Tengine Face Detector

(20181223)

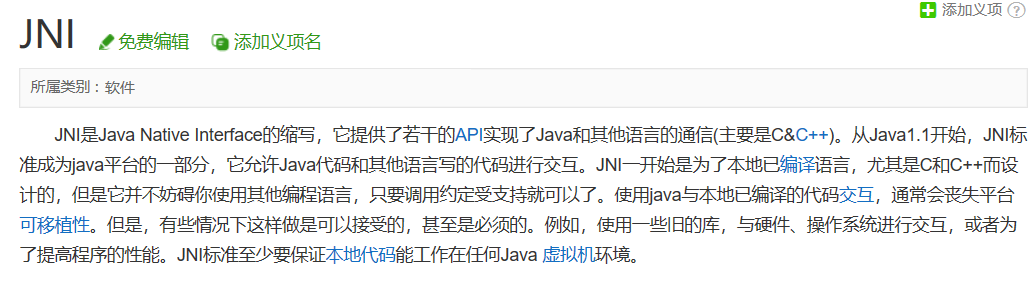
Preface: 在之前学习了Tengine在安卓上的相关配置(01\_Tengine\_Android)和Android Studio上使用OpenCV之后终于步入正题，进入我们这个比赛的内容，检测人脸的特征点。之前其实都没有好好看过代码，所以这篇文章就作为Tengine\_FaceDetector的代码解读吧！其实东西做完了却不知从哪里说起，回想起来这几天做的很多事情，却非常杂乱，行文之间若有逻辑不清之处还望海涵。

1. 整体框架

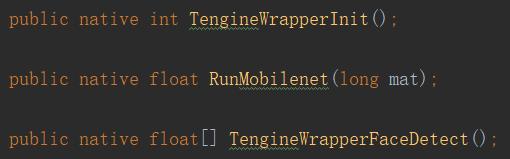
那我们就从这个项目的最终完成版说起吧，之前说了本文主要以解读代码为主，那就先来看看整个项目工程的框架。主要是分成这么几部分，我们一一介绍：

1. Android平台的java语言，这部分主要是对安卓手机界面进行编程，看着《第一行代码》一步步走下去就行了，这部分算是顶层代码吧~，主要就是activity文件以及界面xml文件。
2. JNI接口，这是个啥玩意呢？请看词条：

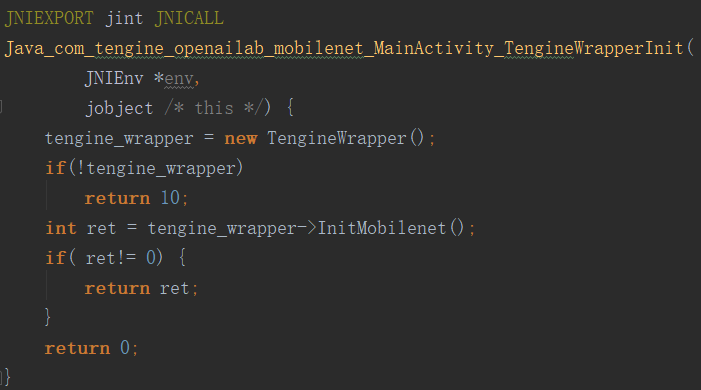
<https://baike.so.com/doc/508314-538197.html>



意思也就是说，JNI可以当做一个接口，使得Java语言能够调用底层的C++/C语言，也可以是其他语言。要使用这些接口需要首先进行函数申明，一般是在MainActivity类的最后：



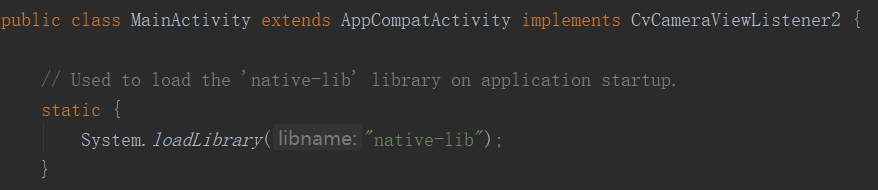
然后呢这些函数就链接到一个叫做native-lib.cpp的文件当中：



这里其实你就会发现一些东西，比如在JNI里面函数的定义与申明的不同，首先是数据类型的转换，java与JNI当中是一一对应的：



这里没有提到数组，其实JNI里面有特定的数组格式：比如jobjectArray、jbooleanArray、jbyteArray、jcharArray、jshortArray、jintArray、jlongArray、jfloatArray、jdoubleArray等，还有jclass、jstring、jthrowable等。那么其实这里就有个问题了，为什么顶层的java语言能够调用JNI接口文件呢？这就是下面这段话的功劳：



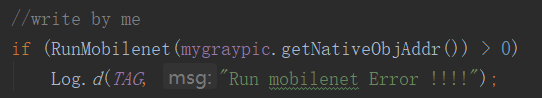
看到了吧这里在MainActivity的一开始就load了这个名字叫做native-lib的库。所以他们能够链接起来。

