## What we are going to do?

In this section, we are going to build a small model (a+b)2 in tensorflow and deploy it in to an android mobile.

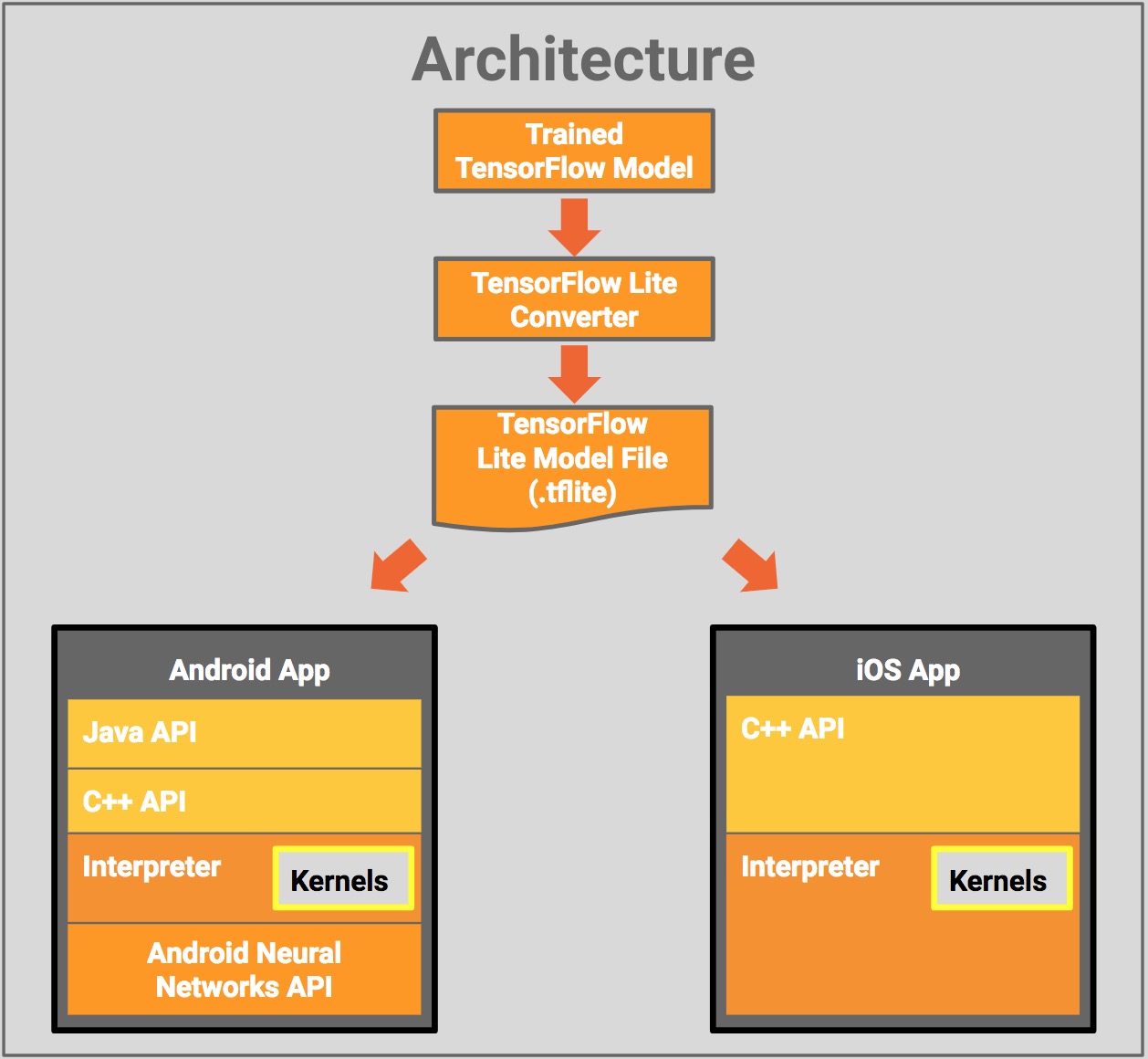
## What you need to know?

To proceed in this section You need a working installation of python, tensorflow dependencies installed and android studio and also some knowledge on python and java android. You can find the instructions on how to install tensorflow [here](https://www.tensorflow.org/install/).

