<https://blog.wildcat.io/2018/04/a-simple-tutorial-about-caffe-tensorflow-model-conversion/>

2018-04-17 • Development

A simple tutorial about Caffe-TensorFlow model conversion

Introduction

Since [Caffe](http://caffe.berkeleyvision.org/) is really a good deep learning framework, there are many pre-trained models of Caffe. It is useful to know how to convert Caffe models into TensorFlow models. The whole process of this model conversion is so tricky that I decided to write it down, wishing it would help others.

Note:

* The source code and other related files of this tutorial can be found at: <https://github.com/imWildCat/a-simple-tutorial-about-caffe-tensorflow-model-conversion>
* The original pre-trained Caffe model in this tutorial is located at: <https://github.com/choosehappy/public/tree/master/DL%20tutorial%20Code/4-lymphocyte/models>

Pre-requisites

1. Operating System: macOS or Linux
2. Install Protobuf library
3. CMake 2.8 or newer
4. Python 2.7 (required by caffe-tensorflow, better if you could use virtualenv)
5. TensorFlow 1.x installed (tested with TensorFlow 1.7)

Note: this tutorial **does not** require to install Caffe except that you would like to convert the mean files.

Major steps

Step 1: Upgrade Caffe .prototxt (optional)

Since many .prototxt files are outdated, they must be upgraded before this kind of model conversion. If you have Caffe installed, you could just use upgrade\_net\_proto\_text ([reference](https://github.com/BVLC/caffe/blob/master/tools/upgrade_net_proto_text.cpp)). However, it is not easy to install Caffe on macOS. [caffe-net-upgrade](https://github.com/lcskrishna/caffe-net-upgrade) could be a good tool to use on Mac.

You could follow the [Build Instructions](https://github.com/lcskrishna/caffe-net-upgrade#build-instructions) to build the upgrade\_caffe\_layers. In this tutorial, we define the path to this executable as [path\_to]/upgrade\_caffe\_layers. Here is a example usage:

➜ [path\_to]/upgrade\_caffe\_layers deploy\_train32.prototxt

Loading prototxt file ...

INFO: Reading the prototxt file from : /Users/wildcat/Downloads/201804/temp/caffe-tensorflow-sample-case-lymphoma/caffe-models/deploy\_train32.prototxt

INFO: prototxt read successful

INFO: Network loaded is : CIFAR10\_quick

INFO: Upgrading V1LayerParameter => LayerParameter

STATUS: upgrade successful.

INFO: upgraded net is written into net.prototxt

Step 2: Convert the model and the mean file

Convert the model

Here we will use [caffe-tensorflow](https://github.com/dhaase-de/caffe-tensorflow-python3) for model conversion. A tricky thing is that the original repository of caffe-tensorflow is out of maintenance so that we are using a forked version: https://github.com/dhaase-de/caffe-tensorflow-python3 . (Although it is claimed to be able to work with Python 3, I can only use it with Python 2)

After *clone* the source code, you can use python ./convert.py to convert the model. For more details, please read: [https://github.com/dhaase-de/caffe-tensorflow-python3#3—convert-your-model](https://github.com/dhaase-de/caffe-tensorflow-python3#3---convert-your-model)

➜ python ./convert.py /path/to/net.prototxt --caffemodel /path/to/5\_caffenet\_train\_w32\_iter\_600000.caffemodel --data-output-path case\_tf.npy --code-output-path case\_tf.py

------------------------------------------------------------

WARNING: PyCaffe not found!

Falling back to a pure protocol buffer implementation.

\* Conversions will be drastically slower.

\* This backend is UNTESTED!

------------------------------------------------------------

Type Name Param Output

----------------------------------------------------------------------------------------------

Data data -- (10, 3, 32, 32)

Convolution conv1 (32, 3, 5, 5) (10, 32, 32, 32)

Pooling pool1 -- (10, 32, 16, 16)

ReLU relu1 -- (10, 32, 16, 16)

Convolution conv2 (32, 32, 5, 5) (10, 32, 16, 16)

Pooling pool2 -- (10, 32, 8, 8)

Convolution conv3 (64, 32, 5, 5) (10, 64, 8, 8)

Pooling pool3 -- (10, 64, 4, 4)

InnerProduct ip1 (64, 1024) (10, 64, 1, 1)

InnerProduct ip2 (2, 64) (10, 2, 1, 1)

Softmax prob -- (10, 2, 1, 1)

Converting data...

Saving data...

Saving source...

Done.

Note:

1. Remember to replace /path/to with your real path to the related files
2. net.prototxt and 5\_caffenet\_train\_w32\_iter\_600000.caffemodel are the model files used in my case, feel free to change them
3. case\_tf.npy stores the weights (parameters) and case\_tf.py stores the neural network architecture.

