2018.10.30

基于 Android 人脸识别

孙浩翔 夏启林

版本信息

版本	日期	更改
V1.0	2018 年 10 月 30 日	创建文档

介绍

基于 Android 的脸部识别程序。经过 C++验证的脸部识别算法，和杭州艾芯智能 AXON M2 官方提供的安卓平台相结合。实现基于深度图像的人脸识别。并实时显示检测结果。

实现目标：1.C++程序移植到 android 平台，生成.apk 文件；

2.算法处理效果图形以及界面布局显示。

环境

操作系统：Windows 10

安卓 SDk：最低版本 Android 4.0

目标版本 Android 4.3 （安卓平台 NanoPC-T4 支持 Android7.0）

编写工具：Android Studio 3.1.4

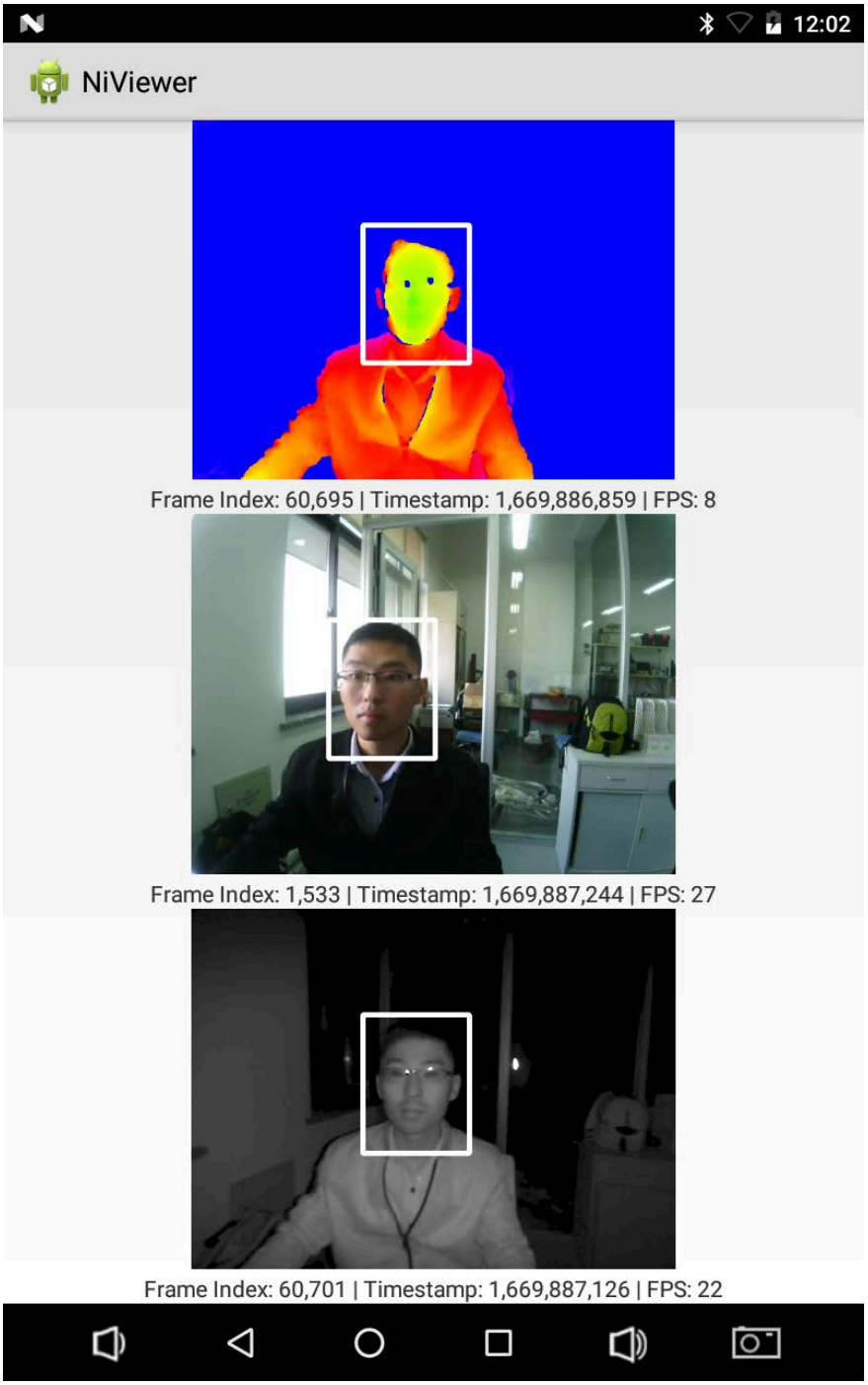
JRE 版本: 1.8.0_152-release-1024-b02 amd64

