



Amlogic

Android JNI 开发指导


Revision: 0.1

Release Date: 2021-08-10

Copyright

© 2021 Amlogic. All rights reserved. No part of this document may be reproduced. Transmitted, transcribed, or translated into any language in any form or by any means with the written permission of Amlogic.

Trademarks

 , and other Amlogic icons are trademarks of Amlogic companies. All other trademarks and registered trademarks are property of their respective companies.

Disclaimer

Amlogic may make improvements and/or changes in this document or in the product described in this document at any time.

This product is not intended for use in medical, life saving, or life sustaining applications.

Circuit diagrams and other information relating to products of Amlogic are included as a means of illustrating typical applications. Consequently, complete information sufficient for production design is not necessarily given. Amlogic makes no representations or warranties with respect to the accuracy or completeness of the contents presented in this document.

Contact Information

- Website: www.amlogic.com
- Pre-sales consultation: contact@amlogic.com
- Technical support: support@amlogic.com

更改记录

版本 0.1 (2021-08-10)

第一版。

Confidential!
nick@khadas.com
2021-09-07 10:49:41

目录

更改记录.....	2
1. Android NN JNI 架构.....	5
2. Android NN JNI Release 包简介	7
3. NN JNI 开发.....	8
3.1 开发环境配置.....	8
3.2 NN JNI 接口说明	9
3.3 app 中 NN JNI 使用	10
4.Android nn jni demo	11

Confidential!
nick@khadas.com
2021-09-07 10:49:41

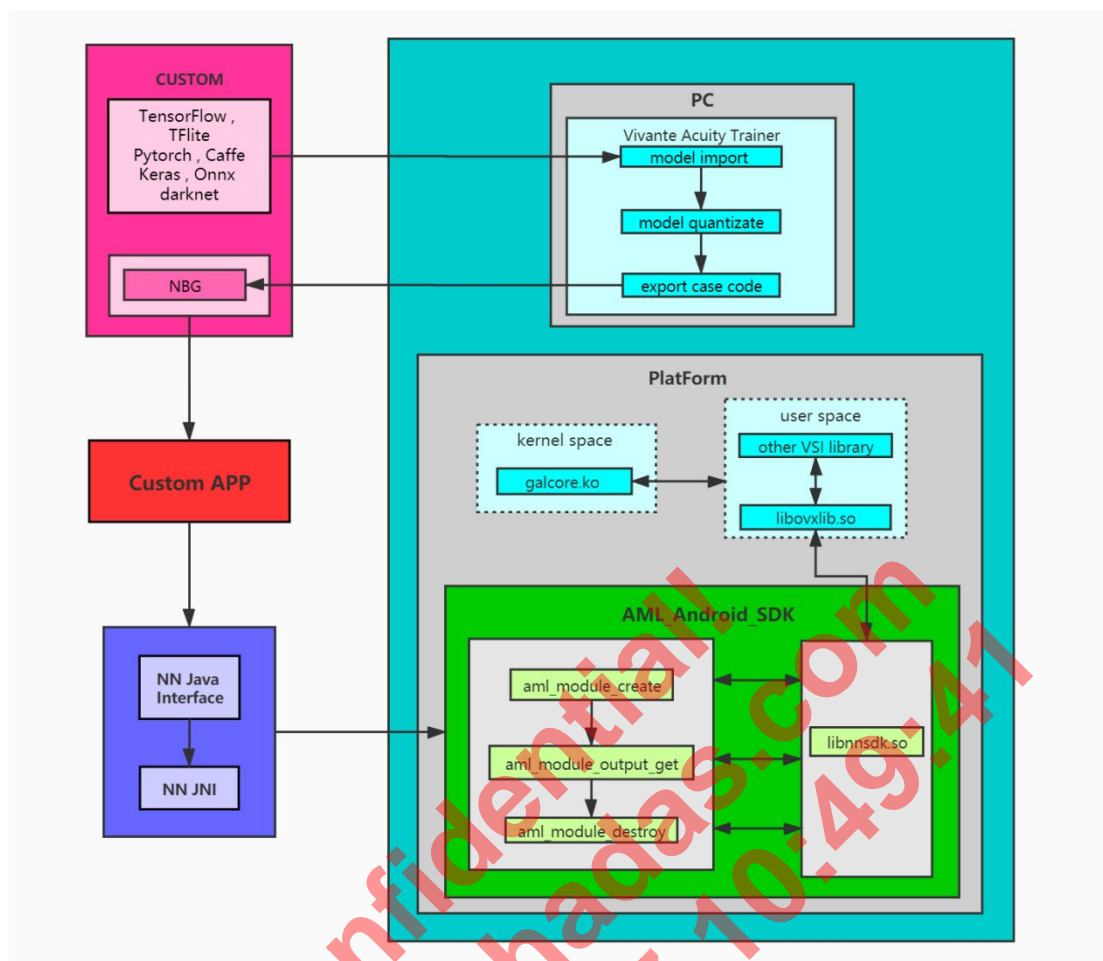
1. Android NN JNI 架构

Android 8.1 之后的版本包含了 Neural Networks API(NNAPI)，该接口可以为 google 基于 tensorflow lite 架构开发和训练的深度学习网络提供底层支持。基于 android 固有的 nnapi 架构与开发流程文档参考《android NN apk 开发流程指导》与《Android&Linux 编译集成指导(0.2)》。这里不做赘述。

