TensorFlow的训练模型在Android和Java的应用及调用

由 谭东 创建， 最后一次修改 2017-09-01

作者：谭东

时间：2017年5月29日

环境：Windows 7

TensorFlow ™

当我们开始学习编程的时候，第一件事往往是学习打印"Hello World"。就好比编程入门有Hello World，机器学习入门有MNIST。

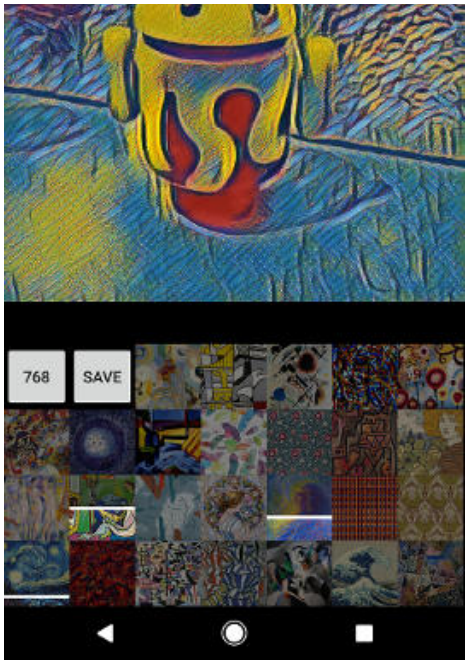
MNIST是一个入门级的计算机视觉数据集，它包含各种手写数字图片：

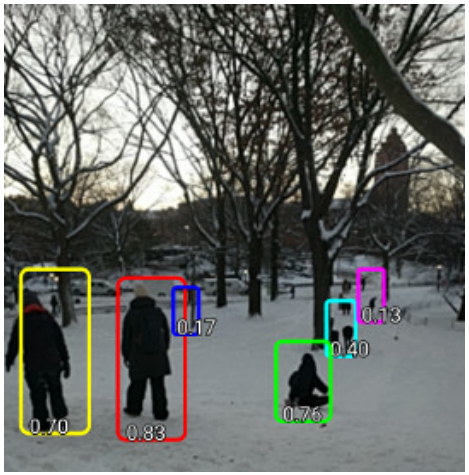
它也包含每一张图片对应的标签，告诉我们这个是数字几。比如，上面这四张图片的标签分别是5，0，4，1。

那我我们就将TensorFlow里的一个训练后的模型数据集，在Android里实现调用使用。

第一个txt文件展示了这个pb训练模型可以识别的东西有哪些。

```
English springer
grey whale
lesser panda
Egyptian cat
ibex
Persian cat
cougar
gazelle
porcupine
sea lion
malamute
badger
Great Dane
Walker hound
Welsh springer spaniel
whippet
Scottish deerhound
killer whale
mink
African elephant
Weimaraner
soft-coated wheaten terrier
Dandie Dinmont
red wolf
Old English sheepdog
jaguar
otterhound
bloodhound
Airedale
hyena
meerkat
giant schnauzer
titi
three-toed sloth
sorrel
```





那么我们接下来就是在android或Java里调用API使用他这个训练模型，实现图像识别功能。

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android 这个是TensorFlow官方的Demo源码。

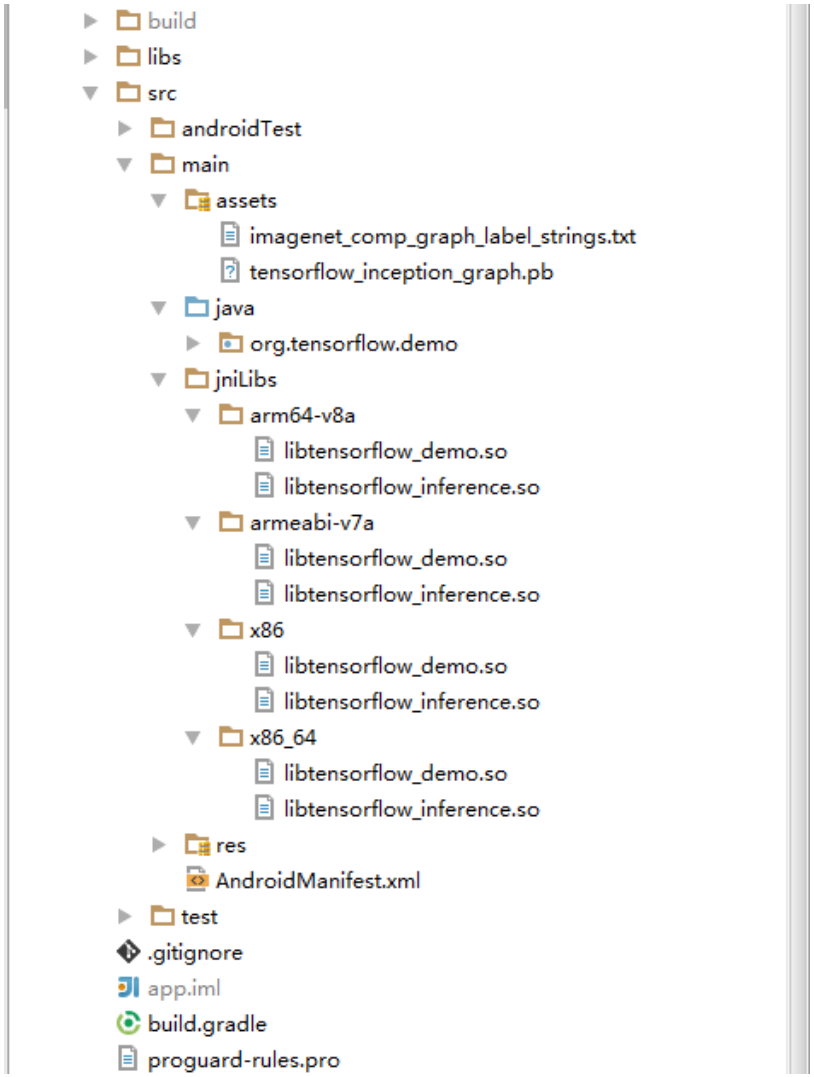
Android想要使用要编译so，毕竟是跨平台调用。

jni在官方Demo里也附带了。

Android和TensorFlow调用API的aar库可以在gradle里引用：

```
compile 'org.tensorflow:tensorflow-android:+'
```