## About Tensoflow mobile:

Tensorflow mobile is nothing but saving the tensorflow program written in python in to small file that can be read by the cpp native libraries what we will install in our android app and can execute and do the inference from the mobile. To do so, JNI (Java native interface) is working as a bridge b/w java and c++.

To know about the idea behind the tensorflow lite please visit <https://www.tensorflow.org/mobile/tflite/>.



Preparing the TF Model

First, we first create a simple model and save its computation graph as a serialized GraphDef file. After training the model, we then save the values of its variables into a checkpoint file. We have to turn these two files into an optimized standalone file, which is all we need to use inside the Android app.

Creating and Saving the Model

For this tutorial, we create a very simple Tensorflow graph that implements a small usecase that will calcite (a+b)2. Here we are saving the input as a and b. And output as c.

This network might seem too simple and lack an actual learning, but I think it is enough to demonstrate the point. I know I could have been more creative here ;)

import tensorflow as tf  
  
a = tf.placeholder(tf.int32, name='a') # input  
b = tf.placeholder(tf.int32, name='b') # input  
times = tf.Variable(name="times", dtype=tf.int32, initial\_value=2)  
c = tf.pow(tf.add(a, b), times, name="c")  
  
saver = tf.train.Saver()  
init\_op = tf.global\_variables\_initializer()  
with tf.Session() as sess:  
 sess.run(init\_op)  
  
 tf.train.write\_graph(sess.graph\_def, '.', 'tfdroid.pbtxt')  
 sess.run(tf.assign(name="times", value=2, ref=times))  
 # save the graph  
  
 # save a checkpoint file, which will store the above assignment  
 saver.save(sess, './tfdroid.ckpt')

Running the above piece of code will produce two files: first, it saves the TF computation graph in a GraphDef text file called tfdroid.pbtxt. Next, it will do a simple assignment (which normally would be done through actual learning) and saves a checkpoint of the model variables in tfdroid.ckpt.

Freezing the Graph

Now that we have these files, we need to freeze the graph by converting the variables in the checkpoint file into Const Ops that contain the values of the variables, and combining them with the GraphDef proto in a single standalone file. Using this file makes it easier to load the model inside a mobile app. Tensorflow provides freeze\_graph in tensorflow.python.toolsfor this purpose:

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 = "c"  
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, "")

Optimizing the Model File

Once we have the frozen graph, we can further optimize the file for inference-only purposes by removing the parts of the graph that are only needed during training. According to the documentation, these include:

* Removing training-only operations like checkpoint saving.
* Stripping out parts of the graph that are never reached.
* Removing debug operations like CheckNumerics.
* Folding batch normalization ops into the pre-calculated weights.
* Fusing common operations into unified versions.

Tensorflow provides optimize\_for\_inference\_lib in tensorflow.python.tools for this purpose:

# 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,  
 ["a", "b"], # an array of the input node(s)  
 ["c"], # an array of output nodes  
 tf.int32.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)

Take note of the input nodes and output nodes in the above code. Our graph only has one input node named I, and one output node named O. These names correspond to the names you use when you define your tensors. You should adjust these based on your graph in case you are using a different one.

Now we have a binary file called optimized\_tfdroid.pb. Now we are ready to build our Android app. If you got exception in creating optimized\_tfdroid.pb. You can use the tfdroid.pb which is not an optimized version of model. Which is some what big in size.

Creating the Android App

We need to get the tensorflow libraries for Android, create an Android app and configure it to use these libraries, and then invoke the tensorflow model inside the app.

Getting the TF Libraries

Although you can [compile the Tensorflow libraries from scratch](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/android), it’s easier to use the prebuilt libraries that are provided in [nightly android builds](https://ci.tensorflow.org/view/Nightly/job/nightly-android/).

For this tutorial, I’m using the lastest successful build at the time of this writing, *Build #44 (Feb 17, 2017 12:05:00 AM)*, which you can download from:

<https://ci.tensorflow.org/view/Nightly/job/nightly-android/44/artifact/>

Inside the nightly android builds, will find the installable package (.apk) for the official demo apps, the Java library (.jar), and the native shared libraries (.so) for different architectures.

Creating an Empty APP

I used Android Studio to create an Android project with an empty activity.

