In this Tensorflow tutorial, I shall explain:

1. *How does a Tensorflow model look like?*
2. *How to save a Tensorflow model?*
3. *How to restore a Tensorflow model for prediction/transfer learning*?
4. How to work with imported pretrained models for fine-tuning and modification

This tutorial assumes that you have some idea about training a neural network. Otherwise, please[follow this tutorial](https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/) and come back here.

**1.What is a Tensorflow model?:**

After you have trained a neural network, you would want to save it for future use and deploying to production. So, what is a Tensorflow model? Tensorflow model primarily contains the network design or graph and values of the network parameters that we have trained. Hence, Tensorflow model has two main files:

**a) Meta graph:**

This is a protocol buffer which saves the complete Tensorflow graph; i.e. all variables, operations, collections etc. This file has **.meta** extension.

**b) Checkpoint file:**

This is a binary file which contains all the values of the weights, biases, gradients and all the other variables saved. This file has an extension .**ckpt.**However, Tensorflow has changed this from version 0.11. Now, instead of single .ckpt file, we have two files:



|  |  |
| --- | --- |
| 1  2  3 | mymodel.data-00000-of-00001  mymodel.index |

.data file is the file that contains our training variables and we shall go after it.

Along with this, Tensorflow also has a file named **checkpoint** which simply keeps a record of latest checkpoint files saved.

So, to summarize, Tensorflow models for versions greater than 0.10 look like this:

while Tensorflow model before 0.11 contained only three files:



|  |  |
| --- | --- |
| 1  2  3  4 | inception\_v1.meta  inception\_v1.ckpt  checkpoint |

Now that we know how a Tensorflow model looks like, let’s learn how to save the model.

**2. Saving a Tensorflow model:**

Let’s say, you are[training a convolutional neural network for image classification](https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/). As a standard practice, you keep a watch on loss and accuracy numbers. Once you see that the network has converged, you can stop the training manually or you will run the training for fixed number of epochs. After the training is done, we want to save all the variables and network graph to a file for future use. So, in Tensorflow, you want to save the graph and values of all the parameters for which we shall be creating an instance of tf.train.Saver() class.

saver = tf.train.Saver()

Remember that Tensorflow variables are only alive inside a session. So, you have to save the model inside a session by calling save method on saver object you just created.



|  |  |
| --- | --- |
| 1  2 | saver.save(sess, 'my-test-model') |

Here, sess is the session object, while ‘my-test-model’ is the name you want to give your model. Let’s see a complete example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import tensorflow as tf  w1 = tf.Variable(tf.random\_normal(shape=[2]), name='w1')  w2 = tf.Variable(tf.random\_normal(shape=[5]), name='w2')  saver = tf.train.Saver()  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())  saver.save(sess, 'my\_test\_model')    # This will save following files in Tensorflow v >= 0.11  # my\_test\_model.data-00000-of-00001  # my\_test\_model.index  # my\_test\_model.meta  # checkpoint |

If we are saving the model after 1000 iterations, we shall call save by passing the step count:

saver.save(sess, 'my\_test\_model',global\_step=1000)

This will just append ‘-1000’ to the model name and following files will be created:



|  |  |
| --- | --- |
| 1  2  3  4  5 | my\_test\_model-1000.index  my\_test\_model-1000.meta  my\_test\_model-1000.data-00000-of-00001  checkpoint |

Let’s say, while training, we are saving our model after every 1000 iterations, so .meta file is created the first time(on 1000th iteration) and we don’t need to recreate the .meta file each time(so, we don’t save the .meta file at 2000, 3000.. or any other iteration). We only save the model for further iterations, as the graph will not change. Hence, when we don’t want to write the meta-graph we use this:



|  |  |
| --- | --- |
| 1  2 | saver.save(sess, 'my-model', global\_step=step,write\_meta\_graph=False) |

If you want to keep only 4 latest models and want to save one model after every 2 hours during training you can use max\_to\_keep and keep\_checkpoint\_every\_n\_hours like this.



|  |  |
| --- | --- |
| 1  2  3 | #saves a model every 2 hours and maximum 4 latest models are saved.  saver = tf.train.Saver(max\_to\_keep=4, keep\_checkpoint\_every\_n\_hours=2) |

**Note, if we don’t specify anything in the tf.train.Saver(), it saves all the variables**. What if, we don’t want to save all the variables and just some of them. We can specify the variables/collections we want to save. While creating the tf.train.Saver instance we pass it a list or a dictionary of variables that we want to save. Let’s look at an example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import tensorflow as tf  w1 = tf.Variable(tf.random\_normal(shape=[2]), name='w1')  w2 = tf.Variable(tf.random\_normal(shape=[5]), name='w2')  saver = tf.train.Saver([w1,w2])  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())  saver.save(sess, 'my\_test\_model',global\_step=1000) |

This can be used to save specific part of Tensorflow graphs when required.

**3. Importing a pre-trained model:**

If you want to use someone else’s pre-trained model for fine-tuning, there are two things you need to do:

**a) Create the network:**

You can create the network by writing python code to create each and every layer manually as the original model. However, if you think about it, we had saved the network in .meta file which we can use to recreate the network using tf.train.import() function like this: saver = tf.train.import\_meta\_graph('my\_test\_model-1000.meta')

Remember, import\_meta\_graph appends the network defined in .meta file to the current graph. So, this will create the graph/network for you but we still need to load the value of the parameters that we had trained on this graph.

