<http://blog.csdn.net/nicholas_wong/article/details/76509763>

https://github.com/omimo/TFDroid

[**Android中运行Tensorflow程序2-编写自己的程序**](http://blog.csdn.net/nicholas_wong/article/details/76509763)

标签： [android](http://www.csdn.net/tag/android)[Tensorflow](http://www.csdn.net/tag/Tensorflow)

2017-08-01 14:09 1264人阅读 [评论](http://blog.csdn.net/nicholas_wong/article/details/76509763#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/nicholas_wong/article/details/76509763#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Linux（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg deep-learning（4） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg Android（3） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/nicholas_wong/article/details/76509763)

官方给出的demo中运行已经打包好的模型，没有解释怎样从零开始构建自己的模型。参考网站<https://omid.al/posts/2017-02-20-Tutorial-Build-Your-First-Tensorflow-Android-App.html>，自己做了一些尝试。

准备我们自己的TF模型

首先，我们创建一个简单的模型，把它的计算图保存为一个序列化的**GraphDef**文件。训练之后，把模型的变量值保存到checkpoint文件中。最后，我们需要把这两个文件变成一个优化了的独立的文件，这个文件是我们在Android App中所需要的所有文件。

创建和保存模型

主要目的是演示过程，所以模型十分简单：一个采用ReLU的单层网络。代码如下：

# Create a simple TF Graph

# By Omid Alemi - Jan 2017

# Works with TF r1.0

import tensorflow as tf

I = tf.placeholder(tf.float32, shape=[None,3], name='I') # input

W = tf.Variable(tf.zeros(shape=[3,2]), dtype=tf.float32, name='W') # weights

b = tf.Variable(tf.zeros(shape=[2]), dtype=tf.float32, name='b') # biases

O = tf.nn.relu(tf.matmul(I, W) + b, name='O') # activation / output

saver = tf.train.Saver()

init\_op = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init\_op)

# save the graph

tf.train.write\_graph(sess.graph\_def, '.', 'tfdroid.pbtxt')

# normally you would do some training here

# but fornow we will just assign something to W

sess.run(tf.assign(W, [[1, 2],[4,5],[7,8]]))

sess.run(tf.assign(b, [1,1]))

#save a checkpoint file, which will store the above assignment

saver.save(sess, 'tfdroid.ckpt')

* 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
* 26
* 27
* 28

运行上面的代码会把模型的计算图保存在**tfdroid.pbtxt**文件中，同时把模型变量的checkpoint保存在**tfdroid.ckpt**中。

冻结图

接下来需要把checkpoint中的变量转化为const Ops，同时把他们和GraphDef proto结合成为一个单独的文件。使用这个更方便我们在app中载入模型。为此，Tensorflow在tensorflow.python.tools中提供了freeze\_graph这个工具。   
冻结图之后，我们就可以对模型文件进行优化：移除那些只在训练过程中才用得上的部分，保留做预测需要的部分。根据文档，这个过程包括以下内容：   
1. 删除只有在训练过程中才用得到的操作，比如保存checkpoint。   
2. 剪枝掉那些永远都用不到的图。   
3. 删除debug操作，比如数据检查。   
4. 把batch normalization操作变成预先计算权值。   
5. Fusing common operations into unified versions。   
冻结图和优化的代码如下：

# Preparing a TF model for usage in Android

# By Omid Alemi - Jan 2017

# Works with TF r1.0

import sys

import tensorflow as tf

from tensorflow.python.tools import freeze\_graph

from tensorflow.python.tools import optimize\_for\_inference\_lib

MODEL\_NAME = 'tfdroid'

# Freeze the graph

input\_graph\_path = MODEL\_NAME+'.pbtxt'

checkpoint\_path = './'+MODEL\_NAME+'.ckpt'

input\_saver\_def\_path = ""

input\_binary = False

output\_node\_names = "O"

restore\_op\_name = "save/restore\_all"

filename\_tensor\_name = "save/Const:0"

output\_frozen\_graph\_name = 'frozen\_'+MODEL\_NAME+'.pb'

output\_optimized\_graph\_name = 'optimized\_'+MODEL\_NAME+'.pb'

clear\_devices = True

freeze\_graph.freeze\_graph(input\_graph\_path, input\_saver\_def\_path,

input\_binary, checkpoint\_path, output\_node\_names,

restore\_op\_name, filename\_tensor\_name,

output\_frozen\_graph\_name, clear\_devices, "")

# Optimize for inference

input\_graph\_def = tf.GraphDef()

with tf.gfile.Open(output\_frozen\_graph\_name, "r") as f:

data = f.read()

input\_graph\_def.ParseFromString(data)

output\_graph\_def = optimize\_for\_inference\_lib.optimize\_for\_inference(

input\_graph\_def,

["I"], # an array of the input node(s)

["O"], # an array of output nodes

tf.float32.as\_datatype\_enum)

# Save the optimized graph

f = tf.gfile.FastGFile(output\_optimized\_graph\_name, "w")

f.write(output\_graph\_def.SerializeToString())

# tf.train.write\_graph(output\_graph\_def, './', output\_optimized\_graph\_name)

* 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
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52