基于 android 标准 NN 架构的方案有其固有的缺点，主要是支持的开源架构较少(仅支持 tflite)，支持的 op 算子较少，以及与底层的接口对接受限于 NPU 厂家的技术开发成熟度，不能完美的发挥底层 NPU 的真实性能，在对性能要求较高的场景下，该方案有些捉衿见肘。基于 android jni 架构直接访问底层硬件是该方案的一种有效补充。

Android APP 层通过 JNI 接口访问底层动态库是一种较常见的实现方式，具有灵活度高，可定制性强，实现简单，方便移植等优点。就 android NN 模块而言，在实际应用时可以更好的适配特定的算子及运行环境，更好的桥接到底层硬件。这是 NN JNI 机制设计的初衷。

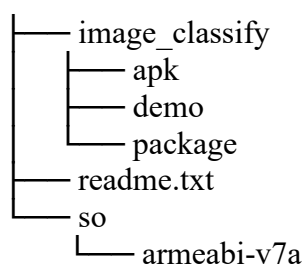
Android NN JNI 的完整框图如下：



如上图所见，用户开发 custom app，要首先使用 acuity tool 生成对应网络的 nbg 文件。Custom app 中需要集成预先使用 ndk 编译好的底层相关的动态库，并使用简单的 jni 函数接口来实现底层动态库的访问。Aml sdk 模块对应 apk 中 libs 目录中的 libnnsdk.so，是封装好的可以直接访问底层 API 的动态库。其他 libs 目录中的动态库，如 libGAL.so、libnnsdk.so、libOpenVX.so、libOpenVXU.so、libovxlib.so 以及 libVSC_Lite.so 是对接底层 NPU 设备的一系列依赖动态库，这些集成库用户不会涉及到更改。

2. Android NN JNI Release 包简介

android release 包基本目录结构为：



release 包可通过 GitHub 获取，link:

https://github.com/Amlogic-NN/AML_NN_SDK/tree/master/Android

如上，release 包中包含使用 jni 访问底层硬件所依赖的 armeabi-v7a so 包，以及简明的 android apk demo。image_classify 提供了 apk，demo 源码和运行所需的文件。该 demo 展示了如何使用 jni 方式开发 nn apk，整体实现非常简单，仅仅是使用分类网络实现输入图片的分类，并将类别在 apk 中显示出来。该版本 libs 仅包含 32bit 相关的库文件，针对 64bit 的完整包会在后续版本提供。

3. NN JNI 开发

3.1 开发环境配置

以下结合 release 包的 demo 说明基本的开发环境搭建。

首先要先安装好 android studio 环境，配置好 ndk。建立 android apk 的基本工程，假设 android apk 工程已建立。

将 release package 开发包中的 libs 文件拷贝到 app/src/main/libs 目录下。

release package 开发包中的 amlogic 文件拷贝到 app/src/main/java/com 目录下。
(该目录可根据实际工程修改，仅包含接口类就可以了)

将 release package 开发包中的 jni 文件拷贝到 app/src/main 目录下。

确认 app 目录下的 build.gradle 文件相关配置：

```
Android{
    ndk {
        abiFilters "armeabi-v7a" //指定 ANDROID_ABI
    }
    sourceSets {
        main {
            jniLibs.srcDirs = ['src/main/libs'] //指定 lib 路径
        }
    }
}
```

本 demo 以 32bit 进行说明，故 abi 仅指定了 armeabi-v7a。

修改 app 目录下的 cmakeLists.txt 文件，添加：

```
set(jni_srcs
    src/main/jni/com_amlogic_nnapi.cc
)
add_library(aml_nnapi SHARED ${jni_srcs})
```

#将 jni 文件编译到工程中，并指定链接：

```
target_link_libraries( # Specifies the target library.
    aml_nnapi
    log
    c
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libnnsdk.so
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libovxlib.so
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libOpenVX.so
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libOpenVXU.so
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libGAL.so
```



```
    ${CMAKE_SOURCE_DIR}/src/main/libs/${ANDROID_ABI}/libVSC_Lite.so  
)
```

aml_nnapi 就是 jni 的接口函数，是 java 与 c 链接的桥梁。

修改后，apk 会将 jni 编译生成的 aml_nnapi 动态库与 libs 中的动态库与 code 统一打包生成 apk。

3.2 NN JNI 接口说明

NN JNI 的主要接口实现在 jni 目录下的 com_amlogic_nnapi.cc 文件中，调用者为 amlogic/nnapi.java，以下结合两者做详细说明。

Java 层包含 amlogic/nnapi 类后，可以使用其中的主要接口方法：

1).nn_model_init

该方法为运行环境的初始化，调用到 jni 中的 model_init 方法，主要是指定 nbg 文件的路径与生成 nbg 文件的开源模型，开源模型值的定义参考附录 amlnn_model_type。该 jni 方法会返回 context 上下文，保存到该 class 的私有成员中。用户可自定义设置 pfile 读取 nbg 文件。

