

# 基于 RV1126 的车牌检测识别系统设计 与实现的算法模型部署流程

湖南典阅教育科技有限公司

2023 年 9 月 25 日制

## 目录

一、 开发环境准备 .....	2
(一) 、Linux 虚拟机准备 .....	2
(二) 、USB 驱动安装 .....	4
(三) 、编译环境准备与更新 .....	6
二、 开发板调试方式 .....	11
硬件链接调试方式: .....	11
三、 开发编译方式说明 .....	13
(一) 、交叉编译 .....	13
(二) 、本地编译 .....	17
四、 模型转换与转换环境搭建 .....	18
(一) 、转换为 rknn 模型环境搭建 .....	18
(二) 、运行模型转换工具环境 .....	19
五、 rknn 模型转换及预编译 .....	21
(一) 、模型转换 Demo 下载 .....	23
(二) 、进入模型转换工具 docker 环境 .....	23
(三) 、模型转换操作说明 .....	24
(四) 、onnx 模型转换为 rknn 模型 .....	26
六、 算法模型部署终端 .....	29
(一) 、开发环境准备 .....	29
(二) 、源码下载以及例程编译 .....	29
(三) 、方案部署 .....	30

## 注意事项

**\*注：针对赛用开发板，请使用赛事方提供的专用电源适配器、ADB 连接线等其他相关配件进行相应的使用操作，按照电子元件的使用规则妥善使用；**

**\*注：该手册提供 RV1126 开发板的开发环境、交叉编译环境、ADB 连接驱动安装所需安装包，以及对算法模型的转换、部署、预编译等内容介绍。**

**\*注：如出现 ADB 连接图标正常，但终端连接 ADB 一直出错，与相关界面显示有异，请重启虚拟机系统**



## 一、开发环境准备

### (一)、Linux 虚拟机准备

#### 1、VMware 环境搭载

##### 1.1、VMware 虚拟机下载

可在官网下载 vmware-pro15 虚拟机版本，官网下载地址如下：

<https://www.vmware.com/cn.html>

也可在百度网盘下载，

链接：<https://pan.baidu.com/s/1DElQ0KyjCJxTQDIOSMjy4w>

提取码：0825

\* 推荐下载方式为网盘下载，防止出现版本不匹配现象；安装教程请参考百度：<https://www.xjx100.cn/news/674329.html?action=onClick>

#### 2、使用 Ubuntu 镜像

Ubuntu 镜像文件百度网盘获取：

链接：<https://pan.baidu.com/s/1fb5YOq4HHiBTB8bIrLhMyw>

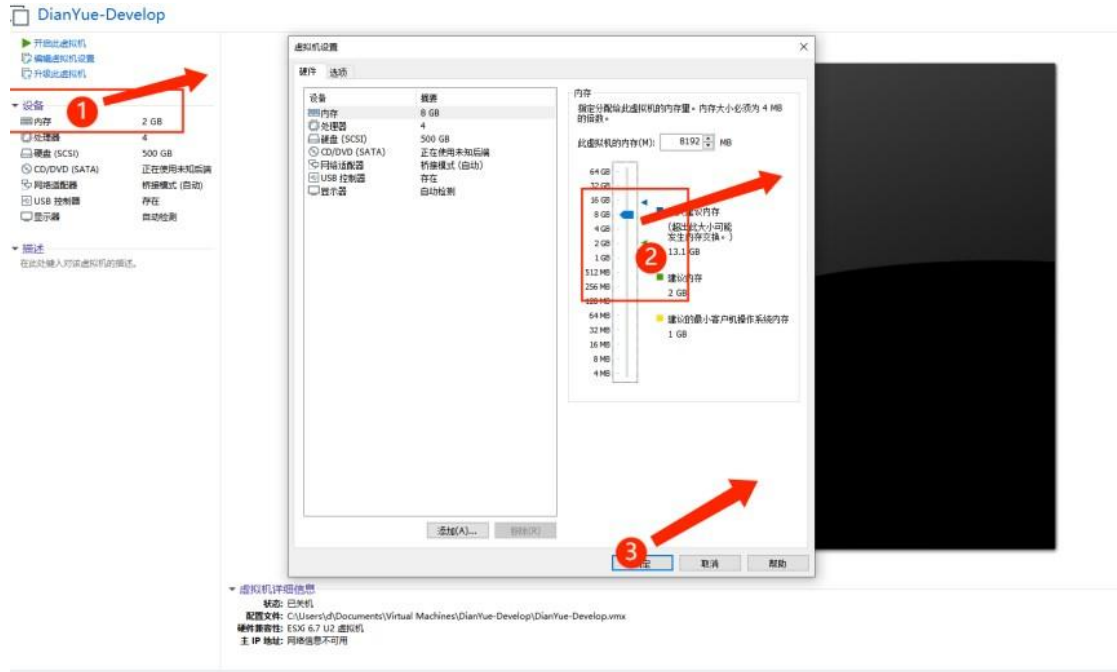
提取码：0825

\* 该镜像文件我可以搭建完成

##### 2.1、对虚拟机进行运行内存扩容操作

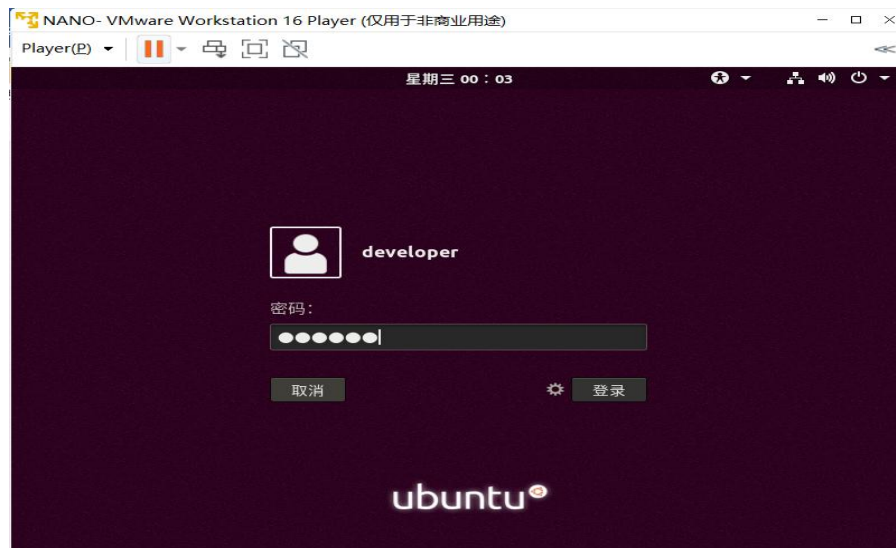
在安装完成虚拟机 Ubuntu 之后，根据需要，我们需对虚拟机的运行内存进行扩容操作，具体操作如下图所示。

在虚拟机界面双击设备栏下的内存一项，如标识 1 所在位置所示，而后在弹出的虚拟机设置界面右侧将标识 2 所在位置的滑标调整为我们所需的内存大小，根据本次比赛需求，请将内存大小设置为 8GB 随后点击标识 3 所在位置确定，内存扩容设置完成。



## 2.2、打开 Ubuntu 虚拟机

点击“播放虚拟机”后，即可进入 Ubuntu 虚拟机。登录密码为“123456”。



## 2.2、Ubuntu 镜像资源

Ubuntu 镜像默认搭建好初级的开发环境，文件如下所示。

工具	描述
----	----

/opt/rv1126_rv1109_sdk	交叉编译工具
qtcreator	qt 编译环境
.bashrc	编译环境配置文件

## （二）、USB 驱动安装

### 1、下载驱动包

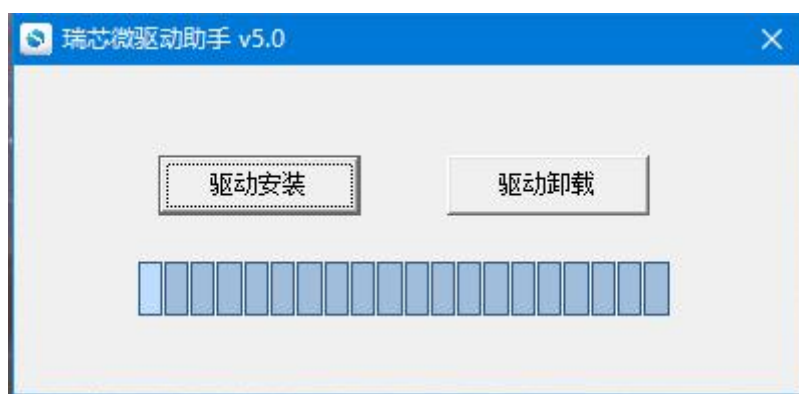
百度网盘下载：

链接：[https://pan.baidu.com/s/1eUmVC5Tn\\_QI2WRCxyOcYPA](https://pan.baidu.com/s/1eUmVC5Tn_QI2WRCxyOcYPA)

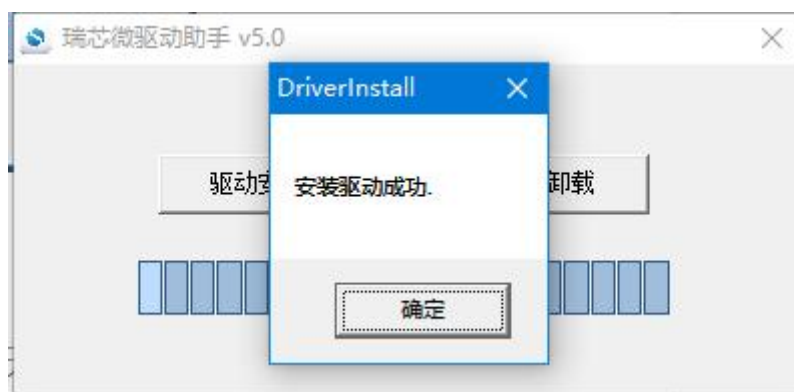
提取码：0825

### 2、安装 USB 驱动

安装驱动前，最好先断开 USB 的物理连接。解压缩 USB 驱动包 DriverAssitant\_v5.0.zip，打开 “DriverInstall.exe” 可看见以下窗口。