那么在做这个项目的时候呢就有一个问题出现了：顶层activity文件中如果我要输入一个图片文件如Mat格式的数据到底层去处理，数据接口应该怎么写呢？这个问题曾经困扰过，因为一开始的时候我是用Mat格式的数据直接传的，后来发现这样传递数据根本没有传到底层，而且还会导致app闪退，幸好这个已经有前人做过了，下面这篇博文就讲述了java层到JNI的Mat数据的接口处理：

<https://blog.csdn.net/brcli/article/details/76407986>

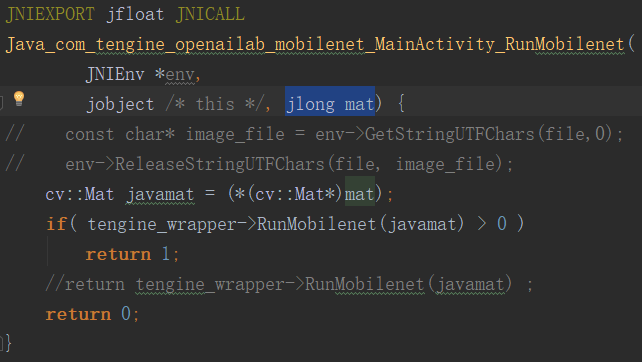
<https://blog.csdn.net/pplxlee/article/details/52713311>

简单说来就是这样：java层新建一个mat数据，然后通过调用JNI接口函数传入mat.getNativeObjAddr()就可以将mat数据传进函数，



顾名思义getNativeObjAddr()就是获取本地对象的地址，返回的是一个jlong类型的数据。所以下面我们在JNI进行调用的时候格式是这样的：

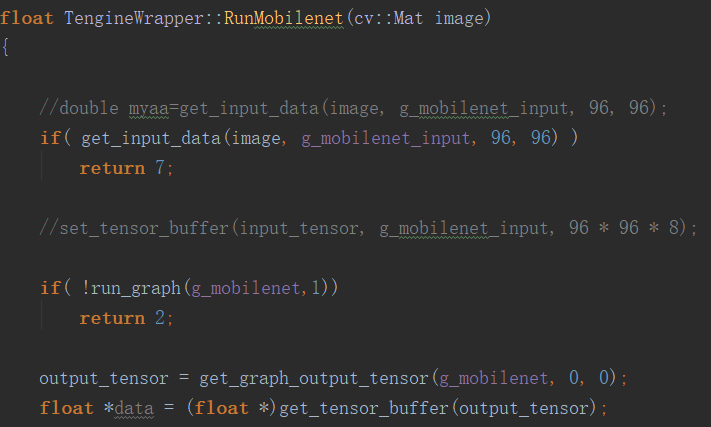




这里稍微解释一下这句：cv::Mat javamat = (\*(cv::Mat\*)mat);

jlong mat是一个地址，先将它转换成mat类型指针，然后取该地址的值作为JNI里面mat变量的值，这就实现了mat数据的传递。

然后我们发现这个新的javamat又被传递到C++/C里面的一个函数叫做RunMobilenet()，下面我们就去看看C++里面是怎么写的。这个RunMobilenet()是class TengineWrapper里面的一个函数，定义在Tengine\_Wrapper.h里面。具体的函数在Tengine\_Wrapper.cpp里面。



这里就可以看到，它的输入就是一个mat类型的数据。

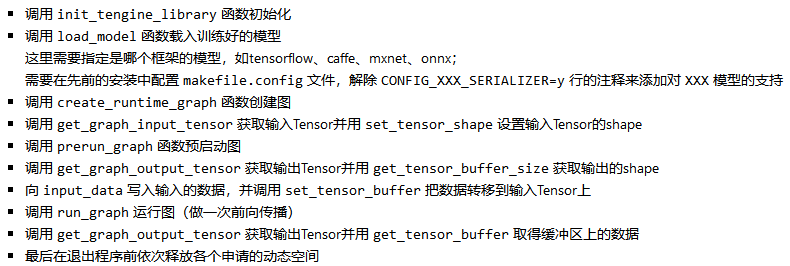
至此就讲完了整个接口的调用情况。

1. 底层C++/C函数

这部分就是底层代码啦，主要就在cpp文件夹里，里面是Tengine框架的调用，我们的数据图像处理就会在这里使用。在第二部分将会重点讲述这部分内容。

1. Tengine使用

大致上的调用流程如下：



他们所对应的代码如下，基本都在Tengine\_Wrapper.cpp里面：

**int** TengineWrapper::InitMobilenet()  
{  
 init\_tengine\_library();  
 **if** (request\_tengine\_version("0.1") < 0)  
 **return** -1;