OpenCV： 2.4.3

OpenNI2

编译构建工具：Gradle 3.1.4 （需要联网下载）

软件界面



说明：1.通过 USB 连接相机，HDMI 外接显示器以及 USB 鼠标，安装 APP；

2.如上图，三张显示图片，第一张是相机实时采集的深度图像经过算法处理后的显示效果图，第二张是对应的 RGB 图，第三张是红外图；

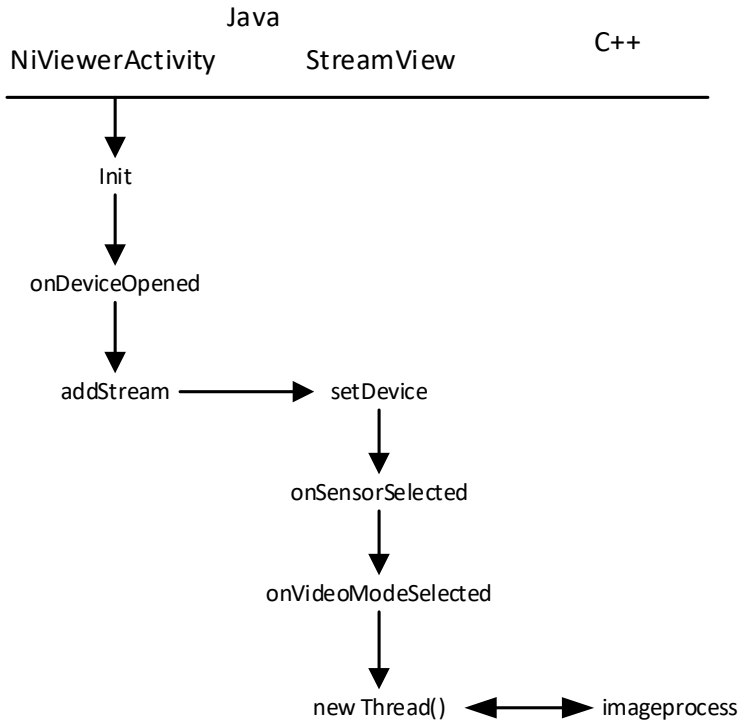
文件结构

Project 工程文件夹显示下 对应文件结构;

```
app
├ build
├ src
|    └ main
|         ├ assets
|         ├ cpp
|         |    ├ include                          // CPP 文件包含目录
|         |    |                                    （opencv 库及外部头文件）
|         |    └ native-lib.cpp                   // CPP 文件
|         ├ java
|         |    └ org.openni
|         |         ├ andoird
|         |         |     ├ tools.niviewer
|         |         |     |     ├ DeviceSelectDialog
|         |         |     |     ├ NiViewerActivity      //  启动 Activity（入口）
|         |         |     |     └ StreamView            //  数据流处理类
|         |         |     ├ OpenNIHelper
|         |         |     └ OpenNIView
|         |         ├ CamParam
|         |         ├ …
|         |         └ VideoStream
|         ├ jni
|         ├ jniLibs
|         |    ├ armeabi-v71                      // OpenCV、OpenNI 的 so 文件
|         |    └ libs
|         ├ res
|         |    ├ drawable-hdpi
|         |    ├ drawable-mdpi
|         |    ├ drawable-xhdpi
|         |    ├ drawable-xxhdpi
|         |    ├ layout
|         |    |    ├ activity_niviewer.xml       //  主窗口的布局文件
|         |    |    └ stream_view.xml             //  数据流处理布局文件
|         |    ├ menu
|         |    └ values
|         └ AndroidManifest.xml                   //  启动配置文件
├ build.gradle                                    // gradle 脚本
└ CMakeLists.txt                                  // CMakeLists 文件
```

运行流程

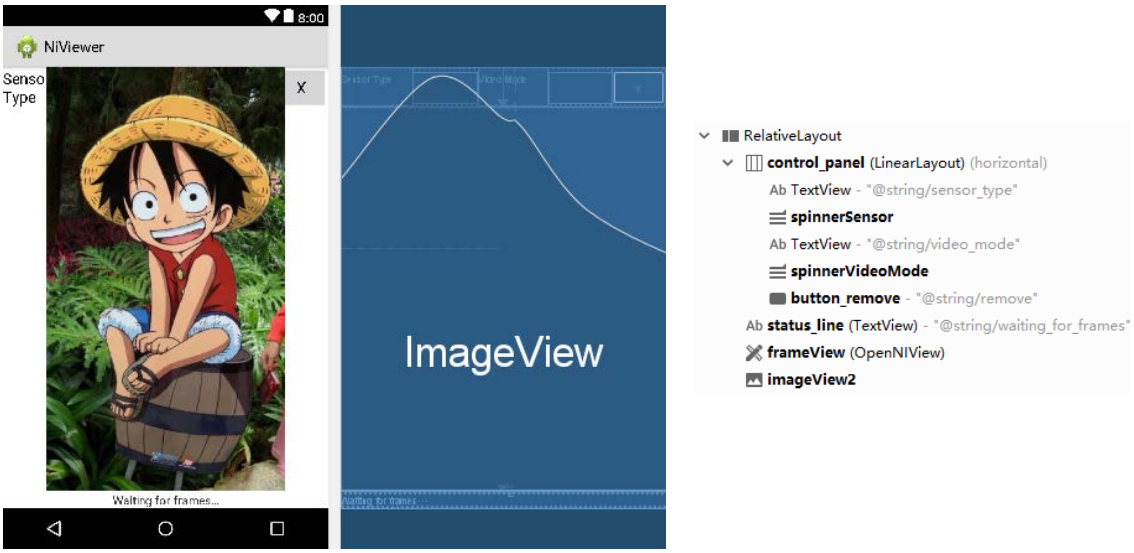
Java 部分



1. 程序启动，根据 AndroidManifest.xml 配置，首先激活 NiViewerActivity 类，进行界面初始化。初始化界面只有一个水平布局控件，后续会调用 StreamView 类添加图像显示内容。初始化 OpenNIHelper 类。
2. 实现 OpenNIHelper.DeviceOpenListener 的 onDeviceOpened 接口。当应用从后台切换到前台显示，外部设备打开的时候会调用监听器的 onDeviceOpened 函数，将显示的 StreamView 控件的设备修改为打开的设备。
3. 如果没有 StreamView 控件存在，调用 addStream 函数创建 StreamView 类，并添加到主显示窗口中。StreamView 类主要负责数据流的读取、解析和显示。初始化 StreamView 类的时候，依次调用 setDevice 函数设置控件显示的图片类型（Depth，Color，IR）。
4. 对设备的选择触发监听器调用 onSensorSelected 函数，根据传入的序号通知 VideoStream 类读取不同类型的图片数据流。根据选择的不同类型，更新视频模式下拉框，并默认选择第一个。
5. 选择视频模式会触发监听器调用 onVideoModeSelected 函数。设置 VideoStream 类视频模式，然后开启子线程。
6. 新线程循环读取视频数据流，经过格式转换后交给 C++程序进行人脸识别并在图片中标识识别结果，经过格式转换后交给 Java，在界面中展示识别结果。
7. Android 中相关的 view 和控件不是线程安全的。通过 Handler 传递消息，实现异步 UI 更新。

注意：

VideoStream 类的窗口布局包含了一个水平布局的 control_panel 控件，包含两个 TextView，两个 spinner 和一个 Button 控件。显示的时候将其高度设置为 1px，目的是为了隐藏功能，同时可以保证原有的程序框架不做大的改动。