点击 “驱动安装”，等待片刻，程序会提示 “安装驱动成功”，表示 USB 驱动已安装成功，部分系统需要重启电脑才能生效。

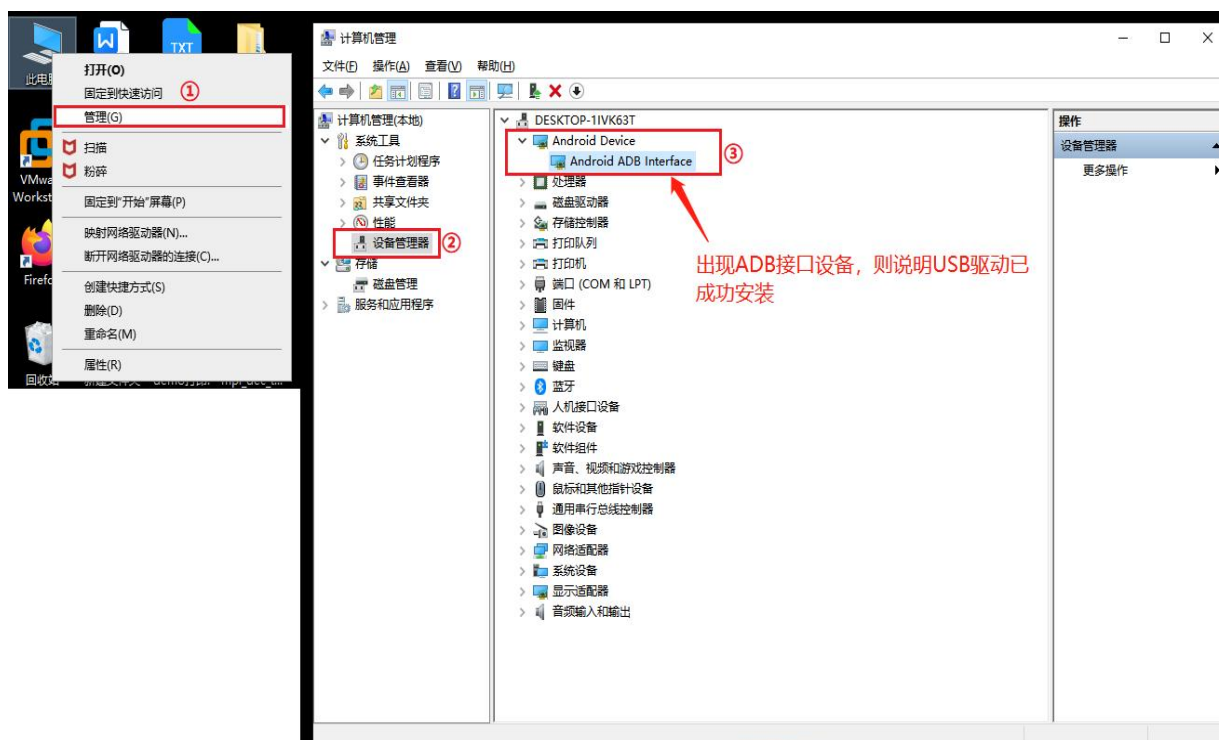


安装驱动成功后，即可开始对板卡执行固件烧录、ADB 调试等操作。

### 3、确认结果

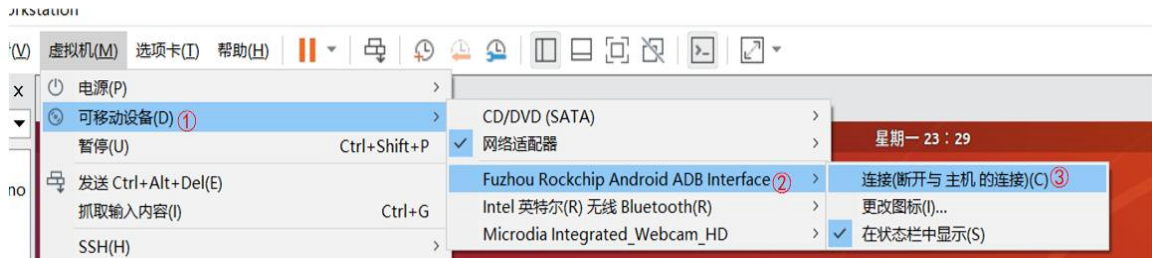
开发板连接电脑后，确认驱动是否已经被正确安装。开发板连接电脑后，硬件连接请查看第二章[《开发板调试方式》](#)。

#### Window 平台：



如果 Window 平台上没有出现 ADB 接口设备, 则 USB 有可能被虚拟机接管。

## 虚拟机:



链接成功会在虚拟机的右下角图标内会显色标注一个 USB 接口设备, 如下所示。



## (三)、编译环境准备与更新

### 1、部署编译环境

#### 1.1、编译环境简介

RV1126 编译环境是一款在线的编译环境, 建立了 PC 端与板卡端实时同步的映射关系, 保障 PC 端进行交叉编译时能实时链接开发板的依赖库。具有以下优点:

- (1) 解决传统交叉编译器因 PC 端和板卡端依赖库不一致, 导致的各种开发问题;
- (2) 免去用户配置传统交叉编译器环境变量更改的工作;
- (3) PC 端与板卡端实时映射, 文件传输速度大为提升, 满足大文件实时传输需求;



(4) 解决 PC 端环境对编译器兼容性的问题，比如 ubuntu20.04 环境可以，ubuntu18.04 环境却无法兼容；

(5) 同时支持开发板上进行本地开发，增加开发方式的灵活性。

**\*注意：在 RV1126 编译环境下进行程序编译必须满足 PC 端与开发板保持 ADB 连接的前提。**

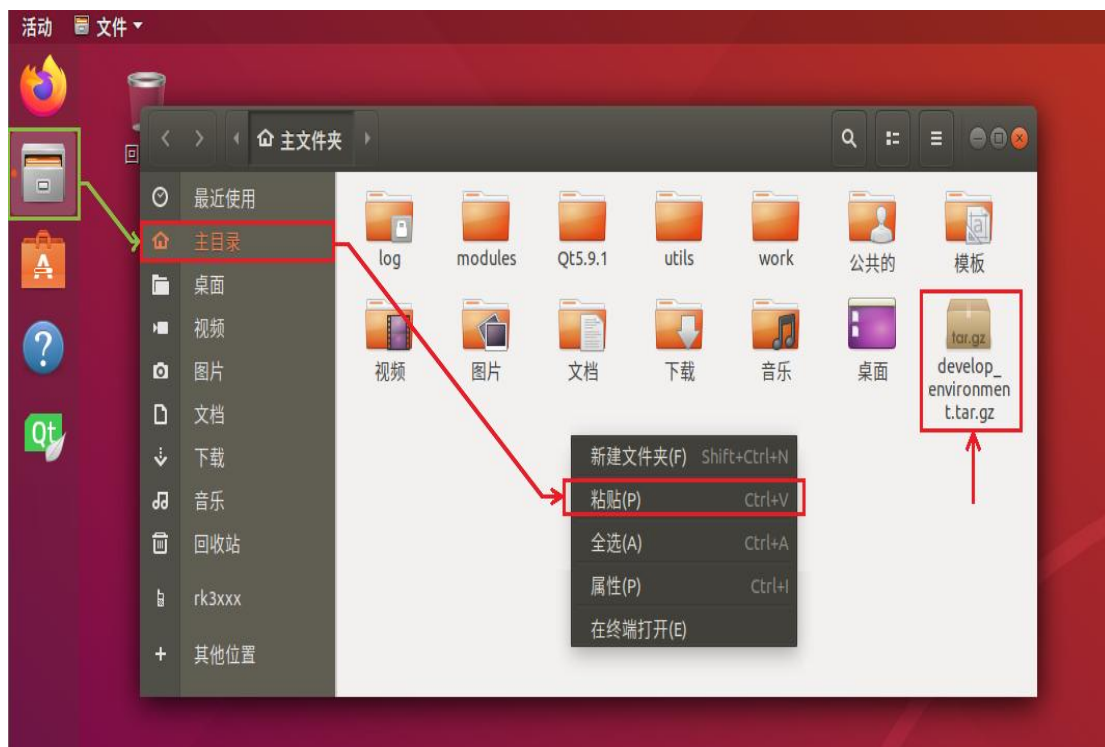
## 1.2、编译环境获取

百度网盘获取

链接：<https://pan.baidu.com/s/1GRig3Er6VjezvmhDYJhwTQ>

提取码：0825

下载 RV1126 编译环境工具包：develop\_environment.tar.gz。然后拷贝放入 PC 端 Ubuntu 系统的/home/developer 中，如下图所示。



## 1.3、编译环境安装

解压 RV1126 编译环境工具包，Ctrl+Alt+T 打开终端，并执行 run.sh 脚本。即

可对 RV1126 编译环境进行安装部署。

```
1 | cd ~
2 | tar -xvf develop_environment.tar.gz
3 | cd develop_environment/
4 | ./run.sh
```

