CV-Tricks.com
Best of Deep Learning & Computer vision (http://cv-tricks.com/)

A quick complete tutorial to save and restore Tensorflow models

by ANKIT SACHAN

In this Tensorflow tutorial, I shall explain:

I. *How does a Tensorflow model look like?*
II. *How to save a Tensorflow model?*
III. *How to restore a Tensorflow model for prediction/transfer learning*?
IV. 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 (http://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 mymodel.data-00000-of-00001
3 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:

```
-rw-rw-r-- 1 ubuntu ubuntu       93 Mar 20 10:32 checkpoint
-rw-rw-r-- 1 ubuntu ubuntu 26532836 Mar 20 10:32 inception_v1_model.data-00000-of-00001
-rw-rw-r-- 1 ubuntu ubuntu    10443 Mar 20 10:32 inception_v1_model.index
-rw-rw-r-- 1 ubuntu ubuntu  1072565 Mar 20 10:32 inception_v1_model.meta
```

while Tensorflow model before 0.11 contained only three files:

```
1
2 inception_v1.meta
3 inception_v1.ckpt
4 checkpoint
```

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

2. Saving a Tensorflow model:

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.

```
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:

```
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:

```
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:

```
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.

```
#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:

```
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')`

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.

```
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:

```
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-

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.


```
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.

```
#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.

```

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.
```

```
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.

```

......
......
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.

#TensorFlow (http://cv-tricks.com/tag/tensorflow/), #Tensorflow tutorial (http://cv-tricks.com/tag/tensorflow-tutorial/)

Note: you're not correct on how to restore operations. The operation in your example here is really a tensor as it's the result of an operation. If you want to, for example, restore a training operation to continue training, your code wouldn't work.

"get_tensor_by_name" would be "get_operation_by_name"

1 ^ | v • Reply • Share ›

Gold Man • 13 hours ago

How to change/update the max_to_keep attribute in the original saver object after importing the meta graph to a new saver object?

^ | v • Reply • Share ›

Jeong Mo Kang • 10 days ago

Hi, nice explanation!
If I run the code below:
with tf.Session(graph=graph) as session:
Initializate the weights and biases
#tf.global_variables_initializer().run()
new_saver = tf.train.import_meta_graph('model-1000.meta')
new_saver.restore(session, tf.train.latest_checkpoint('./'))
for curr_epoch in range(1):#num_epochs
train_cost = train_ler = 0
start = time.time()
for batch in range(3):#num_batches_per_epoch
.....some training.....

It gives error like:
---> new_saver.restore(session, tf.train.latest_checkpoint('./'))
SystemError: <built-in function="" tf_run=""> returned a result with an error set

Can you please help me?

^ | v • Reply • Share ›

Arnaldo Gualberto ➚ Jeong Mo Kang • 9 days ago

If you running on windows and the model is in the current folder, change tf.train.latest_checkpoint('./') by tf.train.latest_checkpoint('')

^ | v • Reply • Share ›

Meixu Song • 19 days ago

nice blog

^ | v • Reply • Share ›

Yan Gao • 2 months ago

Hi, nice article. In the last example, you just imported the vgg net but did not restore parameters. In this case, how can we restore a part of parameters?

^ | v • Reply • Share ›

KKL1 • 2 months ago

Hello, i am trying to do the training after dividing my dataset (due to memory troubles). The first training went good and i have saved the generated model. I was wondering how could I resume the training?
I have tried to restore the model and run the training over again, shall I run all of the predictions and optimizer functions over again, cause i am facing a problem while feeding variables to these functions (like y_pred in your previous tuto). How could we use them or declare them??

