**TensorFlow Convolution Layer**

**Using Convolution Layers in TensorFlow**

Let's now apply what we've learned to build real CNNs in TensorFlow. In the below exercise, you'll be asked to set up the dimensions of the Convolution filters, the weights, the biases. This is in many ways the trickiest part to using CNNs in TensorFlow. Once you have a sense of how to set up the dimensions of these attributes, applying CNNs will be far more straight forward.

**Review**

You should go over the TensorFlow documentation for [**2D convolutions**](https://www.tensorflow.org/api_guides/python/nn#Convolution). Most of the documentation is straightforward, except perhaps the padding argument. The padding might differ depending on whether you pass 'VALID' or 'SAME'.

Here are a few more things worth reviewing:

1. Introduction to TensorFlow -> TensorFlow Variables.
2. How to determine the dimensions of the output based on the input size and the filter size (shown below). You'll use this to determine what the size of your filter should be.
3. new\_height = (input\_height - filter\_height + 2 \* P)/S + 1
4. new\_width = (input\_width - filter\_width + 2 \* P)/S + 1

**Instructions**

1. Finish off each TODO in the conv2d function.
2. Setup the strides, padding and filter weight/bias (F\_w and F\_b) such that the output shape is (1, 2, 2, 3). Note that all of these except strides should be TensorFlow variables.

"""

Setup the strides, padding and filter weight/bias such that

the output shape is (1, 2, 2, 3).

"""

import tensorflow as tf

import numpy as np

# `tf.nn.conv2d` requires the input be 4D (batch\_size, height, width, depth)

# (1, 4, 4, 1)

x = np.array([

[0, 1, 0.5, 10],

[2, 2.5, 1, -8],

[4, 0, 5, 6],

[15, 1, 2, 3]], dtype=np.float32).reshape((1, 4, 4, 1))

X = tf.constant(x)

def conv2d(input):

# Filter (weights and bias)

# The shape of the filter weight is (height, width, input\_depth, output\_depth)

# The shape of the filter bias is (output\_depth,)

# TODO: Define the filter weights `F\_W` and filter bias `F\_b`.

# NOTE: Remember to wrap them in `tf.Variable`, they are trainable parameters after all.

F\_W = tf.Variable(tf.truncated\_normal((2, 2, 1, 3))) # (height, width, input\_depth, output\_depth)

F\_b = tf.Variable(tf.zeros(3)) # (output\_depth)

# TODO: Set the stride for each dimension (batch\_size, height, width, depth)