执行成功如下图所示：

```
正在从"/etc/skel"复制文件...
正在添加用户"developer"到"sudo"组...
正在将用户"developer"加入到"sudo"组中
完成。
Generating locales (this might take a while)...
  zh_CN.UTF-8... done
  zh_SG.UTF-8... done
Generation complete.

Current default time zone: 'Asia/Shanghai'
Local time is now:      Sun Oct  8 11:33:36 CST 2023.
Universal Time is now:  Sun Oct  8 03:33:36 UTC 2023.

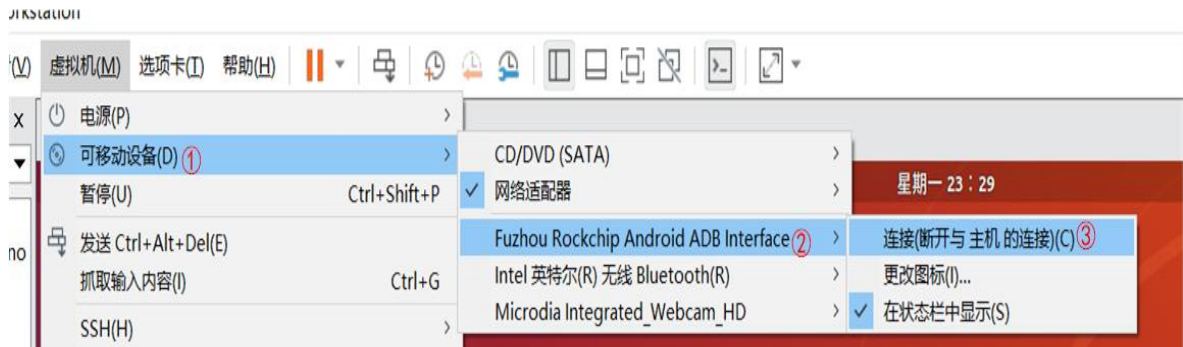
Removing intermediate container 24d2483f90d3
---> a7df669015fe
Step 8/11 : COPY files/mount-nfs.sh /usr/local/bin/
---> 6ed472bcc3f0
Step 9/11 : COPY files/start.sh /usr/local/bin/
---> 5cc296d9a564
Step 10/11 : COPY files/cross.cmake /home/${user}/configs
---> 69e6a7aa83b1
Step 11/11 : COPY files/welcome.txt /home/${user}/configs
---> dca99a5dd838
Successfully built dca99a5dd838
Successfully tagged rv1126-develop:latest
4b49fb56b46fec6530418570c0220a588b2faf2ec83d47879b7826a4104a0c5
rv1126-develop

----- [ 欢迎使用"典阅杯"人工智能算法大赛专用开发版开发环境 ] -----+
开发环境简介：
  在您刚刚运行 ./run.sh 之前，本终端属于 Docker 宿主机环境；运行 ./run.sh 之后本终端
  就进入了 Docker 容器环境。宿主机和开发板分别会与 Docker 容器建立映射关系，您可以在此
  环境中进行 [编译操作] 以及 [应用部署操作]。
  映射关系：
    Docker容器的/opt <==映射到== Docker宿主机的家目录("/home/developer")
    Docker容器的/mnt <==映射到== 开发板的根目录("/")
  注意事项：
    * 编译依赖库部署在开发板上，编译过程中，不能断开adb连接。
    * adb链接断开重连后，开发板与Docker容器建立映射，需要等待一段时间。
-----+
```

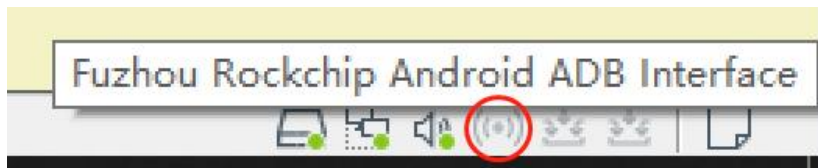
\* 在首次执行 ./run.sh 需要 PC 端虚拟机能够访问互联网。成功安装 RV1126 编译环境后，即可在不联互联网的情况下，通过 ./run.sh 反复进入 RV1126 编译环境。

## 1.4、编译环境使用

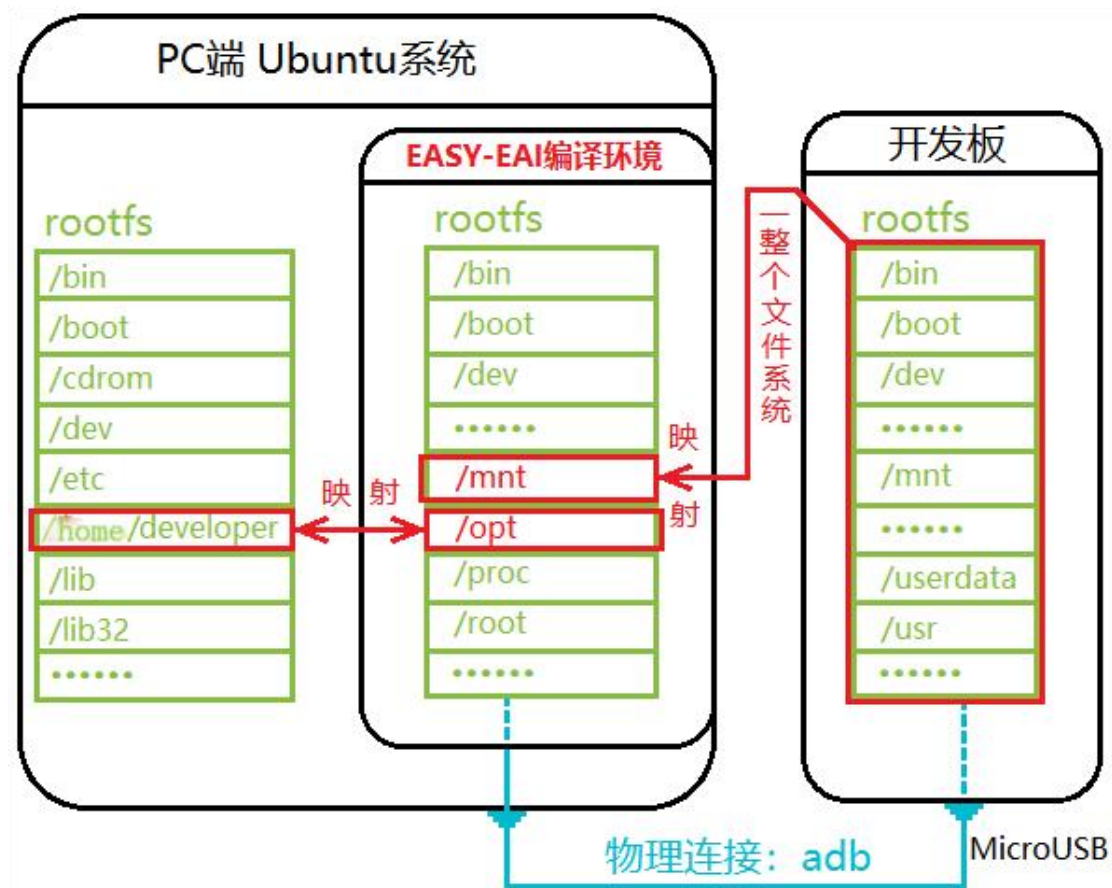
首先**必须确保 adb 链接成功**，硬件连接请查看第二章[《开发板调试方式》](#)连接效果如下图所示。



链接成功会在虚拟机的右下角图标内会出现这个 ADB 设备，如下所示。（若没出现，可参考一下[《USB 驱动安装》](#)）



只要 adb 的链接状态正常，则会自动建立如下方所示的映射关系。



注：adb 链接断开重连后，开发板 rootfs 与 RV1126 编译环境的/mnt 映射建立，需要等待一段时间。

在 RV1126 编译环境中查看/opt, 是 PC Ubuntu 系统的/home/developer, 用户可以在此处放置、编辑以及编译源代码。

```
developer@rv1126-develop:~$
developer@rv1126-develop:~$ ls /opt/
公共的 模板 视频 图片 文档 下载 音乐 桌面 log modules upgrade_tool utils work
developer@rv1126-develop:~$
```

在 RV1126 编译环境中查看/mnt, 是开发板的 rootfs, 里面存放有交叉编译所需的依赖库。在/opt 编译生成好的可执行文件, 建议部署到开发板的 userdata 中。在板卡上通过 apt 工具安装的库也会被立即同步到这个映射中。

```
developer@rv1126-develop:~$
developer@rv1126-develop:~$ ls /mnt
bin boot dev etc home lib lost+found media mnt opt proc root run sbin sdcard srv sys tmp udisk userdata usr var vendor
developer@rv1126-develop:~$
```



关于 RV1126 编译环境的退出，可以执行 exit 命令；再次进入，可以在宿主机中再次执行./run.sh 脚本。

## 2、更新工具依赖库

由于编译工具链的依赖库部署在硬件板卡上，因此需要更新板卡系统固件，即可完成对开发工具链的更新。该步骤赛用开发板无需进行，相关的编译工具依赖库已更新部署完毕。

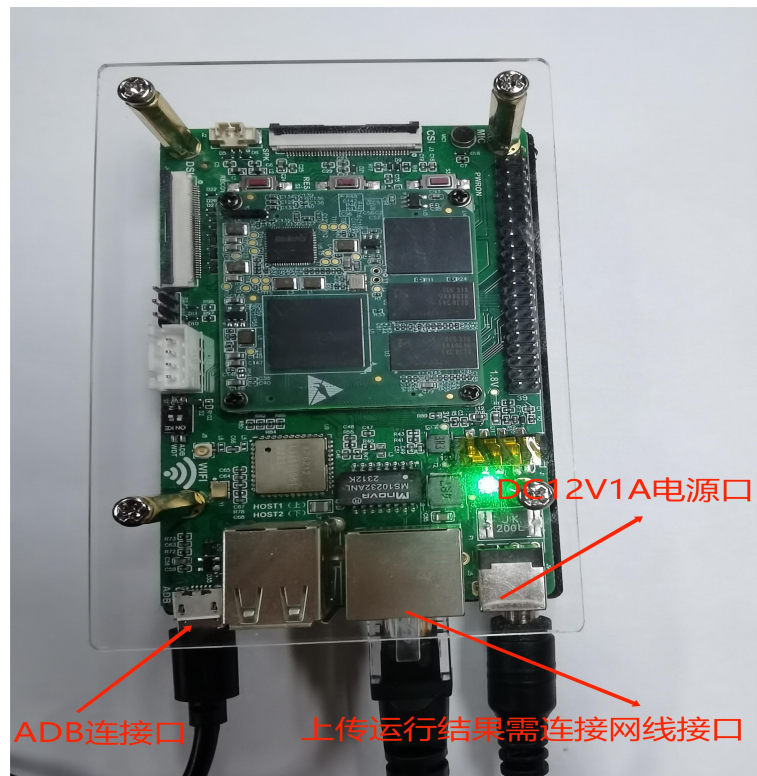
## 二、开发板调试方式

### 硬件链接调试方式：

RV1126 开发板支持多种调试方法，如 ADB 调试、串口调试、网络调试等，在本次比赛中我们采用的是 ADB 调试，在这里我们着重介绍 ADB 调试方式。

在使用 ADB 调试方式前，需要把 ADB 的 Windows 版的 USB 驱动安装好，具体步骤可参考 [《开发环境准备/USB 驱动安装》](#)。

ADB 调试涉及 ADB 接口、电源接口，相关硬件接口如下图所示。



使用 USB 线将 ADB 接口与 PC 端的 USB 口相连，给开发板上电，或通过 RESET 按键重启开发板，PC 上的 Ubuntu 虚拟机就会自动识别到开发板，如下图所示。



