Neural Networks: Tensorflow

Dr. Amjad Hawash

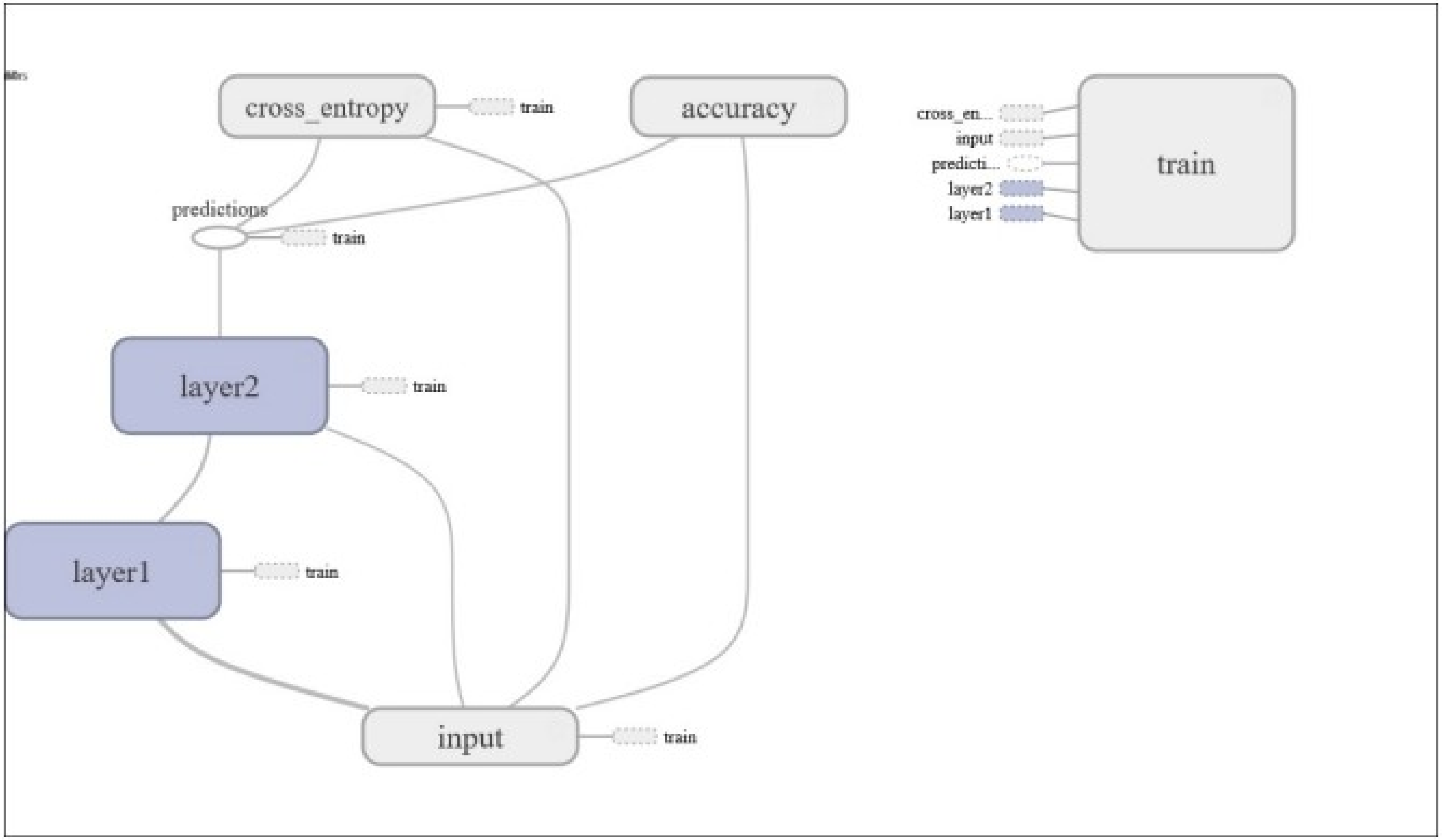
How to use TensorBoard

- TensorBoard is a visualization tool, devoted to analyzing Data Flow Graph and also to better understand the machine learning models.
- It can view different types of statistics about the parameters and details of any part of a computer graph graphically.
- A deep neural network can have up to 36,000 nodes.
- For this reason, TensorBoard collapses nodes in high-level blocks, highlighting the groups with identical structures.
- Doing so allows a better analysis of the graph, focusing only on the core sections of the computation graph.