C++部分

略

配置过程

NDK 配置

NDK 版本 18.1.5063045

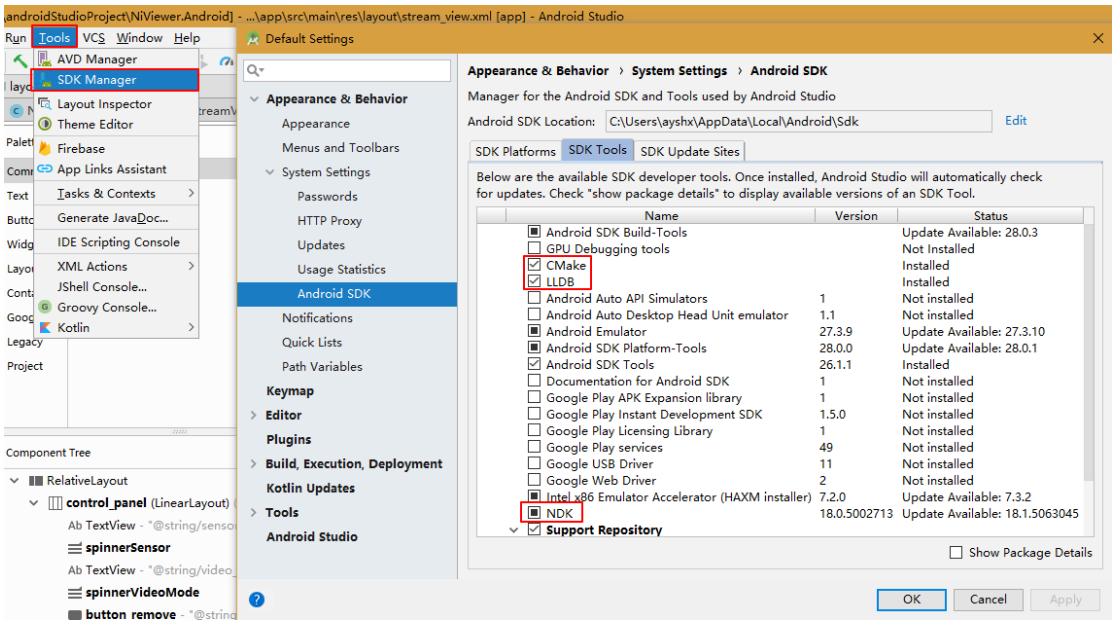
JNI：是 Java Native Interface 的缩写，它提供了若干的 API 实现了 Java 和其他语言的通信（主要是 C&C++）。JNI 是一套编程接口，用来实现 Java 代码与本地的 C/C++代码进行交互。

NDK: NDK 是 Google 开发的一套开发和编译工具集，可以生成动态链接库，主要用于 Android 的 JNI 开发。

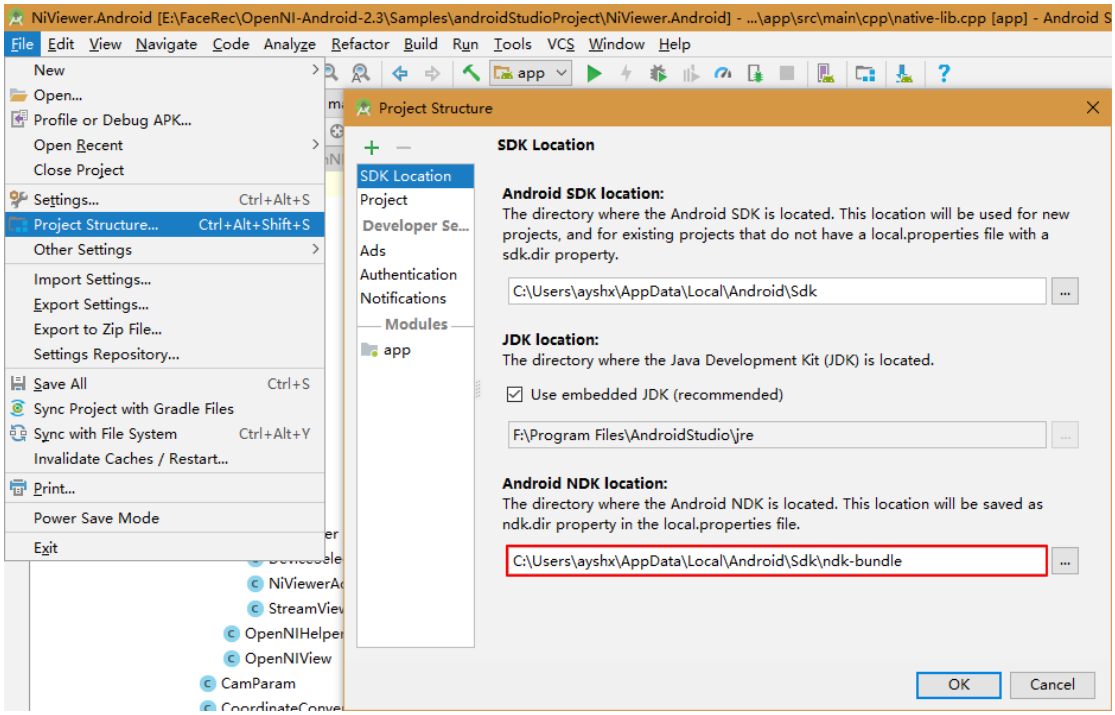
要为应用编译和调试原生代码，需要安装以下组件：
Android 原生开发工具包 (NDK)：这套工具集允许我们为 Android 使用 C 和 C++代码，且其提供众多平台库让我们可以管理原生 Activity 和访问物理设备组件，例如传感器和触摸输入。

CMake：一款外部构建工具，可与 Gradle 搭配使用来构建原生库。如果只计划使用 ndk-build（需要编写 Android.mk 文件和 Application.mk 文件），则不需要此组件，推荐使用 Cmake，编写 CMakLists.txt.
LLDB：一种调试程序，Android Studio 使用它来调试原生代码。

Android Studio 内提供了安装选项，tools->SDK Manager->Android SDK->SDK Tools，勾选 CMake、LLDB 和 NDK 后点击确定自动安装，根据网络环境大概需要十到二十分钟。