把这个设备连接到虚拟机 Ubuntu，连接成功会在虚拟机的右下角图标内会出现一个 USB 接口设备，如下所示。



接着在 Ubuntu 的终端中输入命令“adb shell”就可以开始调试开发板了，如下图所示。

```
bkq@rv11xx-dev: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
bkq@rv11xx-dev:~$ adb shell  
[root@RV1126_RV1109:/]# ls /  
bin          init          media  proc          sdcard        userdata  
busybox.config lib           misc   rockchip_test sys            usr  
data         lib32         mnt    root          timestamp     var  
dev          linuxrc       oem    run           tmp           vendor  
etc          lost+found    opt    sbin          udisk  
[root@RV1126_RV1109:/]#
```

### 三、开发编译方式说明

#### (一)、交叉编译

##### 1、优缺点

###### 优点：

- (1)、采用 x86 架构的 CPU 进行编译，编译速度快。
- (2)、源码编辑方便，开发环境支持各种如 vsCode、qtCreator 等 IDE。

###### 缺点：

- (1)、编译环境需要进行安装部署。
- (2)、程序的调试运行操作相对本地编译不那么直接。

## 2、编译环境简介

RV1126 编译环境是在线编译环境（开发板在线）。建立了 PC 端与板卡端实时同步的映射关系；保障了在 PC 端进行交叉编译时，能与开发板的依赖库进行实时链接。

\* 在线编译：编译过程中必须保持 PC 端与板卡端的 ADB 连接。

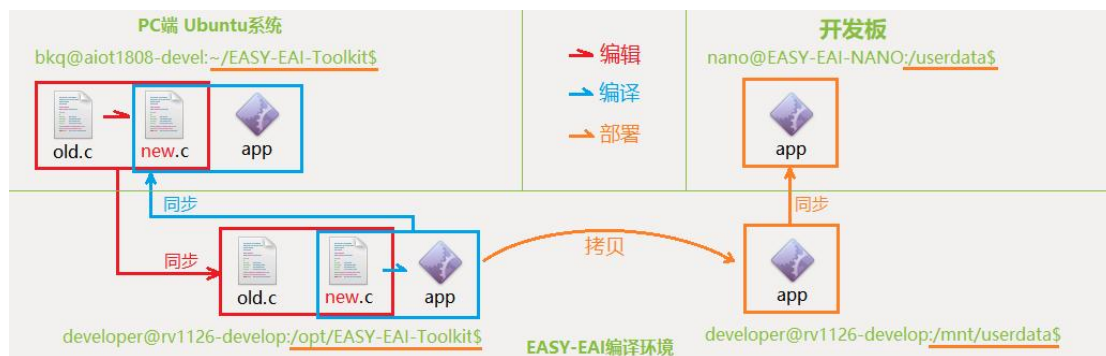
### 2.1、安装部署方式

RV1126 编译环境，相对于传统的通过解压编译工具链配置系统环境变量的交叉编译环境，具有部署更加方便的特点，并且屏蔽了各种兼容性问题。具体的安装步骤在[《开发环境准备/编译环境准备》](#)文章中有详细的说明

### 2.2、映射关系介绍

PC 端 Ubuntu：用于编辑源代码，以及通过 adb 进入板卡环境，进行应用的运行调试。

RV1126 编译环境：用于交叉编译源代码，以及部署应用到板卡中。关系如下图所示。



在 RV1126 编译环境中，PC 端 Ubuntu 系统的 `/home/developer` 会被映射到 RV1126 编译环境中的 `/opt`，因此我们只需要在 PC 端 Ubuntu 编辑了 `/home/developer` 的文件，它就会**自动被同步**到 RV1126 编译环境的 `/opt`。



在 RV1126 编译环境中, 开发板的 rootfs 根目录会被映射到 RV1126 编译环境中的/mnt, 因此我们只需要在 RV1126 编译环境中编辑了/mnt 的文件, 它就会**自动被同步**到开发板的 rootfs 根目录中。

### 3、常规编译手段示例

百度网盘下载

链接: <https://pan.baidu.com/s/1OEdOkK0WpR1U2AHX3xqA1A>

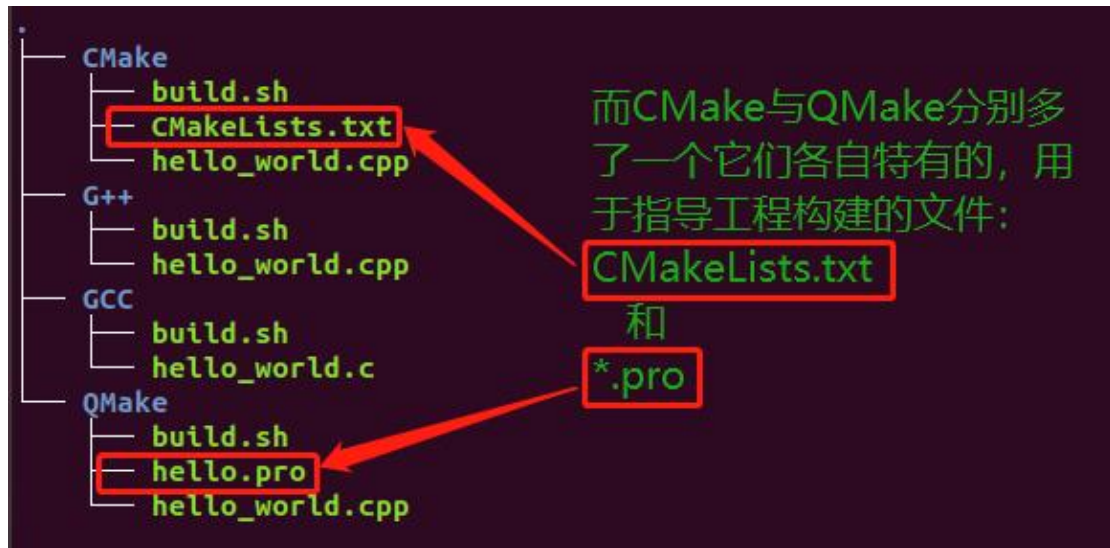
提取码: 0825

此处是四种常规编译方式的编译示例:

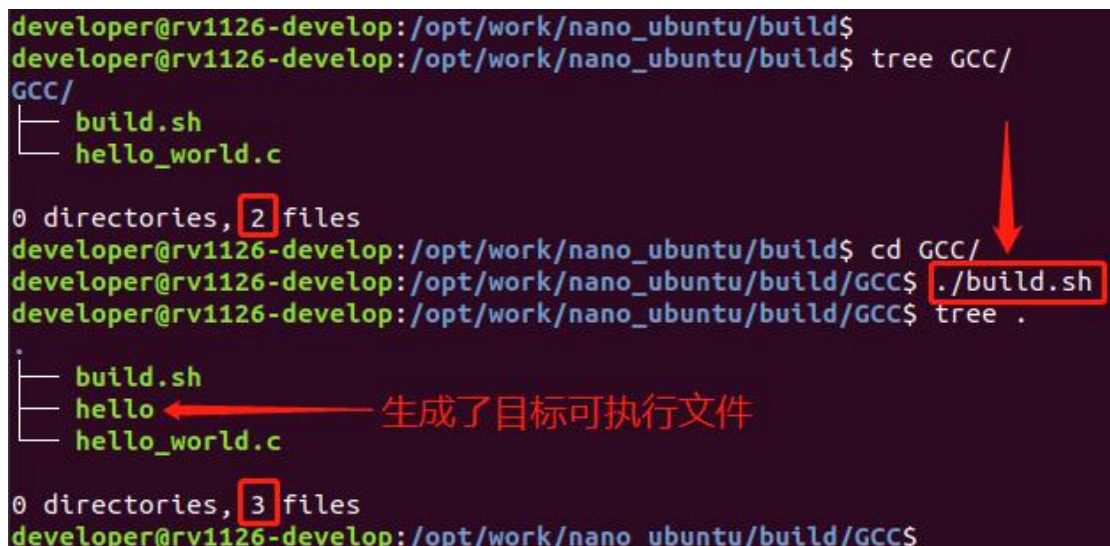


每个示例都包含了一个 build.sh 脚本, 以及一个源代码文件, 如下所示。

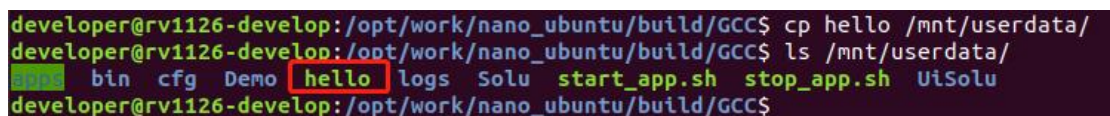




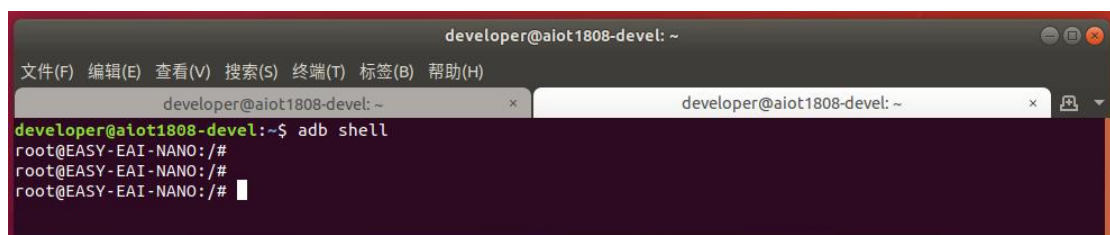
我们随便进入一个示例, 在里面执行./build.sh 操作, 就会生成一个 hello 文件



