

Android&Linux

编译集成指导


Revision: 0.2

Release Date: 2019-12-18

Copyright

© 2019 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

变更说明

版本 0.2 (2019-12-18)

第二版。

1. buildroot_sdk 中增加 linux 环境下 32bit 的编译环境，通过参数控制编译 32bit/64bit 可执行文件。
2. 添加对转出 case 代码中 openvx_case 目录的说明

版本 0.1 (2019-07-22)

第一版。

Confidential!
nick@khadas.com
2021-04-12 20:25:40

目录

变更说明.....	ii
1. 简介.....	4
2. Case 代码介绍.....	5
2.1 简介.....	5
2.2 流程介绍.....	5
2.2.1 Case 代码流程.....	5
2.2.2 主要接口以及参数获取.....	5
3. Android 开发指导.....	8
3.1 简介.....	8
3.2 基于 TFLite 开发.....	8
3.2.1 开发指导.....	8
3.2.2 运行环境确认.....	9
3.2.3 基于 TFLite 开发的 FAQ.....	9
3.3 基于 JNI 方式开发.....	9
3.3.1 开发简介.....	10
3.3.2 编译环境.....	10
3.3.3 Ndk 编译.....	10
4. Linux 开发指导.....	12
4.1 简介.....	12
4.2 buildroot 可执行文件编译.....	12
4.2.1 Sdk 包介绍.....	12
4.2.2 可执行文件编译.....	12
4.2.3 可执行文件执行.....	12
5. FAQ.....	13

1. 简介

本文介绍基于模型转换流程文档转出 case 代码后，在 Linux&Android 下进行编译集成的流程，用以指导 amlogic 相关开发人员以及 amlogic 客户。

Confidential!
nick@khadas.com
2021-04-12 20:25:40

2. Case 代码介绍

2.1 简介

当前模型转换出 case 代码分析如下:

```
577 12月 18 16:55 BUILD
5835 12月 18 16:55 main.c
2000 12月 18 16:55 makefile.linux
12691 12月 18 16:55 mobilenettf.vcxproj
1063 12月 18 16:55 nbg_meta.json
3447390 12月 18 16:55 network_binary.nb
4096 12月 18 16:55 openvx_case
358 12月 18 16:55 vnn_global.h
7505 12月 18 16:55 vnn_mobilenettf.c
977 12月 18 16:55 vnn_mobilenettf.h
3629 12月 18 16:55 vnn_post_process.c
462 12月 18 16:55 vnn_post_process.h
24058 12月 18 16:55 vnn_pre_process.c
1545 12月 18 16:55 vnn_pre_process.h
```

1. mobilenet_tf.nb 生成的 nbg 文件，默认生成 network_binary.nb。可自行改名。
2. vnn_mobilenettf.c/vnn_mobilenettf.h 模型创建和释放对应的头文件和实现。（转 case 代码时候，会生成对应名称的 vnn_modelname.c）
3. vnn_pre_process.c/vnn_pre_process.h 模型前处理文件，case 代码中是在此文件中对读入的图片进行的减均值除以 scale 并进行量化操作。
4. vnn_post_process.c/vnn_post_process.h 模型后处理文件，case 代码是在此文件中获取输出结果并将结果从量化数据转成 float32 数据，并对输出结果做了 top 的处理（由于是单一模板，所有 case 代码均是做的 top5 后处理）。
5. openvx_case: 直接调用 openvx 接口运行 nbg 模型的 case 代码，当前暂时不用。

2.2 流程介绍

2.2.1 Case 代码流程

TODO

2.2.2 主要接口以及参数获取

全局唯一句柄: vsi_nn_graph_t * graph ;

1. 创建句柄: graph = vnn_CreateNeuralNetwork(path_to_xxx.nb);

或直接使用 vnn_xxxmodel.c 里面的 graph = vnn_CreateXXXX(path_to_xxx.nb, NULL);

2. vnn_VerifyGraph(vsi_nn_graph_t *graph);