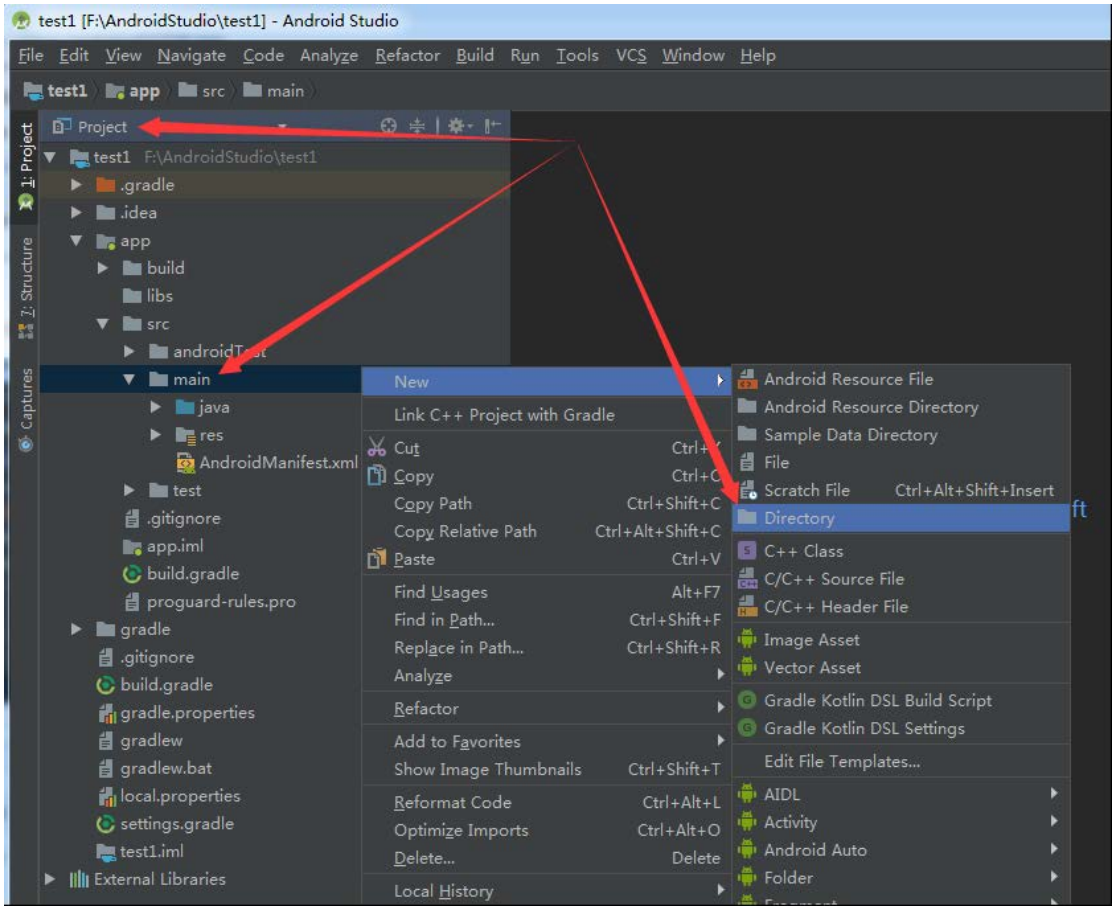
完成后点击“Finish”即可。
可以在工程配置中看到已经配置好的 NDK 目录。

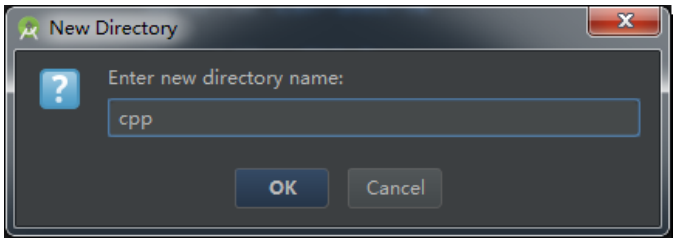


加入 C++支持

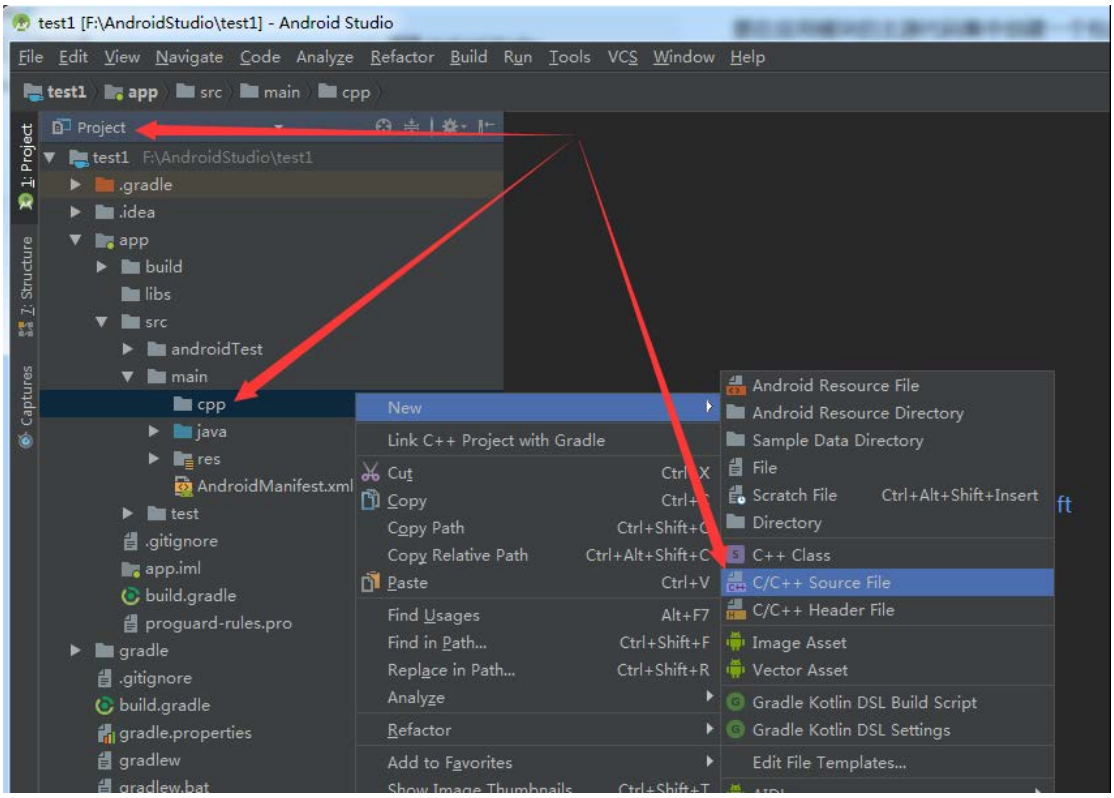
创建 CPP 文件

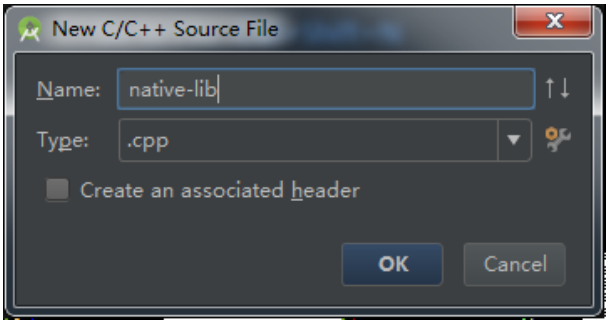
相机官方提供了一个 Android 工程，现对工程进行修改使其支持 C++代码的调用。
在 src 目录下创建一个新的文件夹 cpp。



