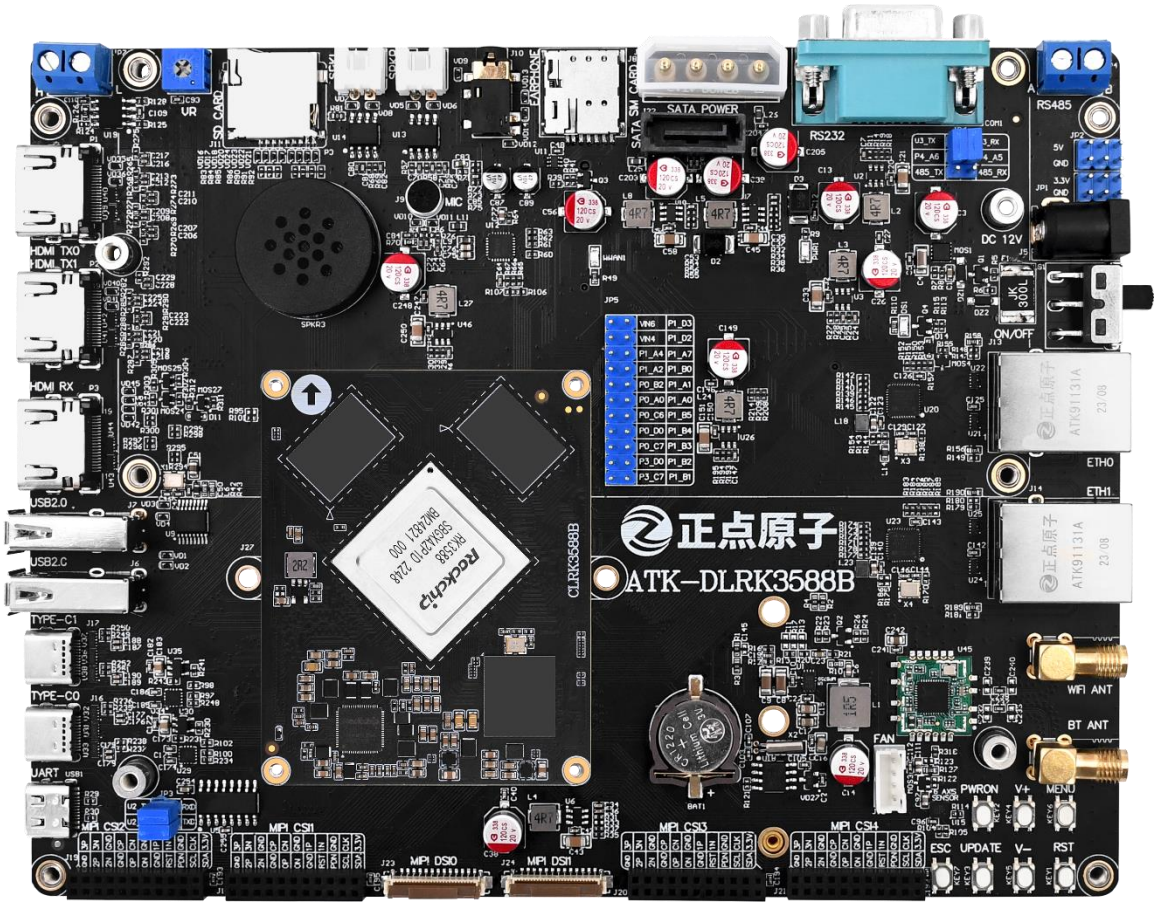
ATK-DLRK3588

Linux5.10 AI Test Manual

V1.1



ALIENTEK

1. Shopping:

TMALL: https://zhengdianyuanzi.tmall.com

TAOBAO: https://openedv.taobao.com

2. Download

Address: http://www.openedv.com/docs/index.html

3. FAE

Website : www.alientek.com

Forum : http://www.openedv.com/forum.php

Videos : www.yuanzige.com

Fax : +86 - 20 - 36773971

Phone : +86 - 20 - 38271790

Disclaimer

The product specifications and instructions mentioned in this document are for reference only and subject to update without prior notice; Unless otherwise agreed, this document is intended as a product guide only, and none of the representations made herein constitutes a warranty of any kind.

In order to get the latest version of product information, please regularly visit the download center or contact the customer service of Taobao ALIENTEK flagship store. Thank you for your tolerance and support.

Revision History:

Version	Version Update Notes	Responsible person	Proofreading	Date
V1.0	release officially	ALIENTEK	ALIENTEK	2024.06.15
V1.1	1. Add test instructions related to rk3568 2. Correct some errors in the illustrations and text of certain sections	ALIENTEK	ALIENTEK	2024.07.10

Catalogue

Brief

With the rapid development of artificial intelligence technology, deep learning has become an important force driving technological innovation. Especially in the field of edge computing, efficient and low-power AI inference capabilities are crucial for realizing intelligent applications. RKNN (Rockchip Neural Network), as a neural network inference framework based on the Rockchip platform, has achieved excellent performance, ease of use, and wide compatibility, and has been widely applied in the AIoT (Artificial Intelligence Internet of Things) field.

This document aims to provide readers with a detailed guide on the testing of RKNN AI routines and the setup of the development environment. By systematically introducing the key steps required for setting up the development environment for RKNN AI routine testing, updating the NPU driver, installing the rknn-toolkit2 conversion environment, adjusting CPU and NPU frequencies, and demonstrating how to apply RKNN for AI inference in different models (such as classification networks, object detection, object segmentation, face detection, OCR text detection and recognition, portrait segmentation, etc.) through specific cases, readers can quickly master the usage skills of RKNN and provide strong support for the development of actual projects.

The content of this document covers all aspects from the basic environment setup to advanced model application, and is suitable for readers such as developers, researchers, and engineers who are interested in RKNN. Through systematic learning and practice, readers will be able to deeply understand the working principle and application scenarios of RKNN, master the AI application development process based on RKNN, and contribute to the intelligent progress in the AIoT field.

In terms of document organization, we first introduced the basic development environment setup required for the RKNN AI routine testing, including an overview of common IDE tools, installation of cross-compilation toolchains, installation and configuration of Anaconda, etc. Then, we detailed the update method of the NPU driver to ensure that RKNN can run normally on the hardware platform. Subsequently, we introduced the installation of the rknn-toolkit2 conversion environment, which is a key step for converting the trained neural network model into the RKNN format. Additionally, we also introduced how to view and adjust the CPU and NPU frequencies to optimize the performance of AI inference.

After completing the setup of the basic environment, we moved on to the specific model testing phase. Firstly, we introduced the testing method for Python inference routines in the buildroot environment, including the installation of RKNN Toolkit Lite2 and the AI routine testing under Python. Then, we detailed the testing methods for various models in the modelzoo, including classification network models, object detection models, object segmentation models, etc. During each model's testing process, we provided detailed steps and explanations so that readers could follow the guidelines for operation.

Finally, we also provided AI routine tests for specific application scenarios, such as face detection video inference routines, OCR text detection and recognition routines, and portrait segmentation routines. These routines demonstrate the application capabilities of RKNN in practical projects, providing valuable references and inspirations for readers.

We hope that through the introduction and guidance in this document, readers can master the skills of RKNN AI routine testing and development environment setup, contributing to intelligent applications in the AIoT field.

Chapter 1. RKNN AI Routine Test Development Environment Setup

In this chapter, we will introduce to you how to set up the AI routine test development environment, so that after you get the development board, you can directly compile the AI routine and then conduct tests.

This chapter will be divided into the following sections:

1. Introduction to Common IDE Tools
2. Installation of Cross-Compilation Tool Chain
3. Installation and Environment Configuration of Anaconda

1.1 Introduction to Common IDE Tools

You can refer to the tutorial "Development Board CD-ROM A Disk → **Basic Materials → 10_user_manual → [ALIENTEK] ATK-DLRK3588 Embedded Linux System Development Manual V1.0**" to install the following development environment that is needed. A brief introduction to these tools follows.

1. VMware Workstations

This tool is used to install our Ubuntu system. The software installation packages in the data disk also provide them. You can install them according to the documentation. The subsequent development compilation and other basic operations are all carried out in the Ubuntu environment installed based on this software.

2. Ubuntu 20.04

This is the platform on which we will conduct AI routine test and development in the future. All the model conversion environments we will encounter later will be configured and converted under this system environment. It is strongly recommended that you install the Ubuntu 20.04 version as per the documentation. It is not recommended to use other Ubuntu environments. Inconsistent environments may lead to different problems. All the content in this document is based on this Ubuntu environment.

3. Vscode

A powerful code editor, a lightweight yet feature-rich cross-platform IDE that supports multiple programming languages and extension plugins. We mainly use this editor to write and debug our code.

4. Mobaxterm

A terminal software integrating multiple functions such as serial ports and SSH, used to connect to our development boards via serial ports or networks for development, debugging, and other functions.

1.2 Install the cross-compilation toolchain

1.2.1 Copy the cross-compilation toolchain

Before compiling the program, we need to install the cross-compilation toolchain into Ubuntu. This toolchain is compiled through the RK3588 buildroot SDK and contains the libraries needed for our AI routines, which can be directly used on the board. Copy the file "atk-dlrk3588-toolchain-arm-buildroot-linux-gnueabihf-x86_64-xxxxxx-x.x.x.run" from the "Development Board CD A - **Basic Materials → 05_tools → 03. Cross Compilation Tools**" on the data disk to any directory in Ubuntu (the cross-compilation toolchain will be continuously updated, so please refer to the actual directory and toolchain name), as shown in Figure 1.2.1.1.

Figure 1.2.1.1 Cross-compilation toolchain

The path for the Linux 5.10 version cross-compiler of the RK3568 development board is located at "Development Board CD-ROM A Disk - Basic Materials → 05_tools → 03, Cross-Compilation

Tools → atk-dlrk3568-toolchain-arm-buildroot-linux-gnueabihf-x86_64_5_10_sdk_20240704-x.x.x.run", as shown in Figure 2.2.1.2.

Figure 1.2.1.2 Cross-compilation toolchain

This chapter only shows the installation method of the rk3588 cross-compilation toolchain. The installation method for rk3568 is similar. We won't go into detail here. For more detailed methods, you can refer to the A disc of the rk3568 development board - Basic Materials → 10_user_manual → 03, Auxiliary Documents → 33 [ALIENTEK] Buildroot System - Cross Compiler Installation and Usage Reference Manual V1.0.pdf.

Here, the author copies the cross-compilation toolchain to the newly created "toolchain" directory in the home directory. You can use the "ls -l" command to check the executable permissions and other details.

```
dominick@ubuntu:~/toolchain$ ls -l
总用量 352604
-rwxrw-rw- 1 dominick dominick 361060669 May  9 00:30 atk-dlrk3588-toolchain-arm-buildroot-linux-gnueabihf-x86_64-20240506-v1.0.run
dominick@ubuntu:~/toolchain$
```

Figure 1.2.1.3 Check the installation files of the cross-compiler toolchain.

If there is no execution permission, and you are using the RK3588 development board, you can execute the following command to grant execution permission.

```
chmod a+x atk-dlrk3588-toolchain-arm-buildroot-linux-gnueabihf-x86_64-20240506-v1.0.run
```

The development board used is RK3568. The following commands can be executed to grant execution permissions.

```
chmod a+x atk-dlrk3568-toolchain-arm-buildroot-linux-gnueabihf-x86_64_5_10_sdk_20240704-v1.0.0.run
```

Note: This cross-compilation toolchain will be updated later and it is not the final version. The purpose of the update is to adapt to more routines. If you need to reinstall, simply uninstall and then install the latest version. The installation process is very simple.

1.2.2 Installation of Cross-Compilation Toolchain

If the development board used is the rk3588 development board, open the terminal and execute the following commands to install the compilation toolchain. The installation process is shown in Figure 2.2.2.1.

```
./atk-dlrk3588-toolchain-arm-buildroot-linux-gnueabihf-x86_64-20240506-v1.0.run
```

If the development board used is the rk3568 development board, open the terminal and execute the following commands to install the compilation toolchain. The installation process is similar to that of the rk3588.

```
./atk-dlrk3568-toolchain-arm-buildroot-linux-gnueabihf-x86_64_5_10_sdk_20240704-v1.0.0.run
```

When the prompt "Enter the target directory for the toolchain (default: /opt/atk-dlrk3588-toolchain):" appears, it indicates whether to select the default installation in the /opt/atk-dlrk3588-toolchain directory. It is recommended to directly select the default installation path and simply press

the Enter key. When the prompt "You are about to install the toolchain to '/opt/atk-dlrk3588-toolchain'. Proceed [Y/n]?" appears, simply press "Y" and then press Enter. After entering the Ubuntu password and pressing Enter, when the prompt "$. source /opt/atk-dlrk3588-toolchain/environment-setup" appears, it indicates that the installation is complete.

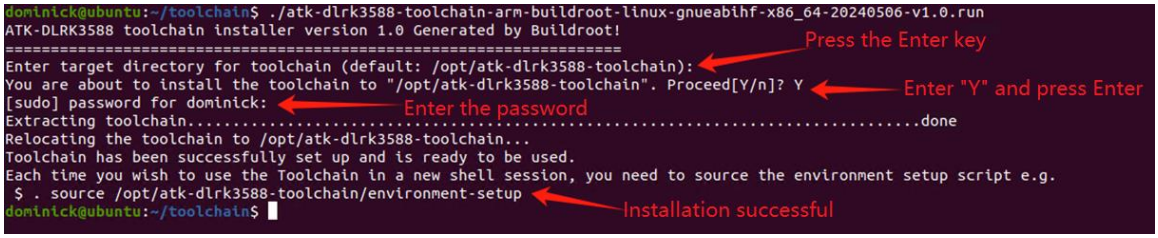


Figure 1.2.2.1 Installation of cross-compiler toolchain installation

At this point, the cross-compiler toolchain has been installed.

1.2.3 Uninstallation of cross-compiler toolchain

Open the terminal of Ubuntu and go to the /opt directory. Execute "ls" to see the folder of the installed cross-compilation toolchain. Then, execute the following command to delete it.

```
cd /opt
sudo rm -rf atk-dlrk3588-toolchain/
```

Figure 1.2.2.1 Uninstall the cross-compilation toolchain

Executing "ls" again will show that this folder is gone, indicating that the cross-compilation toolchain has been successfully uninstalled.

If you delete the cross-compiler for RK3568, execute the following command.

```
cd /opt
sudo rm -rf atk-dlrk3568-5_10_sdk-toolchain/
```

1.3 Installation and environment configuration of Anaconda

1.3.1 Download and installation of Anaconda

Open the Tsinghua mirror repository to download https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/?C=M&O=D.

Since we choose to perform model conversion and performance evaluation operations in the Ubuntu environment, we select Anaconda3-2023.03-1-Linux-x86_64.sh for installation and use. We also provide the A disk of the development board CD - **Basic Materials → 04_software → Anaconda3-2023.03-1-Linux-x86_64.sh**.

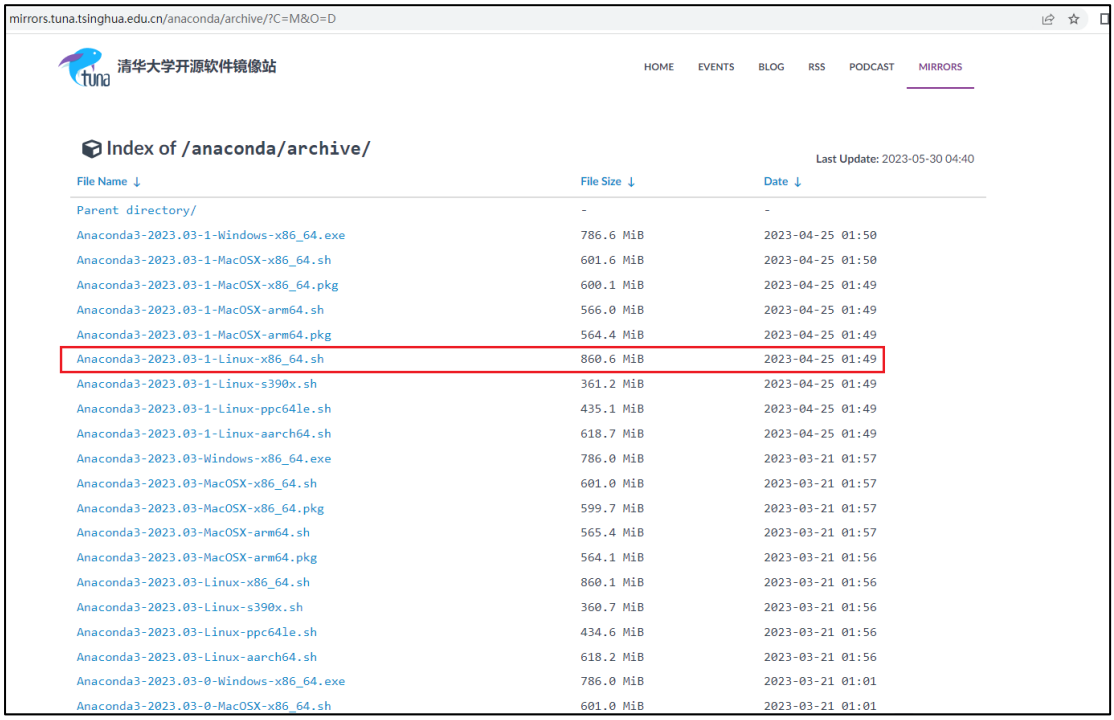


Figure 1.3.1.1 Cross-compilation toolchain installation interface

Here, you can open the terminal and enter the following commands. First, create a new folder named "software" to facilitate future use. Then, enter the "software" folder and download anaconda using wget (here, the "-c" option is added to enable resuming of the download in case of interruption).

```
mkdir ~/software
cd ~/software
wget -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/Anaconda3-2023.03-1-Linux-x86_64.sh
```

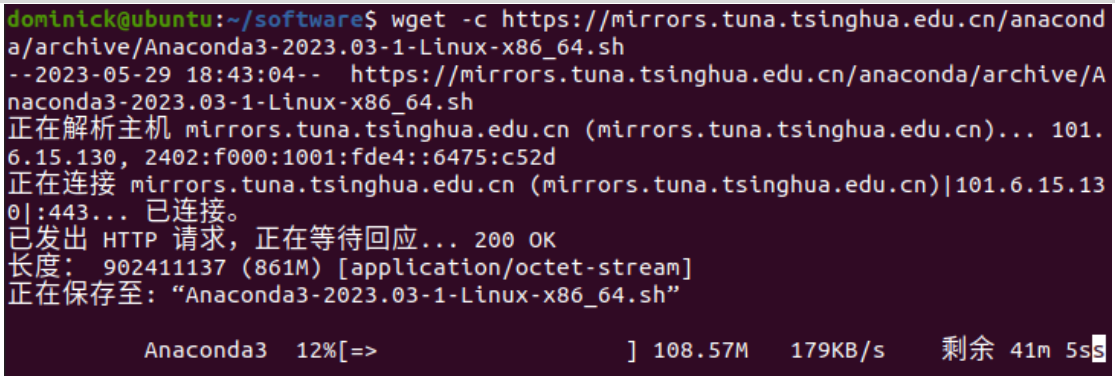


Figure 1.3.1.2 Download the Anaconda3 software.

After the download is complete, use the bash command to install it, and then press Enter once to continue the installation.

```
bash Anaconda3-2023.03-1-Linux-x86_64.sh
```

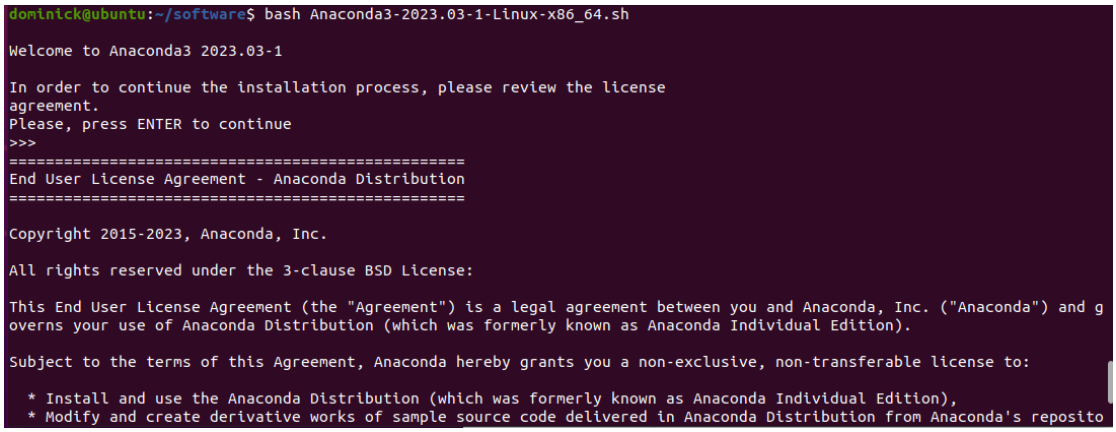


Figure 1.3.1.3 Install Anaconda3 software

Press and hold the Enter key again to read the license until the [no] >>> appears. Enter "yes" and press Enter to agree. When the installation path appears, it will be installed by default in the anaconda3 folder under the user directory. You can also specify your own installation path. Here, we will install it directly to the default location. Press Enter to proceed.