然后只需要在 RV1126 编译环境中把生成的文件拷贝到/mnt 的任意子目录中即可



通过按键 Ctrl+Alt+T 创建一个新窗口, 执行 adb shell 命令, 进入板卡运行环境。



进入板卡后，定位到刚刚拷贝 hello 文件的位置，执行 ls 命令查看该目录，就可以发现 hello 文件已经被同步过来了。

```
root@EASY-EAI-NANO:/#  
root@EASY-EAI-NANO:/# cd userdata/  
root@EASY-EAI-NANO:/userdata# ls  
Demo Solu UiSolu apps bin cfg hello logs start_app.sh stop_app.sh  
root@EASY-EAI-NANO:/userdata#
```

直接运行 hello 文件，即可看到运行结果

```
root@EASY-EAI-NANO:/userdata# ./hello  
hello ! I am EASY-EAI-nano.  
root@EASY-EAI-NANO:/userdata#
```

## 4、常规编译手段说明

由映射关系得知，RV1126 编译环境的/mnt 目录就是开发板的根目录，由于交叉编译工具的依赖库都在开发板上，因此只需要给编译器指定 sysroot 为 /mnt 即可。

GCC/G++:

```
1 | arm-linux-gnueabi-gcc --sysroot=/mnt hello_world.c -o hello  
2 | arm-linux-gnueabi-g++ --sysroot=/mnt hello_world.cpp -o hello
```

CMAKE: 在 CMakeList.txt 文件中包含以下文件即可。

```
1 | include ($ENV{HOME}/configs/cross.cmake)
```

QMAKE: qmake 工具直接安装到开发板中，所以可以直接 RV1126 编译环境中执行 qmake，示例命令如下。

```
1 | /mnt/usr/bin/qmake xxxxxx.pro  
2 | make
```

## （二）、本地编译

### 1、优缺点

优点:

把开发板直接当成一台卡片电脑使用，无须进行繁琐的应用部署。

### 缺点：

(1)、采用 RV1126 的 CPU 进行编译，性能较弱，编译速度慢。

(2)、源码编辑困难，仅有源生的 VI 编辑器可使用，无法使用各种 IDE。

## 2、使用方法

可以利用：adb、调试串口、ssh 三种方式进入开发板后台，然后直接把开发板当做一台装有 ubuntu 系统的电脑使用。

## 四、模型转换与转换环境搭建

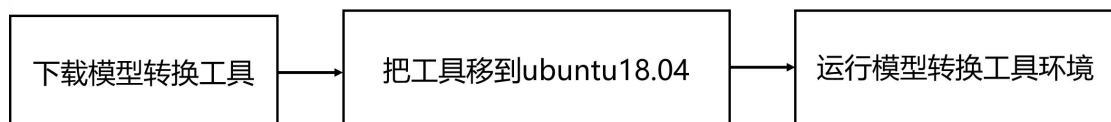
本次比赛，主办方提供基础 onnx 模型，赛事专业开发板部署相应算法模型格式为 rknn 模型，将 onnx 模型转换为 rknn 模型才能在开发板 RV1126 运行。

### (一)、转换为 rknn 模型环境搭建

onnx 模型转换为 rknn 模型后才能在开发板 RV1126 运行，所以需要先搭建 rknn-toolkit 模型转换工具的环境。当然 tensorflow、tensorflow lite、caffe、darknet 等也是通过类似的方法进行模型转换，只是本教程 onnx 为例。

### 1、概述

模型转换环境搭建流程如下所示：



### 2、下载模型转换工具

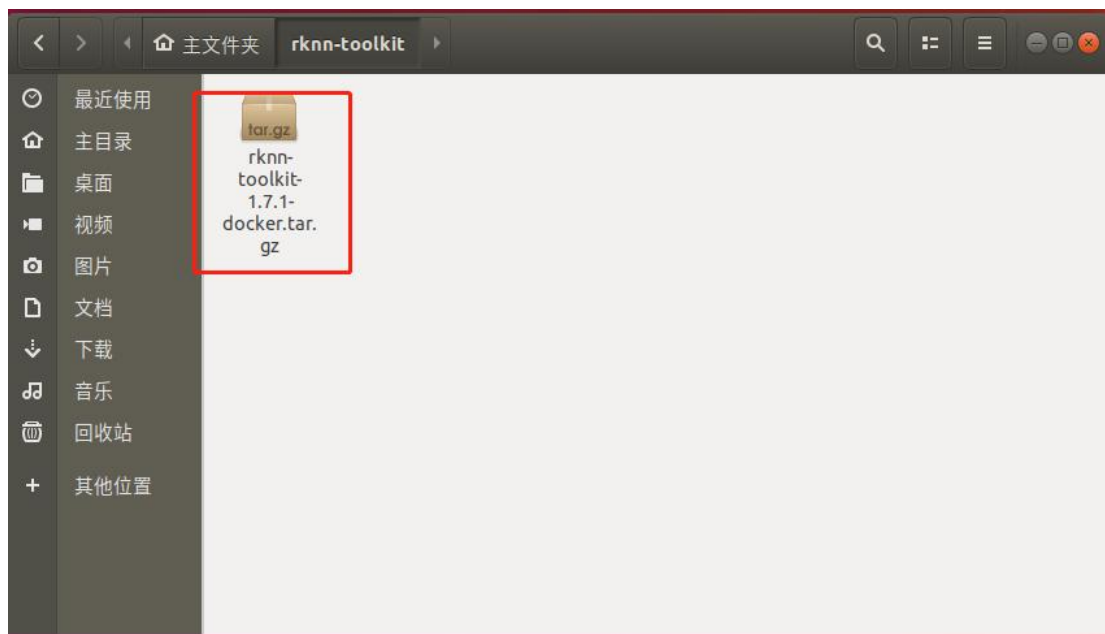
为了保证模型转换工具顺利运行，请在网盘下载；

链接: <https://pan.baidu.com/s/1PSUGE2441YMnmVzlvMBgrQ>

提取码: 0825

### 3、转换工具移植到 Ubuntu 18.04

在主目录下自建一个 rknn-toolkit, 把下载完成的 docker 镜像移到虚拟机 ubuntu18.04 的 rknn-toolkit 目录,如下图所示:



## (二)、运行模型转换工具环境

### 1、打开终端

在该目录打开终端:



## 2、加载 docker 镜像

执行以下指令加载模型转换工具 docker 镜像:

```
docker load --input  
/home/developer/rknn-toolkit/rknn-toolkit-1.7.1-docker.tar.gz
```

注: (以上两行为同一命令, 因命令太长自动换行)

## 3、进入镜像 bash 环境

执行命令: `docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit:1.7.1 /bin/bash`

现象如下图所示:



```
root@5ca22c5f9ded: /
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
developer@EASY-EAI-Develop:~/rknn-toolkit$ docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb
rknn-toolkit:1.7.1 /bin/bash
root@5ca22c5f9ded: /#
```

## 4、测试环境

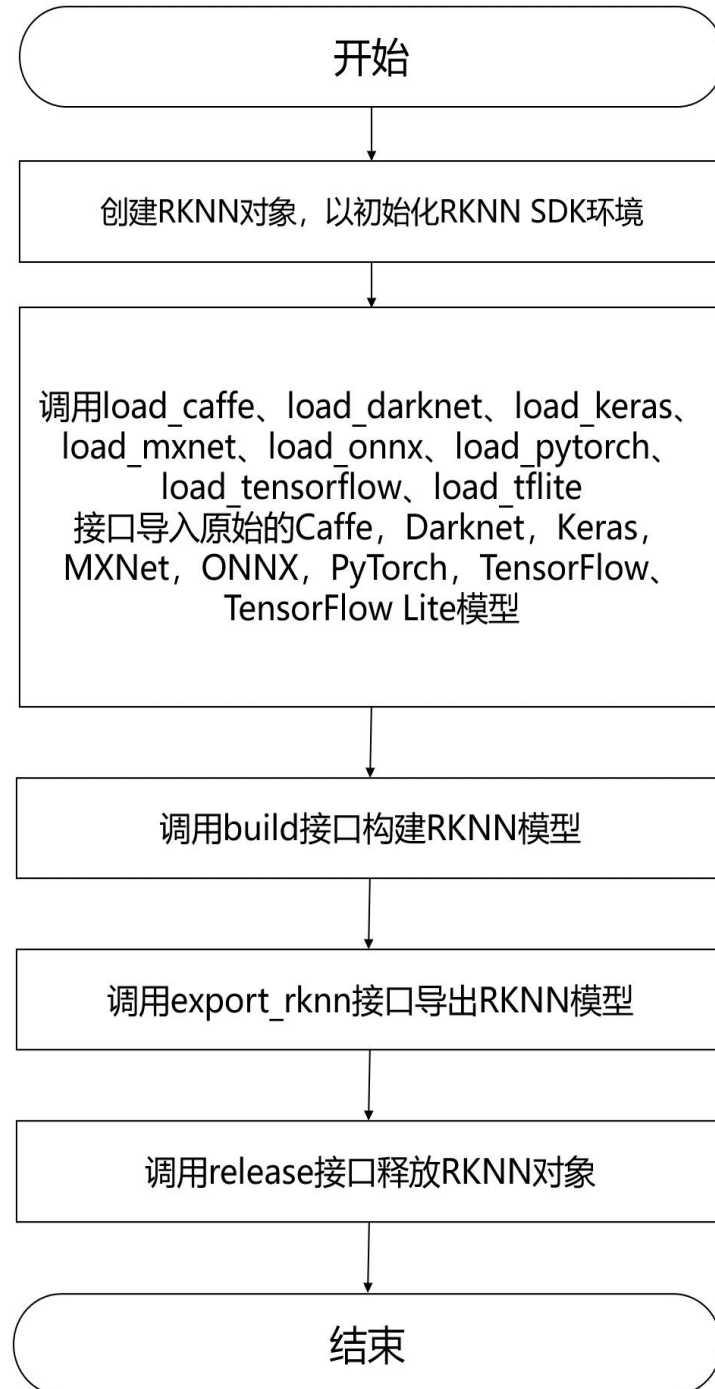
输入“python”加载 python 相关库，尝试加载 rknn 库，如下图环境测试成功:

```
root@5ca22c5f9ded: /
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
developer@EASY-EAI-Develop:~/rknn-toolkit$ docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb
rknn-toolkit:1.7.1 /bin/bash
root@5ca22c5f9ded: /# python
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import rknn
>>>
```