3. Prepare 过程中用到的参数:

输入个数: graph->input.num

获取对应的 vsi_nn_tensor_t * tensor:

```
tensor = vsi_nn_GetTensor( graph, graph->input.tensors[i]);
```

获取 tensor 对应的 size 长度:

```
sz = vsi_nn_GetElementNum(tensor);
```

获取维度信息:

```
dim_num = tensor->attr.dim_num
```

```
width = tensor->attr.size[0];
```

```
height = tensor->attr.size[1];
```

```
channel = tensor->attr.size[2];
```

```
batch = tensor->attr.size[3];
```

量化参数获取:

Int8/int16: 表示小数位的位数 fl:

```
fl = tensor->attr.dtype.fl
```

U8: scale 和 zero_point:

```
scale = tensor->attr.dtype.scale
```

```
zero_point= tensor->attr.dtype.zero_point
```

拷贝数据到对应的 tensor:

```
vsi_nn_CopyDataToTensor( graph, tensor, data);
```

自己做量化:

做完减均值+除以 scale 之后, 将输入的 input 值进行量化:

Int8/16:

```
float fl = pow(2.,tensor->attr.dtype.fl);
```

```
value_int8 = (int)(value_float * fl);
```

U8:

```
value_uint8 = value_float/scale + zero_point;
```

4.推理计算:

```
vnn_ProcessGraph( graph );
```

5、获取输出结果:

输出个数: graph->output.num

获取对应 tensor:

```
tensor = vsi_nn_GetTensor(graph, graph->output.tensors[i])
```

获取结果并将结果转成 float32:

```
stride = vsi_nn_TypeGetBytes(tensor->attr.dtype.vx_type); (int16 为 2,
int8/u8 为 1)
```

```
uint8_t * tensor_data = (uint8_t *)vsi_nn_ConvertTensorToData(graph,
tensor);
```

```
sz = vsi_nn_GetElementNum(tensor);
```

```
buffer = (float *)malloc(sizeof(float) * sz);
```

量化参数获取:

Int8/int16: 表示小数位的位数 fl:

```
fl = tensor->attr.dtype.fl
```

U8: scale 和 zero_point:

```
float scale = tensor->attr.dtype.scale
```

```
int zero_point= tensor->attr.dtype.zero_point
```

将量化数据结果转成 float32 结果:

```
float fl_value = pow(2., -tensor->attr.dtype.fl);
```

```
for (i = 0; i < sz; i++) {
```

```
    buffer[i] = tensor_data[stride * i] * fl_value;
```

```
    或者 buffer[i] = (tensor_data[stride * i] - zero_point) * scale
```

```
}
```

6. debug 手段:

将 tensor 保存到文件中

```
vsi_nn_SaveTensorToBinary(graph, tensor, "data.txt");
```

```
vsi_nn_SaveTensorToText( graph, tensor, "input.txt", NULL );
```

```
vsi_nn_SaveTensorToTextByFp32( graph, tensor, filename, NULL );
```


3.Android 开发指导

3.1 简介

当前我们 Android 平台上也集成了 nn 功能。基于 Android 平台的开发有两种方式：

1、基于 TFlite 开发

优点：如果当前已经基于 tflite 开发了 apk，只需要添加一行代码并将模型量化成 int8 即可。

缺点：1、必须是量化后的 tflite 模型。

2、支持的算子有限，碰到 TFlite 框架不支持算子，适配麻烦

2、基于 JNI 方式开发

优点：支持 caffe、tensorflow、onnx、darknet、tflite 模型。支持灵活，支持的算较多。

缺点：当前 amlogic 暂时没有对接口进行封装。Jni 的开发是基于转出的 case 代码，得自己从 case 代码中提取对应功能的接口并进行封装，编译成 so。

3.2 基于 TFlite 开发

3.2.1 开发指导

Amlogic 对 Android 的 TFlite 架构，底层进行了适配。支持的 layer 在枚举变量 OperationCode 中列出。（android 代码 frameworks/ml/nn/runtime/include 目录下的 NeuralNetworks.h 文件中定义）。实际支持的算子只有 nnapi_delegate.cc 文件中 AddOpsAndParams 接口里面具体使用的一些，可以实际参考下。