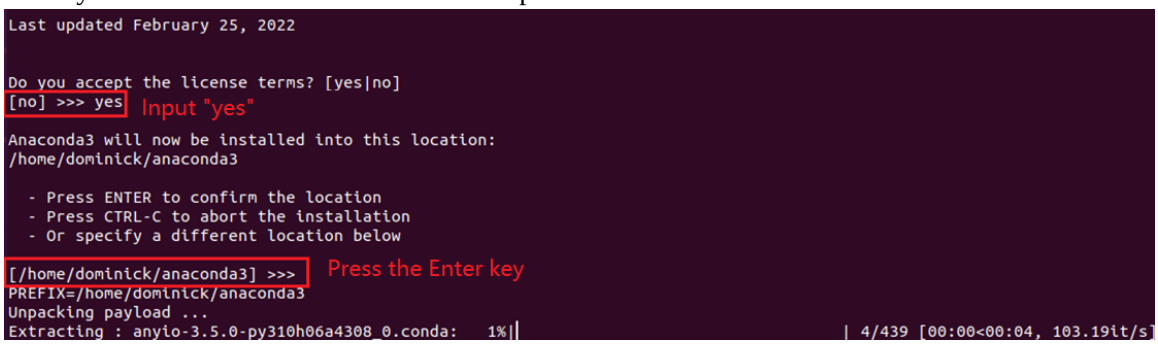


Figure 1.3.1.4 Agree to the license agreement and specify the installation path.

Then enter "yes" and press Enter. When you see "Thank you for installing Anaconda3!", the installation is complete.

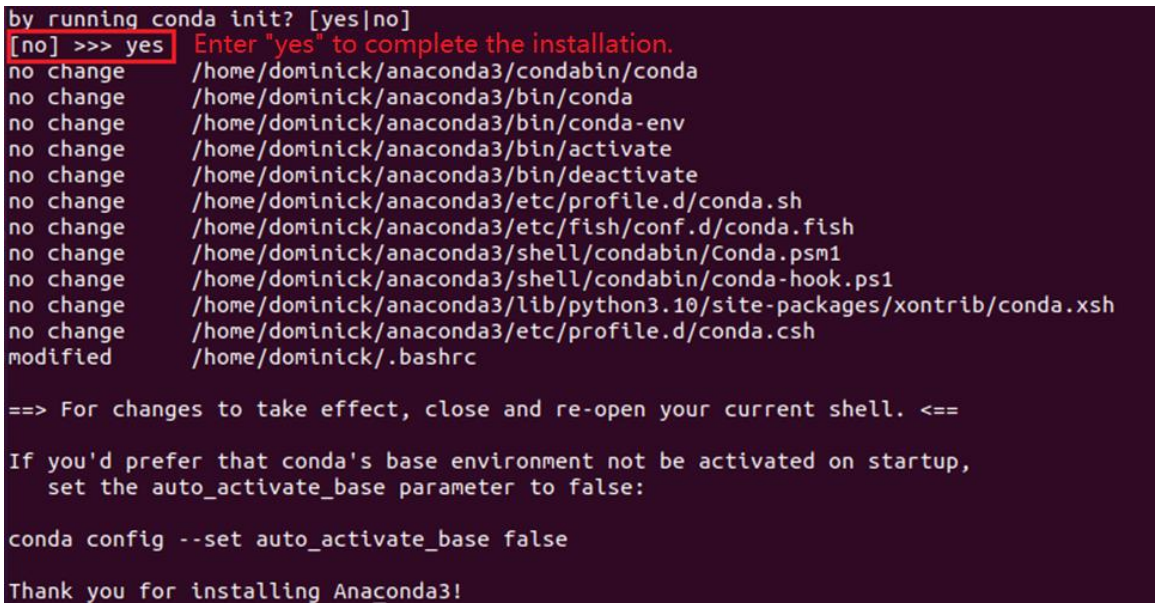


Figure 1.3.1.5 Anaconda installation completed.

After restarting Ubuntu and opening the terminal, you will notice that there is an additional (base) in front. This indicates that you have entered the Conda environment. To exit the Conda environment and return to the Ubuntu environment, simply type "conda deactivate".

Figure 1.3.1.6 Exit the Conda environment

Note!!! This way, every time you boot up, you will automatically enter the Conda environment, which will be very inconvenient. To prevent confusion with other environments such as the ones for compiling the RK3568/RK3588 SDK, we modify the Conda environment variables to set it so that it does not automatically enter the virtual environment of Conda. Execute the following command in the terminal.

```
conda config --set auto_activate_base false
```

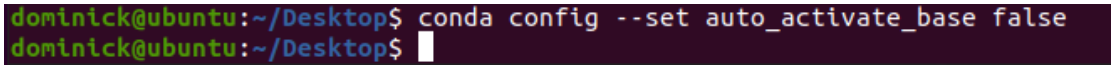


Figure 1.3.1.7 The configuration does not automatically enter the Conda environment.

After restarting, when opening the terminal, it was found that the default Conda environment would not be entered.

1.3.2 Configuration of Anaconda environment

Configure the download source of the Anaconda environment. Since the official download source is located abroad and may be slow, it is recommended to change the download source to the Anaconda mirror source in China.

a. First, execute the following command to check the current mirror source. It shows that there is only one defaults default source.

```
conda config --show channels
```

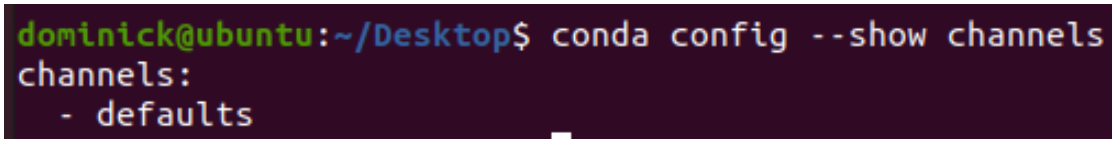


Figure 1.3.2.1 Check the configuration of the Conda environment

b. Next, add the Tsinghua mirror source

```
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
```

Figure 1.3.2.2 Configure the conda environment image source

c. Check the configuration

```
conda config --show
```

```
dominick@ubuntu:~/Desktop$ conda config --show
add_anaconda_token: True
add_pip_as_python_dependency: True
aggressive_update_packages:
  - ca-certificates
  - certifi
  - openssl
allow_conda_downgrades: False
allow_cycles: True
allow_non_channel_urls: False
allow_softlinks: False
allowlist_channels: []
always_copy: False
always_softlink: False
always_yes: None
anaconda_upload: None
auto_activate_base: False
auto_stack: 0
auto_update_conda: True
bld_path:
changeps1: True
channel_alias: https://conda.anaconda.org
channel_priority: flexible
channel_settings: []
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
  - defaults
```

Figure 1.3.2.3 Check if the Conda environment image source has been configured properly

Chapter 2. Update NPU Driver

RKNN-Toollkit2 is a model conversion tool implemented in Python language, which can convert models exported from other training frameworks into RKNN models and provides relatively limited inference interfaces to assist users in testing the model conversion effect. The website link is: https://github.com/rockchip-linux/rknn-toolkit2. RKNPU2 is a board-end component that provides NPU driver and offers functions such as model loading and model inference based on C language. RKNPU2 has a consistent version number. There may be incompatibility issues between different versions, to avoid causing unnecessary troubles, it is recommended that users use the same version of RKNN-Toolkit2 and RKNPU2.

This chapter is divided into the following sections:

1. Download the rknpu2 driver compressed package
2. Update the npu board connection inference runtime library

2.1 Download the rknpu2 driver compression package

In order to facilitate the subsequent use of OTG and the development board connection for AI-related development performance tests and board connection inference functions, it is necessary to update the NPU driver for rk3588/rk3568 (precisely speaking, it is not a driver, but a board-end board connection service. Regarding the rknpu driver, we have updated it to the current latest version 0.9.2). You can view the NPU driver through the following command.

```
dmesg | grep -i rknpu
```

As shown in the figure below, the current version of the RKNPU2 driver is 0.9.2.

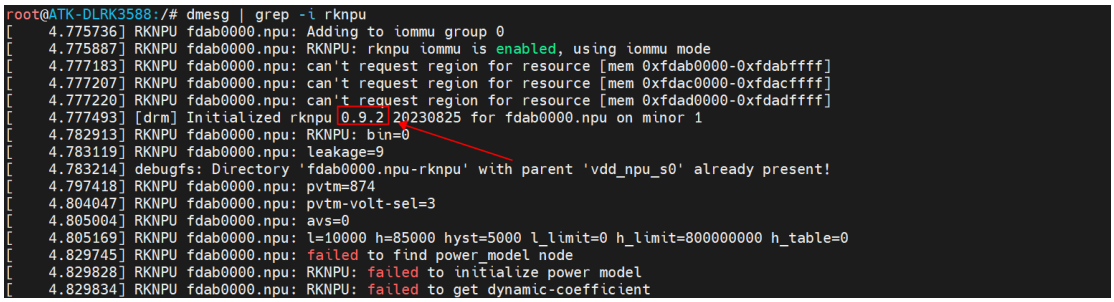


Figure 2.1.1.1 Check the NPU driver version

Here is the practical operation on how to update the NPU driver and board service for rk3588. Open the GitHub download website https://github.com/airockchip/rknn-toolkit2, click on the v2.0.0-bata0 in the Releasess column on the right, and download it. You will find that the compressed file contains the rknn-toolkit2 conversion tool and rknpu driver files. There is also a downloaded compressed file on the development board data CD. Using the RK3588/RK3568 development board, you can directly use the rknn-toolkit2-2.0.0-beta0.zip in the A drive of the development board CD - Basic Data → 01_codes → 01_AI_Examples → 03, Software and Drivers.

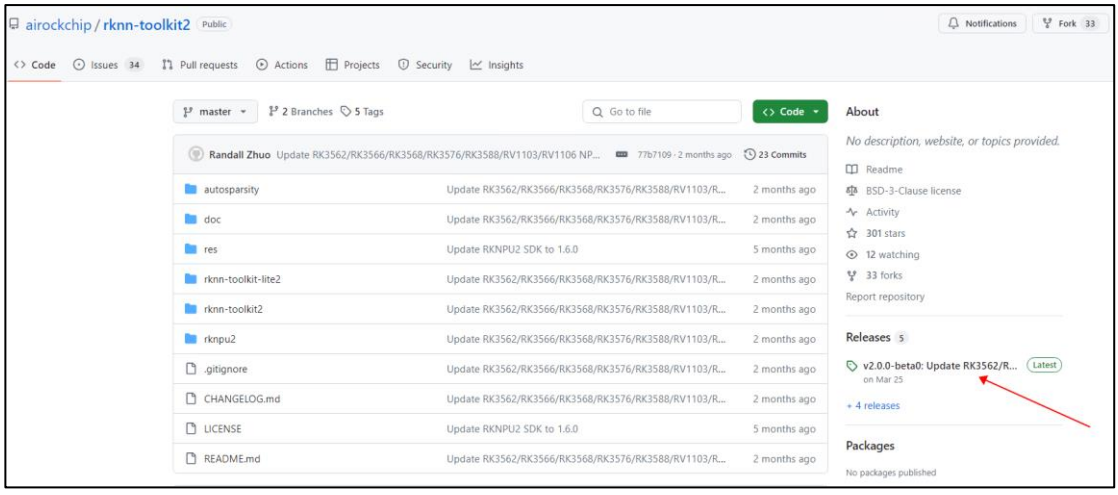


Figure 2.1.1.2 The GitHub official website of the rknn-toolkit2 tool

Click on "v2.0.0-beta0" under "Source code (zip)" to download the compressed file.

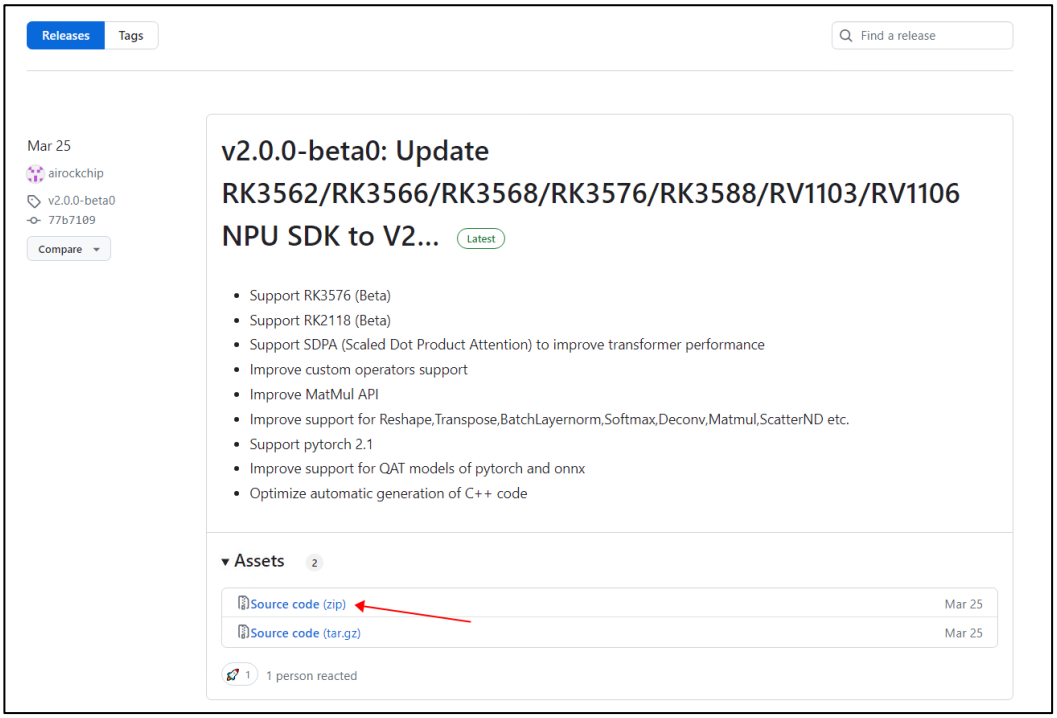


Figure 2.1.1.3 Download the compressed package of the rknn-toolkit2 tool.

After downloading, copy and decompress the compressed package to the "software" directory under Ubuntu. We can refer to the "doc/rknn_server_proxy.md" file in the "rknn-toolkit2-2.0.0-bata0" folder to update the driver files. As shown in the following picture.

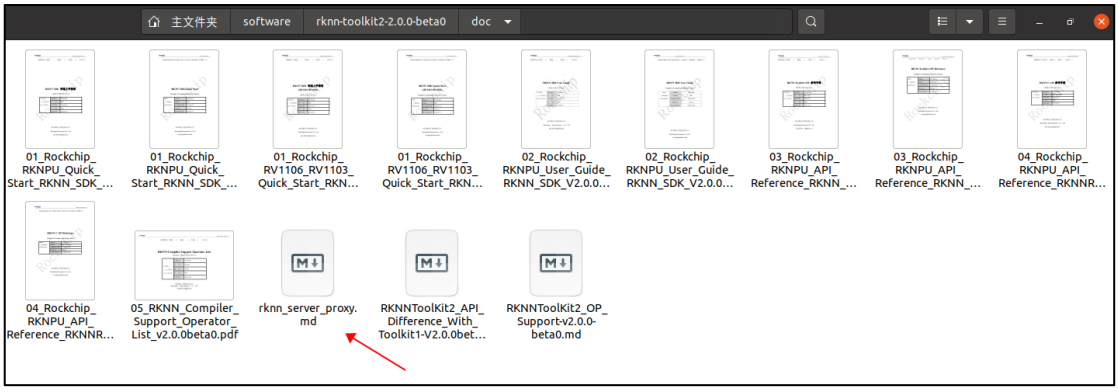


Figure 2.1.1.4 rknn_server_proxy.md file

2.2 Update the inference runtime library for the NPU board

To update the NPU board inference runtime library, we need to transfer the librknnrt.so file in the runtime directory of the downloaded rknpu2 folder, as well as the rknn_server file and scripts to the development board. The main files in these two directories are as follows.

One is the directory rknpu2/runtime/Linux/librknn_api/aarch64/.

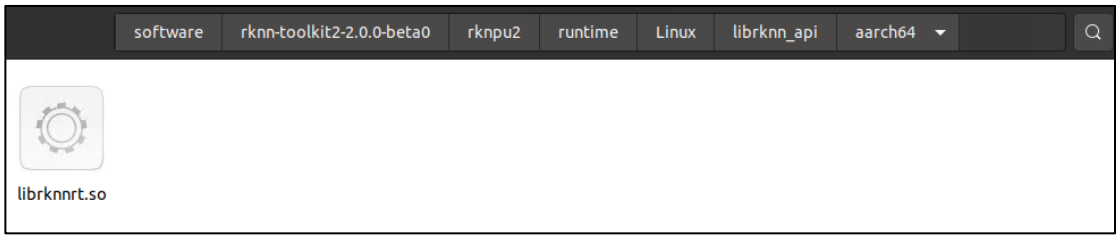


Figure 2.2.1.1 The librknnrt.so file

The other one is in the directory: rknpu2/runtime/Linux/rknn_server/aarch64/usr/bin/

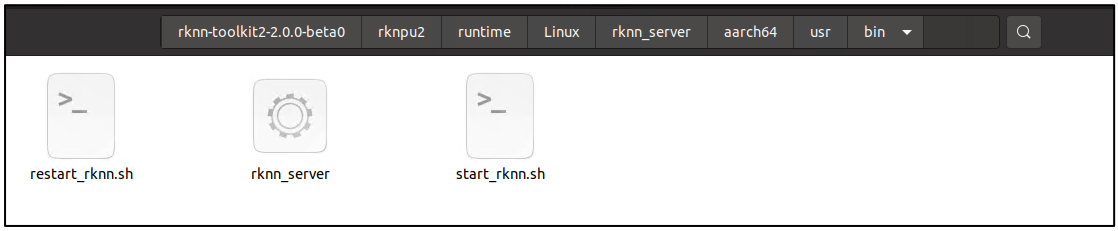


Figure 2.2.1.2 The rknn_server file and related scripts

Before updating the NPU driver service, you need to first connect the power supply of the development board and connect the development board's OTG via a Type-C cable to the computer. Then, enter the rknpu2 driver directory, open the terminal in the NPU driver directory, and use adb for data transfer.

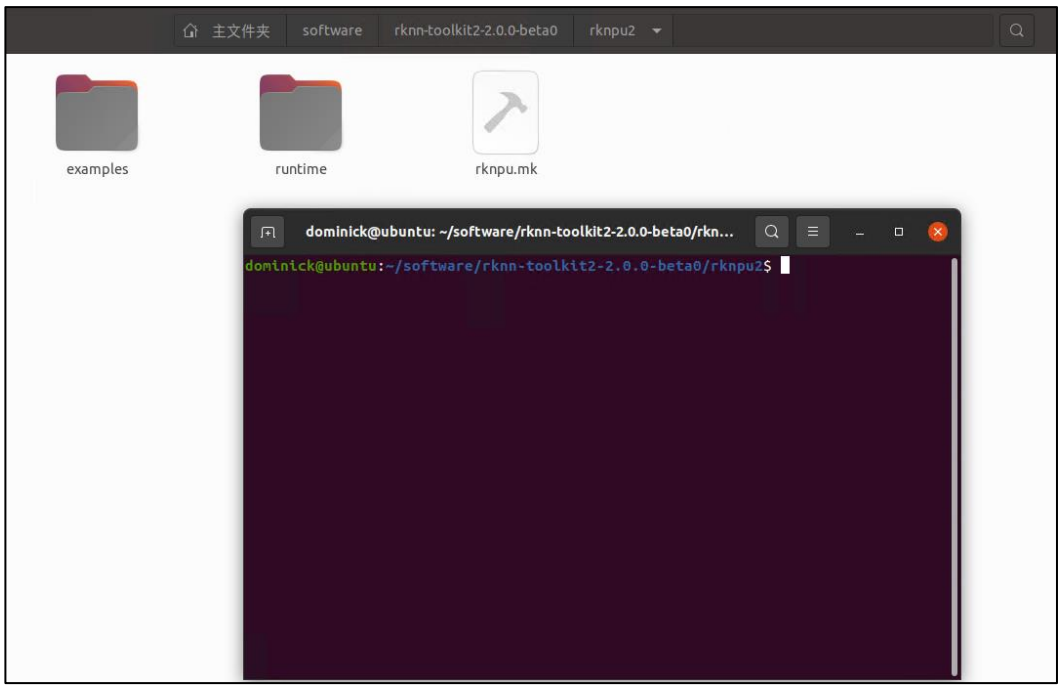


Figure 2.2.1.3 Open the terminal in the rknpu2 directory.