How to use TensorBoard

- Also, the visualization process is interactive; user can pan, zoom, and expand the nodes to display the details.
- The following figure shows a neural network model with TensorBoard:

How to use TensorBoard



A TensorBoard visualization example

How to use TensorBoard

- TensorFlow lets you insert so-called summary operations into the graph.
- These summary operations monitor changing values (during the execution of a computation) written in a log file.
- Then TensorBoard is configured to watch this log file with summary information and display how this information changes over time.

How to use TensorBoard

- Example:
  - import tensorflow as tf
  - a = tf.constant(10,name="a")
  - b = tf.constant(90,name="b")
  - y = tf.Variable(a+b*2, name="y")
  - model = tf.initialize_all_variables()
  - with tf.Session() as session:
    - merged = tf.merge_all_summaries()
    - writer = tf.train.SummaryWriter\
      ("/tmp/tensorflowlogs",session.graph)
    - session.run(model)
    - print(session.run(y))

How to use TensorBoard

- merged = tf.merge_all_summaries()

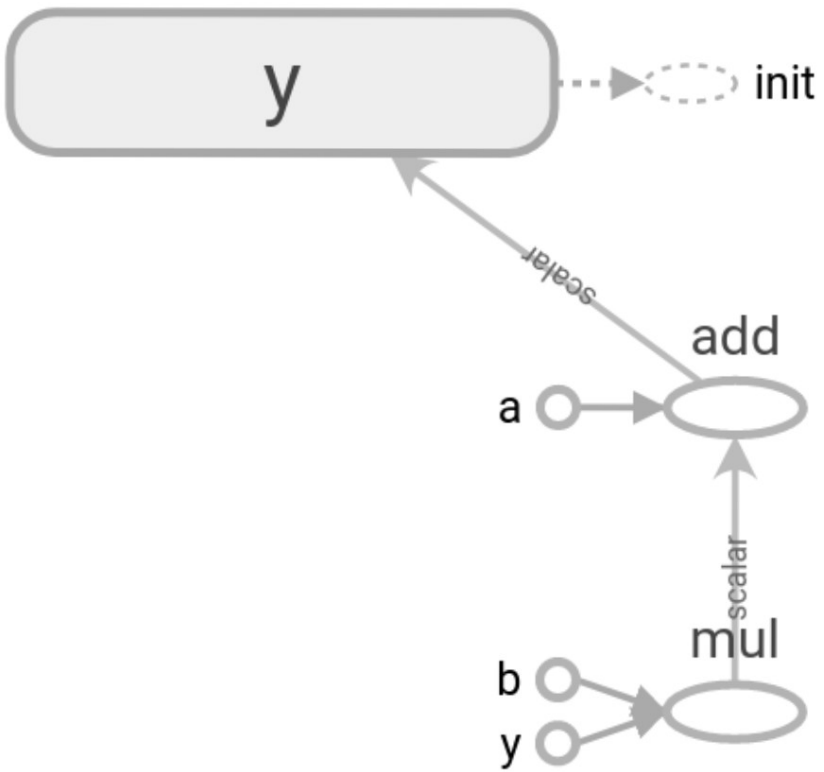
- This instruction must merge all the summaries collected in the default graph.
- Then we create SummaryWriter. It will write all the summaries (in this case the execution graph) obtained from the code's execution into the /tmp/tensorflowlogs directory:
  - writer = tf.train.SummaryWriter\
    ("/tmp/tensorflowlogs",session.graph)

How to use TensorBoard

- Finally, we run the model and so build the Data Flow Graph:
  - session.run(model)
  - print(session.run(y))
- The use of TensorBoard is very simple. Let's open a terminal and enter the following:
  - tensorboard –logdir=/tmp/tensorflowlogs
- A message such as the following should appear:
  - startig tensorboard on port 6006