基本结构：



基本API调用训练模型如下代码类似：

主要的类就是TensorFlowInferenceInterface 、Operation。

那么接下来把官方Demo的这个类调用给出：

他这个是Android的Assets目录读取训练模型， 从

c.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);

这句可以看出。

那么我们可以根据实际训练模型pb文件的位置进行修改引用。

```
/* Copyright 2016 The TensorFlow Authors. All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
==========================================================================*/

package org.tensorflow.demo;

import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.os.Trace;
import android.util.Log;
import java.io.BufferedReader;
import java.io.IOException;
```

```
import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

/** A classifier specialized to label images using TensorFlow. */
public class TensorFlowImageClassifier implements Classifier {
  private static final String TAG = "TensorFlowImageClassifier";

  // Only return this many results with at least this confidence.
  private static final int MAX_RESULTS = 3;
  private static final float THRESHOLD = 0.1f;

  // Config values.
  private String inputName;
  private String outputName;
  private int inputSize;
  private int imageMean;
  private float imageStd;

  // Pre-allocated buffers.
  private Vector<String> labels = new Vector<String>();
  private int[] intValues;
  private float[] floatValues;
  private float[] outputs;
  private String[] outputNames;

  private boolean logStats = false;

  private TensorFlowInferenceInterface inferenceInterface;

  private TensorFlowImageClassifier() {}
```

```
   * @param inputSize The input size. A square image of inputSize x inputSize is
assumed.
   * @param imageMean The assumed mean of the image values.
   * @param imageStd The assumed std of the image values.
   * @param inputName The label of the image input node.
   * @param outputName The label of the output node.
   * @throws IOException
   */
  public static Classifier create(
      AssetManager assetManager,
      String modelFilename,
      String labelFilename,
      int inputSize,
      int imageMean,
      float imageStd,
      String inputName,
      String outputName) {
    TensorFlowImageClassifier c = new TensorFlowImageClassifier();
    c.inputName = inputName;
    c.outputName = outputName;

    // Read the label names into memory.
    // TODO(andrewharp): make this handle non-assets.
    String actualFilename = labelFilename.split("file:///android_asset/")[1];
    Log.i(TAG, "Reading labels from: " + actualFilename);
    BufferedReader br = null;
    try {
      br = new BufferedReader(new
InputStreamReader(assetManager.open(actualFilename)));
```

```
    }

    c.inferenceInterface = new TensorFlowInferenceInterface(assetManager,
modelFilename);

    // The shape of the output is [N, NUM_CLASSES], where N is the batch size.
    final Operation operation = c.inferenceInterface.graphOperation(outputName);
    final int numClasses = (int) operation.output(0).shape().size(1);
    Log.i(TAG, "Read " + c.labels.size() + " labels, output layer size is " +
numClasses);

    // Ideally, inputSize could have been retrieved from the shape of the input
operation.  Alas,
    // the placeholder node for input in the graphdef typically used does not
specify a shape, so it
    // must be passed in as a parameter.
    c.inputSize = inputSize;
    c.imageMean = imageMean;
    c.imageStd = imageStd;

    // Pre-allocate buffers.
    c.outputNames = new String[] {outputName};
    c.intValues = new int[inputSize * inputSize];
    c.floatValues = new float[inputSize * inputSize * 3];
    c.outputs = new float[numClasses];

    return c;
  }
```

```
    // on the provided parameters.
    bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(),
bitmap.getHeight());
    for (int i = 0; i < intValues.length; ++i) {
      final int val = intValues[i];
      floatValues[i * 3 + 0] = (((val >> 16) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 1] = (((val >> 8) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 2] = ((val & 0xFF) - imageMean) / imageStd;
    }
    Trace.endSection();

    // Copy the input data into TensorFlow.
    Trace.beginSection("feed");
    inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
    Trace.endSection();

    // Run the inference call.
    Trace.beginSection("run");
    inferenceInterface.run(outputNames, logStats);
    Trace.endSection();

    // Copy the output Tensor back into the output array.
    Trace.beginSection("fetch");
    inferenceInterface.fetch(outputName, outputs);
    Trace.endSection();

    // Find the best classifications.
    PriorityQueue<Recognition> pq =
        new PriorityQueue<Recognition>(
```

```
            }
          });
    for (int i = 0; i < outputs.length; ++i) {
      if (outputs[i] > THRESHOLD) {
        pq.add(
            new Recognition(
                "" + i, labels.size() > i ? labels.get(i) : "unknown",
outputs[i], null));
      }
    }
    final ArrayList<Recognition> recognitions = new ArrayList<Recognition>();
    int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
    for (int i = 0; i < recognitionsSize; ++i) {
      recognitions.add(pq.poll());
    }
    Trace.endSection(); // "recognizeImage"
    return recognitions;
  }

  @Override
  public void enableStatLogging(boolean logStats) {
    this.logStats = logStats;
  }

  @Override
  public String getStatString() {
    return inferenceInterface.getStatString();
  }
```