First, check if adb is installed. If not, you can execute the following command to install it.

```
sudo apt install adb
```

Execute the following commands one by one in the Ubuntu terminal to transfer the file to the development board.

```
adb push runtime/Linux/librknn_api/aarch64/librknnrt.so /usr/lib
```

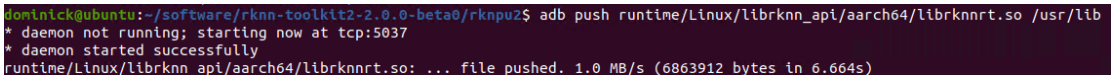


Figure 2.2.1.4 Transfer the RKNN inference library via ADB to the development board.

```
adb push runtime/Linux/rknn_server/aarch64/usr/bin/* /usr/bin
```

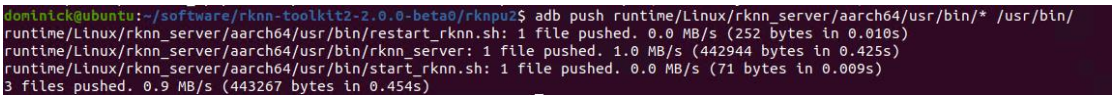


Figure 2.2.1.5 Transfer the rknn_server and scripts to the development board via adb.

Enter the shell environment of the development board in the Ubuntu terminal, grant execution permissions, and restart the service.

```
adb shell
chmod +x /usr/bin/rknn_server
chmod +x /usr/bin/start_rknn.sh
chmod +x /usr/bin/restart_rknn.sh
restart_rknn.sh
```

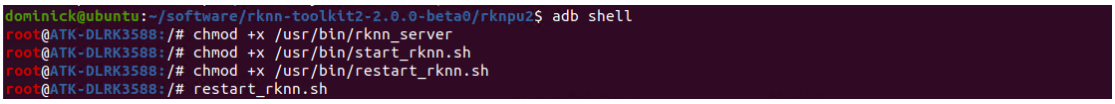


Figure 2.2.1.6 Enter the shell command line of the development board.

If you execute the restart_rknn.sh script on the development board and restart the service, the version number will be printed.

Figure 2.2.1.7 Check the version number of rknn

Execute the following command to query the version of rknn_server, and it will print the version of rknn_server.

```
strings /usr/bin/rknn_server | grep build
```

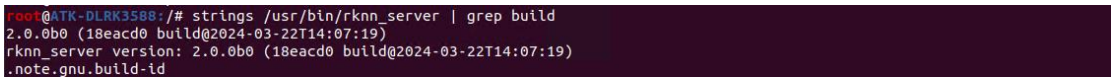


Figure 2.2.1.8 Check the version of the rknn_server service

Execute the following command to query the version of librknnrt.so, and it will print the version of librknnrt.

```
strings /usr/lib/librknnrt.so | grep version
```

```
root@ATK-DLRK3588:/# strings /usr/lib/librknnrt.so | grep version
librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
rknn_set_input_shape is deprecated next version! please call rknn_set_input_shapes()
rknn_query, info_len(%d) < sizeof(rknn_sdk_version)(%d)!
RKNN model version is
, but current librknnrt.so is support model version <=
please try updating to the latest version of the toolkit2 and runtime from: https://console.zbox.filez.com/l/I00fc3 (PWD: rknn)
RKNN Model Information, version: %d, toolkit version: %s, target: %s, target platform: %s, framework name: %s, framework layout
: %s, model inference type: %s
RKNN Model version: %d.%d.%d not match with rknn runtime version: %d.%d.%d
version
(compiler version:
Current driver version: %d.%d.%d, recommend to upgrade the driver to the new version: >= %d.%d.%d
This shared library is for RK356X/RK3588/RV1103/RV1106/RK3576/RK2118 platforms only, hw_version = %d
RKNN Driver Information, version: %d.%d.%d
Mismatch driver version, %s requires driver version >= %d.%d.%d, but you have driver version: %d.%d.%d which is incompatible!
Illegal core num, rknn model version illegal
wrong version
failed to check rknpu hardware version: %#x
Unsupport LayerNormalization! Please lower the OPSET version of the onnx model to below 16.
Do not support OpenCL version:
Parse device version[
.gnu.version
.gnu.version_r
```

Figure 2.2.1.9 Check the runtime version of librknnrt

Input "exit" to exit the development board system and return to the Ubuntu terminal. The method for updating the library on RK3568 is the same.

Chapter 3. Install the rknn-toolkit2 conversion environment

As the application of deep learning becomes increasingly widespread in various fields, it has become increasingly important for embedded and edge computing devices to be able to process deep learning tasks in real time and efficiently. However, due to the hardware limitations of these devices (such as computing power, memory size, etc.), directly running traditional deep learning models on them is often impractical.

The emergence of rknn-toolkit2 has solved this problem. It provides a complete solution that allows developers to convert already trained deep learning models (such as traditional TensorFlow, Pytorch, onnx format models) into optimized rknn format. These format models can fully utilize the hardware characteristics of devices like RK3568, RK3588, etc., thereby ensuring the model performance while reducing the demand for computing resources.

This chapter will be divided into the following sections:

1. Create a new conda environment
2. Install the rknn-toolkit2 tool

3.1 Create a new Conda environment

Previously, we downloaded Anaconda. To prevent confusion in the subsequent compilation environment, this comes in handy here. First, we create a new Conda virtual environment and then install the rknn-toolkit2. Open the terminal and execute the following command: Create a new Conda environment named "python3.8-tk2-2.0" with a Python version of Python 3.8.

```
conda create --name python3.8-tk2-2.0 python=3.8
```

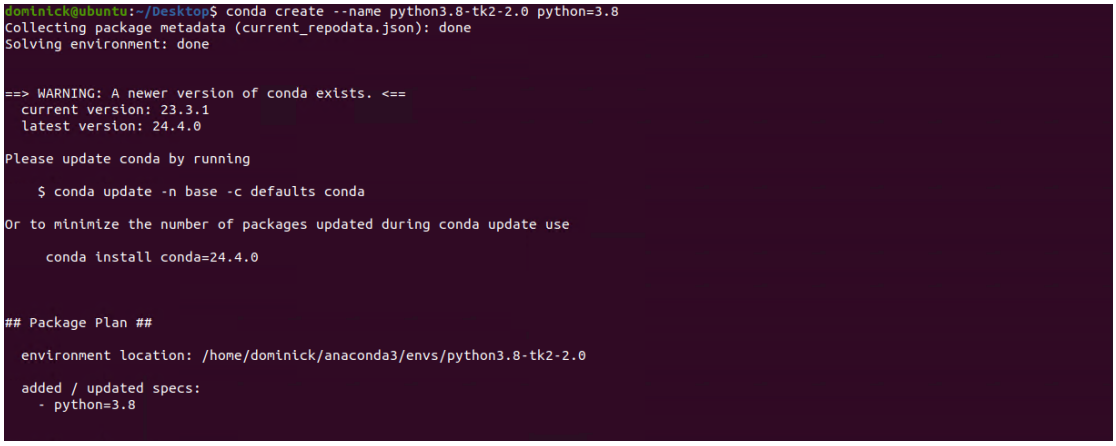


Figure 3.1.1.1 Create a Conda environment

The program stops at "Proceed". Press "y" and then press "Enter".

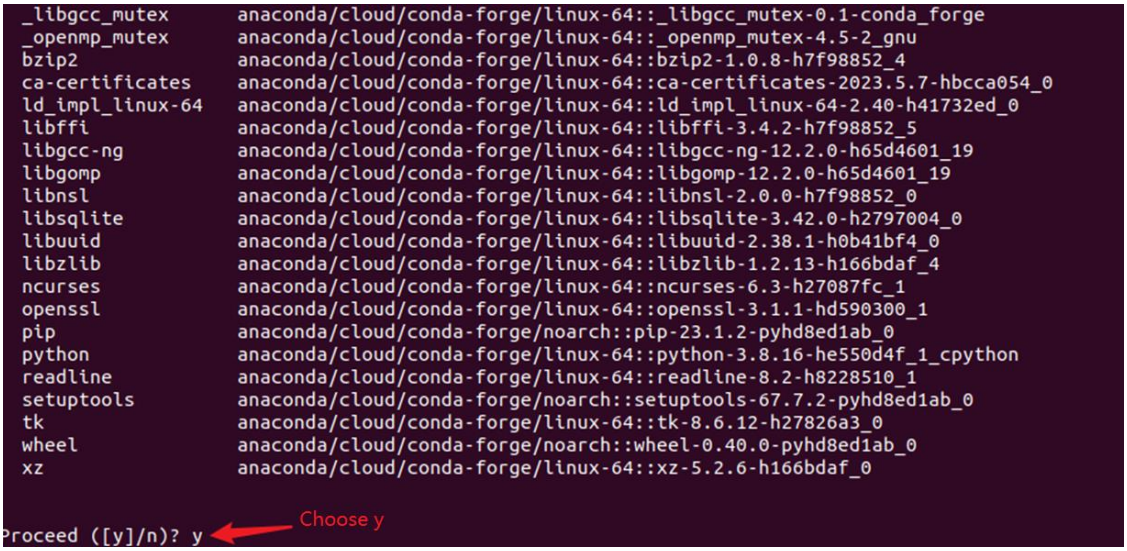


Figure 3.1.1.2 Select "y" and press Enter

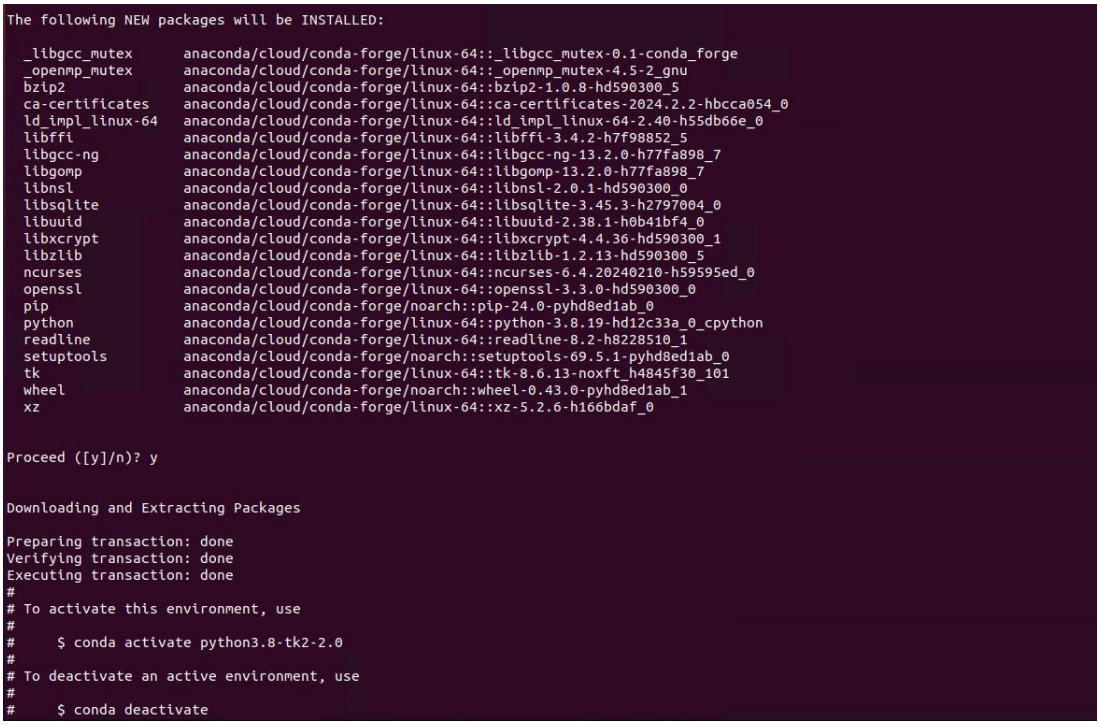


Figure 3.1.1.3 The conda environment has been successfully created.

At this point, the conda environment has been successfully set up. You can enter the conda environment by entering the following command in the command line terminal.

```
conda activate python3.8-tk2-2.0
```

To exit the conda environment, use the following command in the conda environment.

```
conda deactivate
```

3.2 Install the rknn-toolkit2 tool

3.2.1 Install the dependencies for rknn-toolkit2

Enter the rknn-toolkit2-2.0.0-bata0/rknn-toolkit2/packages folder, open the terminal and execute the following command to activate the conda environment.

```
conda activate python3.8-tk2-2.0
```

If the terminal command line shows "(python3.8-tk2-2.0)", it indicates that the newly created conda environment named "py3.8" has been successfully entered.

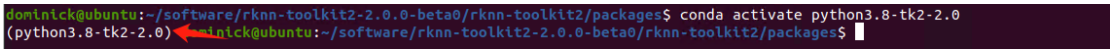


Figure 3.2.1.1 Enter the Conda environment

Before installing the rknn-toolkit2, it is necessary to install the related dependent libraries first. Before executing the following commands, it is necessary to check whether you have entered the rknn-toolkit2/packages folder. If not, you need to enter the packages folder first.

```
pip install -r requirements_cp38-2.0.0b0.txt -i https://mirror.baidu.com/pypi/simple
```

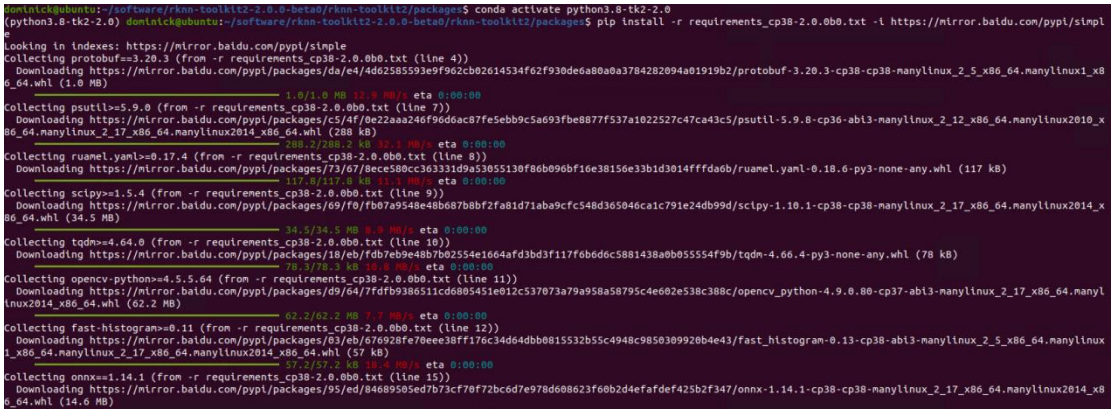


Figure 3.2.1.2 Dependencies required for installing rknn-toolkit2

The -i at the end of the parameters, such as "https://mirror.baidu.com/pypi/simple", means specifying the source as the pip mirror source as https://mirror.baidu.com/pypi/simple. By opening requirements_cp38-2.0.0b0.txt, you can see that this mirror source is the recommended download source by Rockchip Microelectronics. Therefore, we will try to perform the download and installation operation according to the official recommended source.

Figure 3.2.1.3 Installation source for the required libraries of rknn-toolkit2

When the following message "Successfully installed" is displayed, it indicates that the related dependent libraries have been installed successfully.

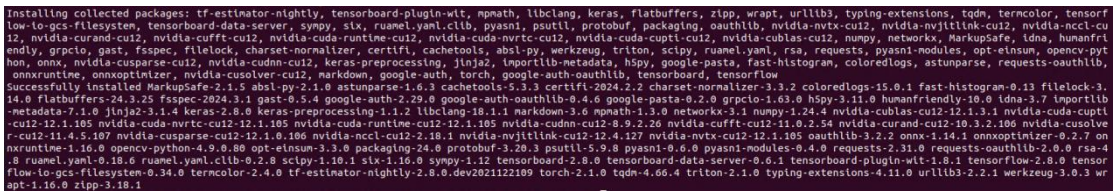


Figure 3.2.1.4 The required libraries for rknn-toolkit2 have been installed successfully.

3.2.2 Installation of the rknn-toolkit2 tool

Stay in the conda environment py3.8 and enter the rknn-toolkit2/packages directory. Execute the following commands.

```
pip install rknn_toolkit2-2.0.0b0+9bab5682-cp38-cp38-linux_x86_64.whl
```

After the installation is completed, a red box will appear in the last line, indicating "Successfully installed rknn-toolkit2-2.0.0+9bab5682", which means the installation was successful.

Figure 3.2.2.1 The required libraries for rknn-toolkit2 have been installed successfully.

3.2.3 Testing rknn-toolkit2

We enter the routine directory of examples/tflite/mobilenet_v1 to test whether the mobilenet_v1 routine works properly. If it can be converted and inferred normally, then it indicates that the rknn-toolkit2 installation is successful. Execute the following command.

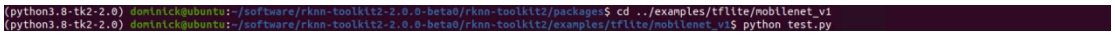


Figure 3.2.3.1 Check if the rknn-toolkit2 has been installed successfully

```
cd ../examples/tflite/mobilenet_v1
python test.py1
```

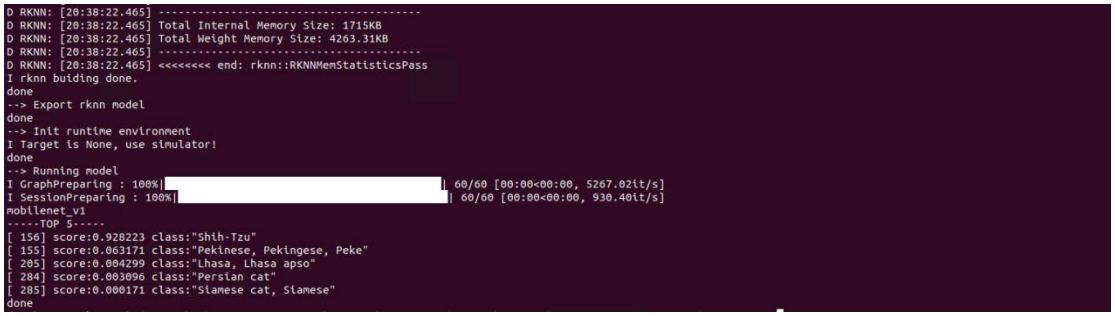


Figure 3.2.3.2 Run the conversion routine for the mobilenet_v1 model.

It can print out the top-5 classification results of the model, and the test is normal.

Chapter 4. Check and adjust the CPU and NPU frequencies

Pulse frequency is used to automatically adjust the working frequency based on system requirements and load conditions. This adjustment can be switched between different frequency modes to balance performance and energy efficiency. To facilitate everyone's better utilization of the development boards of rk3568/rk3588 to achieve the best performance on the Linux platform, a series of commands and tools for checking and setting the pulse frequency and NPU occupancy rate are provided. Next, we will introduce the methods for checking and setting the pulse frequency and NPU frequency and NPU pulse frequency for the CPU and NPU respectively.

This chapter will be divided into the following sections:

1. Check if debugfs is mounted
2. Check and fix the CPU frequency
3. Check the NPU occupancy rate
4. Check the NPU frequency and NPU pulse frequency
5. Quick pulse frequency setting

4.1 Check if debugfs is mounted

Before checking the CPU/GPU/NPU frequencies and usage rates, it is necessary to check if debugfs is mounted. You can use the following command to check, and you will see that the ALIENTEK factory system defaults to mounting debugfs.

```
mount | grep debug
```

```
root@ATK-DLRK3588:/# mount | grep debug
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
```

Figure 4.1.1.1 Mount debugfs

The command for viewing and mounting debugfs on rk3568 is the same.