运行上述代码之后，我们可以得到**frozen\_tfdroid.pb**和**optimized\_tfdroid.pb**两个文件。如果运行过程中提示utf8decode错误，请尝试用python2.7运行。

freeze\_graph.freeze\_graph参数解析

有以下几个参数：   
1. input\_graph：必须，要输入的计算图的路径   
2. input\_saver：必须，不太懂，给它赋值为''（空字符串）   
3. input\_binary：必须，输入的是二进制数据或者是文件   
4. input\_checkpoint：必须，checkpoint文件的位置   
5. output\_node\_names：必须，字符串，内容是输出节点的名字，多个节点名字之间用,隔开   
6. restore\_op\_name：从模型中恢复变量的名字，默认设置为'save/restore\_all'   
7. filename\_tensor\_name：已弃用，默认设置为save/Const:0   
8. output\_graph：必选，保存输出文件   
9. clear\_devices：设置为True   
10. initializer\_nodes：必须，不理解   
11. variable\_names\_blacklist：不理解

optimize\_for\_inference\_lib.optimize\_for\_inference函数的参数解析

1. input\_graph\_def：包括训的模型的一个GraphDef
2. input\_node\_names：一个列表，列表的元素是字符串，一个字符串是一个输入节点的name
3. output\_node\_names：一个列表，列表的元素是字符串，一个字符串是一个输出节点的name
4. placeholder\_type\_enum：一个AttrValue enum（只有一个输入）或者它的列表（如果有多个输入），指明输入数据的格式。

接下来，我们构建自己的Android App。

创建Android App

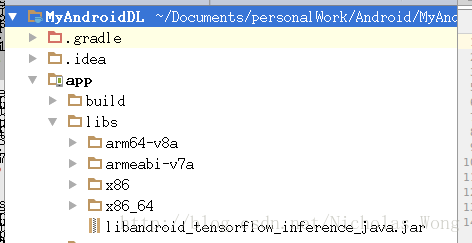
创建一个新的App

使用Android Studio创建只有一个空activity的project。

获取TF Libraries

当然可以从源码开始编译TF Libraries（参考网站[compile the Tensorflow libraries from scratch](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/android)），但是使用[nightly android builds](https://ci.tensorflow.org/view/Nightly/job/nightly-android/)提供的编译好的接口会更方便一些。从[网站](https://ci.tensorflow.org/view/Nightly/job/nightly-android/44/artifact/)下载。

在project中使用TF Libraries

下载编译好的接口之后，解压缩，把libandroid\_tensorflow\_inference\_java.jar和libtensorflow\_inference.so中所有的文件夹都拷贝到project的app/libs/里面，在Android Studio中可以看到如下结构   


然后修改app/build.gradle，增加如下内容，使系统知道这些libraries在什么位置。

sourceSets {

main {

jniLibs.srcDirs = ['libs']

}

}

* 1
* 2
* 3
* 4
* 5

修改后的app/build.gradle内容如下：

apply plugin: 'com.android.application'

android {

compileSdkVersion 26

buildToolsVersion "26.0.1"

defaultConfig {

applicationId "com.example.dong.myandroiddl"

minSdkVersion 15

targetSdkVersion 26

versionCode 1

versionName "1.0"

testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

sourceSets {

main {

jniLibs.srcDirs = ['libs']

}

}

}

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {

exclude group: 'com.android.support', module: 'support-annotations'

})

compile 'com.android.support:appcompat-v7:26.+'

compile 'com.android.support.constraint:constraint-layout:1.0.2'

testCompile 'junit:junit:4.12'

}

* 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
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37

拷贝TF Model

在app/src/main/中创建assets/文件夹，把optimized\_tfdroid.pb文件拷贝进来。

导入TF Inference Interfaces

在MainActivity.java中导入ensorFlowInferenceInterface包。

import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

* 1

导入tensorflow\_inference库。

static {

System.loadLibrary("tensorflow\_inference");

}

* 1
* 2
* 3

然后设置一些辅助变量。

private static final String MODEL\_FILE = "file:///android\_asset/optimized\_tfdroid.pb";

private static final String INPUT\_NODE = "I";

private static final String OUTPUT\_NODE = "O";

private static final int[] INPUT\_SIZE = {1,3};

* 1
* 2
* 3
* 4
* 5

创建TensorFlowInferenceInterface接口的对象

private TensorFlowInferenceInterface inferenceInterface;

* 1

在onCreate()函数中初始化接口和加载模型文件：

inferenceInterface = new TensorFlowInferenceInterface();

inferenceInterface.initializeTensorFlow(getAssets(), MODEL\_FILE);

* 1
* 2

开始预测

首先，给INPUT\_NODE赋值。

float[] inputFloats = {num1, num2, num3};

inferenceInterface.fillNodeFloat(INPUT\_NODE, INPUT\_SIZE, inputFloats);

* 1
* 2

然后，调用runInference（）方法来计算OUTPUT\_NODE。

inferenceInterface.runInference(new String[] {OUTPUT\_NODE});

* 1

计算完成之后，从OUTPUT\_NODE中获取值。

float[] resu = {0, 0};

inferenceInterface.readNodeFloat(OUTPUT\_NODE, resu);

* 1
* 2

项目代码可以从[github](https://github.com/omimo/TFDroid)上下载。