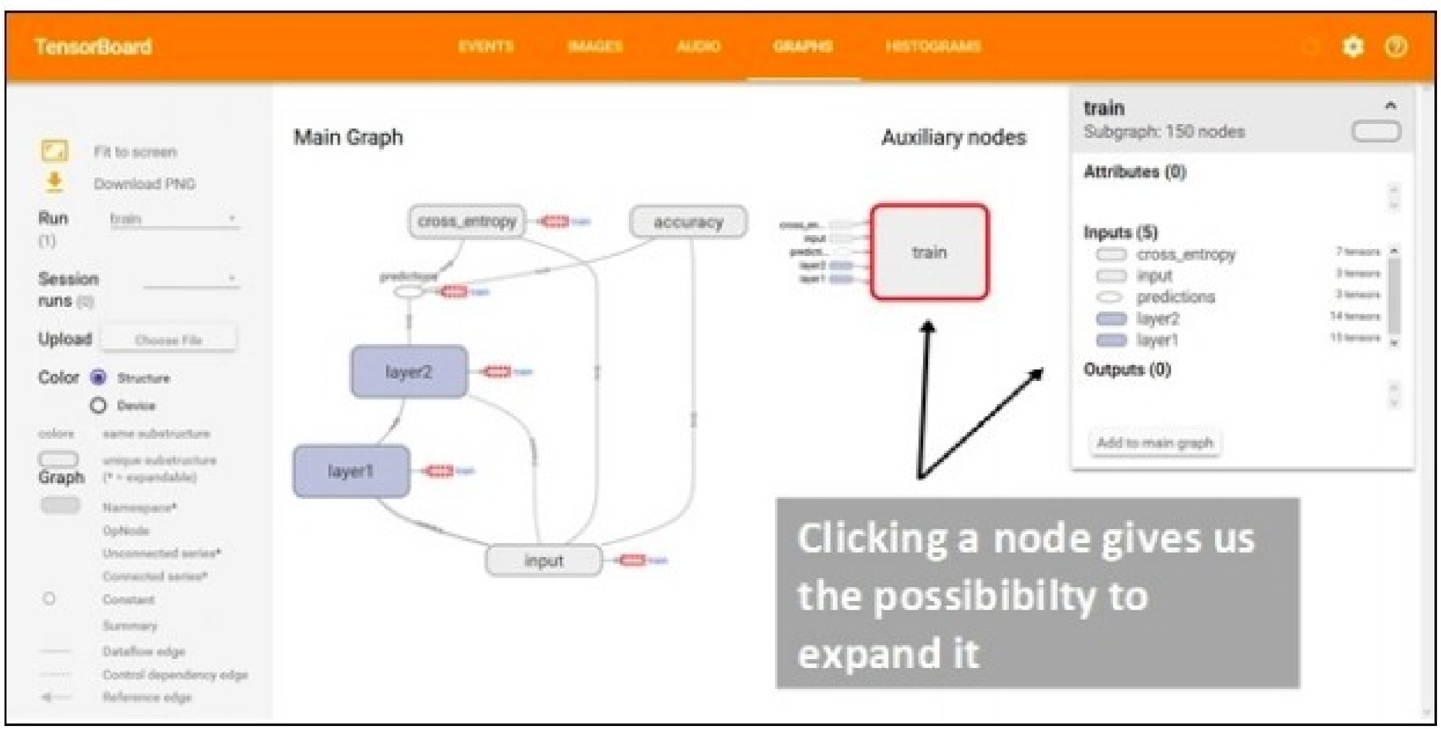
How to use TensorBoard

- Then, by opening a web browser, we should display the Data Flow Graph with auxiliary nodes:



How to use TensorBoard

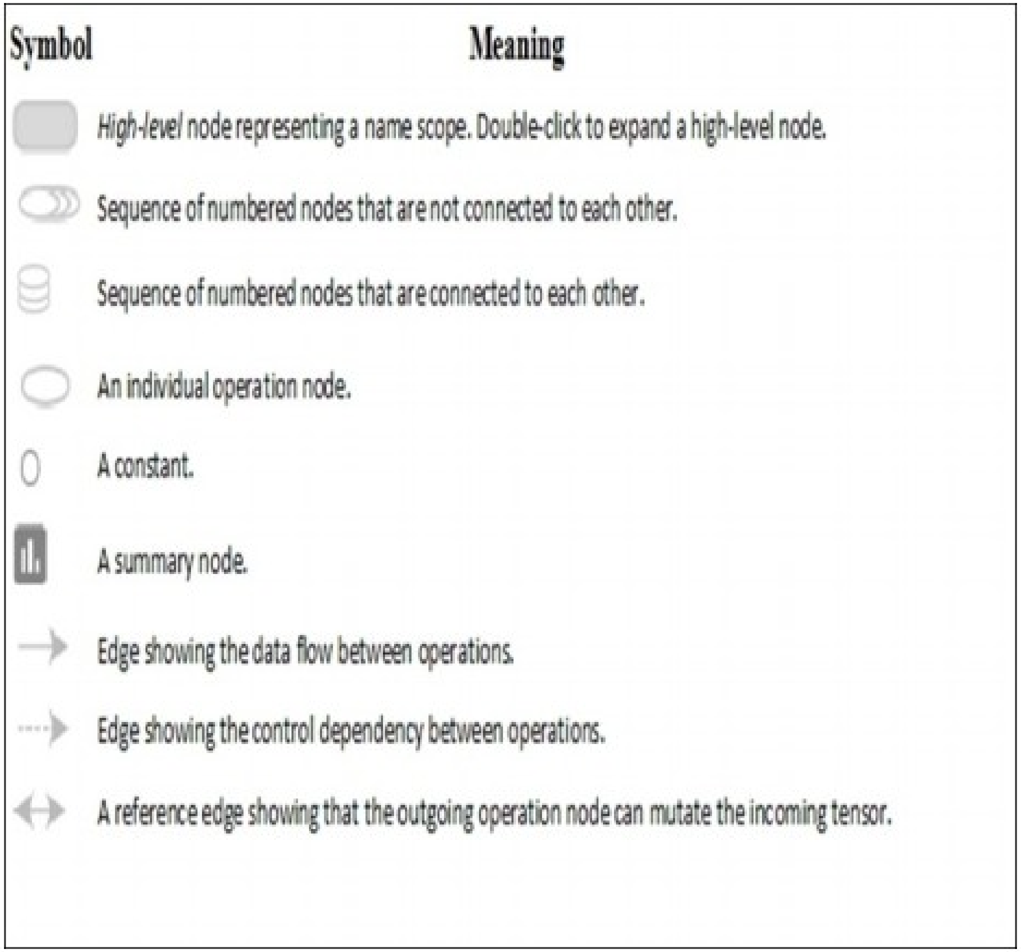
- Now we will be able to explore the Data Flow Graph:



Explore the Data Flow Graph display with TensorBoard

How to use TensorBoard

- TensorBoard uses special icons for constants and summary nodes.
- To summarize, we report in the next figure the table of node symbols displayed:



Symbol	Meaning
	High-level node representing a name scope. Double-click to expand a high-level node.
	Sequence of numbered nodes that are not connected to each other.
	Sequence of numbered nodes that are connected to each other.
	An individual operation node.
	A constant.
	A summary node.
	Edge showing the data flow between operations.
	Edge showing the control dependency between operations.
	A reference edge showing that the outgoing operation node can mutate the incoming tensor.