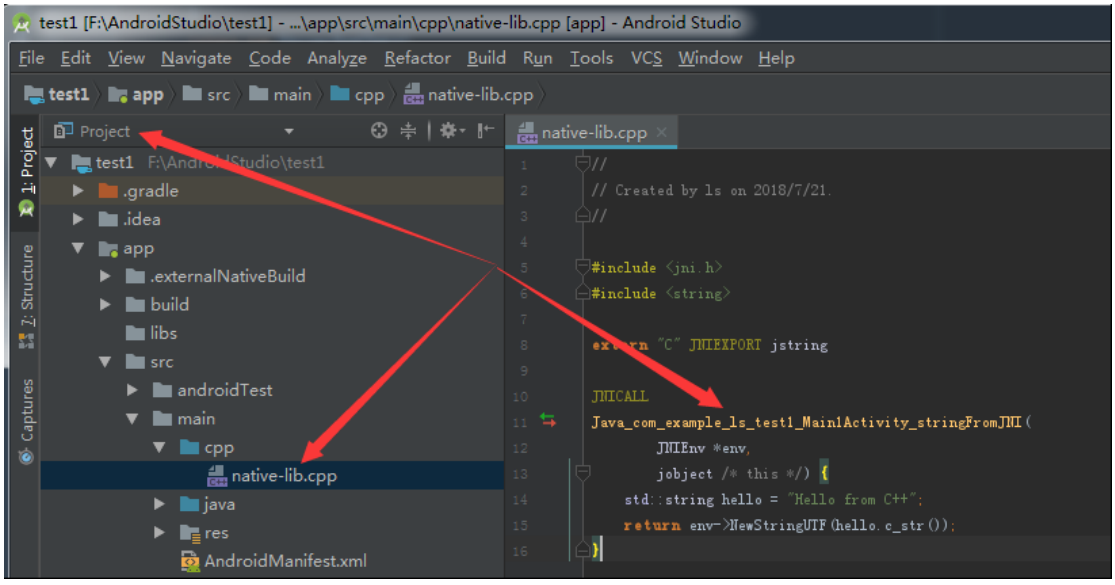


新建一个 CPP 文件 native-lib.cpp。写一个测试函数。



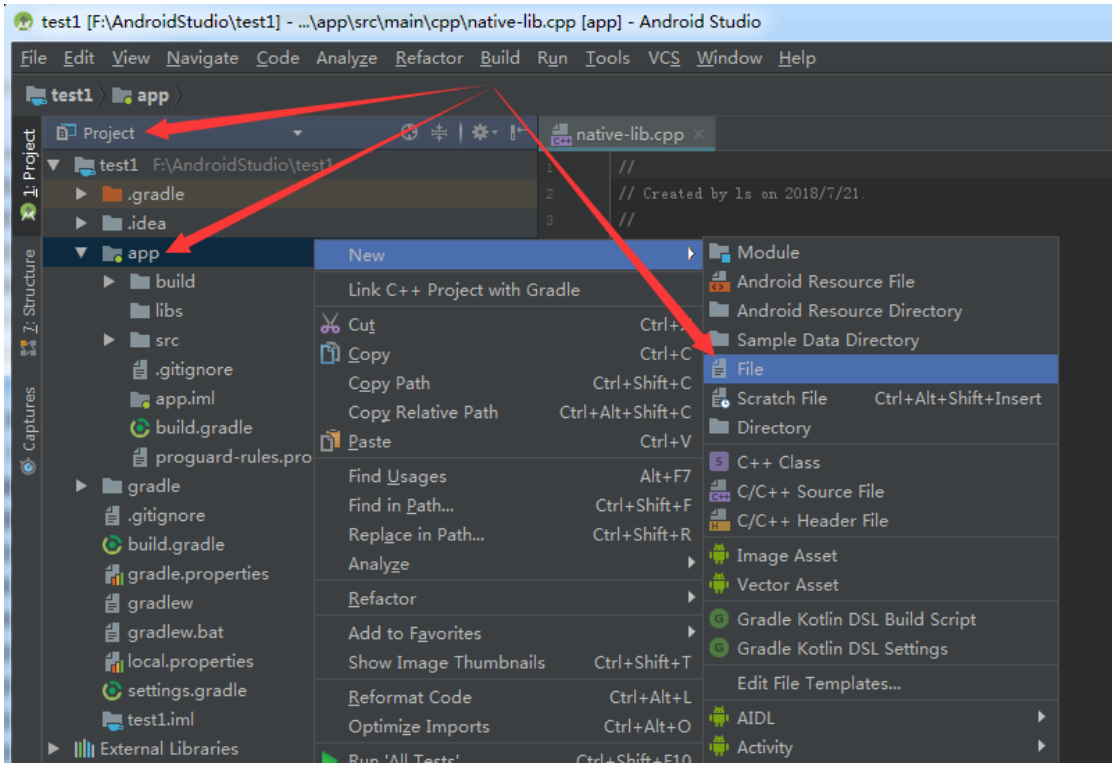


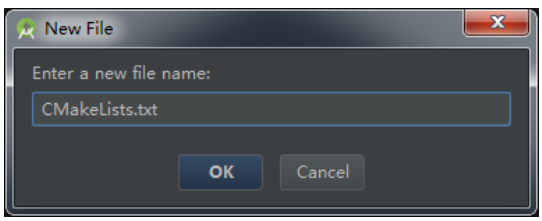
测试函数需要注意函数命名规则，Java_包名_类名_函数名(JNIEnv *env, jobject[，其他参数])。



```cpp
#include <jni.h>
#include <string>

using namespace std;

extern "C" JNIEXPORT jstring

JNICALL
Java_org_openni_android_tools_niviewer_NiViewerActivity_stringFromJNI(
        JNIEnv *env,
        jobject /* this */) {

    std::string hello = "Hello from C++";
    return env->NewStringUTF(hello.c_str());

}
```