4.2 View and fix CPU frequency

4.2.1 View the current running frequency of the CPU

View the running frequency of each CPU core of the current CPU on the serial terminal of rk3588.

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu5/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu6/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu7/cpufreq/scaling_cur_freq
```

```
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
408000
```

Figure 4.2.1.1 Check the current running frequency of the CPU of rk3588

On the serial terminal of rk3568, check the running frequency of each CPU core of the current CPU.

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq
```

```
root@ATK-DLRK3568:/# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1416000
```

Figure 4.2.1.2 Check the current running frequency of the CPU of rk3568

4.2.2 Fixed CPU Frequency

1. View the available frequency of the CPU

Execute the following command in the serial terminal of the rk3568 development board to view the available frequency of the CPU.

```
cat /sys/devices/system/cpu/cpufreq/policy0/scaling_available_frequencies
```

408000 600000 816000 1104000 1416000 1608000 1800000 1992000

If it is rk3588, then execute the following command in the serial terminal to check the available CPU frequency.

```
cat /sys/devices/system/cpu/cpufreq/policy0/cpuinfo_cur_freq
cat /sys/devices/system/cpu/cpufreq/policy4/cpuinfo_cur_freq
cat /sys/devices/system/cpu/cpufreq/policy6/cpuinfo_cur_freq
```

```
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpufreq/policy0/scaling_available_frequencies
408000 600000 816000 1008000 1200000 1416000 1608000 1800000
```

Figure 4.2.2.1 Check the available CPU frequency

2. Set the CPU frequency to performance mode

If it is rk3588, you can set the CPU to performance mode by using the following command.

```
echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
echo performance > /sys/devices/system/cpu/cpufreq/policy4/scaling_governor
echo performance > /sys/devices/system/cpu/cpufreq/policy6/scaling_governor
```

If it is the RK3568, you can set the CPU to the performance mode by using the following command.

```
echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

To view the CPU frequency, use the following command. Here, policy0 represents the option that can control and display the frequencies of CPU0 to CPU3, policy4 represents the option that can control and display the frequencies of CPU4 and CPU5, and policy6 represents the option that can control and display the frequencies of CPU6 and CPU7. Among them, rk3568 only has policy0, while rk3588 has policy0, policy4, and policy6.

```
cat /sys/devices/system/cpu/cpufreq/policy0/cpuinfo_cur_freq
cat /sys/devices/system/cpu/cpufreq/policy4/cpuinfo_cur_freq
cat /sys/devices/system/cpu/cpufreq/policy6/cpuinfo_cur_freq
```

After the settings are completed, execute the following command on the RK3588 terminal to check the current CPU frequencies for each core.

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu5/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu6/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu7/cpufreq/scaling_cur_freq
```

```
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1800000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
1800000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq
1800000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq
1800000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_cur_freq
2352000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu5/cpufreq/scaling_cur_freq
2352000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu6/cpufreq/scaling_cur_freq
2256000
root@ATK-DLRK3588:/# cat /sys/devices/system/cpu/cpu7/cpufreq/scaling_cur_freq
2256000
```

Figure 4.2.2.2 Check the current CPU running frequency of rk3588

As shown in the above figure, each CPU core has been adjusted to the highest running frequency supported by the CPU. If it is rk3568, execute the following command.

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
```

cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq

cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq

```
root@ATK-DLRK3568:/# echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
root@ATK-DLRK3568:/# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1992000
root@ATK-DLRK3568:/# cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq
1992000
root@ATK-DLRK3568:/# cat /sys/devices/system/cpu/cpu2/cpufreq/scaling_cur_freq
1992000
root@ATK-DLRK3568:/# cat /sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq
1992000
```

Figure 4.2.2.3 Check the current CPU running frequency of rk3568

4.3 Check the NPU occupancy rate

During the operation of the NPU, you can use the following command to check the current occupancy rate of the NPU. The rk3588 has three NPU cores, and it will print the usage rates of the three NPU cores.

cat /sys/kernel/debug/rknpu/load

```
root@ATK-DLRK3588:/# cat /sys/kernel/debug/rknpu/load
NPU load:  Core0:  0%, Core1:  0%, Core2:  0%,
```

Figure 4.3.1.1 Check the NPU occupancy rate of rk3588

rk3568 has only one NPU core, and it will only print the occupancy rate of that single NPU core.

```
root@ATK-DLRK3568:/# cat /sys/kernel/debug/rknpu/load
NPU load:  0%
```

Figure 4.3.1.2 Check the NPU occupancy rate of rk3568

4.4 Check NPU frequency and fixed frequency setting

4.4.1 View NPU frequency

On rk3588, view the current NPU frequency. The 250000000 under clk_npu_dsu0 is the current NPU frequency.

cat /sys/kernel/debug/clk/clk_summary | grep clk_npu

```
root@ATK-DLRK3588:/# cat /sys/kernel/debug/clk/clk_summary | grep clk_npu
 scmi_clk_npu                      0        3        0   200000000          0     0  50000
    tclk_npu_wdt                   0        0        0    24000000          0     0  50000
    clk_nputimer_root              0        0        0    24000000          0     0  50000
       clk_nputimer1               0        0        0    24000000          0     0  50000
       clk_nputimer0               0        0        0    24000000          0     0  50000
    clk_npu_pvtm                   0        0        0    24000000          0     0  50000
            hclk_npu_cm0_root      0        1        0   396000000          0     0  50000
               fclk_npu_cm0_core       0        0        0   396000000          0     0  50000
            hclk_npu_root          0        6        0   198000000          0     0  50000
               hclk_npu0           0        4        0   198000000          0     0  50000
               hclk_npu2           0        4        0   198000000          0     0  50000
               hclk_npu1           0        4        0   198000000          0     0  50000
         clk_npu_dsu0              0        6        0   250000000          0     0  50000
            aclk_npu0              0        4        0   250000000          0     0  50000
            aclk_npu2              0        4        0   250000000          0     0  50000
            aclk_npu1              0        4        0   250000000          0     0  50000
            pclk_npu_root          0        6        0   100000000          0     0  50000
               pclk_npu_wdt        0        0        0   100000000          0     0  50000
               pclk_npu_timer      0        0        0   100000000          0     0  50000
               pclk_npu_grf        0        0        0   100000000          0     0  50000
               pclk_npu_pvtm       0        0        0   100000000          0     0  50000
    clk_npu_cm0_rtc                0        0        0       32768          0     0  50000
```

Figure 4.4.1.1 Check the frequency of the NPU of rk3588

On rk3568, check the current frequency of the NPU. The 200000000 under clk_npu_dsu0 is the current NPU frequency.

cat /sys/kernel/debug/clk/clk_summary | grep clk_npu

```
root@ATK-DLRK3568:/# cat /sys/kernel/debug/clk/clk_summary | grep clk_npu
    clk_npu_pvtm                    0        0        0    24000000          0     0  50000
       clk_npu_np5                  0        0        0   342857143          0     0  50000
       clk_npu_src                  0        1        0   200000000          0     0  50000
          clk_npu_pre_ndft          0        1        0   200000000          0     0  50000
             clk_npu_pvtpll         0        0        0   200000000          0     0  50000
             clk_npu_pvtm_core          0        0        0   200000000          0     0  50000
             clk_npu                0        4        0   200000000          0     0  50000
                aclk_npu_pre        0        2        0   200000000          0     0  50000
                   aclk_npu         0        3        0   200000000          0     0  50000
                pclk_npu_pre        0        1        0    33333334          0     0  50000
                   pclk_npu_pvtm          0        0        0    33333334          0     0  50000
                hclk_npu_pre        0        2        0    50000000          0     0  50000
                   hclk_npu         0        3        0    50000000          0     0  50000
```

Figure 4.4.1.2 Check the frequency of the NPU of rk3568

4.4.2 Fixed-frequency of NPU

1. Check the available frequency of the NPU

Check the available frequency of the NPU on rk3588.

```
cat /sys/class/devfreq/fdab0000.npu/available_frequencies
```

300000000 400000000 500000000 600000000 700000000 800000000 900000000 1000000000

```
root@ATK-DLRK3588:/# cat /sys/class/devfreq/fdab0000.npu/available_frequencies
300000000 400000000 500000000 600000000 700000000 800000000 900000000 1000000000
```

Figure 4.4.2.1 Check the available frequency of the NPU on the rk3588

Check the available frequency of the NPU on the rk3568.

```
cat /sys/class/devfreq/fde40000.npu/available_frequencies
```

```
root@ATK-DLRK3568:/# cat /sys/class/devfreq/fde40000.npu/available_frequencies
200000000 297000000 400000000 600000000 700000000 800000000 900000000
```

Figure 4.4.2.2 View the available frequency of the NPU on rk3568

2. Set the frequency of the NPU

On the rk3588 development board, before setting the frequency of the NPU, it is necessary to first turn on the power supply.

```
echo userspace > /sys/class/devfreq/fdab0000.npu/governor
echo 1000000000 > /sys/class/devfreq/fdab0000.npu/userspace/set_freq
```

```
root@ATK-DLRK3588:/# echo userspace > /sys/class/devfreq/fdab0000.npu/governor
root@ATK-DLRK3588:/# echo 1000000000 > /sys/class/devfreq/fdab0000.npu/userspace/set_freq
[ 2914.041724] RKNPU fdab0000.npu: RKNPU: set rknpu freq: 1000000000, volt: 812500
root@ATK-DLRK3588:/# cat /sys/class/devfreq/fdab0000.npu/cur_freq
1000000000
```

Figure 4.4.2.3 Set the frequency of the NPU to the maximum.

After the setting is completed, you can check the current frequency of the NPU.

```
cat /sys/class/devfreq/fdab0000.npu/cur_freq
```

```
root@ATK-DLRK3588:/# cat /sys/class/devfreq/fdab0000.npu/cur_freq
1000000000
```

Figure 4.4.2.4 Check the current frequency of the NPU.

If you are using the RK3568 development board, execute the following command.

```
echo userspace > /sys/class/devfreq/fde40000.npu/governor
echo 900000000 > /sys/class/devfreq/fde40000.npu/userspace/set_freq
```

```
root@ATK-DLRK3568:/# echo userspace > /sys/class/devfreq/fde40000.npu/governor
root@ATK-DLRK3568:/# echo 900000000 > /sys/class/devfreq/fde40000.npu/userspace/set_freq
[ 1471.079306] RKNPU fde40000.npu: RKNPU: set rknpu freq: 900000000, volt: 925000
```

Figure 4.4.2.5 Set the frequency of the NPU to the maximum.

After the settings are completed, you can view the current frequency of the NPU.

```
cat /sys/class/devfreq/fde40000.npu/cur_freq
```

```
root@ATK-DLRK3568:/# cat /sys/class/devfreq/fde40000.npu/cur_freq
900000000
```

Figure 4.4.2.6 Check the current NPU frequency

4.5 Quick Frequency Setting

To facilitate everyone to quickly start the development process and achieve better results after getting the board, the development board materials provide a frequency setting script. For the RK3588 development board, the path is the A disk of the development board CD-ROM - Basic Materials → 01_codes → 01, AI Examples → 01, Source Code → 00, Frequency Setting Script → scaling_frequency.sh. Using SSH or ADB, push it to any path on the development board. Here, we push it to the /userdata directory of the development board. Open the serial port terminal on the board and enter the directory where the script is located to execute the following commands.

```
cd /userdata
chmod +x scaling_frequency.sh
./scaling_frequency.sh -c rk3588
```

The execution result is shown in the following figure. It can be seen that the CPU, DDR frequency and NPU frequency of rk3588 have been successfully set to the highest performance level.

```
root@ATK-DLRK3588:/userdata# chmod +x scaling_frequency.sh
root@ATK-DLRK3588:/userdata# ./scaling_frequency.sh -c rk3588
Try seting frequency for rk3588
    Setting strategy as 4
    NPU seting to 1000000000
    CPU seting to 2256000
    DDR seting to 2112000000
CPU: seting frequency
  Core0
    try set 1800000
    query   1800000
    Seting Success
  Core4
    try set 2256000
    query   2352000
    Seting Failed
  Core6
    try set 2256000
    query   2256000
    Seting Success
NPU: seting frequency
[   63.770834] RKNPU fdab0000.npu: RKNPU: set rknpu freq: 1000000000, volt: 812500
Firmware seems not support seting NPU frequency
    wanted 1000000000
    check  1000000000
DDR: seting frequency
    try set 2112000000
    query   2112000000
    Seting Success
root@ATK-DLRK3588:/userdata#
```

Figure 4.5.1.1 Execute the fixed-frequency script using rk3588

If it is a rk3568 development board, the path is the A disk of the development board's CD-ROM - Basic Data → 01_codes → 01, AI Routine → 04, Linux 5.10 AI Routine Source Code → 00, Fixed Frequency Script → scaling_frequency.sh. Push it to any path on the development board via SSH or adb. Here, we will push it to the /userdata directory of the development board. Open the serial port terminal on the board and enter the directory where the script is located to execute the following commands. Execute the following commands.

```
cd /userdata
chmod +x scaling_frequency.sh
./scaling_frequency.sh -c rk3588
```

The execution result is shown in the following figure. It can be seen that the CPU, DDR frequency and NPU frequency of rk3568 have been successfully set to the highest performance level.

```
root@ATK-DLRK3568:/userdata# chmod +x scaling_frequency.sh
root@ATK-DLRK3568:/userdata# ./scaling_frequency.sh -c rk3568
Try seting frequency for rk3568
    Setting strategy as 3
    NPU seting to 1000000000
    CPU seting to 1800000
    DDR seting to 1560000000
CPU: seting frequency
    try set 1800000
    query   1800000
    Seting Success
NPU: seting frequency
[  130.620077] RKNPU fde40000.npu: RKNPU: set rknpu freq: 900000000, volt: 925000
    try set 1000000000
    query   900000000
    Seting Failed
DDR: seting frequency
    try set 1560000000
    query   1560000000
    Seting Success
```

Figure 4.5.1.2 Executing the fixed-frequency script with rk3568

Chapter 5. Test the Python inference routine under buildroot

The root file system of the factory system of the Linux system of ALIENTEK rk3568/rk3588 is built by buildroot, so the support for Python is not so good. However, ALIENTEK has adapted the main related libraries of Python inference, so some of the provided routines can be executed for inference through Python. Now, let's introduce how to make our Python routines run.

This chapter will be divided into the following sections:

1. Install RKNN Toolkit Lite2
2. Test the AI routines under Python
3. Test the self-trained lenet model under Python

5.1 Install RKNN Toolkit Lite2

RKNN Toolkit Lite2 provides Python programming interfaces for the Rockchip NPU platform, helping users deploy RKNN models and accelerate the implementation of AI applications. Next, we will explain how to install the rknn-toolkit-lite2 environment on the board.

Open the serial terminal, enter the Python command and press Enter to find that the pre-installed Python version on the rk3568/rk3588 development board is 3.10.5. In the development board terminal, enter exit() and press Enter to exit.

```
root@ATK-DLRK3588:/# python
Python 3.10.5 (main, May  6 2024, 18:40:58) [GCC 10.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 5.1.1.1 Check the Python version

Open the rknn-toolkit2-2.0.0 folder that was extracted in Chapter 2, and go to the rknn_toolkit_lite2/packages directory within it. Open the Ubuntu terminal in the current directory and connect the development board to the OTG interface. Then, enter the following command to transfer the whl file to the userdata directory on the development board.

```
adb push rknn_toolkit_lite2-2.0.0b0-cp310-cp310-linux_aarch64.whl /userdata
```

```
dominick@ubuntu:~/software/rknn-toolkit2-2.0.0-beta0/rknn-toolkit-lite2/packages$ adb push rknn_toolkit_lite2-2.0.0
b0-cp310-cp310-linux_aarch64.whl /userdata
dominick@ubuntu:~/software/rknn-toolkit2-2.0.0-beta0/rknn-toolkit-lite2/packages$
```

Figure 5.1.1.2 Push the rknn_toolkit_lite to the development board.

After the transmission is completed, open the serial terminal or SSH terminal of the development board, enter the directory userdata, and input the following commands to install.

```
cd /userdata
pip install rknn_toolkit_lite2-2.0.0b0-cp310-cp310-linux_aarch64.whl
```

After installation, it will be as shown in the following picture.

```
root@ATK-DLRK3588:/# cd /userdata/
root@ATK-DLRK3588:/userdata# pip install rknn_toolkit_lite2-2.0.0b0-cp310-cp310-linux_aarch64.whl
Processing ./rknn_toolkit_lite2-2.0.0b0-cp310-cp310-linux_aarch64.whl
Requirement already satisfied: psutil in /usr/lib/python3.10/site-packages (from rknn-toolkit-lite2==2.0.0b0) (5.8.0)
Requirement already satisfied: numpy in /usr/lib/python3.10/site-packages (from rknn-toolkit-lite2==2.0.0b0) (1.21.2)
Requirement already satisfied: ruamel.yaml in /usr/lib/python3.10/site-packages (from rknn-toolkit-lite2==2.0.0b0) (0.17.31)
Requirement already satisfied: ruamel.yaml.clib>=0.2.7 in /usr/lib/python3.10/site-packages (from ruamel.yaml->rknn-toolkit-lite2==2.0.0b
0) (0.2.7)
Installing collected packages: rknn-toolkit-lite2
Successfully installed rknn-toolkit-lite2-2.0.0b0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is
 recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

Figure 5.1.1.3 Use the pip command to install rknn_toolkit_lite

5.2 Test the AI routine in Python

In the previous section, we have installed RKNN Toolkit Lite 2. In this section, we will verify whether the environment we installed can work properly. Also, in the rknn_toolkit_lite2 directory under Ubuntu, open the terminal and input the following command to transfer the resnet18 routine in the examples directory to the userdata directory of the development board via adb.

```
adb push examples/resnet18/ /userdata
```

```
dominick@ubuntu:~/software/rknn-toolkit2-2.0.0-beta0/rknn-toolkit-lite2$ adb push examples/resnet18/ /userdata
examples/resnet18/: 8 files pushed. 1.0 MB/s (47685637 bytes in 45.645s)
```

Figure 5.2.1.1 Push ResNet to the development board.

After the transmission is complete, open the serial port terminal of the development board, enter the /userdata/resnet18/ directory and input the following command to execute to see the effect.

```
cd /userdata/resnet18/
```

```
python test.py
```

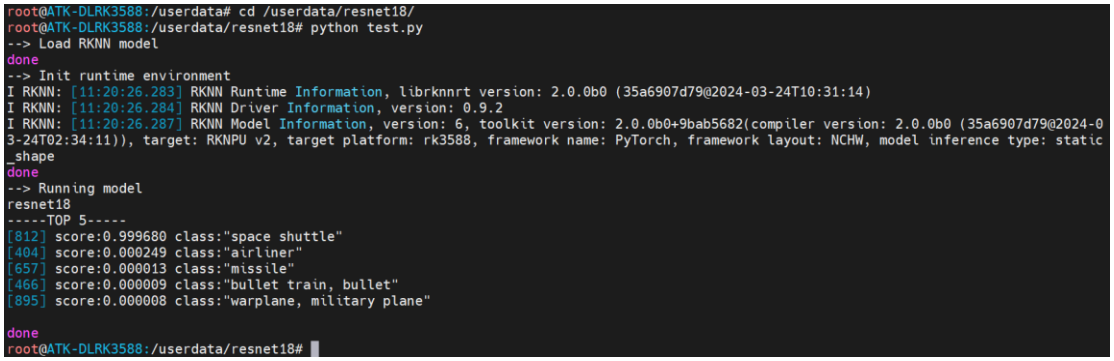


Figure 5.2.1.2 Execute the inference program of the ResNet model

This routine uses the "space_shuttle_224.jpg" provided in the routine for inference. The model is a ResNet model converted from the official RKNN model of Rockchip Microelectronics through the ImageNet dataset. Finally, you can see the printed indices of the five categories with the highest confidence levels. 812 represents the space shuttle. Regarding the category indices of the ImageNet dataset, you can search for them online. No further introduction will be provided here.

5.3 Testing of the self-trained LeNet model in Python

In the previous section, we installed the board-end Python inference rknn_toolkit_lite2 tool. Next, we will test the AI model routine written in Python by ALIENTEK. The first and foremost thing to do is our essential model for deep learning, LeNet. It is one of the classic convolutional neural network (CNN) architectures in the field of deep learning, proposed by Yann LeCun et al. in 1998. It is one of the pioneers of convolutional neural networks and is widely regarded as the founding work of deep learning. We trained a five-layer classic network model of LENET using the handwritten digit recognition (mnist) dataset and could view its model structure through the Netron tool.

LeNet was originally designed for handwritten digit recognition, especially for the automatic recognition task of US postal codes. Its design concept was revolutionary at that time, introducing convolutional layers and sub-sampling (pooling) layers, which decomposed the input image into multiple layers of feature representations, effectively extracting the local features of the image. Next, we will introduce the steps for running this routine on the board-end. If using the rk3588 development board, first copy the files 01_lenet from the A disk of the development board's CD-ROM - Basic Materials → 01_codes → 01, AI Routine → 01, Source Code → 01_lenet folder to the development board using adb or ssh methods. It is recommended to copy all the routines to Ubuntu first. Here, we still use the adb method for transmission. If using the RK3568 development board, the files are in the A disk of the development board's CD-ROM - Basic Materials → 01_codes → 01, AI Routine → 04, Linux5_10 AI Routine Source Code → 01_lenet folder. All other operations are the same.

We first create a folder named aidemo in the /userdata directory of the development board to facilitate the unified placement of all routines for testing. Then transfer the 01_lenet routine there. On the development board, enter the following command.