至此，模型转换工具环境搭建完成。

## 五、rknn 模型转换及预编译

RV1126 支持 rknn 后缀的模型的评估及运行，对于常见的 tensorflow、tensorflow lite、caffe、darknet、onnx 和 Pytorch 模型都可以通过我们提供的 toolkit 工具将其转换至 rknn 模型，而对于其他框架训练出来的模型，也可以先将其转至 onnx 模型再转换为 rknn 模型。模型转换操作流程下图所示：





## (一)、模型转换 Demo 下载

百度网盘下载

链接: [https://pan.baidu.com/s/1\\_G095bBxVf6KmOrQLuzogw](https://pan.baidu.com/s/1_G095bBxVf6KmOrQLuzogw)

提取码: 0825

把 model\_convert.tar.bz2 解压到虚拟机, 如下图所示:

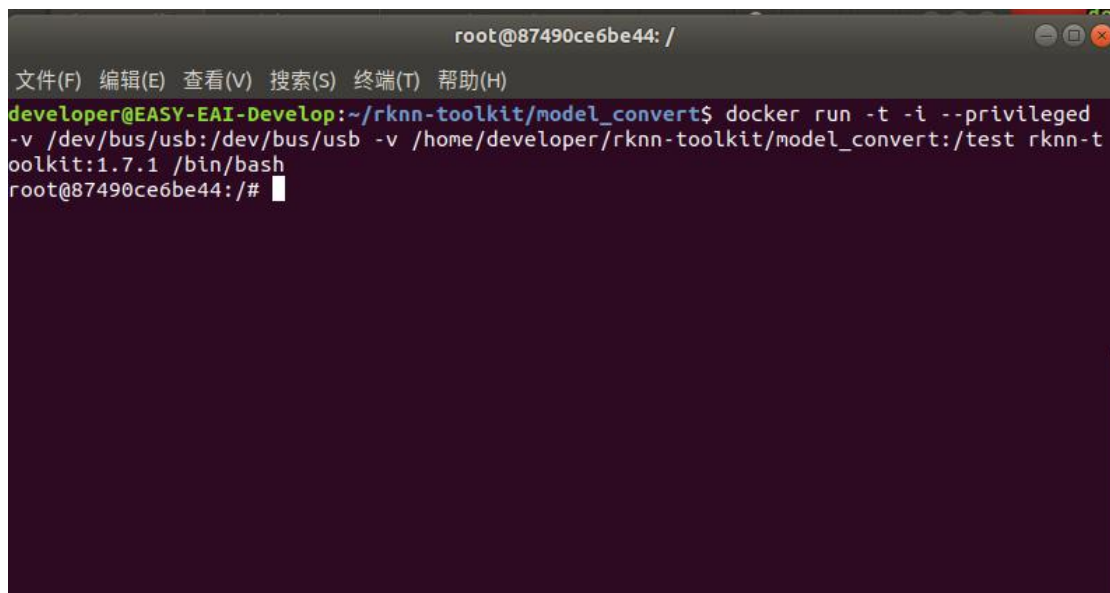


## (二)、进入模型转换工具 docker 环境

执行以下指令把工作区域映射进 docker 镜像, 其中 /home/developer/rknn-toolkit/model\_convert 为工作区域, /test 为映射到 docker 镜像, /dev/bus/usb:/dev/bus/usb 为映射 usb 到 docker 镜像, 执行成功如下图所示:

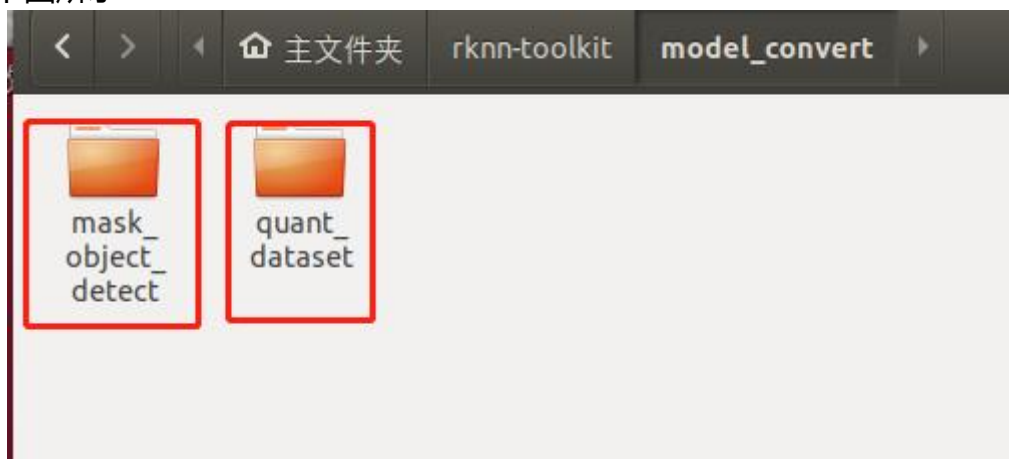
```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v
```

```
/home/developer/rknn-toolkit/model_convert:/test rknn-toolkit:1.7.1 /bin/bash
```

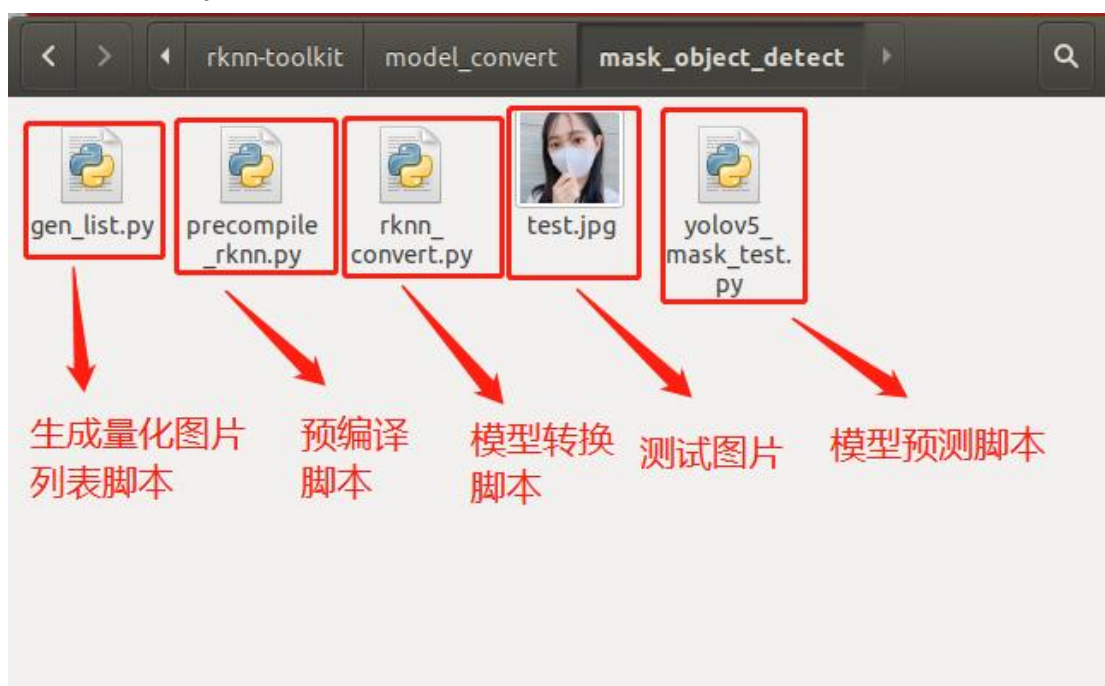


### (三)、模型转换操作说明

模型转换测试 Demo 由 mask\_object\_detect 和 quant\_dataset 组成。  
coco\_object\_detect 存放软件脚本, quant\_dataset 存放量化模型所需的数据。  
如下图所示:



mask\_object\_detect 文件夹存放以下内容, 如下图所示:



在 docker 环境切换到模型转换工作目录, 如下图所示:

```
root@6e9bafd88f54: /test/mask_object_detect
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@6e9bafd88f54:~# cd /test/mask_object_detect/
root@6e9bafd88f54:/test/mask_object_detect#
```

执行 `gen_list.py` 生成量化图片列表，命令行现象如下图所示：

执行命令：`python gen_list.py`

```
root@6e9bafd88f54: /test/mask_object_detect
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@6e9bafd88f54:~# cd /test/mask_object_detect/
root@6e9bafd88f54:/test/mask_object_detect# python gen_list.py
len of all 500
root@6e9bafd88f54:/test/mask_object_detect#
```