Node symbols in TensorBoard

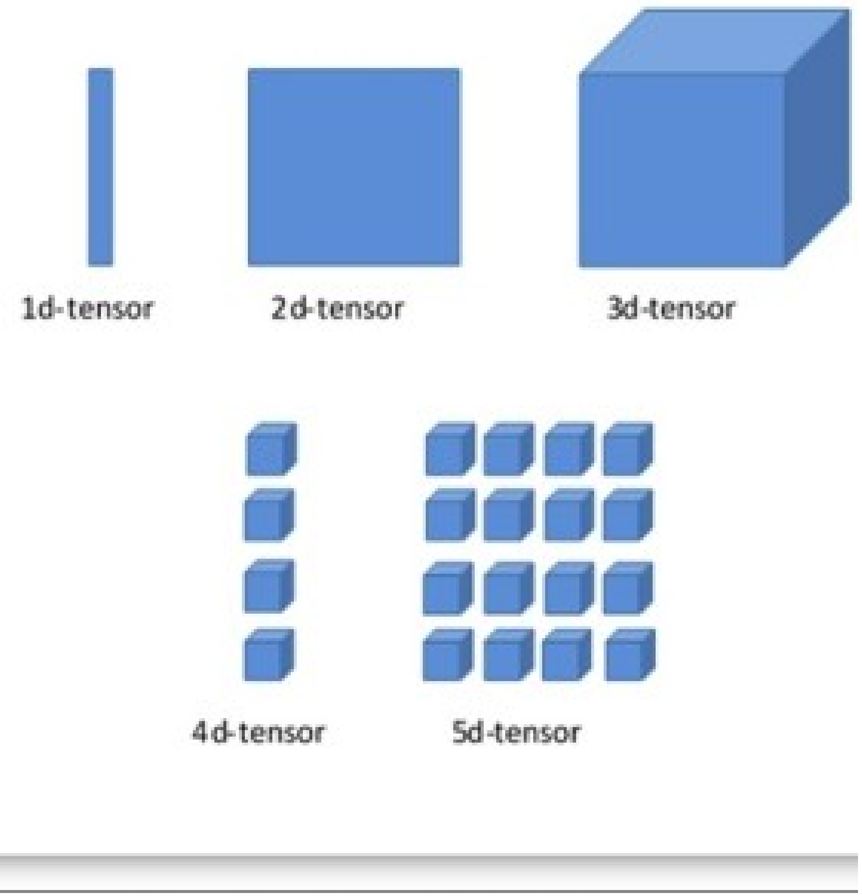
The tensor data structure

- Tensors are the basic data structures in TensorFlow.
- They represent the connecting edges in a Data Flow Graph.
- A tensor simply identifies a multidimensional array or list.
- It can be identified by three parameters, rank, shape, and type:
  - Rank: identifies the number of dimensions of the tensor. For example, a rank 2 tensor is a matrix and a rank 1 tensor is a vector.
  - shape: The shape of a tensor is the number of rows and columns it has.
  - type: It is the data type assigned to the tensor's elements.

The tensor data structure

- To build a tensor, we can:
  - Build an n-dimensional array; for example, by using the NumPy library
  - Convert the n-dimensional array into a TensorFlow tensor

The tensor data structure



Visualization of multidimensional tensors

One-dimensional tensors

- To build a one-dimensional tensor, we use the Numpy array(s) command, where s is a Python list:
  - import numpy as np
  - tensor_1d = np.array([1.3, 1, 4.0, 23.99])
  - print tensor_1d → [1.3 1. 4. 23.99]
- Indexing:
  - print tensor_1d[0] →1.3
  - print tensor_1d[2] → 4.0

One-dimensional tensors

- Finally, you can view the basic attributes of the tensor, the rank of the tensor:
  - tensor_1d.ndim → 1
- The tuple of the tensor's dimension is as follows:
  - tensor_1d.shape → (4L,)
- The data type in the tensor:
  - tensor_1d.dtype → dtype('float64')

One-dimensional tensors

- Now, let's see how to convert a NumPy array into a TensorFlow tensor:
  - import tensorflow as tf
- The TensorFlow function tf_convert_to_tensor converts Python objects of various types to tensor objects.
- It accepts tensor objects, Numpy arrays, Python lists, and Python scalars:
  - tf_tensor=tf.convert_to_tensor(tensor_1d,dtype=tf.float64)
- Running the Session , we can visualize the tensor and its elements as follows:
  - with tf.Session() as sess:
    - print sess.run(tf_tensor)
    - print sess.run(tf_tensor[0])
    - print sess.run(tf_tensor[2])

Two-dimensional tensors

- To create a two-dimensional tensor or matrix, we again use array(s), but s will be a sequence of array:
  - import numpy as np
  - tensor_2d=np.array([(1,2,3,4),(4,5,6,7),(8,9,10,11), (12,13,14,15)])
  - print tensor_2d
- The output:
    - [[1 2 3 4]
    - [4 5 6 7]
    - [8 9 10 11]
    - [12 13 14 15]]
  - Print tensor_2d[3][3]
    - 15
  - Print tensor_2d[0:2,0:2]
    - array([[1, 2],
    - [4, 5]])

Tensor handling

- we can apply a little more complex operations to these data structures.
  - Import the libraries:
    - import TensorFlow as tf
    - import numpy as np
  - build two integer arrays. These represents two 3×3 matrices:
    - matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype='int32')
    - matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype='int32')
  - Visualize them:
    - print "matrix1="
    - print matrix1
    -
    - print "matrix2 ="
    - print matrix2