```
cd /userdata
mkdir aidemo
```

```
root@ATK-DLRK3588:/userdata/resnet18# cd /userdata/
root@ATK-DLRK3588:/userdata# mkdir aidemo
```

Figure 5.3.1.2 Execute the inference program of the ResNet model.

Then go to the directory where the Ubuntu routine is located, transfer the routine to the development board via adb, and open the terminal in Ubuntu and enter the following commands.

```
adb push 01_lenet/ /userdata/aidemo
```

After the transmission is completed, return to the development board terminal and enter the corresponding directory to execute the following commands.

```
cd /userdata/aidemo/01_lenet/
python atk_lenet_demo.py
```

```
root@ATK-DLRK3588:/userdata/aidemo/01-lenet# python atk_lenet_demo.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [11:59:49.395] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [11:59:49.395] RKNN Driver Information, version: 0.9.2
I RKNN: [11:59:49.395] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-0
3-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name: TFLite, framework layout: NHWC, model inference type: static_
shape
--> Running model
LeNet
-----TOP 5-----
[5]: 0.99951171875
[0 1 2 3 4 6 7 8 9]: 6.103515625e-05
[0 1 2 3 4 6 7 8 9]: 6.103515625e-05
[0 1 2 3 4 6 7 8 9]: 6.103515625e-05
[0 1 2 3 4 6 7 8 9]: 6.103515625e-05

done
```

Figure 5.3.1.3 Execute the inference program for the self-trained LeNet model.

In the code, we input a handwritten digit image of the number 5 for inference.

Figure 5.3.1.4 The 28x28 grayscale image of the number "5"

From the output result, we can see that the probability of index 5 is 1.0. The handwritten digit dataset is a dataset of 10 categories from 0 to 9, which exactly corresponds to the data sets of 0 to 9. Thus, we can see that the model is running normally. Everyone can also try to manually write some digits and resize them to 28x28 pixels to modify the source code for reading the image for inference.

Chapter 6. modelzoo Introduction and Testing

In this chapter, we will introduce how to use the modelzoo models for testing and running on the ALIENTEK RK3588. Modelzoo is a collection of various models provided by Rockchip. The website is as follows: https://github.com/airockchip/rknn_model_zoo. The models covered in this chapter are quite numerous and are divided into the following sections.

This chapter will be divided into the following sections:

1. Introduction to modelzoo
2. Testing the MobileNet model
3. Testing the ResNet model
4. Testing the YOLOv5 model
5. Testing the YOLOv6 model
6. Test the yolov7 model
7. Test the yolov8 model
8. Test the yolox model
9. Test the ppyoloe model
10. Test the deeplabv3 model
11. Test the yolov5_seg model
12. Test the yolov8_seg model
13. Test the ppseg model
14. Test the RetinaFace model
15. Test the PPOCR model
16. Test the LPRNet model
17. Test the lite_transform model

6.1 modelzoo Introduction

The RKNN Model Zoo provides a large number of example codes for model conversion and deployment, aiming to help users quickly run various models on the development board equipped with Rockchip chips. If using the rk3588 development board, copy the compressed files from the A disk of the development board CD - Basic Data\01_codes\01, AI Routine\01, Source Code\06_modelzoo to the Ubuntu system and then decompress it. If using the rk3568 development board, copy the compressed files from the A disk of the development board CD - Basic Data\01_codes \01, AI Routine\04, Linux5_10 AI Routine Source Code\06_modelzoo to the Ubuntu system and then decompress it. The project directory is as follows:

- 3rdparty Third-party libraries
- datasets Data sets
- examples Sample codes
- utils Common methods, file operations, drawing tools, etc.
- build-android.sh Compilation script for compiling Android system development boards for the target
- build-linux.sh Compilation script for compiling Linux system development boards for the target

6.1.1 Preparation before testing

Export environment variables, enable the compiler, be aware that after reopening the terminal, you need to enable the compiler again, unless it is directly written into the startup /etc/profile file. If using the rk3588 development board, execute the following command.

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0/examples$ export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buil
droot-linux-gnu
```

Figure 6.1.1.1 Enable the compiler environment

If you are using the rk3568 development board, execute the following command.

```
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0/examples$ export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buil
droot-linux-gnu
```

Figure 6.1.1.2 Enable the compiler environment

Download the model using the model download script, move it to the "model" directory of the mobilenet routine, and execute the following command to download the model.

```
chmod +x build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ chmod +x build-linux.sh
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ls -l
总用量 104
drwxrwxr-x 10 dominick dominick  4096 Mar 24 23:05 3rdparty
drwxrwxr-x  2 dominick dominick  4096 Mar 24 23:05 asset
-rw-rw-r--  1 dominick dominick  4713 Mar 24 23:05 build-android.sh
-rwxrwxr-x  1 dominick dominick  5005 Mar 24 23:05 build-linux.sh
drwxrwxr-x  6 dominick dominick  4096 Mar 24 23:05 datasets
drwxrwxr-x  2 dominick dominick  4096 Mar 24 23:05 docs
drwxrwxr-x 18 dominick dominick  4096 Mar 24 23:05 examples
-rw-rw-r--  1 dominick dominick 11357 Mar 24 23:05 LICENSE
drwxrwxr-x  2 dominick dominick  4096 Mar 24 23:05 py_utils
-rw-rw-r--  1 dominick dominick 16023 Mar 24 23:05 README_CN.md
-rw-rw-r--  1 dominick dominick 18772 Mar 24 23:05 README.md
-rw-rw-r--  1 dominick dominick 11713 Mar 24 23:05 scaling_frequency.sh
drwxrwxr-x  2 dominick dominick  4096 Mar 24 23:05 utils
```

Figure 6.1.1.3 Grant execute permission to the compilation script

6.2 Test the mobileNet model

First, you need to download the model. Download the model by executing the model download script. Before execution, you need to check if the Ubuntu network is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd rknn_model_zoo-2.0.0/examples/mobilenet/model
sh download_model.sh
```

Figure 6.2.1.1 Download the MobileNet model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the directory of the MobileNet routine under the modelzoo.

```
cd rknn_model_zoo-2.0.0/examples/mobilenet/python
```

If you are using the rk3588 development board, execute the following command to convert the mobileNet model.

```
python3 mobilenet.py --target rk3588
```

Figure 6.2.1.2 Convert the MobileNet model to the RKNN model.

After the model conversion is completed, the running results will be displayed.

```
done
--> Init runtime environment
I Target is None, use simulator!
done
--> Running model
W inference: The 'data_format' is not set, and its default value is 'nhwc'!
I GraphPreparing : 100%|                                | 102/102 [00:00<00:00, 5158.11it/s]
I SessionPreparing : 100%|                              | 102/102 [00:00<00:00, 1278.96it/s]
--> PostProcess
-----TOP 5-----
[494] score=0.98 class="n03017168 chime, bell, gong"
[653] score=0.01 class="n03764736 milk can"
[469] score=0.00 class="n02939185 caldron, cauldron"
[505] score=0.00 class="n03063689 coffeepot"
[463] score=0.00 class="n02909870 bucket, pail"
done
```

Figure 6.2.1.3 Convert the MobileNet model to the RKNN model completed.

If you are using the RK3568 development board, execute the following command to convert the MobileNet model.

```
python3 mobilenet.py --target rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo. First, enable the compiler. If you are using the rk3588 development board, execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d mobilenet
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ./build-linux.sh -t rk3588 -a aarch64 -d mobilenet
./build-linux.sh -t rk3588 -a aarch64 -d mobilenet
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=mobilenet
BUILD_DEMO_PATH=examples/mobilenet/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/install/rk3588_linux_aarch64/rknn_mobilenet_demo
BUILD_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_mobilenet_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
```

Figure 6.2.1.4 Compile the mobilenet program

After compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_mobilenet_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_mobilenet_demo/ /userdata/aidemo
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_mobilenet_demo/ /userdat
a/aidemo
install/rk3588_linux_aarch64/rknn_mobilenet_demo/: 6 files pushed. 0.7 MB/s (12281132 bytes in 16.192s)
```

Figure 6.2.1.5 Push the MobileNet program and model to the development board.

If using the RK3568 development board, execute the following command to compile and push it to the board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
./build-linux.sh -t rk3568 -a aarch64 -d mobilenet
adb push install/rk3568_linux_aarch64/rknn_mobilenet_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_mobilenet_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bell.jpg for inference. The execution result is as shown in the figure below.

```
./rknn_mobilenet_demo model/mobilenet_v2.rknn model/bell.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_mobilenet_demo# ./rknn_mobilenet_demo model/mobilenet_v2.rknn model/bell.jpg
num_lines=1001
model input num: 1, output num: 1
input tensors:
  index=0, name=input, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-14, scale=
0.018658
output tensors:
  index=0, name=output, n_dims=2, dims=[1, 1000, 0, 0], n_elems=1000, size=1000, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-55, scale
=0.141923
model is NHWC input fmt
model input height=224, width=224, channel=3
origin size=500x333 crop size=496x320
input image: 500 x 333, subsampling: 4:4:4, colorspace: YCbCr, orientation: 1
src width=500 height=333 fmt=0x1 virAddr=0x0x55a7014b20 fd=0
dst width=224 height=224 fmt=0x1 virAddr=0x0x55a708ea60 fd=0
color=0x0
rga_api version 1.10.1_[0]
Error on improcess STATUS=-1
RGA error message: Unsupported function: src unsupport width stride 500, rgb888 width stride should be 16 aligned!
try convert image use cpu
finish
rknn_run
[494] score=0.993227 class=n03017168 chime, bell, gong
[469] score=0.002560 class=n02939185 caldron, cauldron
[747] score=0.000466 class=n04023962 punching bag, punch bag, punching ball, punchball
[792] score=0.000466 class=n04208210 shovel
[618] score=0.000405 class=n03633091 ladle
```

Figure 6.2.1.6 The program reasoning is executed at the board end.

From the results shown in the above figure, it can be seen that the confidence level for category 494 is 0.993227. The category is "chime, bell, gong", which means "bell", and the images used for recognition are as follows.

```
[494] score=0.993227 class=n03017168 chime, bell, gong
```

Figure 6.2.1.7 bell.jpg image

6.3 Testing the ResNet model

The ResNet routine also requires downloading the model. The model can be downloaded by executing the model download script. Before execution, it is necessary to first check if the Ubuntu network is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd rknn_model_zoo-2.0.0/examples/resnet/model
sh download_model.sh
```

Figure 6.3.1.1 Execute the script to download the model.

The model has been downloaded. Open the Ubuntu terminal and execute the following command to enter the conda environment.

```
conda activate python3.8-tk2-2.0
```

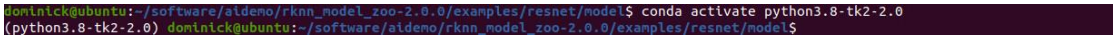


Figure 6.3.1.2 Enable the Conda environment

Enter the resnet routine directory under the rknn_model_zoo-2.0.0 routine in the aidemo directory, and execute the following command to convert the model.

```
cd rknn_model_zoo-2.0.0/examples/resnet/python
python3 resnet.py ../model/resnet50-v2-7.onnx rk3588
```

Figure 6.3.1.3 Convert ResNet model

After the model conversion is completed, the running results will be displayed. If you are using the RK3568 development board, execute the following command to convert the MobileNet model.

```
python3 resnet.py --target rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo. First, enable the compiler. If you are using the rk3588 development board, execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d resnet
```

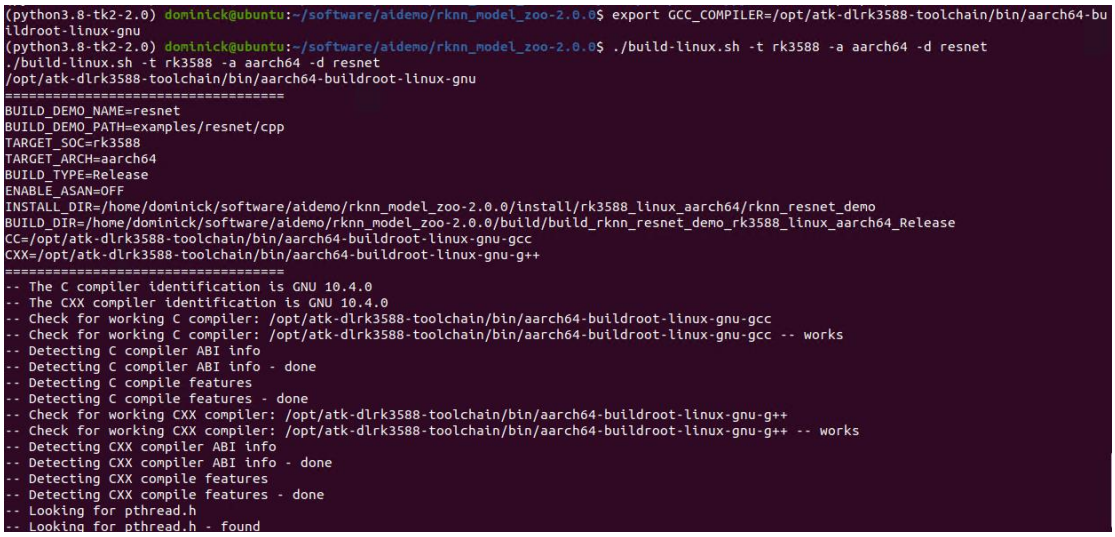


Figure 6.3.1.4 Compile the ResNet test program.

After compilation is completed, executable files and packaged models, etc. will be generated in the "modelzoo" directory under the "install/rk3588_linux_aarch64/rknn_resnet_demo" folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_resnet_demo/ /userdata/aidemo
```

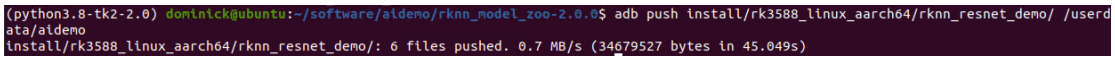


Figure 6.3.1.5 Push the ResNet model and program

If the RK3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d resnet
adb push install/rk3568_linux_aarch64/rknn_resnet_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_resnet_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use dog_224x224.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_resnet_demo
./rknn_resnet_demo model/resnet50-v2-7.rknn model/dog_224x224.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_resnet_demo# ./rknn_resnet_demo model/resnet50-v2-7.rknn model/dog_224x224.jpg
num_lines=1001
model input num: 1, output num: 1
input tensors:
  index=0, name=data, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-14, scale=0
.018658
output tensors:
  index=0, name=resnetv24_dense0_fwd, n_dims=2, dims=[1, 1000, 0, 0], n_elems=1000, size=1000, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE,
 zp=-62, scale=0.146746
model is NHWC input fmt
model input height=224, width=224, channel=3
origin size=224x224 crop size=224x224
input image: 224 x 224, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
src width=224 height=224 fmt=0x1 virAddr=0x0x55a3935710 fd=0
dst width=224 height=224 fmt=0x1 virAddr=0x0x55a395e780 fd=0
color=0x0
rga_api version 1.10.1_[0]
rknn_run
[155] score=0.792182 class=n02086240 Shih-Tzu
[154] score=0.182606 class=n02086079 Pekinese, Pekingese, Peke
[262] score=0.013012 class=n02112706 Brabancon griffon
[152] score=0.002237 class=n02085782 Japanese spaniel
[283] score=0.001931 class=n02123394 Persian cat
```

Figure 6.3.1.6 The board end executes the ResNet program for inference.

From the results shown in the above figure, it can be seen that the confidence level for category 155 is 0.792182. The category is 155, which is "Shih-Tzu", the recognition image used is as follows.

```
[155] score=0.792182 class=n02086240 Shih-Tzu
[154] score=0.182606 class=n02086079 Pekinese, Pekingese, Peke
[262] score=0.013012 class=n02112706 Brabancon griffon
[152] score=0.002237 class=n02085782 Japanese spaniel
[283] score=0.001931 class=n02123394 Persian cat
```

Figure 6.3.1.7 dog_224x224.jpg

6.4 Testing the yolov5 model

To download the model, execute the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd rknn_model_zoo-2.0.0/examples/yolov5/model
sh download_model.sh
```

```
dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov5/model$ sh download_model.sh
--2024-05-11 02:23:42--  https://ftrg.zbox.filez.com/v2/delivery/data/95f00b0fc900458ba134f8b180b3f7a1/examples/yolov5/yolo
v5s_relu.onnx
正在解析主机 ftrg.zbox.filez.com (ftrg.zbox.filez.com)... 180.184.171.46
正在连接 ftrg.zbox.filez.com (ftrg.zbox.filez.com)|180.184.171.46|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200
长度： 28935807 (28M) [application/octet-stream]
正在保存至：“./yolov5s_relu.onnx”

./yolov5s_relu.onnx           100%[==================================================>]  27.59M  8.69MB/s    用时 3.2s

2024-05-11 02:23:46 (8.69 MB/s) - 已保存 “./yolov5s_relu.onnx” [28935807/28935807])
```

Figure 6.4.1.1 Execute the script to download the yolov5 model.

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the yolov5 routine directory under rknn_model_zoo-2.0.0.

```
cd rknn_model_zoo-2.0.0/examples/yolov5/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov5 model.

```
python3  convert.py  ../model/yolov5s_relu.onnx  rk3588
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov5/python$ python3 convert.py ../model/yolov5s_rel
u.onnx rk3588
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
I It is recommended onnx opset 19, but your onnx model opset is 12!
I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for successful convert!
I Loading : 100%|| 125/125 [00:00<00:00, 22885.68it/s]
done
--> Building model
I GraphPreparing : 100%|| 149/149 [00:00<00:00, 6611.28it/s]
I Quantizating : 100%|| 149/149 [00:05<00:00, 28.95it/s]
W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for performance!
                  Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'output0' is changed from 'float32' to 'int8' in rknn model for performance!
                  Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '286' is changed from 'float32' to 'int8' in rknn model for performance!
                  Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '288' is changed from 'float32' to 'int8' in rknn model for performance!
                  Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn buiding done.
done
--> Export rknn model
done
```

Figure 6.4.1.2 Convert the YOLOv5 model to a RKNN model.

If you are using the RK3568 development board, execute the following command to convert the YOLOv5s_relu model.

