

Amlogic

Android NN JNI Development Guide


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

Revision History

Issue 0.1 (2021-08-10)

This is the Initial release.

Confidential!
nick@khadas.com
2021-09-07 10:49:40

Contents

Revision History	3
Contents	4
1. Android NN JNI Architecture	5
2. Android NN JNI Release Package introduction	7
3. NN JNI Development	8
3.1 development environment configuration	8
3.2 NN JNI Interface introduction	9
3.3 How to use NN JNI in App	9
4. Android nn jni demo	11

Confidential!
nick@khadas.com
2021-09-07 10:49:40

1. Android NN JNI Architecture

Android versions after 8.1 include the neural networks API (nnapi), which can provide low-level support for Google's network deep learning developed and trained based on tensorflow lite architecture. For developments using NNAPI, please refer to *Android NN APK development process guidance* and *Android & Linux compilation integration guidance (0.2)*.

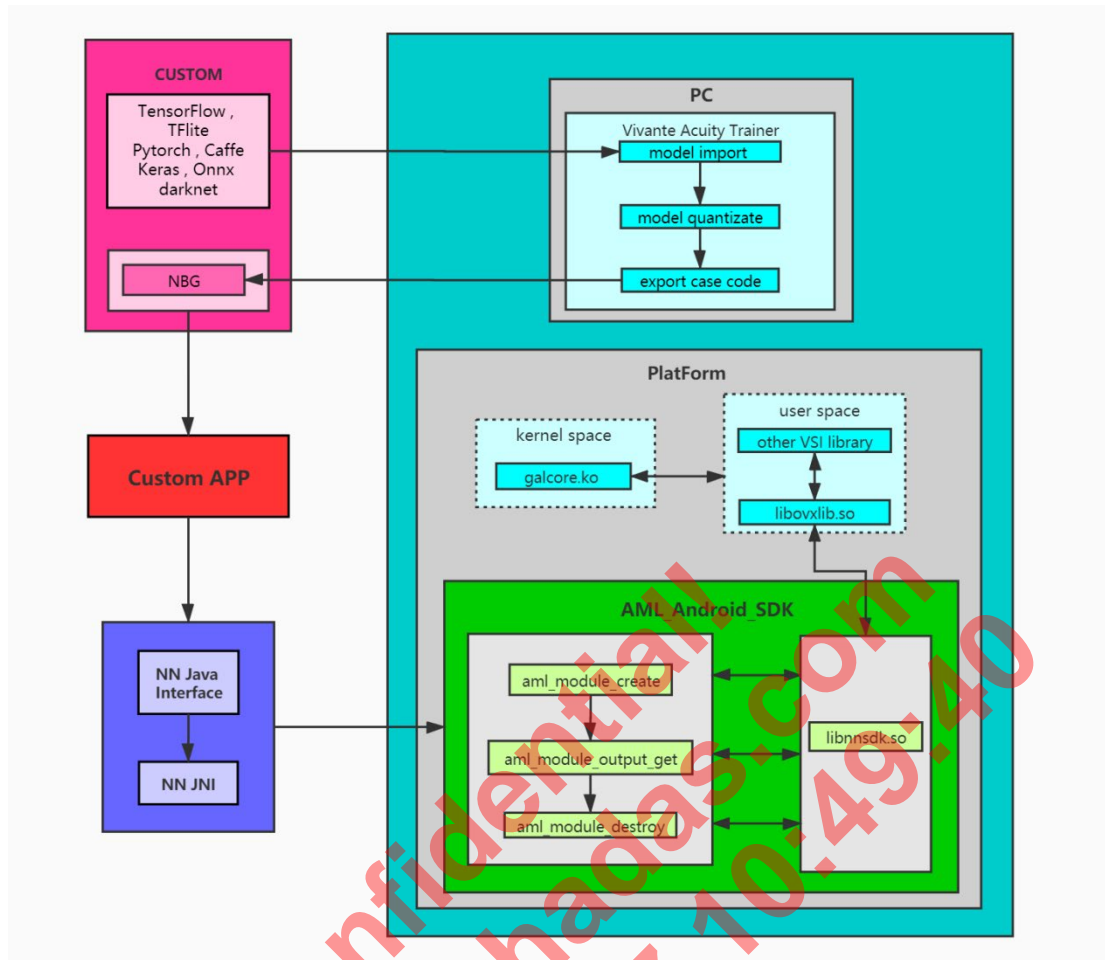
The solution based on android standard NN architecture has the following disadvantages:

- Less supports open source architecture (only supports tflite)
- Less supports operators,
- Could not take all advantages of the underlying NPU hardware (limited by the maturity of the mapping between NN and low-level API)

This solution may not be perfect in case high performance is required. Access to the low-level hardware directly based on android Java Native Interface (JNI) architecture can be a necessary supplement to this solution.

Access to underlying dynamic library through JNI interfaces from Android App is a commonly used method, which is highly flexible, more customization, ease to implement/migrate. This enables Android NN to be able to better adapted to specific operators, running platform and low-level hardware more better though NN JNI(And this is the original motivation of NN JNI.)

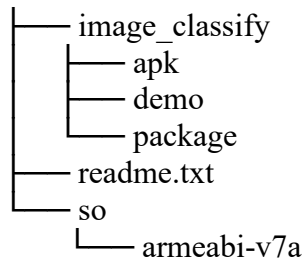
The following picture describes the Android NN JNI architecture:



As Shown in the figure above, the first step to develop a custom app, is to use Acuity tool to generate the NGB file. The custom app must integrate with the related low-level dynamic library pre-compiled using ndk ahead of time. and accesses the low-level dynamic libraries using JNI. The libnnsdk.so under libs directory in apk is the compiled dynamic library for Aml SDK. It provides a set of interfaces to access low-level APIs directly. Other dynamic libraries in libs directory, such as libGAL.so、libnnsdk.so、libOpenVX.so、libOpenVXU.so、libovxlib.so and libVSC_Lite.so, is a set of dependent dynamic library for accessing low-level NPU hardware which are not likely be used by users.

2. Android NN JNI Release Package introduction

The basic directory structure of android release package is as follows:



the release package can be got though GitHub, the website address is:

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

As shown in the figure above, the release package includes armeabi-v7a so package that JNI relies on to access low-level hardware, and simple android apk demo. Image_classify includes APK, demo source code and required packages. The demo shows how to use JNI to develop nn apk, which is a simple implementation using classification network to classify the input figures and display the classification result in APK. The versions of libs are for 32bit related library files. Complete package for 64bit will be provided in the subsequent releases.

3. NN JNI Development

3.1 development environment configuration

Here is a description of how to build a development environment, using a demo in release package as an example.

First install the android studio and configure ndk. Create the basic project of Android APK,(assuming that the Android APK project has already been created).

Copy the libs file in release package to app/src/main/libs directory.

Copy the Amlogic file in release package to app/src/main/java/com (the directory can be modified according to the actual project, only contains interface classes).

Copy the jni file in release package to app/src/main directory.

Check the build.gradle configuration under the app Directory:

```

Android{
    ndk {
        abiFilters "armeabi-v7a" // Specify ANDROID_ABI
    }
    sourceSets {
        main {
            jniLibs.srcDirs = ["src/main/libs"] //Specify lib path
        }
    }
}

```

This demo is described in 32bit, so abi only specifies armeabi-v7a.