**b) Load the parameters:**

We can restore the parameters of the network by calling restore on this saver which is an instance of tf.train.Saver() class.



|  |  |
| --- | --- |
| 1  2  3  4 | with tf.Session() as sess:    new\_saver = tf.train.import\_meta\_graph('my\_test\_model-1000.meta')    new\_saver.restore(sess, tf.train.latest\_checkpoint('./')) |

After this, the value of tensors like w1 and w2 has been restored and can be accessed:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | with tf.Session() as sess:      saver = tf.train.import\_meta\_graph('my-model-1000.meta')      saver.restore(sess,tf.train.latest\_checkpoint('./'))      print(sess.run('w1:0'))  ##Model has been restored. Above statement will print the saved value of w1. |

So, now you have understood how saving and importing works for a Tensorflow model. In the next section, I have described a practical usage of above to load any pre-trained model.

**4. Working with restored models**

Now that you have understood how to save and restore Tensorflow models, Let’s develop a practical guide to restore any pre-trained model and use it for prediction, fine-tuning or further training. Whenever you are working with Tensorflow, you define a graph which is fed examples(training data) and some hyperparameters like learning rate, global step etc. It’s a standard practice to feed all the training data and hyperparameters using placeholders. Let’s build a small network using placeholders and save it. Note that when the network is saved, values of the placeholders are not saved.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import tensorflow as tf    #Prepare to feed input, i.e. feed\_dict and placeholders  w1 = tf.placeholder("float", name="w1")  w2 = tf.placeholder("float", name="w2")  b1= tf.Variable(2.0,name="bias")  feed\_dict ={w1:4,w2:8}    #Define a test operation that we will restore  w3 = tf.add(w1,w2)  w4 = tf.multiply(w3,b1,name="op\_to\_restore")  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())    #Create a saver object which will save all the variables  saver = tf.train.Saver()    #Run the operation by feeding input  print sess.run(w4,feed\_dict)  #Prints 24 which is sum of (w1+w2)\*b1    #Now, save the graph  saver.save(sess, 'my\_test\_model',global\_step=1000) |

Now, when we want to restore it, we not only have to restore the graph and weights, but also prepare a new feed\_dict that will feed the new training data to the network. We can get reference to these saved operations and placeholder variables via **graph.get\_tensor\_by\_name()** method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | #How to access saved variable/Tensor/placeholders  w1 = graph.get\_tensor\_by\_name("w1:0")    ## How to access saved operation  op\_to\_restore = graph.get\_tensor\_by\_name("op\_to\_restore:0") |

If we just want to run the same network with different data, you can simply pass the new data via feed\_dict to the network.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | import tensorflow as tf    sess=tf.Session()  #First let's load meta graph and restore weights  saver = tf.train.import\_meta\_graph('my\_test\_model-1000.meta')  saver.restore(sess,tf.train.latest\_checkpoint('./'))      # Now, let's access and create placeholders variables and  # create feed-dict to feed new data    graph = tf.get\_default\_graph()  w1 = graph.get\_tensor\_by\_name("w1:0")  w2 = graph.get\_tensor\_by\_name("w2:0")  feed\_dict ={w1:13.0,w2:17.0}    #Now, access the op that you want to run.  op\_to\_restore = graph.get\_tensor\_by\_name("op\_to\_restore:0")    print sess.run(op\_to\_restore,feed\_dict)  #This will print 60 which is calculated  #using new values of w1 and w2 and saved value of b1. |

What if you want to add more operations to the graph by adding more layers and then train it. Of course you can do that too. See here:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import tensorflow as tf    sess=tf.Session()  #First let's load meta graph and restore weights  saver = tf.train.import\_meta\_graph('my\_test\_model-1000.meta')  saver.restore(sess,tf.train.latest\_checkpoint('./'))      # Now, let's access and create placeholders variables and  # create feed-dict to feed new data    graph = tf.get\_default\_graph()  w1 = graph.get\_tensor\_by\_name("w1:0")  w2 = graph.get\_tensor\_by\_name("w2:0")  feed\_dict ={w1:13.0,w2:17.0}    #Now, access the op that you want to run.  op\_to\_restore = graph.get\_tensor\_by\_name("op\_to\_restore:0")    #Add more to the current graph  add\_on\_op = tf.multiply(op\_to\_restore,2)    print sess.run(add\_on\_op,feed\_dict)  #This will print 120. |

But, can you restore part of the old graph and add-on to that for fine-tuning ? Of-course you can, just access the appropriate operation by graph.get\_tensor\_by\_name() method and build graph on top of that. Here is a real world example. Here we load a vgg pre-trained network using meta graph and change the number of outputs to 2 in the last layer for fine-tuning with new data.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | ......  ......  saver = tf.train.import\_meta\_graph('vgg.meta')  # Access the graph  graph = tf.get\_default\_graph()  ## Prepare the feed\_dict for feeding data for fine-tuning    #Access the appropriate output for fine-tuning  fc7= graph.get\_tensor\_by\_name('fc7:0')    #use this if you only want to change gradients of the last layer  fc7 = tf.stop\_gradient(fc7) # It's an identity function  fc7\_shape= fc7.get\_shape().as\_list()    new\_outputs=2  weights = tf.Variable(tf.truncated\_normal([fc7\_shape[3], num\_outputs], stddev=0.05))  biases = tf.Variable(tf.constant(0.05, shape=[num\_outputs]))  output = tf.matmul(fc7, weights) + biases  pred = tf.nn.softmax(output)    # Now, you run this with fine-tuning data in sess.run() |

Hopefully, this gives you very clear understanding of how Tensorflow models are saved and restored. Please feel free to share your questions or doubts in the comments section.