```
python3  convert.py  ../model/yolov5s_relu.onnx  rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov5
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ./build-linux.sh -t rk3588 -a aarch64 -d yolov5
./build-linux.sh -t rk3588 -a aarch64 -d yolov5
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolov5
BUILD_DEMO_PATH=examples/yolov5/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/install/rk3588_linux_aarch64/rknn_yolov5_demo
BUILD_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_yolov5_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
```

Figure 6.4.1.3 Compile the board-end inference program for yolov5.

After the compilation is completed, executable files and packaged models, etc. will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolov5_demo folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov5_demo/ /userdata/aidemo
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_yolov5_demo/ /userd
ata/aidemo
install/rk3588_linux_aarch64/rknn_yolov5_demo/: 6 files pushed. 0.7 MB/s (16587910 bytes in 21.465s)
```

Figure 6.4.1.4 Push the yolov5 program to the development board

If you are using the rk3568 development board, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov5
adb push install/rk3568_linux_aarch64/rknn_yolov5_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov5_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov5_demo
./rknn_yolov5_demo model/yolov5.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov5_demo# ./rknn_yolov5_demo model/yolov5.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 3
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, sc
ale=0.003922
output tensors:
  index=0, name=output0, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, s
cale=0.003922
  index=1, name=286, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0
.003922
  index=2, name=288, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0
.003922
model is NHWC input fmt
model input height=640, width=640, channel=3
origin size=640x640 crop size=640x640
input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x556a556330 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x556a682340 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.1_[0]
rknn_run
person @ (209 243 286 510) 0.880
person @ (479 238 560 526) 0.871
person @ (109 238 231 534) 0.840
bus @ (91 129 555 464) 0.692
person @ (79 353 121 517) 0.301
write image path: out.png width=640 height=640 channel=3 data=0x556a556330
```

Figure 6.4.1.5 Execute the board-end inference program of yolov5

As can be seen from the results in the above figure, the following categories were detected using bus.jpg, and the images used for recognition are as follows.

Figure 6.4.1.6 bus.jpg image

6.5 Testing the yolov6 model

The yolov6 routine also requires downloading the model. It can be downloaded by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd rknn_model_zoo-2.0.0/examples/yolov6/model
sh download_model.sh
```

Figure 6.5.1.1 Download the yolov6 model

The model download is complete. Open the Ubuntu terminal and execute the following command to enter the conda environment.

```
conda activate python3.8-tk2-2.0
```

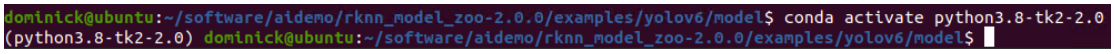


Figure 6.5.1.2 Enter the Conda environment

Navigate to the yolov6 example directory within the rknn_model_zoo-2.0.0 routine in the aidemo directory, and execute the following command to convert the model.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov6/python
python3 convert.py ../model/yolov6n.onnx rk3588
```

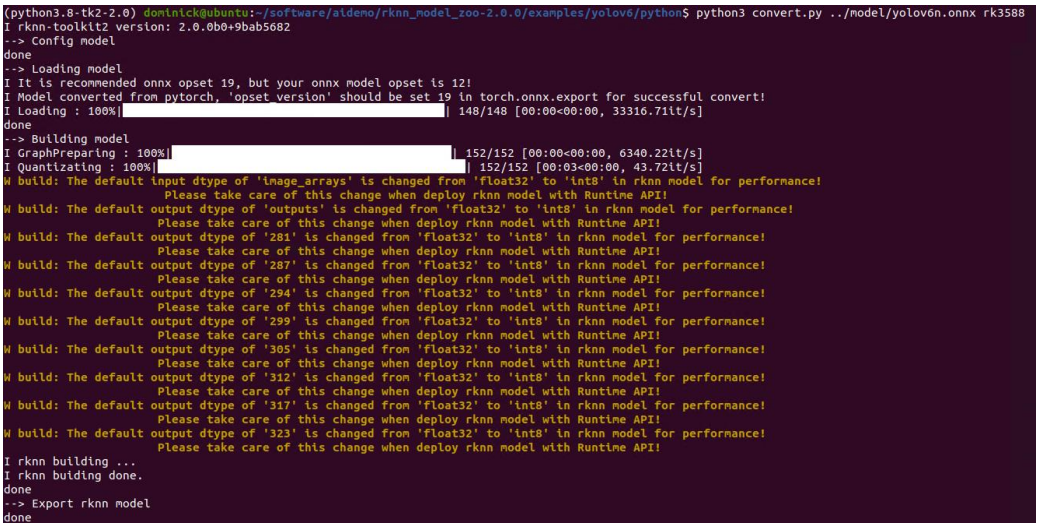


Figure 6.5.1.3 Convert the yolov6 model

After the model conversion is completed, the running results will be displayed. If you are using the rk3568 development board, you can execute the following command to convert the yolov6 model.

```
python3 convert.py ../model/yolov6n.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov6
```

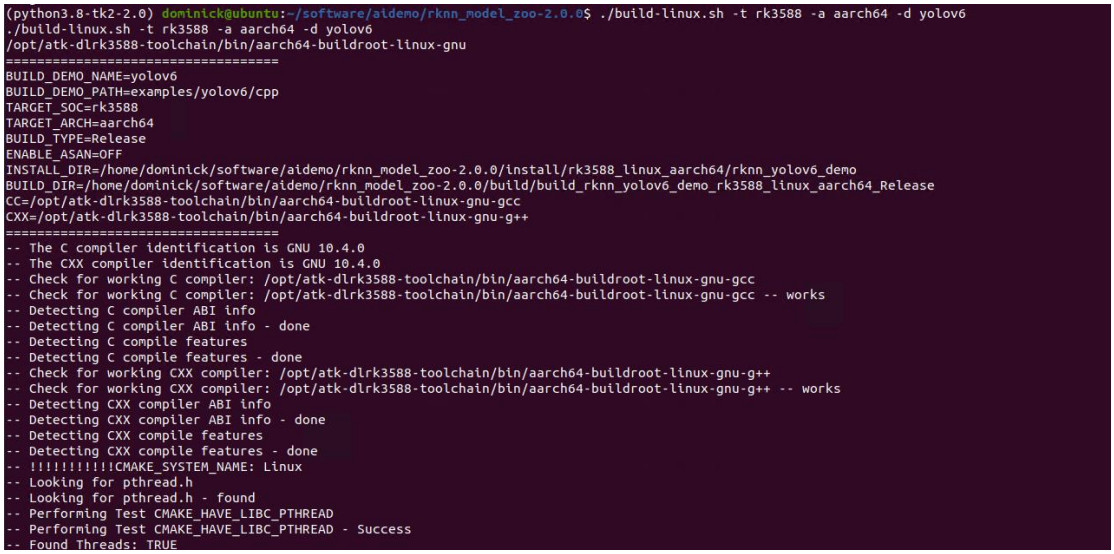


Figure 6.5.1.4 Compile the yolov6 program

After the compilation is completed, executable files and packaged models, etc. will be generated in the install/rk3588_linux_aarch64/rknn_yolov6_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov6_demo/ /userdata/aidemo
```

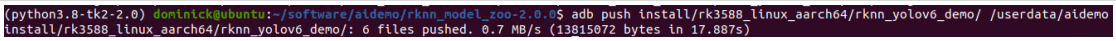


Figure 6.5.1.5 Push the yolov6 model and program to the development board.

If you are using the rk3568 development board, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov6
adb push install/rk3568_linux_aarch64/rknn_yolov6_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov6_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov6_demo
./rknn_yolov6_demo model/yolov6.rknn model/bus.jpg
```

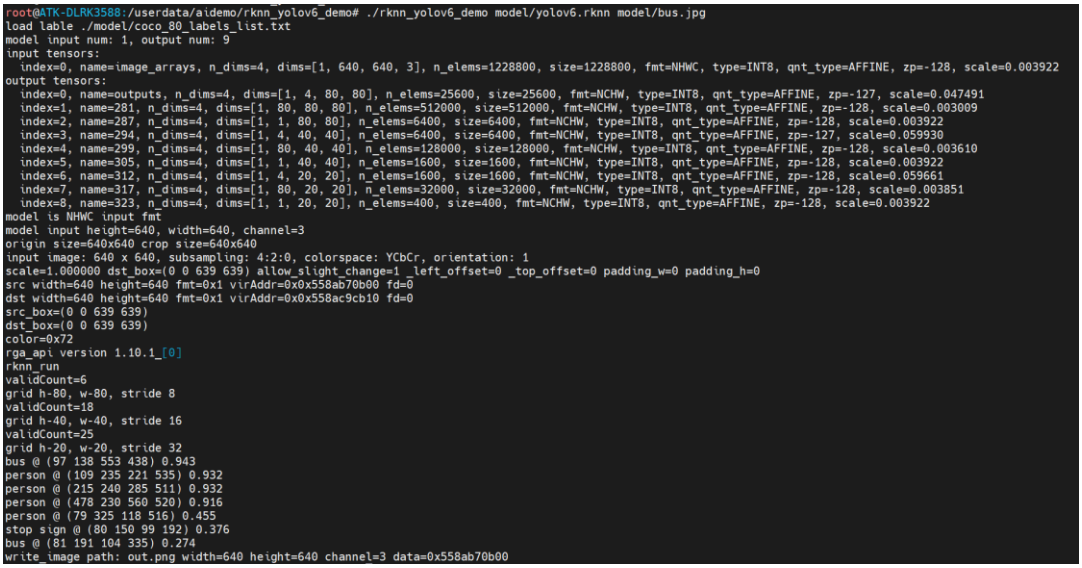


Figure 6.5.1.6 Execute the inference program of the yolov6 model.

From the results shown in the above figure, it can be seen that the following categories were detected through bus.jpg. The images used for recognition are the same as those used for testing by yolov5.

```
bus @ (97 138 553 438) 0.943
person @ (109 235 221 535) 0.932
person @ (215 240 285 511) 0.932
person @ (478 230 560 520) 0.916
person @ (79 325 118 516) 0.455
stop sign @ (80 150 99 192) 0.376
bus @ (81 191 104 335) 0.274
```

6.6 Test the yolov7 model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd rknn_model_zoo-2.0.0/examples/yolov7/model
sh download_model.sh
```

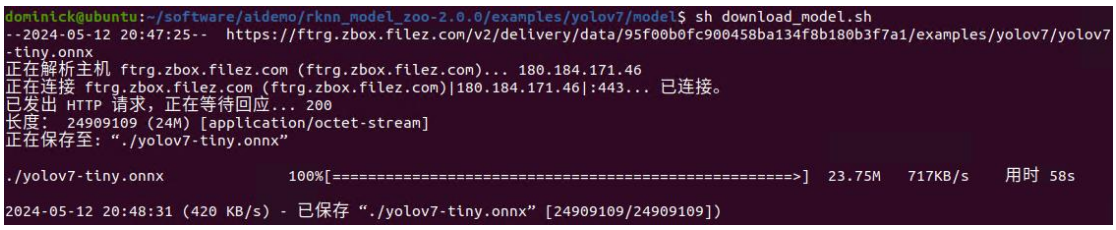


Figure 6.6.1.1 Download the yolov7 model

Open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the yolov7 routine under the rknn_model_zoo-2.0.0 directory.

```
cd rknn_model_zoo-2.0.0/examples/yolov7/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov7 model.

```
python3 convert.py ../model/yolov7-tiny.onnx rk3588
```

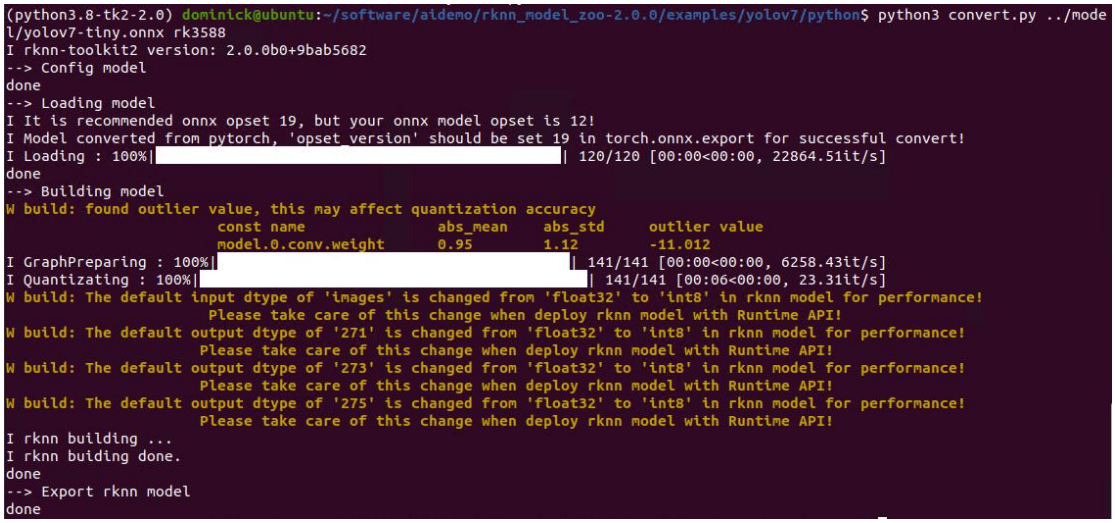


Figure 6.6.1.2 Convert the yolov7 model

If you are using the rk3568 development board, execute the following command to convert the yolov7-tiny model.

```
python3 convert.py ../model/yolov7-tiny.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov7
```

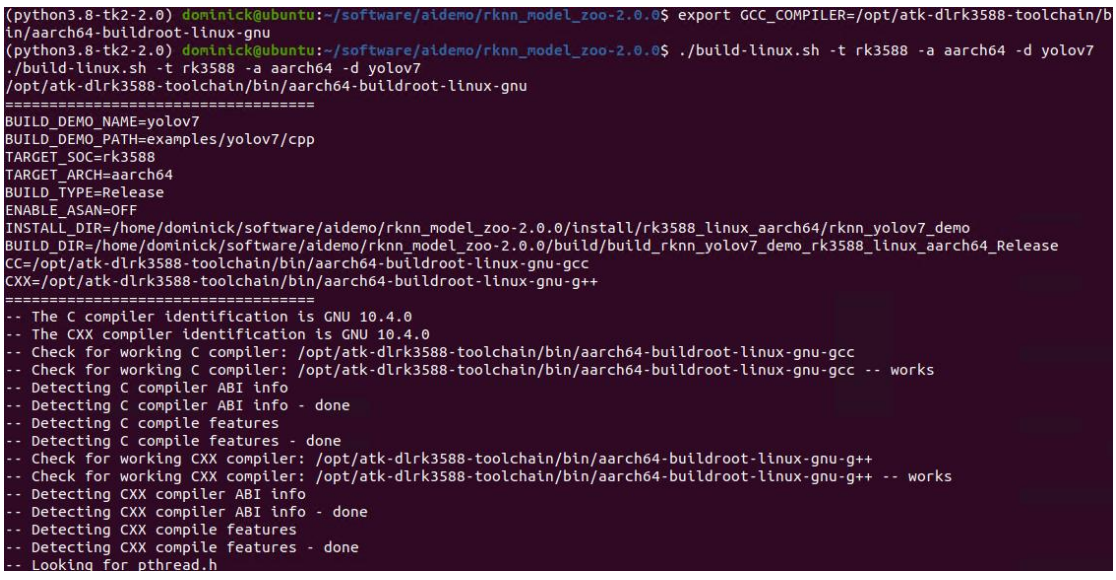


Figure 6.6.1.3 Compile the inference program for the yolov7 model.

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_yolov7_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov7_demo/ /userdata/aidemo
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_yolov7_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_yolov7_demo/: 6 files pushed. 1.0 MB/s (15540962 bytes in 14.488s)
```

Figure 6.6.1.4 Push the YOLOv7 model inference program to the development board.

If you are using the RK3568 development board, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov7
adb push install/rk3568_linux_aarch64/rknn_yolov7_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov7_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov7_demo
./rknn_yolov7_demo model/yolov7.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov7_demo# ./rknn_yolov7_demo model/yolov7.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 3
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=271, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=1, name=273, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=2, name=275, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
model is NHWC input fmt
model input height=640, width=640, channel=3
init_yolov7_model use: 31.701000 ms
origin size=640x640 crop size=640x640
input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x557f1f46d0 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x557f324ad0 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.1_[0]
convert_image_with_letterbox use: 3.648000 ms
rknn_inputs_set use: 4.689000 ms
rknn_run use: 20.065001 ms
rknn_outputs_get use: 2.396000 ms
post_process use: 0.102000 ms
inference_yolov7_model use: 30.989000 ms
person @ (212 243 285 511) 0.898
person @ (475 243 561 519) 0.847
person @ (111 236 220 529) 0.839
bus @ (88 135 542 445) 0.820
person @ (79 331 126 524) 0.328
write_image path: out.png width=640 height=640 channel=3 data=0x557f1f46d0
```

Figure 6.6.1.5 The yolov7 model program is executed at the board end for inference.

An out.png image will be generated in the current directory. You can copy it to your computer for viewing.

```
person @ (212 243 285 511) 0.898
person @ (475 243 561 519) 0.847
person @ (111 236 220 529) 0.839
bus @ (88 135 542 445) 0.820
person @ (79 331 126 524) 0.328
```

6.7 Test the yolov8 model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov8/model
sh download_model.sh
```

Figure 6.7.1.1 Download the yolov8 model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the yolov8 routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov8/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov8n model.

```
python3 convert.py ../model/yolov8n.onnx rk3588
```

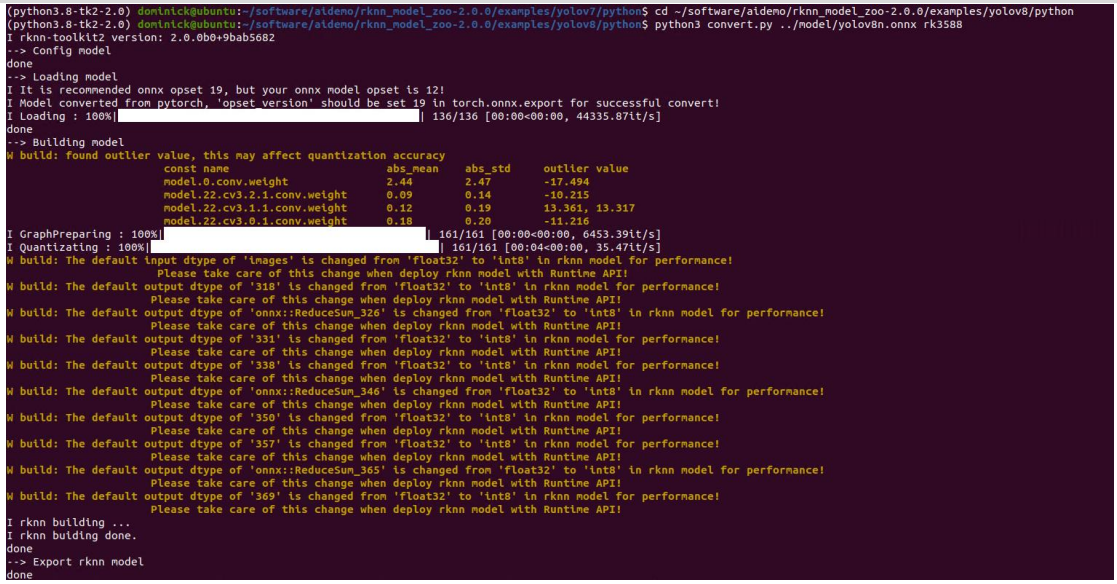


Figure 6.7.1.2 Convert the YOLOv8 model to a RKNN model.

If you are using the RK3568 development board, execute the following command to convert the YOLOv8n model.

```
python3 convert.py ../model/yolov8n.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
```

```
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov8
```

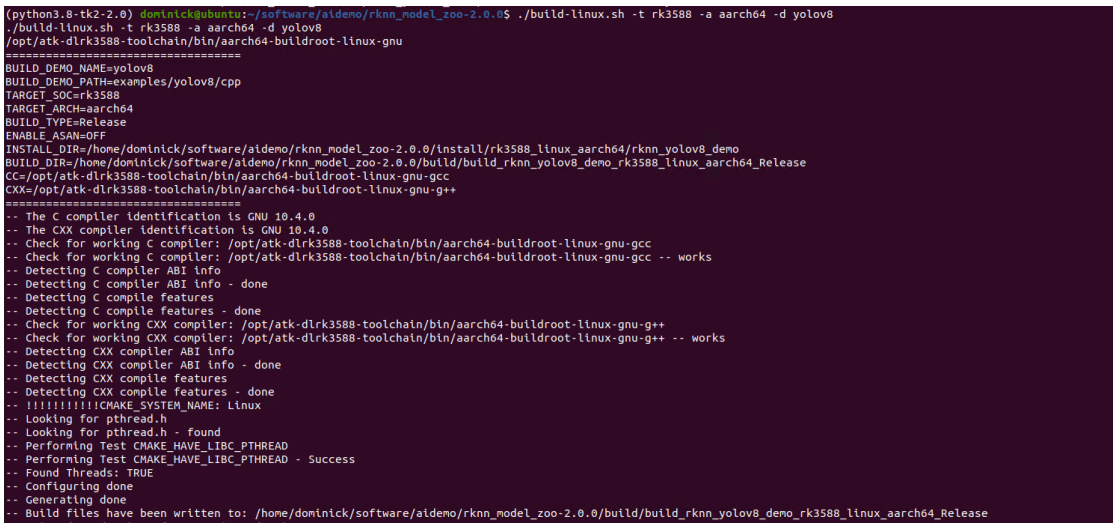


Figure 6.7.1.3 Compile the inference program for the yolov8 model on the board.