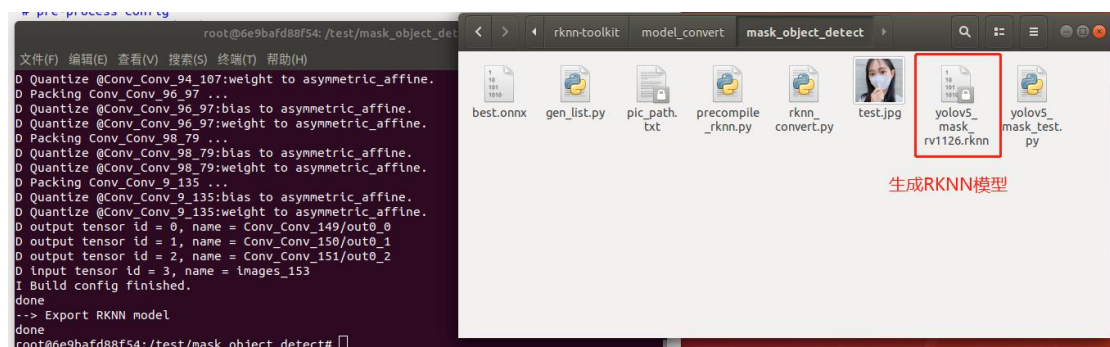
生成“量化图片列表”如下文件夹所示：



#### （四）、onnx 模型转换为 rknn 模型

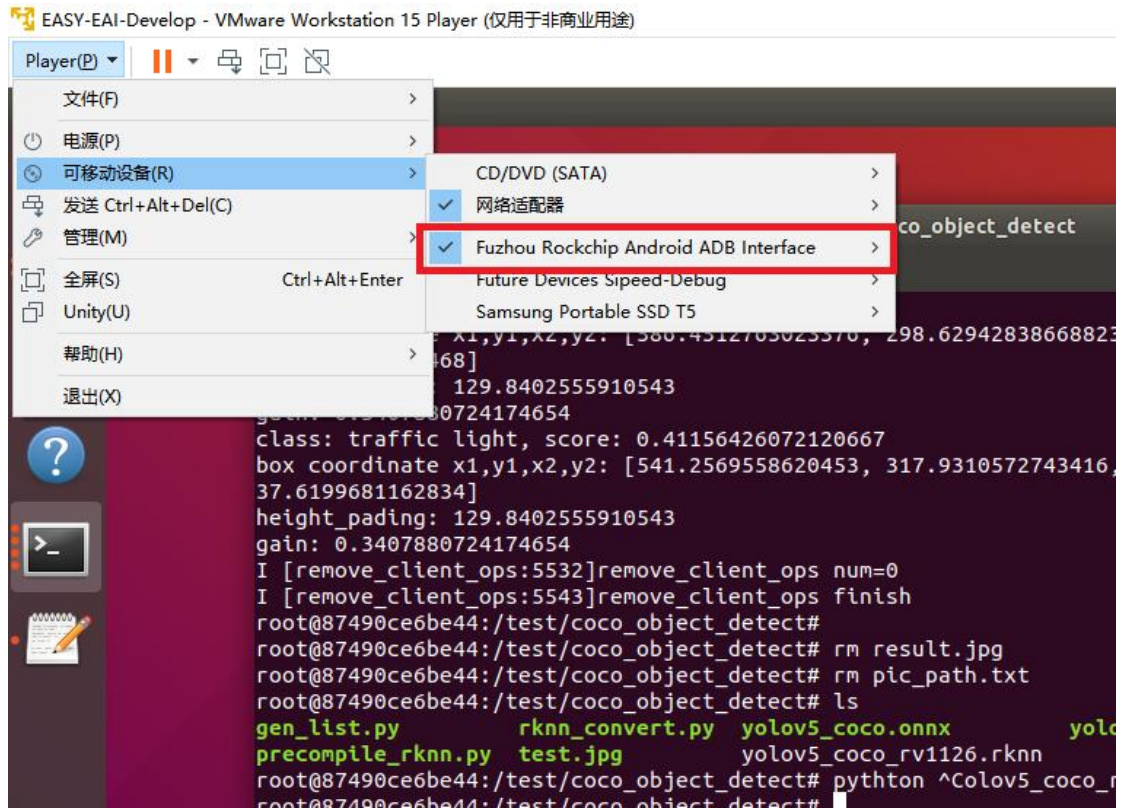
rknn\_convert.py 脚本默认进行 int8 量化操作，把 onnx 模型 best.onnx 放到 mask\_object\_detect 目录，并执行 rknn\_convert.py 脚本进行模型转换：

执行命令：python rknn\_convert.py

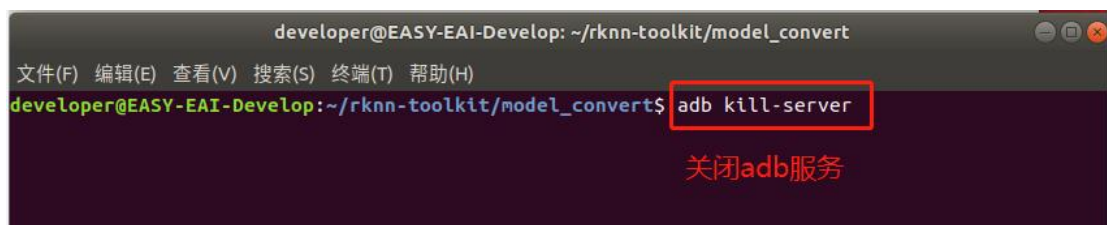


#### 模型预编译

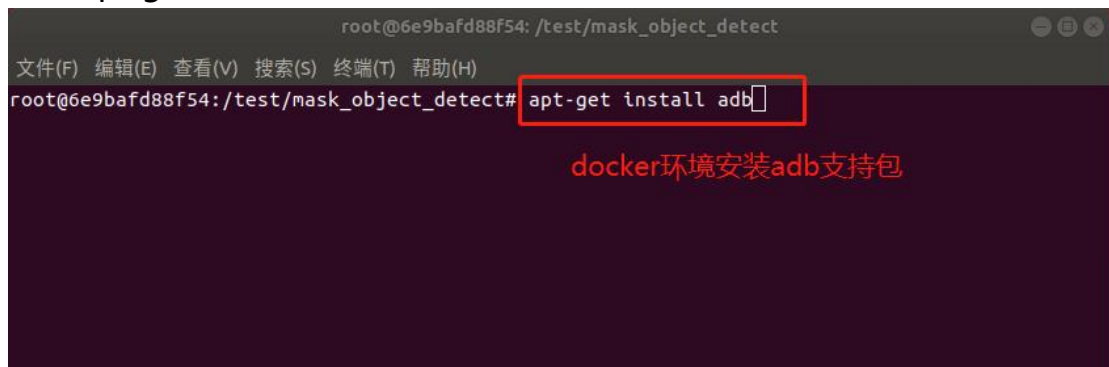
由于 rknn 模型用 NPU API 在 RV1126 加载的时候启动速度会好慢，在评估完模型精度没问题的情况下，建议进行模型预编译。预编译的时候需要通过 RV1126 主板的环境，所以请务必接上 adb 口与 ubuntu 保证稳定连接。虚拟机要保证接上 adb 设备：



由于在虚拟机里 ubuntu 环境与 docker 环境对 adb 设备资源是竞争关系, 所以需要关掉 ubuntu 环境下的 adb 服务, 首先在 PC 端 Ubuntu 中断开与 ADB 连接, 如下图所示:



然后按键 Ctrl+Alt+T 创建一个新窗口终端, 进入 docker 环境, 在 docker 里面通过 apt-get 安装 adb 软件包, 如下图所示:



在 docker 环境里执行 adb devices,现象如下图所示则设备连接成功:

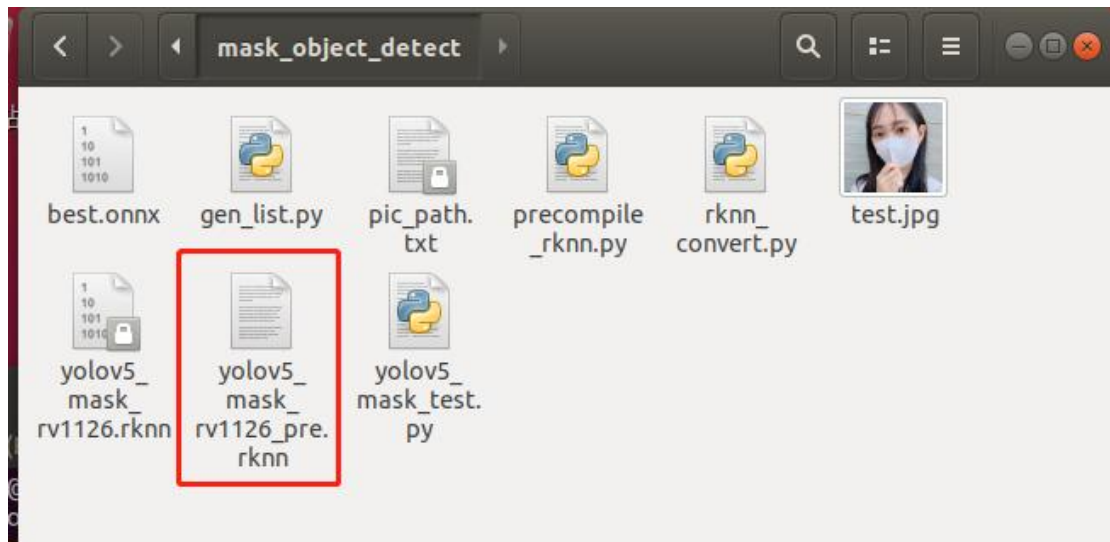


```
root@3a02e292de74: /test/mask_object_detect
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Setting up android-liblog (1:8.1.0+r23-5~18.04) ...
Setting up libusb-1.0-0:amd64 (2:1.0.21-2) ...
Setting up android-libboringssl (8.1.0+r23-2~18.04) ...
Setting up android-sdk-platform-tools-common (27.0.0+10~18.04.2) ...
Setting up android-libcrypto-utils (1:8.1.0+r23-5~18.04) ...
Setting up android-libbase (1:8.1.0+r23-5~18.04) ...
Setting up android-libcutils (1:8.1.0+r23-5~18.04) ...
Setting up android-libadb (1:8.1.0+r23-5~18.04) ...
Setting up adb (1:8.1.0+r23-5~18.04) ...
Processing triggers for libc-bin (2.27-3ubuntu1.4) ...
root@3a02e292de74:/# cd /test/
mask_object_detect/ quant_dataset/
root@3a02e292de74:/# cd /test/mask_object_detect/
root@3a02e292de74:/test/mask_object_detect# ls
best.onnx      precompile_rknn.py  test.jpg
gen_list.py    result.jpg          yolov5_mask_rv1126.rknn
pic_path.txt   rknn_convert.py     yolov5_mask_test.py
root@3a02e292de74:/test/mask_object_detect# adb devices
List of devices attached
* daemon not running; starting now at tcp:5037
* daemon started successfully
7d59a99448995a01    device
root@3a02e292de74:/test/mask_object_detect#
```