(Optional) Convert the mean file (Caffe needed)

Since many Caffe models use mean files for *normalization*, we must also convert the mean file to .npy, loading it in TensorFlow. Otherwise, the prediction cannot be right.

# Ref: https://github.com/BVLC/caffe/issues/290#issuecomment-62846228

# Modified by WildCat

import caffe

import numpy as np

import sys

if len(sys.argv) != 3:

print("Usage: python convert\_protomean.py proto.mean out.npy")

blob = caffe.proto.caffe\_pb2.BlobProto()

data = open('./original-caffe-models/DB\_train\_w32\_5.binaryproto', 'rb').read()

blob.ParseFromString(data)

arr = np.array(caffe.io.blobproto\_to\_array(blob))

out = arr[0]

np.save('mean.npy', out)

Again, please feel free to modify the path and name of the .binaryproto and mean.py files.

Step 3: Finish the conversion by making predictions

import numpy as np

import tensorflow as tf

from case\_tf import CIFAR10\_quick

def check\_correct(prob, path):

neg\_prob, pos\_prob= prob

is\_pos = path.find('\_p\_') != -1 # find '\_p\_' in the file name

if not is\_pos and is\_pos == (pos\_prob > neg\_prob):

print(prob, path, 'True negative')

return is\_pos == (pos\_prob > neg\_prob)

# load the converted mean file

means = np.load('mean.npy')

mean\_tensor = tf.transpose(tf.convert\_to\_tensor(means, dtype=tf.float32), [1, 2, 0])

def classify():

'''Classify the given images using GoogleNet.'''

model\_data\_path = './case\_tf.npy'

image\_file\_name\_pattern = './subs/\*.png'

NUM\_OF\_IMAGES = 100

# according to the .prototxt

IMAGE\_SIZE = 32

IMAGE\_CHANNELS = 3

# Create a placeholder for the input image

input\_node = tf.placeholder(tf.float32, shape=(None, IMAGE\_SIZE, IMAGE\_SIZE, IMAGE\_CHANNELS))

# Construct the network

net = CIFAR10\_quick({'data': input\_node})

# Create an image producer (loads and processes images in parallel)

# image\_producer = dataset.ImageProducer(image\_paths=image\_paths)

# custom: read images

filename\_queue = tf.train.string\_input\_producer(tf.train.match\_filenames\_once(image\_file\_name\_pattern))

reader = tf.WholeFileReader()

key, value = reader.read(filename\_queue)

my\_img = tf.image.decode\_png(value)

with tf.Session() as sess:

sess.run(tf.local\_variables\_initializer())

sess.run(tf.global\_variables\_initializer())

coord = tf.train.Coordinator()

threads = tf.train.start\_queue\_runners(coord=coord)

print('Load weights...')

net.load(data\_path=model\_data\_path, session=sess)

image\_list = []

image\_path\_list = []

print('Making predictions...')

for \_ in range(0, NUM\_OF\_IMAGES):

single\_image = sess.run(my\_img)

# Note (3 April) convert image channel sequence from RGB to BGR

reversed\_image = tf.reverse(single\_image, [-1])

reversed\_image = tf.cast(reversed\_image, tf.float32)

final\_image = tf.subtract(reversed\_image, mean\_tensor)

image\_list.append(final\_image)

image\_path\_list.append(sess.run(key))

input\_images = sess.run(tf.stack(image\_list))

probs = sess.run(net.get\_output(), feed\_dict={input\_node: input\_images})

acc\_list = []

predictions = zip(probs, image\_path\_list)

for prob, path in predictions:

acc\_list.append(check\_correct(prob, path))

print('accuracy: {}'.format(acc\_list.count(True) / float(len(acc\_list))))

for prob, path in predictions[:20]:

print('Image: {}, prob: {}'.format(path, prob))

coord.request\_stop()

coord.join(threads, stop\_grace\_period\_secs=2)

if \_\_name\_\_ == '\_\_main\_\_':

classify()

Note:

1. Part of the converted code (case\_tf.py) might not be correct, for example, change the layer name pattern from .conv(5, 5, 32, 1, 1, relu=False, name=conv1) to .conv(5, 5, 32, 1, 1, relu=False, name='conv1')
2. We have to convert the image channel from RGB to BGR because the original caffe model was trained using BGR convention due to OpenCV:
3. reversed\_image = tf.reverse(single\_image, [-1])

After this step, you could run this model successfully.

Conclusion

It is really a time-consuming task to convert a Caffe model to TensorFlow though this article is not so long. I wish that this article will help you to deal with this kind of problem.