在 app 目录下创建 CMakeLists.txt 文件





创建 CMake 构建脚本

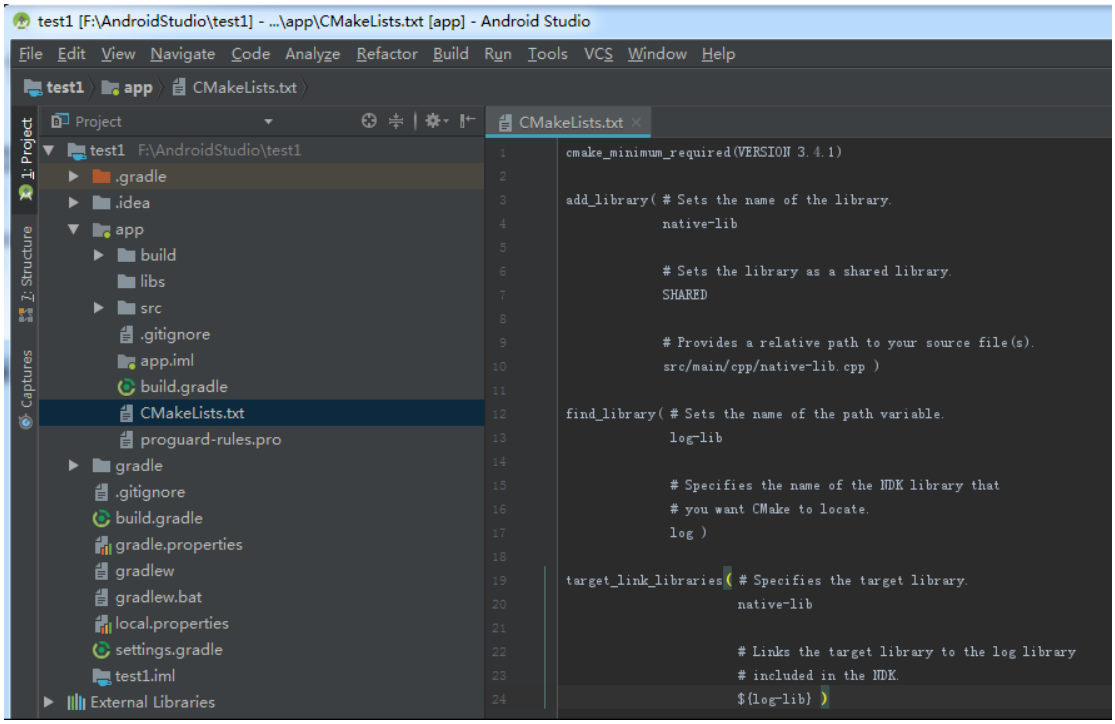
CMake 是一种跨平台编译工具，比 make 更为高级，使用起来要方便得多。CMake 主要是编写 CMakeLists.txt 文件，然后用 cmake 命令将 CMakeLists.txt 文件转化为 make 所需要的 makefile 文件，最后用 make 命令编译源码生成可执行程序或共享库。

Android Studio 采用 CMake 脚本语法配置 C 编译器的环境，如果之前有过使用 CMAKE 的经验，或许这并非难题，但对于初学者而言，CMAKE 的脚本语法，还是略过于生涩，而且 AS 对该文件的配置并不友好，居然没有代码提示，于是不得不查很多文档。但好在，NDK 的开发大多不是大型的 C++项目，也不太需要过于复杂的设置。

这里 CMakeLists 文件的目的就是告诉编译器需要编译的文件，文件和类之间的依赖关系，需要引入的动态库、头文件位置等编译信息。

写入以下内容:

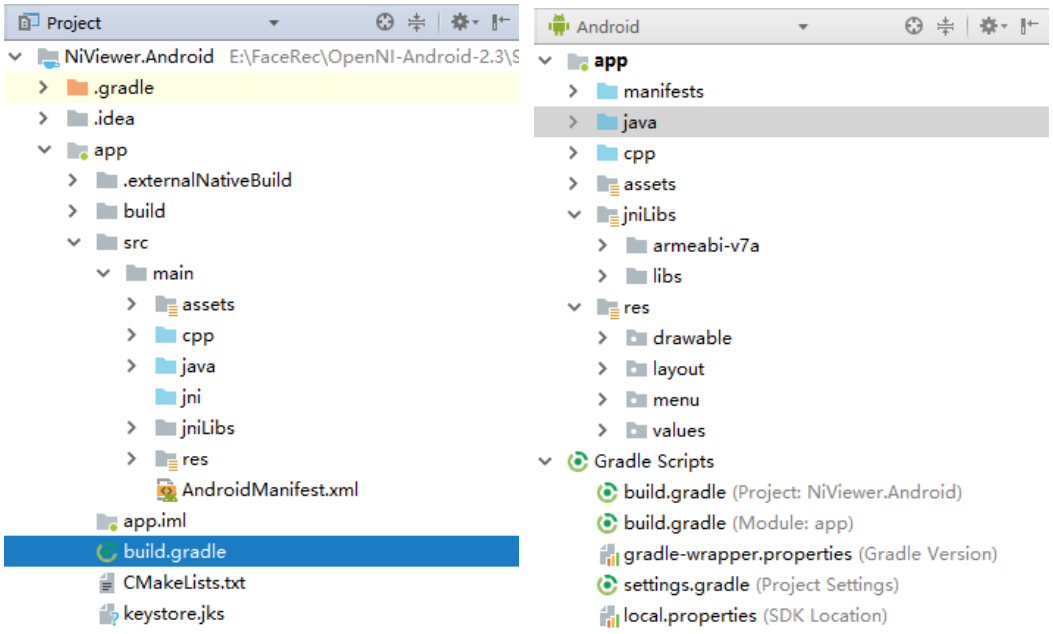
```
#规定 CMake 版本
cmake_minimum_required(VERSION 3.4.1)
add_library( # Sets the name of the library.
           native-lib
# Sets the library as a shared library.
           SHARED
# Provides a relative path to your source file(s).
           src/main/cpp/native-lib.cpp )
find_library( # Sets the name of the path variable.
            log-lib
# Specifies the name of the NDK library that
            # you want CMake to locate.
            log )
target_link_libraries( # Specifies the target library.
                    native-lib
# Links the target library to the log library
                    # included in the NDK.
                    ${log-lib} )
```



向 gradle 注册构建请求

简单的说，Gradle 是一个构建工具，它是用来帮助我们构建 app 的，构建包括编译、打包等过程。我们可以为 Gradle 指定构建规则，然后它就会根据我们的“命令”自动为我们构建 app。Android Studio 中默认就使用 Gradle 来完成应用的构建。