2).nn_model_input_set

该方法指定输入数据，其中 input_type 参考附录 amlnn_input_type 定义，一般视觉应用该值为 0。input_num 为多输入条件下对应的 input index 值，从 0 开始。对单 input 输入，input_num 为 0。该 api 调用到 jni 的 model_inputs_set 函数。

3).nn_model_outputs_get

获取网络的运行结果。调用到 jni 的 model_outputs_get。该方法需要重点说明下，在 jni 中，新开发网络底层 output 的返回类型为 nn_output，可参考头文件看下其详细定义。在 jni 层拿到该值后，可以选择在 jni 层做简单解析，将解析后的结果返回给 java 层（outbuf 数组），也可以将 output 的全部数据返回给 java，在 java 层做全部后处理。参考 demo 实现，demo 中 top5 的解析就是在 jni 层做的，可以选择将解析好的 top5 传给 java 显示。该部分选择较灵活，可根据场景做不同选择。

4).nn_model_destroy

退出时清理运行环境，调用到 jni 的 `model_destroy` 方法。

以上方法更详细说明可以参考《DDK_6.4.*.*_SDK_V1.*.* API 描述.docx》。

3.3 app 中 NN JNI 使用

在上文基础上，使用 `nn_jni` 做应用开发就水到渠成了。一般其基本的调用流程如下：

```
import com.amlogic.nnapi;    //导入 nnapi 类

float[] out = new float[1024]; //output buffer, 如分类 demo 输出为 1000 类

nnapi nndemo= new nnapi(); //实例化

nndemo.nn_model_init(model_path,network_type); //初始化

nndemo.nn_model_inputs_set(0,0,224*224*3,udata); //input 224*224*3

nndemo.nn_model_outputs_get(out,nnapi.CUSTOM_NETWORK,nnapi.AML_O
UTDATA_FLOAT32); //get output to out buffer

/////////some postprocess/////////

nndemo.nn_model_destroy(); //app over
```

该流程与 linux 环境下基本一致，细节可以参考《DDK_6.4.*.*_SDK_V1.*.* API 描述.docx》。

4.Android nn jni demo

为更好说明 nn jni 使用方法，提供了实际操作的简易 demo apk。

Jni_app 目录下的 app 即为 android studio 工程下 app 对应目录。该目录下的 build.gradle 与 CMakeList.txt 的修改说明见上文，并 app/src/main/libs 目录已按上文拷贝到 android 工程中。本部分仅就调用流程做简要说明。

在 src/main/java/com/jnittest/MainActivity.java 中定义了 NN 的运行流程，当点击”START”按钮时，触发 public void onClick(View view)函数。在该函数中调用了完整的 NN-JNI 流程，也可直接下载安装 jnidemo.apk。

安装 APK 后，直接点击 start，demo 运行并输出最后的分类结果。

主要的注意点如下：

1).nndemo.nn_model_init((mpath.getAbsolutePath()+File.separator+"inceptionv1.nb").getBytes(),TENSORFLOW);

该部分加载的为 inceptionv1.nb 模型，该模型基于 tensorflow 架构，故 type 为 1。

依据上面 path，需要预先将该 nb push 到对应的路径下，如

/sdcard/Android/data/com.jnittest/files/路径，com.jnittest 即为实际 apk 的 name。

对应 nnapi.java 中的定义为：

```
public int nn_model_init(byte[] nbg_model_file, char type)
```

2) nn_model_inputs_set

```
nndemo.nn_model_inputs_set(0,0,224*224*3,udata);
```

本 demo 的数据输入为 2242243 的图片做分类，故需要将要分类的图片放到相应的 path 中，如/sdcard/Android/data/com.jnittest/files/images/，上面所提到的路径为 apk 安装后默认放置数据文件的地方，是有访问权限的，也可以放在其他地方，只要在 app 中可以正常访问就行。

对应 nnapi.java 中的定义为：

```
public int nn_model_inputs_set(int input_type,int input_num,int input_size,byte[] data)
```

input_type 表示输入数据类型，0 表示 RGB24_RAW_DATA，请见 nn_api.h；input_num 为多输入条件下对应的 input index 值，从 0 开始。input_size 表示输入数据总大小。data 表示输入数据 buffer。

3) nn_model_outputs_get

```
nndemo.nn_model_outputs_get(out,nnapi.CUSTOM_NETWORK,nnapi.AML_OUTDATA_FLOAT32);
```

对应 nnapi.java 中的定义为:

```
public int nn_model_outputs_get(float[] outbuf,int netid,int format)
```

outbuf 表示输出 buffer。netid 表示模型序号，用户自定义网络序号为 **CUSTOM_NETWORK**。format 表示输出数据类型，**AML_OUTDATA_FLOAT32** 表示输出 float 数据，netid 和 format 详细定义请参考 nn_api.h。

Confidential!
nick@khadas.com
2021-09-07 10:49:41