运行 `precompile_rknn.py` 脚本把模型执行预编译，执行效果如下图所示，生成预编译模型 `yolov5_mask_rv1126_pre.rknn`：

```
root@3a02e292de74: /test/mask_object_detect
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@3a02e292de74:/test/mask_object_detect# python precompile_rknn.py
precompile ./yolov5_mask_rv1126.rknn to ./yolov5_mask_rv1126_pre.rknn
W Verbose file path is invalid, debug info will not dump to file.
D Using CPPUTILS: True
D target set by user is: rv1126
D Starting ntp or adb, target is RV1126, host is None
D Start adb...
D Connect to Device success!
I NPUTransfer: Starting NPU Transfer Client, Transfer version 2.1.0 (b5861e7@202
0-11-23T11:50:36)
D NPUTransfer: Transfer spec = local:transfer_proxy
D NPUTransfer: Transfer interface successfully opened, fd = 4
D RKNNAPI: =====
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 1.7.1 (566a9b6 build: 2021-10-28 14:53:41)
D RKNNAPI:   DRV: 1.7.1 (0cfd4a1 build: 2021-11-24 09:33:04)
D RKNNAPI: =====
root@3a02e292de74:/test/mask_object_detect#
```

至此预编译部署完成，模型转换步骤已全部完成。生成如下预编译后的 int8 量化模型：



\* 本 demo 教程所使用的 onnx 模型优化操作仅作为参考，参赛队伍需要对主办方提供的算法模型进行算法优化（压缩、剪枝、量化、蒸馏等），优化后的效果将作为评分点进行评分。

## 六、算法模型部署终端

本小节展示 yolov5 模型的在 RV1126 的部署过程，该模型仅经过简单训练供示例使用，不保证模型精度。

### （一）、开发环境准备

编译环境的部署在前面[《开发环境准备/编译环境准备与更新》](#)，在 PC 端 Ubuntu 系统中执行 run 脚本，进入 RV1126 编译环境，具体如下所示。

```
developer@aiot1808-devel:~/develop_environment$ ls
developer@aiot1808-devel:~/develop_environment$ clean.sh Dockerfile files run.sh
developer@aiot1808-devel:~/develop_environment$ ./run.sh
rv1126-develop
developer@rv1126-develop:~$
developer@rv1126-develop:~$
developer@rv1126-develop:~$
```

上部分属于PC端Ubuntu环境

从这里开始就进入了【RV1126 编译环境】

### （二）、源码下载以及例程编译

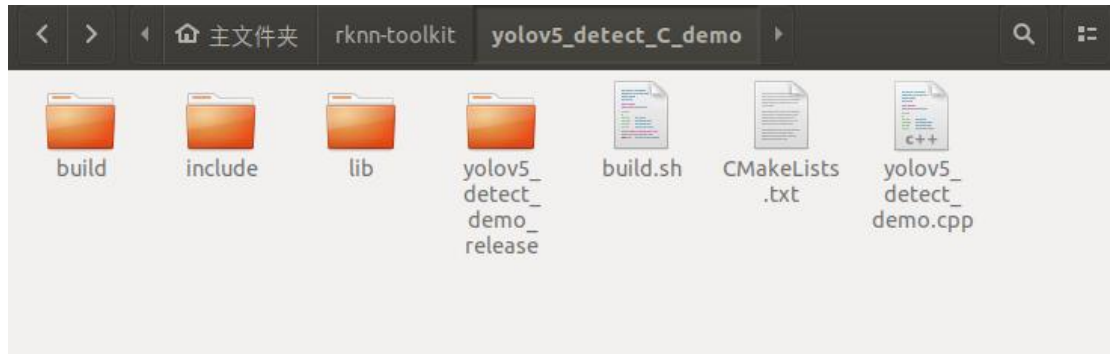
百度网盘下载

链接：<https://pan.baidu.com/s/1VQlqgob8GfVbA3rZlcEAyw>

提取码：0825

下载程序包移至 Ubuntu 环境后，执行以下指令解压：

```
tar -xvf yolov5_detect_C_demo.tar.bz2
```

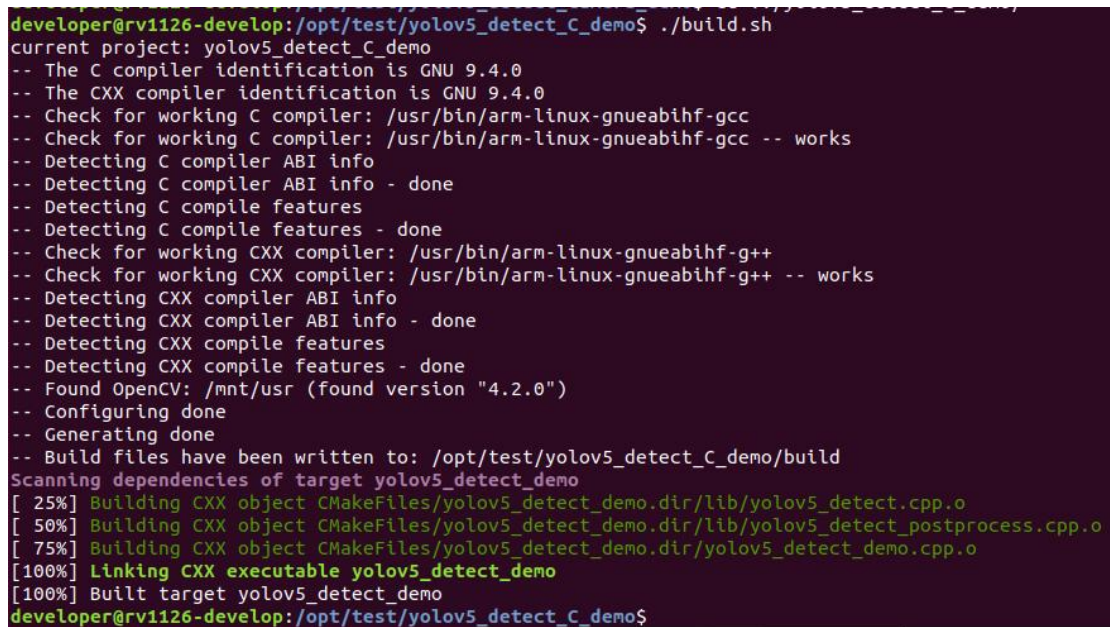


在 RV1126 编译环境下，切换到例程目录执行编译操作：

```
cd /opt/rknn-toolkit/yolov5_detect_C_demo
```

```
./build.sh
```

\* 由于依赖库部署在板卡上，因此交叉编译过程中必须保持 adb 连接。



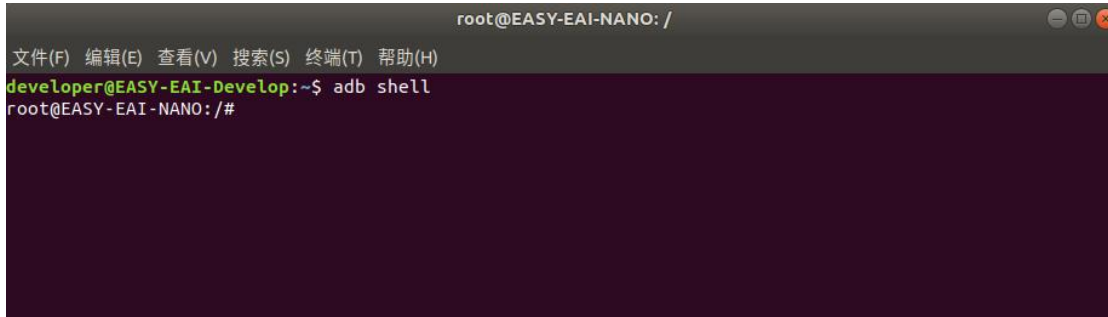
### （三）、方案部署

在 RV1126 编译环境下，在例程目录执行以下指令把可执行程序推送到开发板端：

```
cp yolov5_detect_demo_release/ /mnt/userdata/ -rf
```

通过按键 Ctrl+Alt+T 创建一个新窗口，执行 adb shell 命令，进入板卡运行环境：





```
root@EASY-EAI-NANO: /
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
developer@EASY-EAI-Develop:~$ adb shell
root@EASY-EAI-NANO: /#
```

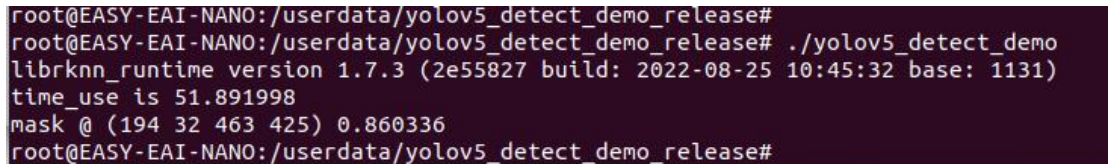
进入板卡后，定位到例程上传的位置，如下所示：

```
cd /userdata/yolov5_detect_demo_release/
```

运行例程命令如下所示：

```
./yolov5_detect_demo
```

运行例程命令，执行结果如下图所示，算法执行时间约为 50ms：



```
root@EASY-EAI-NANO:/userdata/yolov5_detect_demo_release#
root@EASY-EAI-NANO:/userdata/yolov5_detect_demo_release# ./yolov5_detect_demo
librknn_runtime version 1.7.3 (2e55827 build: 2022-08-25 10:45:32 base: 1131)
time_use is 51.891998
mask @ (194 32 463 425) 0.860336
root@EASY-EAI-NANO:/userdata/yolov5_detect_demo_release#
```

退出板卡环境，取回测试图片：

退出板卡：exit

取出测试图片：

```
adb pull /userdata/yolov5_detect_demo_release/result.jpg
```

进入指定得文件位置便可查看测试后所得到的图片。