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_yolov8_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov8_demo/ /userdata/aidemo
```

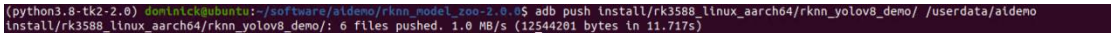


Figure 6.7.1.4 Push the yolov8 model and the board-end inference program.

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov8
adb push install/rk3568_linux_aarch64/rknn_yolov8_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov8_demo under the directory pushed to the development board, and execute the following command in the terminal of the development board. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov8_demo
./rknn_yolov8_demo model/yolov8.rknn model/bus.jpg
```

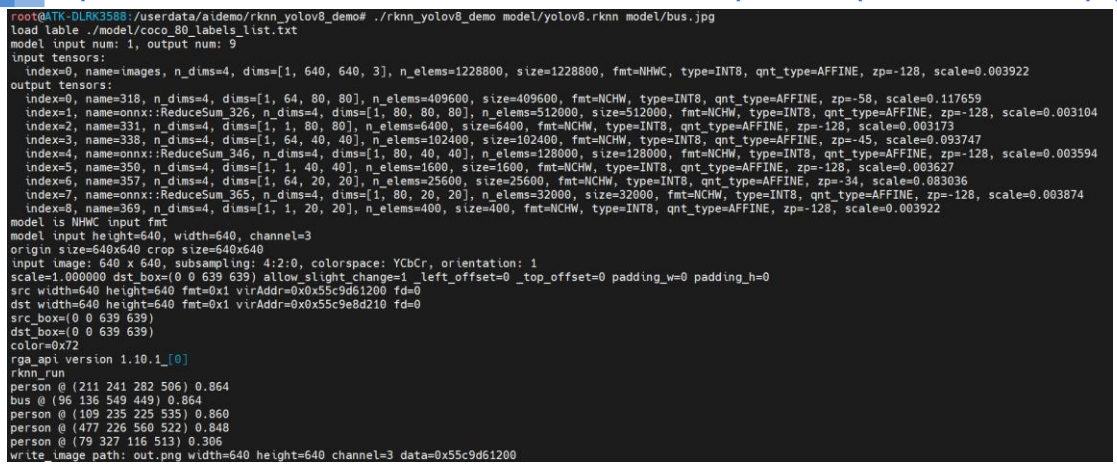


Figure 6.7.1.5 Run the inference program of the yolov8 model on the edge device.

In the current directory, an out.png image will be generated, which can be copied to the computer for viewing.

6.8 Test the yolox model

Download the model by executing the model download script. Before execution, it is necessary to first check whether the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolox/model
sh download_model.sh
```

Figure 6.8.1.1 Download the YOLOX model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the YOLOX routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolox/python
```

If you are using the rk3588 development board, execute the following command to convert the yolox model.

```
python3 convert.py ../model/yolox_s.onnx rk3588
```

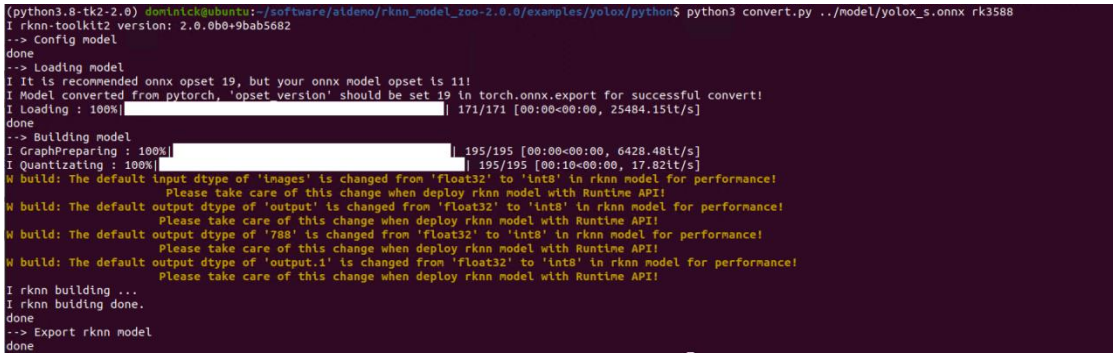


Figure 6.8.1.2 Convert the YOLOX model to a RKNN model.

If you are using the RK3568 development board, execute the following command to convert the YOLOX_s model.

```
python3  convert.py  ../model/yolox_s.onnx  rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolox
```

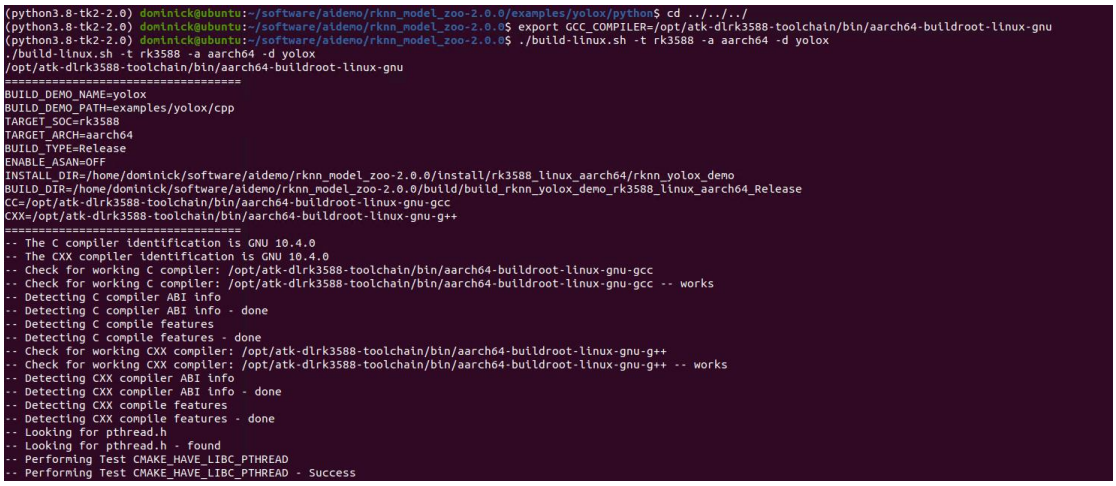


Figure 6.8.1.3 Compile the inference program for the yolox model on the board.

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_yolox_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolox_demo/ /userdata/aidemo
```

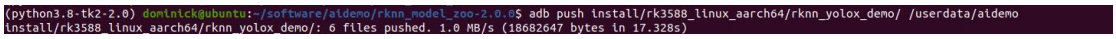


Figure 6.8.1.4 Push the YOLOX model and inference program to the development board.

If using the RK3568 development board, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
./build-linux.sh -t rk3568 -a aarch64 -d yolox
adb push install/rk3568_linux_aarch64/rknn_yolox_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolox_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_yolox_demo
./rknn_yolox_demo model/yolox.rknn model/bus.jpg
```

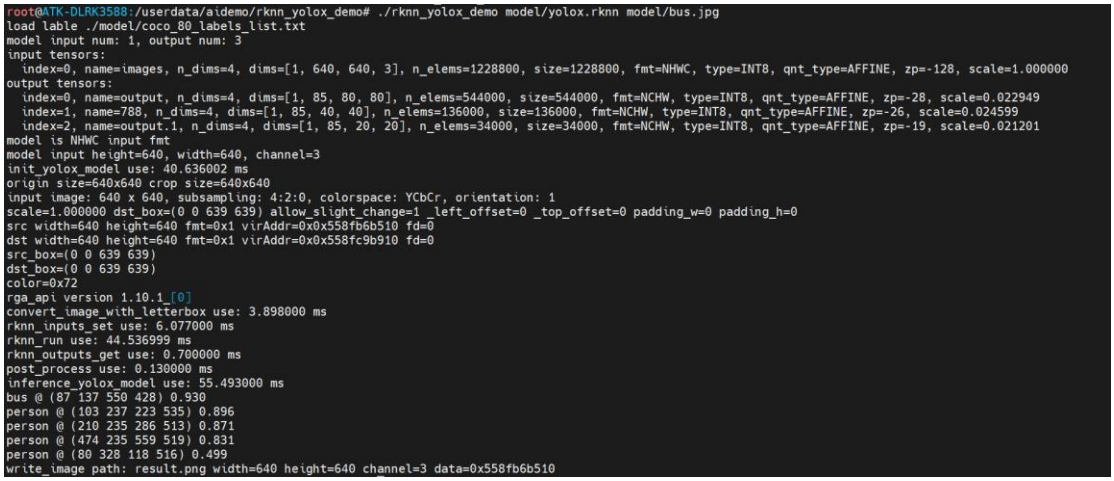


Figure 6.8.1.5 Execute the inference program of the yolox model at the edge.

A result.png image will be generated in the current directory. It can be copied to the computer via network scp or USB drive for viewing.

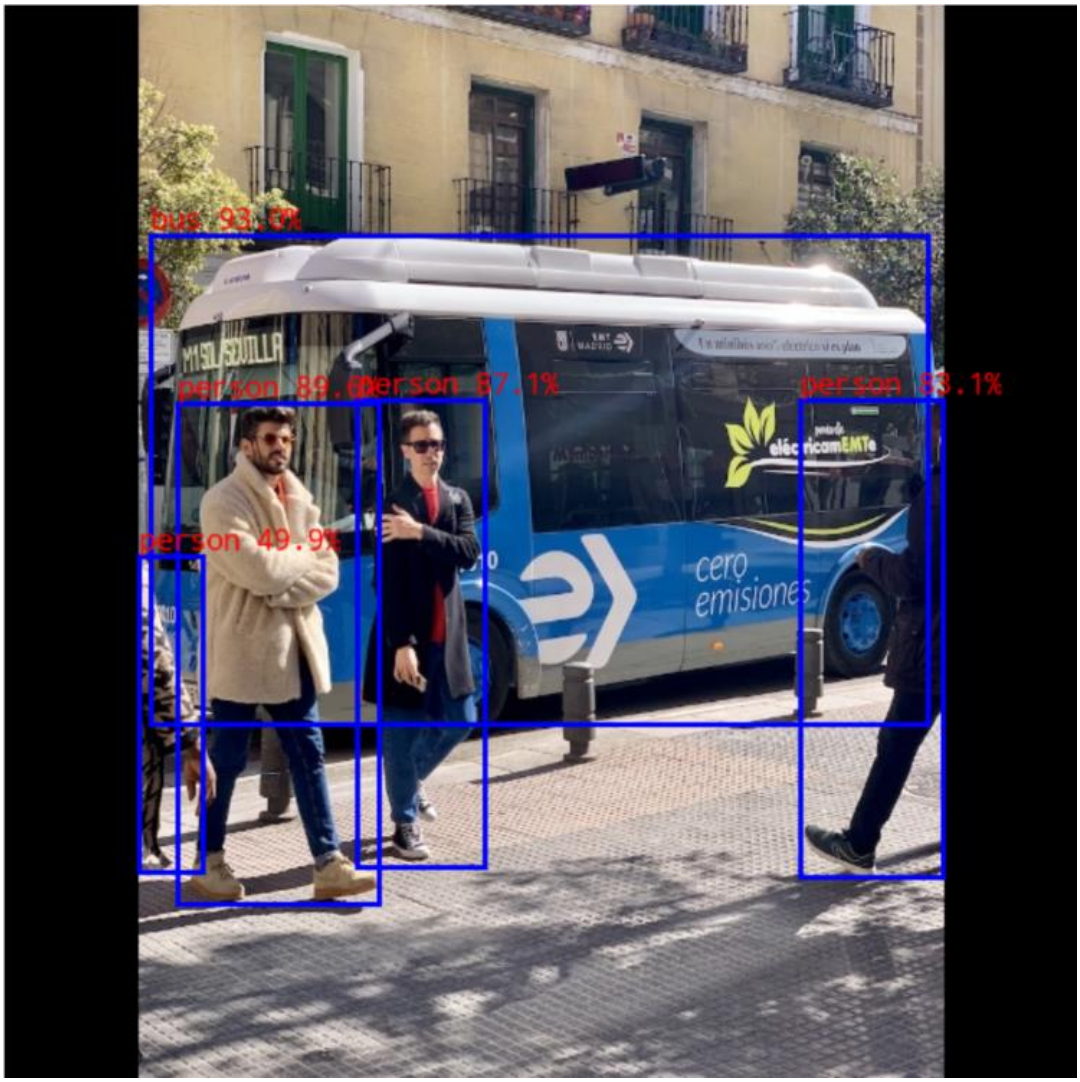


Figure 6.8.1.6 result.png image

6.9 Testing the ppyoloe model

To download the model, execute the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/ppyoloe/model
sh download_model.sh
```

Figure 6.9.1.1 Download the ppyoloe model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory under the ppyoloe routine in the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/ppyoloe/python
```

If you are using the rk3588 development board, execute the following command to convert the ppyoloe_s model.

```
python3 convert.py ../model/ppyoloe_s.onnx rk3588
```

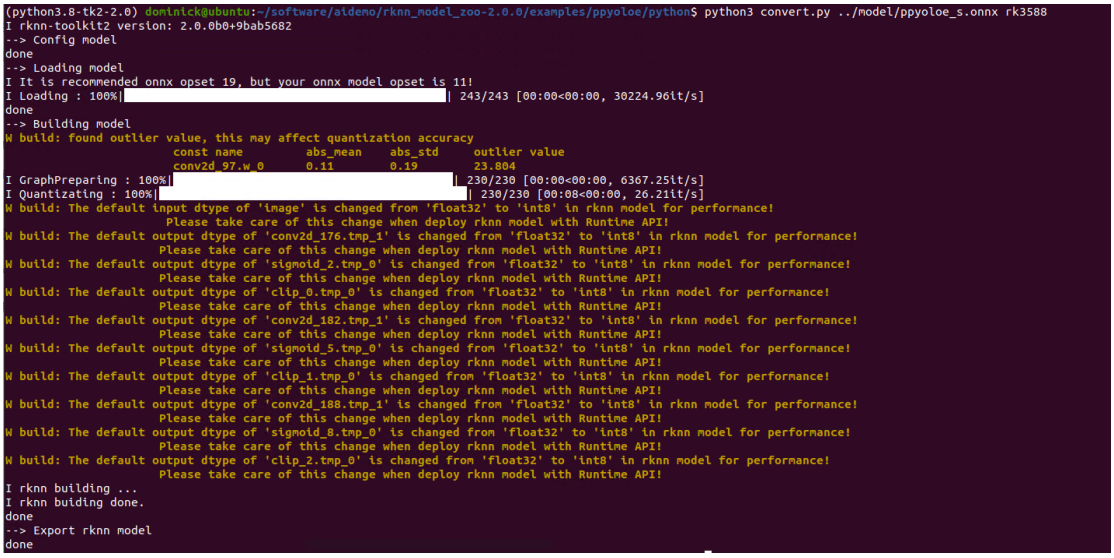


Figure 6.9.1.2 Convert the ppyoloe model to the rknn model.

If you are using the RK3568 development board, execute the following command to convert the PPyoloe_s model.

```
python3 convert.py ../model/ppyoloe_s.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d ppyoloe
```

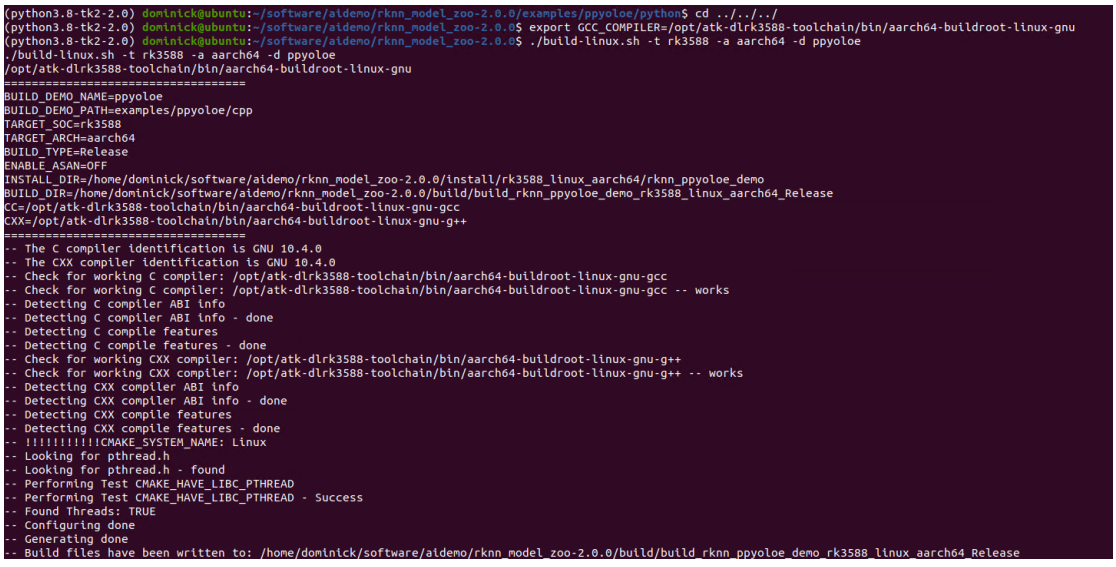


Figure 6.9.1.3 Compile the inference executable program for the ppyoloe model at the board end.

After compilation is complete, executable files and packaged models, etc. will be generated in the install/rk3588_linux_aarch64/rknn_ppyoloe_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_ppyoloe_demo/ /userdata/aidemo
```

Figure 6.9.1.4 Push the pppyoloe model

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d ppyoloe
adb push install/rk3568_linux_aarch64/rknn_ppyoloe_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_ppyoloe_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_ppyoloe_demo
./rknn_ppyoloe_demo model/ppyoloe.rknn model/bus.jpg
```

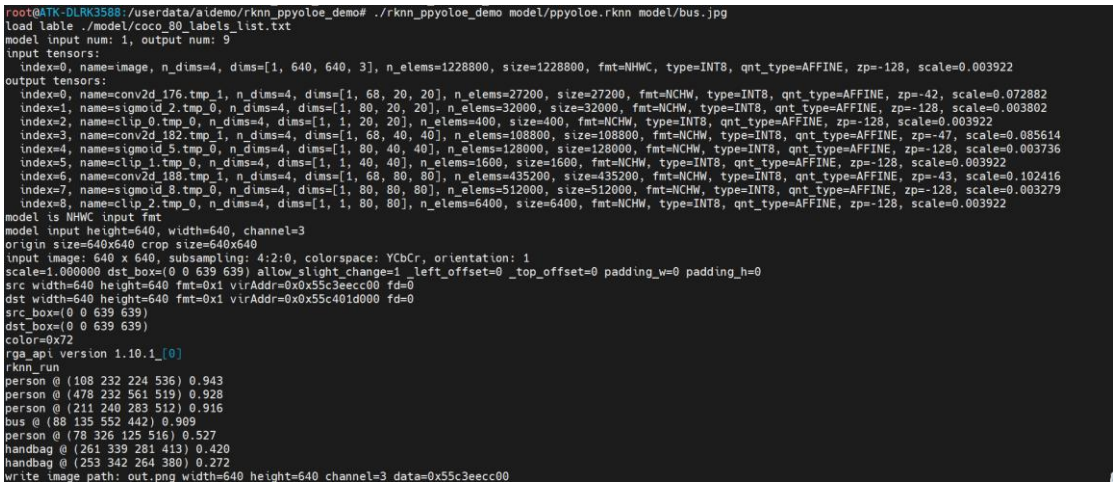


Figure 6.9.1.5 Board end reasoning Ppyoloe model

An out.png image will be generated in the current directory. It can be copied to the computer via network scp or USB drive for viewing.