APK 运行在 NN 上的适配：

1、确保 tflite 模型是 8bit 量化模型，如果不是，请使用 Tensorflow 官方提供的 TFLiteConverter 工具，对模型进行量化。

(1) 编译 tf 可执行文件：bazel build tensorflow/contrib/lite/toco:toco

(2) 量化：

```
bazel-bin/tensorflow/contrib/lite/toco/toco \
--input_file=$(pwd)/mobilenet_v1_1.0_224/frozen_graph.pb \
--input_format=TENSORFLOW_GRAPHDEF \
--output_format=TFLITE \
--output_file=/tmp/mobilenet_v1_1.0_224.tflite \
--inference_type=FLOAT \
--input_type=FLOAT \
--input_arrays=input \
--output_arrays=MobilenetV1/Predictions/Reshape_1 \
```

```
--input_shapes=1,224,224,3
```

2、使能 NN，使底层跑在 NN 上，代码中添加：
tfliteOptions.setUseNNAPI(true);

3.2.2 运行环境确认

3.2.1 章节完成后，就完成了基于 TFLite 开发的 apk 对 NN 的适配。在 apk 运行之前，需确认上板子上的 android 版本是否有 NN 的环境。Adb shell 链接之后执行下面命令：

1. 确认是否有 nn server 启动
: ps -A |grep neuralnetwork
2. 确认 npu 模块是否加载
: lsmod |grep galcore
3. 确认 npu 节点是否正常（权限为 664）
: ls /dev/galcore -l

上面几步确认 OK，说明当前 android 版本有 NN 的环境，当前运行 apk 可以通过 nn server 调用到 NPU 中。

3.2.3 基于 TFLite 开发的 FAQ

1、TFLite 模型加载后，启用 NNAPI 报如下错误：

```
E/tflite: Op code 98 is currently not delegated to NNAPI
E/tflite: Returning error since TFLite returned failure nnapi_delegate.cc:738.
E/tflite: Failed to build graph for NNAPI
```

根因：上文介绍了，TFLite 架构走 NN 是底层对 TFLite 定义的一些 layer 的适配，即 NeuralNetworks.h 中定义的 layer。如果 TFLite 模型中存在该头文件中未定义的 custom layer，会出现上面的这种情况。因为设置 NNAPI 为 true 之后，TFLite framework 会对 TFLite 的每一层 layer 进行 check，如果存在 NeuralNetworks.h 中未定义到的层，会报错返回。

解决：碰到这种情况的解决方法有，将 TFLite 模型进行拆分，把 NeuralNetworks.h 中未定义到的层移除之后生成新的 TFLite 模型，移除的层可以用 CPU 进行实现。或者是使用 JNI 的方式进行开发。

3.3 基于 JNI 方式开发

Android 下的另外一种开发方式就是基于 JNI 方式进行开发。这种方式较 TFLite 开发，支撑的 layer 更多，更加灵活。当前我们 Buildroot 平台上能跑的模型，都能使用这种方式进行 Android apk 的开发。

3.3.1 开发简介

当前 amlogic 提供一个基于 ndk 编译的 sdk 包，包含了编译 case 代码所需要的 so 以及头文件。开发者可以用该 sdk 包，编译出 case 代码对应的可执行文件，并且也可以编译基于 case 代码提取的 apk 开发所需接口对应的 so 文件。

3.3.2 编译环境

1. 安装 ndk 环境

a. 下载 ndk 包

https://dl.google.com/android/repository/android-ndk-r17-linux-x86_64.zip

b. 解压后配置环境变量

打开 ~/.bashrc 并将下面两行添加到文件末尾（或者添加到/etc/profile 中）

```
export NDKROOT=/usr/ndk/android-ndk-r17b
```

```
export PATH=$NDKROOT:$PATH
```

保存退出。更新下环境变量：source ~/.bashrc

c. 确认 ndk 是否安装 ok

在 shell 中输入“ndk-build”命令来检查你的安装是否成功，如果不是显示“ndk-build not found”，表明 ndk 环境配置 OK。

2. sdk 包介绍

android_sdk	2019/12/18 16:53	文件夹
conversion_scripts_nbg_unify	2019/12/18 17:03	文件夹
ReadMe.txt	2019/11/2 15:54	文本文档

