for (height, width, input_depth, output_depth). I'm going to use

`'VALID'` padding for all convolutional and pooling layers. I find it easier to work with when you're trying to decrease the output size. This means by formula for the new height and width is:

```
out_height = ceil(float(in_height - filter_height + 1) / float(stri
des[1]))
out_width  = ceil(float(in_width - filter_width + 1) / float(stride
s[2]))
```

Plugging in values:

```
out_height = ceil(float(32 - 5 + 1) / float(1)) = 28
out_width = ceil(float(32 - 5 + 1) / float(1)) = 28
```

Clearly, setting the height and width strides to anything other than 1 will result in a much smaller output size than I'm looking for.

```
conv1 = tf.nn.relu(conv1)
```

A standard `ReLU` activation. You might have chosen another activation.

```
# 14x14x6
conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
1], padding='VALID')
```

The formula to calculate the new height and width is the same as the one with the used for the convolution layer. Plugging in values:

```
new_height = ceil(float(28 - 2 + 1) / float(2)) = ceil(13.5) = 14
new_width = ceil(float(28 - 2 + 1) / float(2)) = ceil(13.5) = 14
new_width = ceil(float(28 - 2 + 1) / float(2)) = ceil(13.5) = 14
```

The next round of `convolution -> activation -> pooling` is pretty much the same, just with different values, so I won't go over it.

```
fc1 = flatten(conv2)
# (5 * 5 * 16, 120)
fc1_shape = (fc1.get_shape().as_list()[-1], 120)
```

The `flatten` function flattens a `Tensor` such that it becomes 2D. It's generally assumed the first dimension of a `Tensor` is the batch size so that remains unaltered. After the 2nd pooling layer the output shape should be **5x5x16** (ignoring batch size). Applying `flatten` will multiply the length of each dimension together resulting in **400**.

Now that the `Tensor` is 2D, it's ready to be used in fully connected layers.

NOTE: I could have set `fc1_shape` to `(400, 120)` manually and it would be fine since I calculated that's the correct product of the output shape. However, it would reasonable to assume I might want to alter a previous layer which might change the output shape. In that case, `(400, 120)` would be incorrect.

```
fc1_W = tf.Variable(tf.truncated_normal(shape=(fc1_shape)))
fc1_b = tf.Variable(tf.zeros(120))
fc1 = tf.matmul(fc1, fc1_W) + fc1_b
fc1 = tf.nn.relu(fc1)

fc2_W = tf.Variable(tf.truncated_normal(shape=(120, 10)))
fc2_b = tf.Variable(tf.zeros(10))
return tf.matmul(fc1, fc2_W) + fc2_b
```

You're already familiar with fully connected layers so I won't go into much detail. Note the output sizes **120** and **10** output sizes **120** and **10**.

Congratulations! You're now a convolution and pooling expert!

NEXT