^ | v • Reply • Share ›

KKL1 ➚ KKL1 • 2 months ago

Whenever i reload y_pred and all of the prediction and optimizer functions using these instructions:
sess = tf.Session()
saver1 = tf.train.import_meta_graph('my_test_model.ckpt.meta')
saver1= tf.train.Saver(tf.global_variables())
saver1.restore(sess, tf.train.latest_checkpoint('./'))
graph = tf.get_default_graph()
x= sess.graph.get_tensor_by_name("x:0")
y_true=sess.graph.get_tensor_by_name("y_true:0")
y_true_cls = tf.argmax(y_true, dimension=1)
y_pred_cls = tf.argmax(y_pred, dimension=1)
cost = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
sess.run(tf.global_variables_initializer())
i get this error:
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)

ValueError: cannot add op with name Variable/Adam as that name is already used

^ | v • Reply • Share ›

Hi, nice explanation.
I am training a conv neural net and the code saves samples after every , say 1000 iterations and saves sessions or checkpoints after , say 10,000 iterations. The training is complete. But I wish to write a code to use this model on test data / new data. How can we do that?
Reply • Share ›

Lee YG • 3 months ago
Hello, I see your good explanation here, and have a question.
If I want to train two seperate model (like some data CNN and ImageCNN) and combine the with fully connected layer, Is it possible to do it? If so, I wonder how It will figure out the backpropagation so.
Reply • Share ›

Ankit Mod ➔ Lee YG • 2 months ago
yes, it's possible, but there could be complications during backward flow of gradients depending what your networks are trying to achieve. So, tread with caution.
Reply • Share ›

bruce lau • 3 months ago
Thanks for your kindly work,
I got the same problem as Akhil,
And my tf version is 1.0.1 @ windows,
can you give me some advice?
Reply • Share ›

Ankit Mod ➔ bruce lau • 3 months ago
Hi Bruce,

I have updated the tutorial with a more appropriate method which works across versions. Please check at your convenience.

Thank you
Reply • Share ›

Ankit Mod ➔ bruce lau • 3 months ago
Thank you commenting. There was another related bug due to which loading graph and accessing variables (without building network via code) was not working till tensorflow version 1.1.0-rc1. So, the minimum version to load graph and access variables without building network via python code is 1.1.0. I would suggest you to either upgrade your Tensorflow or write python code to build the complete exact network which you want to load. Hope this helps.
Reply • Share ›

Akhil Chaturvedi • 3 months ago
Hi,
Thanks for this article. It was really useful. I had a question about it.
When I try to load the parameters in a different python file I am unable to. I tried replicating this code using different files for the saving and loading, and I get the error - "NameError: name 'w1' is not defined", when I do print sess.run(w1).
Can you help me work around this issue?
Reply • Share ›

Ankit Mod ➔ Akhil Chaturvedi • 3 months ago
Hi Akhil,

I have updated the tutorial with a more appropriate method which works across versions. Please check at your convenience.
Thank you
Reply • Share ›

Ankit Mod ➔ Akhil Chaturvedi • 3 months ago
Hi Akhil, Thank you for asking. This was an issue in earlier versions of Tensorflow which was fixed with this commit. https://github.com/tensorfl... . Please check the version of your Tensorflow installation and upgrade. I have updated the article with this information.
Reply • Share ›

Mohanad Kaleia ➔ Akhil Chaturvedi • 3 months ago
I have same question, how to restore the model from another python script file, I got same error,
Anyone got how to solve it?
Reply • Share ›

Ankit Mod ➔ Mohanad Kaleia • 3 months ago
Hi Mohanad,

I have updated the tutorial with a more appropriate method which works across versions. Please check at your convenience.
Thank you
Reply • Share ›

Ankit Mod ➔ Mohanad Kaleia • 3 months ago

If you are facing this, please check your Tensorflow version. This was fixed by a commit on Feb 5 (https://github.com/tensorfl.... All Tensorflow versions starting v1.0.0 support this functionality. Thank you for your feedback. I am adding this information in the blog.

Reply • Share ›

Wojciech Orzechowski ➚ Akhil Chaturvedi • 3 months ago

You added name to your layer. In that case you need to train the network all over again. They are ways to go around it (I guess), but it will be more problematic.

Reply • Share ›

Ankit Mod ➚ Wojciech Orzechowski • 3 months ago

Could you please elaborate? There are no layers used in this tutorial.

Reply • Share ›