Project 是显示文件夹下所有文件，正确配置 Gradle 文件后，Android 下会显示所有跟项目相关的文件结构和文件。

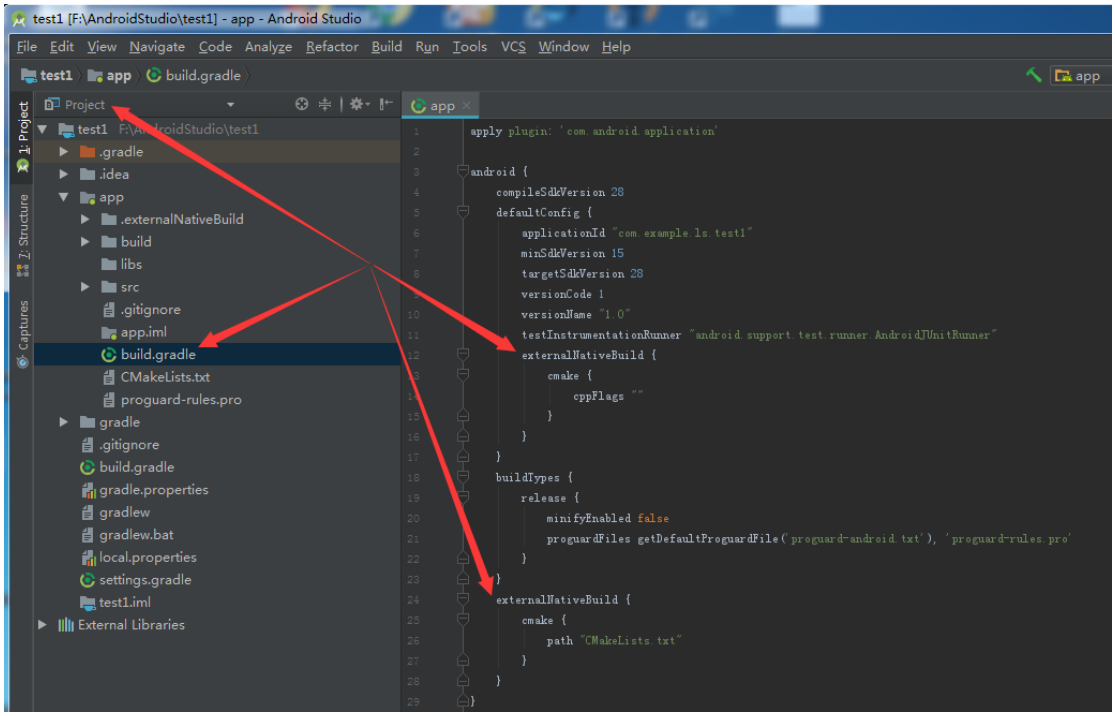


打开 build.gradle 向 android/defaultConfig 节区追加以下内容：

```
externalNativeBuild {
    cmake {
        cppFlags "-std=c++11 -frtti -fexceptions"
        abiFilters 'armeabi-v7a'
    }
}
```

向 android 节区追加以下内容:

```
externalNativeBuild{
    cmake{
        path "CMakeLists.txt"
    }
}
```



测试

在 NiViewerActivity 类中加入库:

```
static {
    System.loadLibrary("native-lib");
}
```

加入函数声明:

```
public native String stringFromJNI( );
```

程序中调用即可:

```
Toast.makeText(this,stringFromJNI(),Toast.LENGTH_LONG).show();
```

简单的完整程序如下:

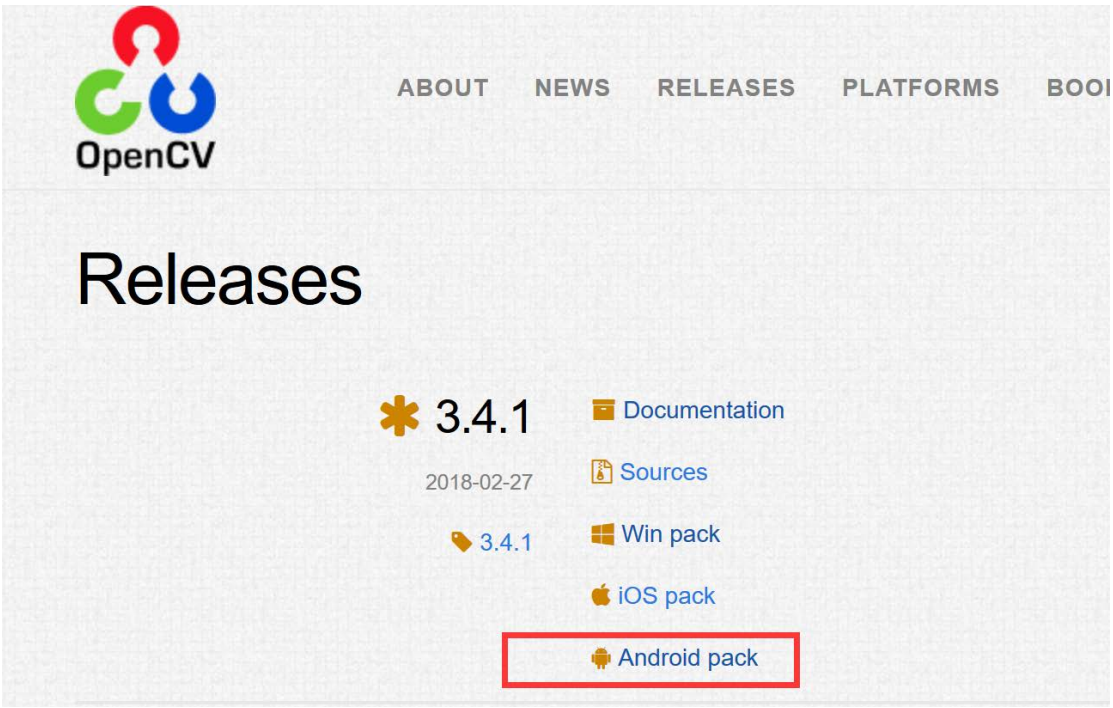
```
package com.example.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    // Used to load the 'native-lib' library on application
startup.
    static {
        System.loadLibrary("native-lib");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example of a call to a native method
        TextView tv = (TextView) findViewById(R.id.sample_text);
        tv.setText(stringFromJNI());
    }

    /**
     * A native method that is implemented by the 'native-lib'
native library,
     * which is packaged with this application.
     */
    public native String stringFromJNI();
}
```

运行后可以看到 Toast 消息框中显示 Hello from C++。

加入 OpenCV 的支持

下载相关文件

OpenCV 可以直接在官网上下，官方网址为：https://opencv.org/releases.html。选择 Android pack 版本即可，大约 310MB。



OpenCV 相关文件

• include 文件
在下载好的 OpenCV 压缩包中，打开路径下的.\opencv-2.4.3-android-sdk\OpenCV-android-sdk\sdk\native\jni 有一个 include 文件夹，把这个文件夹复制粘贴至我们的 OpenCVTest 项目中，路径为 src/main/cpp
• jni 文件
然后是动态库(.so 文件)，打开路径下的.\opencv-2.4.3-android-sdk\OpenCV-android-sdk\sdk\native\libs\ armeabi-v7a 文件夹，这个文件夹里面是 arm 版本的 so 文件。复制粘贴到我们的项目中，路径为 src/main/jniLibs/ armeabi-v7a。和 OpenNI 的 so 文件放在一起。路径是可以自己选择的，只是习惯上这样按类别放。
注意：由于 c++部分调用的 opencv 版本是 2.4.3，故建议保持一致性