Tensor handling

- To use these matrices in our TensorFlow environment, they must be transformed into a tensor data structure:
  - matrix1 = tf.constant(matrix1)
  - matrix2 = tf.constant(matrix2)
- We used the TensorFlow constant operator to perform the transformation.
- The matrices are ready to be manipulated with TensorFlow operators.
- In this case, we calculate a matrix multiplication and a matrix sum:
  - matrix_product = tf.matmul(matrix1, matrix2)
  - matrix_sum = tf.add(matrix1,matrix2)

Tensor handling

- The following matrix will be used to compute a matrix determinant:
  - matrix_3 = np.array([(2,7,2),(1,4,2),
  - (9,0,2)],dtype='float32')
  - print "matrix3 ="
  - print matrix_3
  - matrix_det = tf.matrix_determinant(matrix_3)
- It's time to create our graph and run the session, with the tensors and operators created:
  - with tf.Session() as sess:
    - result1 = sess.run(matrix_product)
    - result2 = sess.run(matrix_sum)
    - result3 = sess.run(matrix_det)

Tensor handling

- The results will be printed out by running the following command:
  - print "matrix1*matrix2 ="
  - print result1
  - print "matrix1 + matrix2 ="
  - print result2
  - print "matrix3 determinant result ="
  - print result3

Tensor handling

- TensorFlow provides numerous math operations on tensors.

TensorFlow operator	Description
`tf.add`	Returns the sum
`tf.sub`	Returns subtraction
`tf.mul`	Returns the multiplication
`tf.div`	Returns the division
`tf.mod`	Returns the module
`tf.abs`	Returns the absolute value
`tf.neg`	Returns the negative value
`tf.sign`	Returns the sign
`tf.inv`	Returns the inverse
`tf.square`	Returns the square
`tf.round`	Returns the nearest integer
`tf.sqrt`	Returns the square root
`tf.pow`	Returns the power
`tf.exp`	Returns the exponential
`tf.log`	Returns the logarithm
`tf.maximum`	Returns the maximum
`tf.minimum`	Returns the minimum
`tf.cos`	Returns the cosine
`tf.sin`	Returns the sine

Three-dimensional tensors

- The following commands build a three-dimensional tensor:
  - import numpy as np
  - tensor_3d = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
- The three-dimensional tensor created is a 2x2x2 matrix:
- To retrieve an element from a three-dimensional tensor, we use an expression of the following form:
  - tensor_3d[plane,row,col]

Handling tensors with TensorFlow

- A color digital image that is a MxNx3 size matrix (a three order tensor) (R,G,B).
  - import matplotlib.image as mp_image
  - filename = "packt.jpeg"
  - input_image = mp_image.imread(filename)
  - //rank and the shape
  - print 'input dim = {}'.format(input_image.ndim)
  - print 'input shape = {}'.format(input_image.shape)
  - import matplotlib.pyplot as plt
  - plt.imshow(input_image)
  - plt.show()

Handling tensors with TensorFlow

- Slice is a bidimensional segment of the starting image, where each pixel has the RGB components, so we need a placeholder to store all the values of the slice:
  - import TensorFlow as tf my_image = tf.placeholder("uint8",[None,None,3])
- Then we use the TensorFlow operator slice to create a sub-image:
  - slice = tf.slice(my_image,[10,0,0],[16,-1,-1])
- The last step is to build a TensorFlow working session:
  - with tf.Session() as session:
    - result = session.run(slice,feed_dict={my_image: input_image})
    - print(result.shape)
  - plt.imshow(result)
  - plt.show()

Handling tensors with TensorFlow

- Geometric transformation of the input image, using the transpose operator:
  - import tensorflow as tf
- Associate the input image to a variable we call x :
  - x = tf.Variable(input_image,name='x')
- Initialize our model:
  - model = tf.initialize_all_variables()
- build up the session with that we run our code:
  - with tf.Session() as session:
    - x = tf.transpose(x, perm=[1,0,2])
    - session.run(model)
    - result=session.run(x)
  - plt.imshow(result)
  - plt.show()