**const char**\* mobilenet\_tf\_model = "/data/local/tmp/model3model.pb";  
 //const char\* mobilenet\_caffe\_proto = "/sdcard/openailab/models/mobilenet\_deploy.prototxt";  
 //const char\* mobilenet\_caffe\_model = "/sdcard/openailab/models/mobilenet.caffemodel";  
 **const char**\* format = "tensorflow";  
  
 **if** (load\_model("mobilenet", format, mobilenet\_tf\_model) < 0)  
 //if (load\_model("mobilenet", format, mobilenet\_caffe\_proto, mobilenet\_caffe\_model) < 0)  
 **return** 4;  
  
 g\_mobilenet = create\_runtime\_graph("graph0","mobilenet",NULL);  
 **if** (!check\_graph\_valid(g\_mobilenet))  
 **return** 5;  
  
 **const int** img\_h = 96;  
 **const int** img\_w = 96;  
  
 //int image\_size = img\_h \* img\_w \* 3;  
 **int** image\_size = img\_h \* img\_w;  
 g\_mobilenet\_input = (**float** \*) malloc(**sizeof**(**float**) \* image\_size);  
  
 //int dims[4] = {1, 3, img\_h, img\_w};  
 **int** dims[4] = {1, 1, img\_h, img\_w};  
  
 input\_tensor = get\_graph\_input\_tensor(g\_mobilenet, 0, 0);  
 **if**(!check\_tensor\_valid(input\_tensor))  
 **return** 6;  
  
 set\_tensor\_shape(input\_tensor, dims, 4);  
 set\_tensor\_buffer(input\_tensor, g\_mobilenet\_input, image\_size \* 4);  
  
 **if**( prerun\_graph(g\_mobilenet)!=0 )  
 **return** 1;  
  
 **return** 0;  
}

我对这里prerun预运行的理解是执行此条后输入数据的接口就做好了，直接给输入数据指针g\_mobilenet\_input赋值就可以了。

**float** TengineWrapper::get\_input\_data(cv::Mat sample, **float** \* data, **int** img\_h, **int** img\_w)  
{  
 **if** (sample.empty())  
 **return** 1;  
 cv::Mat img;  
 img = sample;  
 cv::resize(img, img, cv::Size(img\_h, img\_w));  
 img.convertTo(img, CV\_32FC1);  
 **float** \*img\_data = (**float** \*)img.data;  
 **for** (**int** h = 0; h < img\_h; h++)  
 {  
 **for** (**int** w = 0; w < img\_w; w++)  
 {  
 data[h \* img\_w + w] = (\*img\_data)\*0.0039215686;  
 img\_data++;  
 }  
 }  
 **return** 0;  
}

这段主要就是给输入赋值，直接运行到这里给输入数据指针data赋值就可以了，而这个函数的调用如下所示，可以看到就是之前的g\_mobilenet\_input

get\_input\_data(image, g\_mobilenet\_input, 96, 96)

然后就是运行图了，运行完成之后就可以将输出取出，用数据指针data表示：

**if**( !run\_graph(g\_mobilenet,1))  
 **return** 2;  
  
output\_tensor = get\_graph\_output\_tensor(g\_mobilenet, 0, 0);  
**float** \*data = (**float** \*)get\_tensor\_buffer(output\_tensor);

最后对输出数据进行处理，最好将new类型的数据都delete一下，否则也有可能让程序崩溃：

**float** \*data = (**float** \*)get\_tensor\_buffer(output\_tensor);  
  
**float** (\*datatmp)[30]=(**float** (\*)[30])**new float**[30];  
**for**(**int** i=0;i<30;i++)  
{  
 (\*datatmp)[i]=data[i];  
 //(\*datatmp)[i]=i+0.1;  
}  
**delete** datatmp;  
**return** datatmp;

数据返回到JNI层，随后从JNI层返回到java层使用。至此使用Tengine就可以基本完成了，但是最后能够依次释放一下各个申请的动态空间，但是好像不释放也没啥关系：

**int** TengineWrapper::ReleaseMobilenet()  
{  
 sleep(1);  
 free(g\_mobilenet\_input);  
 postrun\_graph(g\_mobilenet);  
 destroy\_runtime\_graph(g\_mobilenet);  
 remove\_model("mobilenet");  
 **return** 0;  
}

1. Tengine\_FaceDetector

最后再讲述一下这个Tengine\_FaceDetector的基本流程吧：

1. Java层使用一个org.opencv.android.JavaCameraView布局来调用摄像头获取图像
2. 使用opencv自带的分类器实现人脸框图
3. 将人脸框图灰度图传给底层使用tensorflow模型（model3model.pb文件）输出人脸的15个特征点坐标，15个特征点坐标在模型输出的 1\*30数组元素里面。分别是x1 y1 x2 y2 ......
4. 得到15个特征点的数据以后可以在java层画出特征点位置。