Modify cmakeLists.txt under app directory and add:

```

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

#compile the JNI file into project and specify the link:
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 is the interface provided by JNI, it is the bridge between Java and C.

After modification, APK will pack the aml_nnapi dynamic library generated by JNI, the dynamic libraries in libs, and code together, to generate APK.

3.2 NN JNI Interface introduction

The main interfaces of NN JNI are implemented in com_amlogic_nnapi.cc under the JNI directory, they are called by amlogic/nnapi.java (explained further below). Once amlogic/nnapi class is included at the Java layer, its main class methods can be used:

1. n_model_init

This method is used to initialize the running environment, it calls the model_init method in JNI. It is used to configure the path of nbg file and open-source model used to generate the nbg file. (Refer to amlnn_model_type in appendix for the definition of open-source models.). The jni method will return a context, which will be saved in a private member of the class. Customer can customize pfile to read nbg file.

2. nn_model_input_set

This method is used to configure the input data. The definition of input_type in this method can be found in appendix: amlnn_input_type. For general visual applications, it should be set to 0. Input_num is the input index value (start from 0) in case there are multiple inputs. For single input, The input_num should be 0. This API calls model_inpput_set in JNI.

3. nn_model_outputs_get

This method is used to get the running result of the network. This method calls model_outputs_get in jni. The returned type from low-level API to JNI is nn_output (refer to the header file for a detailed definition). Once JNI get the return value, it can choose to do simple interpretation of the result, then return the result after interpretation (in outbuf array) to java layer, or pass the return value to java layer directly, and perform post-processing at the java layer. Take the demo implementation as an example, in the demo, top5 interpretation is done in JNA layer then passed to java layer for display. The two strategies can be chosen flexibly according to application needs.

4. nn_model_destroy

This method calls model_destroy in JNI to clear up the running environment while exiting.

For more detailed information about above methods, please refer to *DDK_6.4.x.x_SDK_V1.x.x API*

3.3 How to use NN JNI in App

The following code demonstrates how to develop APP using nn jni:

```
import com.amlogic.nnapi; //Import nnapi class

float[] out = new float[1024]; //Output buffer, for example, the output buffer for
classification demo is 1000

nnapi nndemo= new nnapi(); //Instantiation

nndemo.nn_model_init(model_path,network_type); //Initialization
```

```
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_OUTDAT  
A_FLOAT32); //get output to out buffer  
////////some postprocess/////////  
nndemo.nn_model_destroy(); //app over
```

The process is similar to that in Linux. For more details, please refer to DDK_6.4.x.x_SDK_V1.x.x API. Docx.

Confidential!
nick@khadas.com
2021-09-07 10:49:40

4. Android nn_jni demo

To better explain the use of NN JNI, a simple demo APK is provided for practice. The app directory under android studio project is actually the app under Jni_app.

The modification to build.gradle and CMakeList.txt under this directory has been described above. Also, app/src/main/libs directory has been copied to the android project as previously described. This chapter only describes the calling sequence in brief. The running sequence of NN is defined in src/main/java/com/jnittest/MainActivity.java. When "START" button is clicked, the public void onClick(View view) function will be triggered. In this function, the whole NN-JNI sequence is called. You can download and install inidemo.apk directly instead.

After APK is installed, click "START" button, the demo will run and output the classified result.

The key points are listed below:

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

This loads the inceptionv1.nb model, which is based on the tensorflow architecture, so the type is 1. inceptionv1.nb model needs to be pushed to the path given above in advance. For example, /sdcard/Android/data/com.jnittest/files/, and com.jnittest is the name of the actual APK.

The prototype of `nn_model_init` defined in `nnapi.java` is:

```
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);
```

The input data of this demo is pictures with 224*224*3, so it needs to put the pictures to be classified in the corresponding path, such as /sdcard/Android/data/com.jnittest/files/images/ (This is the default place to save data file after apk is installed). The path must be accessible. The pictures can be put in any other place as long as it can be accessed by the app. The prototype of `nn_model_inputs_set` defined in `nnapi.java` is:

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

`input_type`: the type of input data, 0 - RGB24_RAW_DATA. For more available values, please see `nn_api.h`;

`input_num`: input index in case of multiple inputs, counted from 0.

`input_size`: the total size of input data

3. `data`: the buffer of input data.`nn_model_outputs_get`

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

The prototype of `nn_model_outputs_get` defined in `nnapi.java` is:

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

outbuf: output buffer.

Netid: model ID, For user-defined network ID use CUSTOM_NETWORK.

Format: the type of the output data, For float type use AML_OUTDATA_FLOAT32. For detailed definition of netid and format, please refer to nn_api.h.

Confidential!
nick@khadas.com
2021-09-07 10:49:40