配置文件

修改 app 文件夹下的 build.gradle 文件

```
apply plugin: 'com.android.application'
android {

    compileSdkVersion 28
    buildToolsVersion "28.0.1"
    defaultConfig {
        applicationId "org.openni.android.tools.niviewer"
        minSdkVersion 14
        targetSdkVersion 28
```

```
        externalNativeBuild {
            cmake {
                cppFlags "-std=c++11 -frtti -fexceptions"
                abiFilters 'armeabi-v7a'
            }
        }
    }
    sourceSets{
        main{

            jniLibs.srcDirs = ['src/main/jniLibs']
            //jniLibs.srcDirs = ['src/main/jniLibs/libs']
        }
    }
    splits {
        abi {
            enable true
            reset()
            include 'armeabi-v7a'
            universalApk true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }

    externalNativeBuild{
        cmake{
            path "CMakeLists.txt"
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
```

```
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

修改 CMakeLists 文件

```
# For more information about using CMake with Android Studio, read the
# documentation: https://d.android.com/studio/projects/add-native-code.html

# Sets the minimum version of CMake required to build the native library.

cmake_minimum_required(VERSION 3.4.1)

# 设置 include 文件夹的地址
include_directories(${CMAKE_SOURCE_DIR}/src/main/cpp/include)
include_directories(${CMAKE_SOURCE_DIR}/src/main/cpp)

# 设置 opencv 的动态库
add_library(libopencv_java SHARED IMPORTED)
set_target_properties(libopencv_java PROPERTIES IMPORTED_LOCATION
${CMAKE_SOURCE_DIR}/src/main/jniLibs/libs/armeabi-v7a/libopencv_java.so)



# Creates and names a library, sets it as either STATIC
# or SHARED, and provides the relative paths to its source code.
# You can define multiple libraries, and CMake builds them for you.
# Gradle automatically packages shared libraries with your APK.

add_library( # Sets the name of the library.
             native-lib

             # Sets the library as a shared library.
             SHARED

             # Provides a relative path to your source file(s).
             src/main/cpp/native-lib.cpp )

# Searches for a specified prebuilt library and stores the path as a
# variable. Because CMake includes system libraries in the search path by
# default, you only need to specify the name of the public NDK library
# you want to add. CMake verifies that the library exists before
# completing its build.

find_library( # Sets the name of the path variable.
              log-lib
```

```
              # Specifies the name of the NDK library that
              # you want CMake to locate.
              log )

# Specifies libraries CMake should link to your target library. You
# can link multiple libraries, such as libraries you define in this
# build script, prebuilt third-party libraries, or system libraries.

target_link_libraries( # Specifies the target library.
                       native-lib libopencv_java

                       # Links the target library to the log library
                       # included in the NDK.
                       ${log-lib} )
```

测试

这里提供已经简单的程序，适用于空工程。针对人脸识别的调用见具体代码实现。

- **MainActivity.java 文件**

```
package com.example.videomedicine.opencvtest;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener{

    // Used to load the 'native-lib' library on application startup.
    static {
        System.loadLibrary("native-lib");
    }

    private Button btn_1;
    private Button btn_2;
    private ImageView imageView;
```

```
    private Bitmap bitmap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn_1 = (Button)findViewById(R.id.button_1);
        imageView = (ImageView)findViewById(R.id.image);
        bitmap =
BitmapFactory.decodeResource(getResources(),R.drawable.luffy);
        imageView.setImageBitmap(bitmap);
        btn_1.setOnClickListener(this);

        btn_2 = (Button)findViewById(R.id.button_2);
        btn_2.setOnClickListener(this);
    }
    public void showImage(){
        bitmap =
BitmapFactory.decodeResource(getResources(),R.drawable.luffy);
        imageView.setImageBitmap(bitmap);
    }

    public void gray(){
        int w = bitmap.getWidth();
        int h = bitmap.getHeight();
        int[] piexls = new int[w*h];
        bitmap.getPixels(piexls,0,w,0,0,w,h);
        int[] resultData =Bitmap2Grey(piexls,w,h);
        Bitmap resultImage = Bitmap.createBitmap(w,h,
Bitmap.Config.ARGB_8888);
        resultImage.setPixels(resultData,0,w,0,0,w,h);
        imageView.setImageBitmap(resultImage);
    }

    @Override
    public void onClick(View view){
        switch(view.getId()){
            case R.id.button_1:showImage();break;
            case R.id.button_2:gray();break;
        }
    }

    /**
```

```
    * A native method that is implemented by the 'native-lib' native
 library,
    * which is packaged with this application.
    */
    public native int[] Bitmap2Grey(int[] pixels,int w,int h);

    @Override
    public void onResume(){
        super.onResume();
    }
}
```

- **native-lib.cpp 文件**

```
#include <jni.h>
#include<opencv2/opencv.hpp>
#include<iostream>
using namespace cv;
using namespace std;

extern "C" JNIEXPORT jintArray

JNICALL
Java_com_example_videomedicine_opencvtest_MainActivity_Bitmap2Grey(
        JNIEnv *env,
        jobject /* this */,jintArray buf,jint w,jint h) {
    jint *cbuf;
    jboolean ptfalse = false;
    cbuf = env->GetIntArrayElements(buf, &ptfalse);
    if(cbuf == NULL){
        return 0;
    }

    Mat imgData(h, w, CV_8UC4, (unsigned char*)cbuf);
    // 注意，Android 的 Bitmap 是 ARGB 四通道,而不是 RGB 三通道
    cvtColor(imgData,imgData,CV_BGRA2GRAY);
    cvtColor(imgData,imgData,CV_GRAY2BGRA);

    int size=w * h;
    jintArray result = env->NewIntArray(size);
    env->SetIntArrayRegion(result, 0, size, (jint*)imgData.data);
    env->ReleaseIntArrayElements(buf, cbuf, 0);
    return result;
}
```

- **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/image"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            app:srcCompat="@drawable/luffy" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="horizontal">

        <Button
            android:id="@+id/button_2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="灰度图" />

        <Button
            android:id="@+id/button_1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="色图" />
```

```
    </LinearLayout>

</LinearLayout>
```

参考

- Android NDK 学习笔记：Android Studio3.1+CMAKE+OpenCV3.4 配置
  https://blog.csdn.net/cv_jason/article/details/79758823
- Android Studio 向项目添加 C/C++原生代码教程
  http://www.cnblogs.com/lsdb/p/9337285.html
- Android Handler 消息传递机制
  https://www.cnblogs.com/zhaohongtian/p/6801596.html