ReadMe.txt: 介绍 ndk 环境的安装以及如何编译 so 和可执行文件。

android_sdk: 编译 case 代码所需的头文件以及 so

conversion_scripts_nbg_unify: acuity tool 里带的 demo 转出的 case 代码程序，在该目录下 ndk-build 即可编译。

3.3.3 Ndk 编译

1、编译 Android 可执行文件

参照当前 demo: conversion_scripts_nbg_unify，在该目录下 ndk-build 即可编译。

编译自己的 case 代码：

- 1、创建与 conversion_scripts_nbg_unify 的同级目录 code_dir
- 2、在 code_dir 目录下创建 jni 目录，将 case 代码到 jni 目录下
- 3、将 conversion_scripts_nbg_unify\jni 下的 Android.mk、Applicatoin.mk 拷贝到 jni 目录下
- 4、修改 Android.mk 文件里的 common_src_files（列出所有.c 文件）
- 5、在 code_dir 目录下执行：ndk-build 即可

2、执行可执行文件

- 1、将 android_sdk\lib 目录下的 libjpeg.so.push 到/vendor/lib 下，并改名成 libjpeg-t.so
- 2、可执行文件在 libs/armeabi-v7a,添加执行权限，chmod +x libs/armeabi-v7a/xxxx
- 3、执行命令：libs/armeabi-v7a/xxxx jni/network_binary.nb xxxx.jpg

3、编译 Android so 文件

- 1、基于上面编译可执行文件的步骤
- 2、修改 Android.mk 最后几行，修改如下：

```
44  
45 #LOCAL_CFLAGS += -pie -fPIE  
46 #LOCAL_LDFLAGS += -pie -fPIE  
47 #include $(BUILD_EXECUTABLE)  
48  
49 include $(BUILD_SHARED_LIBRARY)  
50
```

- 3、执行编译命令：ndk-build

注：当前只编译 32 位 so。


4. Linux 开发指导

4.1 简介

基于 buildroot 的开发，当前提供一个独立于 buildroot 大版本，能编译 case 代码的 sdk 包。当前指导奖转出来的 case 代码编译成 buildroot 下的可执行文件。（buildroot 其他开发环境当前不确认，所以暂时只指导 case 代码编译成可执行文件）

4.2 buildroot 可执行文件编译

4.2.1 Sdk 包介绍

 buildroot_sdk	2019/12/13 10:08	文件夹
 nbg_unify_mobilenet_tf	2019/12/13 10:22	文件夹

buildroot_sdk: buildroot 编译的 sdk 包，里面包含编译所需的头文件/so 以及其他资源

nbg_unify_mobilenet_tf: demo 程序

4.2.2 可执行文件编译

- 1、修改 demo 程序中 build_vx.sh 文件的 line18: export AQROOT=/path_to/buildroot_sdk (需设置绝对路径)
- 2、然后在该目录下执行: ./build_vx.sh arm/arm64 分别编译 32bit/64bit 可执行文件。
- 3、编译自己的 cae 代码，将 case 代码拷贝到 nbg_unify_mobilenet_tf 同级目录
- 4、将 nbg_unify_mobilenet_tf 中的 build_vx.sh 拷贝一份到 case 代码目录中。
- 5、编译执行: ./build_vx.sh arm/arm64。可执行文件生成在 bin_r 目录中。

4.2.3 可执行文件执行

- 1、将编译的可执行文件以及 xxx.nb 和图片拷贝到板子上
- 2、执行命令: ./xxxx xxxx.nb xxxx.jpg 即可

注: 1、case 代码只支持 jpeg 格式图片。
2、默认只能处理与模型大小相同尺寸的图片
3、可以先执行: export VSI_USE_IMAGE_PROCESS=1, 之后就能传入不同尺寸的图片。

5. FAQ

Confidential!
nick@khadas.com
2021-04-12 20:25:40