Add the TF Libraries to Your Project

Once you have all the output artifacts from the nightly build, you need to add some of them to your project and let your build system know that you are going to use them. For this example, these include the Java and libtensorflow\_inference native library.

Copy the libandroid\_tensorflow\_inference\_java.jar and the architecture folders inside of the libtensorflow\_inference.so folder to app/libs/. The libs/ folder should look like:

libs

|\_\_\_\_arm64-v8a

| |\_\_\_\_libtensorflow\_inference.so

|\_\_\_\_armeabi-v7a

| |\_\_\_\_libtensorflow\_inference.so

|\_\_\_\_libandroid\_tensorflow\_inference\_java.jar

|\_\_\_\_x86

| |\_\_\_\_libtensorflow\_inference.so

|\_\_\_\_x86\_64

| |\_\_\_\_libtensorflow\_inference.so

You need to let your build system know where these libraries are located by putting the following lines inside of the android block in app/build.gradle:

    sourceSets {  
        main {  
            jniLibs.srcDirs = ['libs']  
        }  
    }

Copying the TF Model

Create an *Android Asset Folder* for the app and put the optimized\_tfdroid.pb or tfdroid.pb that we just created inside (app/src/main/assets/).

## Creating an activity

Click on the project and create an empty activity name MainActivity. And in the layout of that activity paste the following xml.

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/activity\_main"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context="com.example.vavinash.tensorflowsample.MainActivity">

<EditText

android:id="@+id/editNum1"

android:layout\_width="100dp"

android:layout\_height="wrap\_content"

android:layout\_alignParentTop="true"

android:layout\_marginEnd="13dp"

android:layout\_marginTop="129dp"

android:layout\_toStartOf="@+id/button"

android:ems="10"

android:hint="a"

android:inputType="textPersonName"

android:textAlignment="center" />

<EditText

android:id="@+id/editNum2"

android:layout\_width="100dp"

android:layout\_height="wrap\_content"

android:layout\_alignBaseline="@+id/editNum1"

android:layout\_alignBottom="@+id/editNum1"

android:layout\_toEndOf="@+id/button"

android:ems="10"

android:hint="b"

android:inputType="textPersonName"

android:textAlignment="center" />

<Button

android:text="Run"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/button"

android:layout\_below="@+id/editNum2"

android:layout\_centerHorizontal="true"

android:layout\_marginTop="50dp" />

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Output"

android:id="@+id/txtViewResult"

android:layout\_marginTop="85dp"

android:textAlignment="center"

android:layout\_alignTop="@+id/button"

android:layout\_centerHorizontal="true" />

</RelativeLayout>

In the mainactivity.java paste the flowing code.

package com.example.vavinash.tensorflowsample;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Button;

import android.view.View;

import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

public class MainActivity extends AppCompatActivity {

//change with the file name of your own model generated in python tensorflow.

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

//here we are using this interface to perform the inference with our generated model. It internally uses c++ libraries and JNI.

private TensorFlowInferenceInterface inferenceInterface;

static {

System.loadLibrary("tensorflow\_inference");

}

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

inferenceInterface = new TensorFlowInferenceInterface();

//instantiatind and setting our model file as input.

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

final Button button = (Button) findViewById(R.id.button);

button.setOnClickListener(new View.OnClickListener() {

public void onClick(View v) {

final EditText editNum1 = (EditText) findViewById(R.id.editNum1);

final EditText editNum2 = (EditText) findViewById(R.id.editNum2);

float num1 = Float.parseFloat(editNum1.getText().toString());

float num2 = Float.parseFloat(editNum2.getText().toString());

int[] i = {1};

int[] a = {((int) num1)};

int[] b = {((int) num2)};

//Setting input for variable a and b in our model.

inferenceInterface.fillNodeInt("a",i,a);

inferenceInterface.fillNodeInt("b",i,b);

//performing the inference and getting the output in variable c

inferenceInterface.runInference(new String[] {"c"});

//reading received output

int[] c = {0};

inferenceInterface.readNodeInt("c", c);

//projecting to user.

final TextView textViewR = (TextView) findViewById(R.id.txtViewResult);

textViewR.setText(Integer.toString(c[0]));

}

});

}

}

Now run the app.