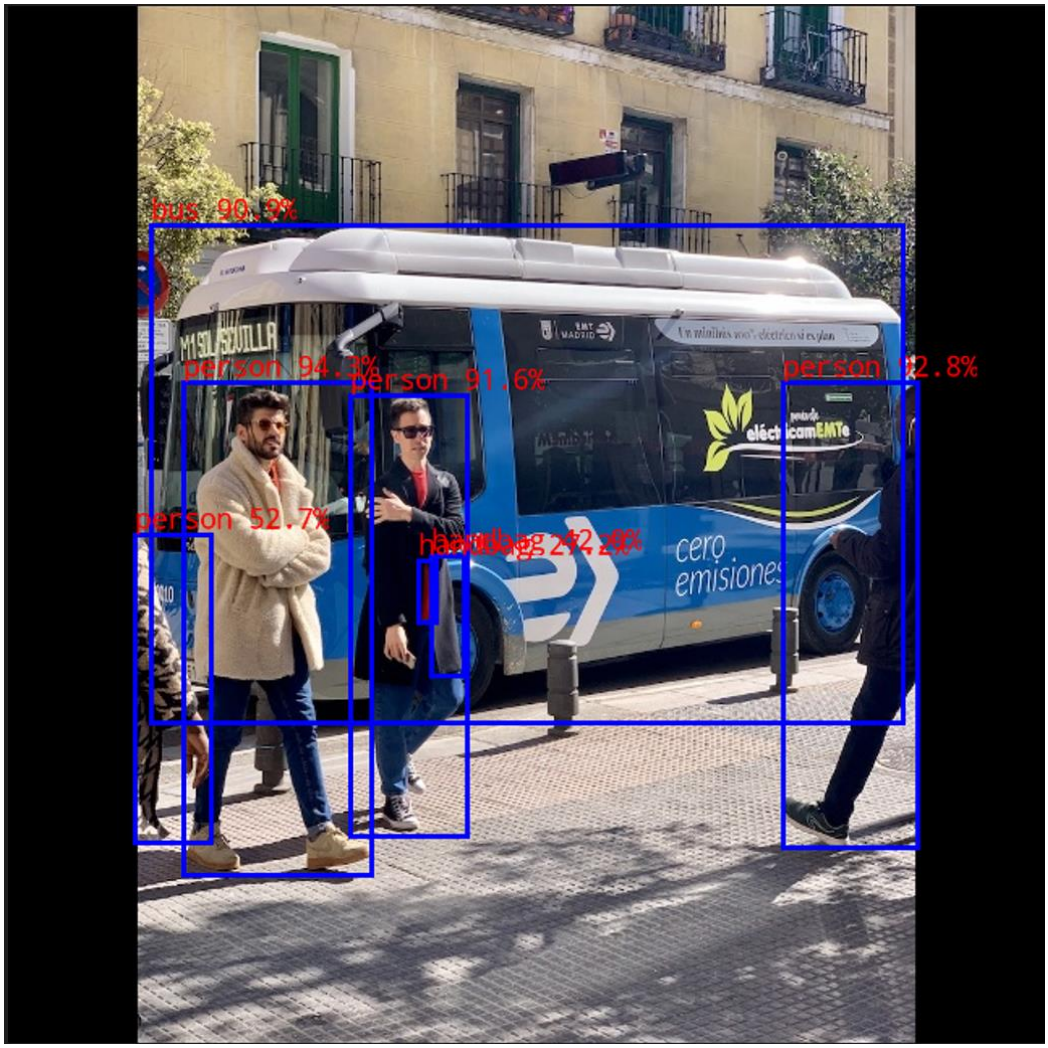


Figure 6.9.1.6 result.png image

6.10 Testing the DeepLabv3 model

To download the model, execute the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/deeplabv3/model
sh download_model.sh
```

Figure 6.10.1.1 Download the DeepLabv3 model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the deeplabv3 routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/deeplabv3/python
```

If you are using the rk3588 development board, execute the following command to convert the deeplab-v3-plus-mobilenet-v2.pb model.

```
python3 convert.py ../model/deeplab-v3-plus-mobilenet-v2.pb rk3588
```

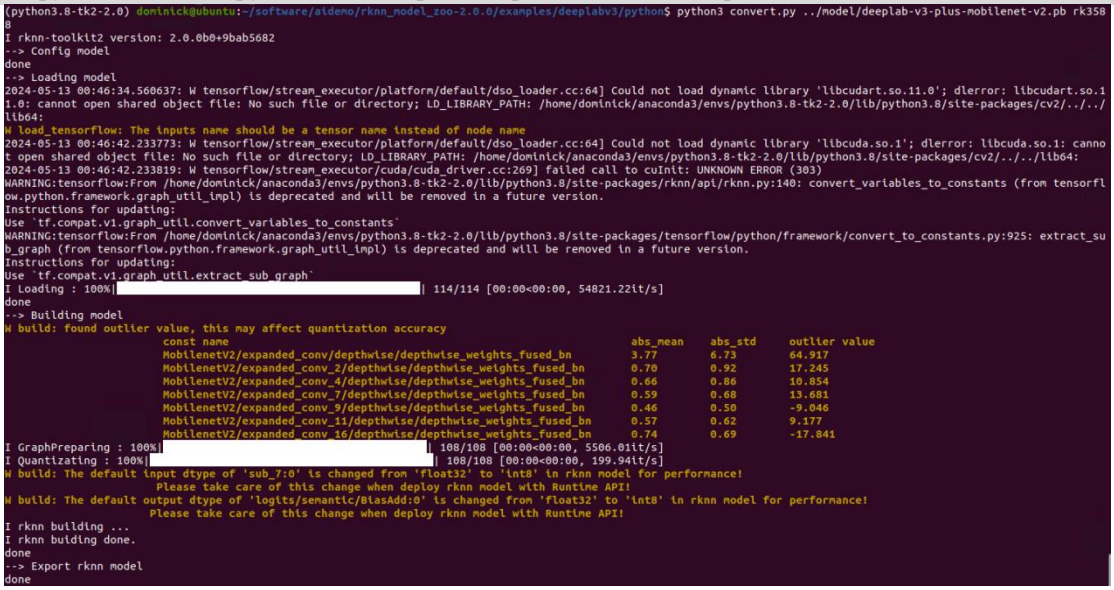


Figure 6.10.1.2 Convert the DeepLabV3 model to the RKNN format model.

If you are using the RK3568 development board, execute the following command to convert the deeplab-v3-plus-mobilenet-v2.pb model.

```
python3 convert.py ../model/deeplab-v3-plus-mobilenet-v2.pb rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d deeplabv3
```

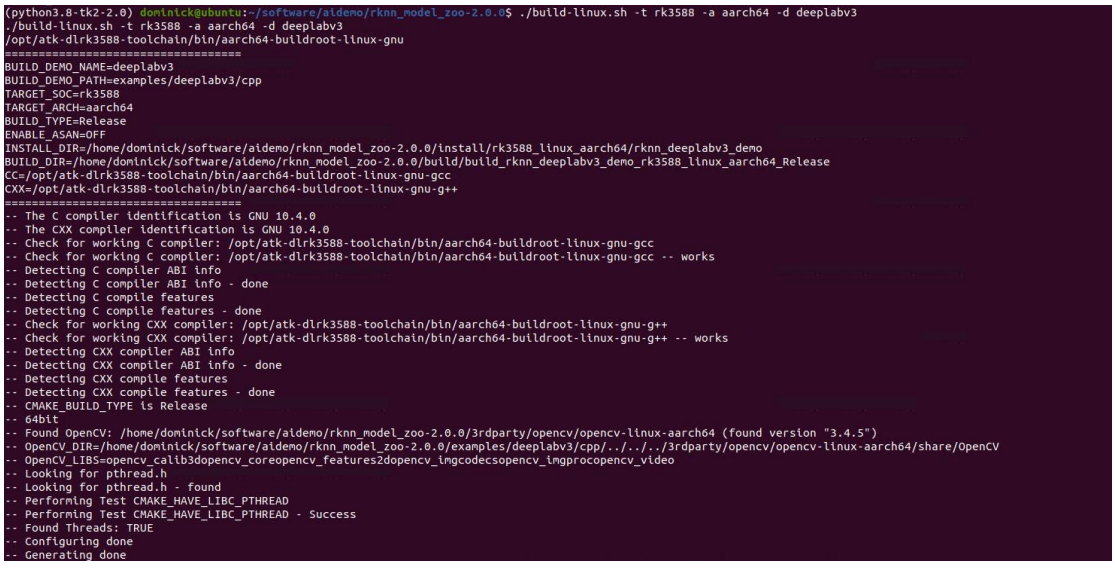


Figure 6.10.1.3 Compile the board-end inference program for the deeplabv3 model.

After compilation is completed, executable files and packaged models, etc. will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_deeplabv3_demo directory. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_deeplabv3_demo/ /userdata/aidemo
```

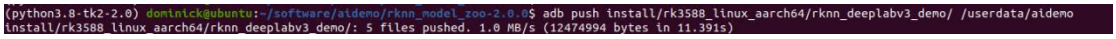


Figure 6.10.1.4 Push the DeepLabV3 model and the board-end inference program.

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d deeplabv3
adb push install/rk3568_linux_aarch64/rknn_deeplabv3_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_deeplabv3_demo which is pushed to the development board, and execute the following command in the development board terminal. Use test_image.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_deeplabv3_demo
./rknn_deeplabv3_demo model/deeplab-v3-plus-mobilenet-v2.rknn model/test_image.jpg
```

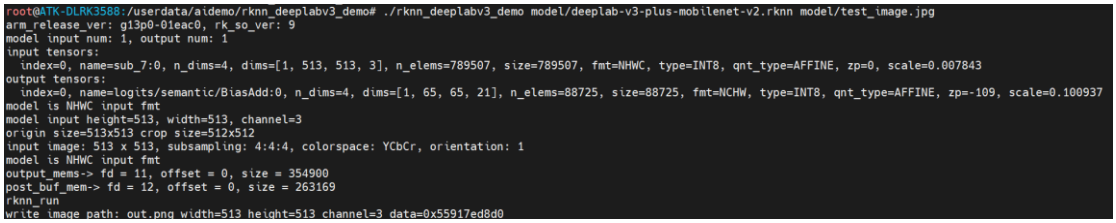


Figure 6.10.1.5 Board end reasoning Deeplabv3 model

An out.png image will be generated in the current directory. It can be copied to the computer via network SCP or USB drive for viewing.

Figure 6.10.1.6 test_image.jpg image

Figure 6.10.1.7 out.png image

6.11 Test the yolov5_seg model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov5_seg/model
sh download_model.sh
```

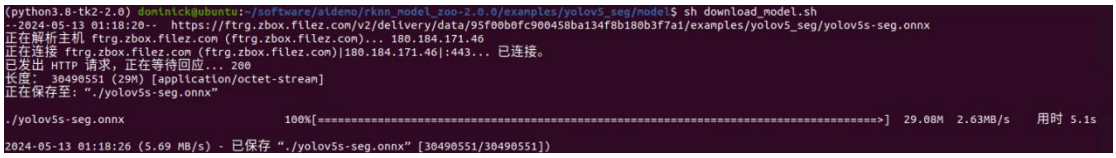


Figure 6.11.1.1 Download the yolov5_seg model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the yolov5_seg routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov5_seg/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov5-seg.onnx model.

```
python3 convert.py ../model/yolov5-seg.onnx rk3588
```

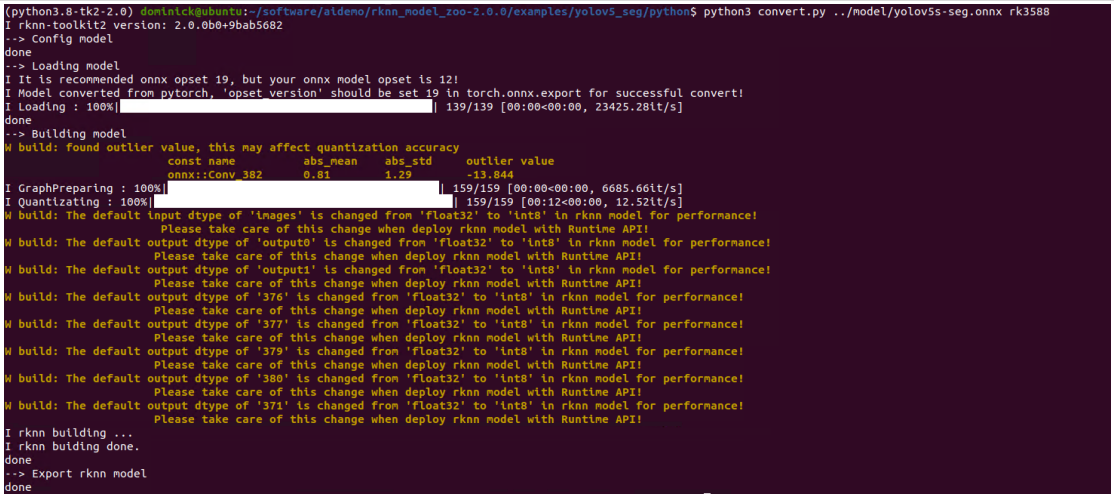


Figure 6.11.1.2 Convert the yolov5_seg model to a rknn model.

If you are using the rk3568 development board, execute the following command to convert the yolov5-seg.onnx model.

```
python3 convert.py ../model/yolov5-seg.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d yolov5_seg
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ./build-linux.sh -t rk3588 -a aarch64 -d yolov5_seg
./build-linux.sh -t rk3588 -a aarch64 -d yolov5_seg
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==============================
BUILD_DEMO_NAME=yolov5_seg
BUILD_DEMO_PATH=examples/yolov5_seg/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/install/rk3588_linux_aarch64/rknn_yolov5_seg_demo
BUILD_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_yolov5_seg_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==============================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- 64bit
-- Found OpenCV: /home/dominick/software/aidemo/rknn_model_zoo-2.0.0/3rdparty/opencv/opencv-linux-aarch64 (found version "3.4.5")
-- OpenCV_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov5_seg/cpp/../../../3rdparty/opencv/opencv-linux-aarch64/share/OpenCV
-- OpenCV_LIBS=opencv_calib3dopencv_coreopencv_features2dopencv_imgcodecsopencv_imgprocopencv_video
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
```

Figure 6.11.1.3 Compile the inference program for the yolov5_seg model on the board.

After the compilation is completed, executable files and packaged models will be generated in the modelzoo directory under install/rk3588_linux_aarch64/rknn_yolov5_seg_demo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov5_seg_demo/ /userdata/aidemo
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_yolov5_seg_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_yolov5_seg_demo/: 6 files pushed. 1.0 MB/s (20224868 bytes in 18.763s)
```

Figure 6.11.1.4 Push the yolov5_seg model and the board-end inference program.

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov5_seg
adb push install/rk3568_linux_aarch64/rknn_yolov5_seg_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov5_seg_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov5_seg_demo
./rknn_yolov5_seg_demo model/yolov5_seg.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov5_seg_demo# ./rknn_yolov5_seg_demo model/yolov5_seg.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 7
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=output0, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=1, name=output1, n_dims=4, dims=[1, 96, 80, 80], n_elems=614400, size=614400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=20, scale=0.022222
  index=2, name=376, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=377, n_dims=4, dims=[1, 96, 40, 40], n_elems=153600, size=153600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=29, scale=0.023239
  index=4, name=379, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003918
  index=5, name=380, n_dims=4, dims=[1, 96, 20, 20], n_elems=38400, size=38400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=32, scale=0.024074
  index=6, name=371, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-116, scale=0.022475
model is NHWC input fmt
model input height=640, width=640, channel=3
origin size=640x640 crop size=640x640
input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x5573cba720 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x5573deab20 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.1_[0]
rknn_run
matmul_by_cpu_uint8 use: 13.343000 ms
resize_by_opencv_uint8 use: 3.277000 ms
crop_mask_uint8 use: 4.614000 ms
seg_reverse use: 0.052000 ms
person @ (213 239 284 516) 0.882
person @ (109 239 224 536) 0.865
person @ (473 230 559 522) 0.842
bus @ (97 134 548 461) 0.824
person @ (79 325 124 520) 0.501
write image path: out.png width=640 height=640 channel=3 data=0x5573cba720
```

Figure 6.11.1.5 Execute the yolov5_seg board-end inference program.

An "out.png" image will be generated in the current directory. You can copy it to your computer via network SCP or USB drive for viewing.

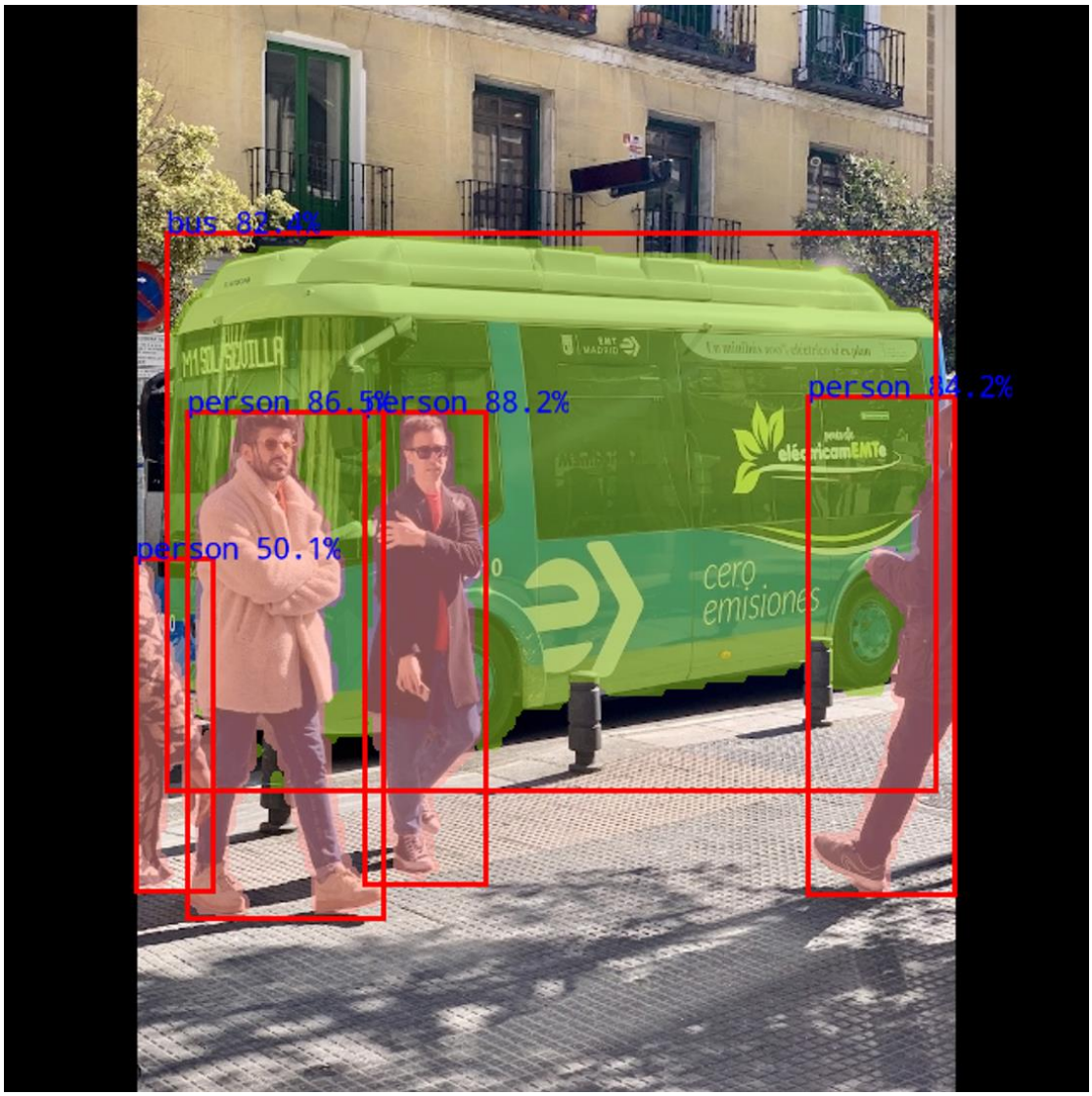


Figure 6.11.1.5 out.png image

6.12 Test yolov8_seg model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov8_seg/model
sh download_model.sh
```

Figure 6.12.1.1 Download the yolov8_seg model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the yolov8_seg routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/yolov8_seg/python
```

If you are using the rk3588 development board, execute the following command to convert the yolov8-seg.onnx model.

```
python3 convert.py ../model/yolov8s-seg.onnx rk3588
```

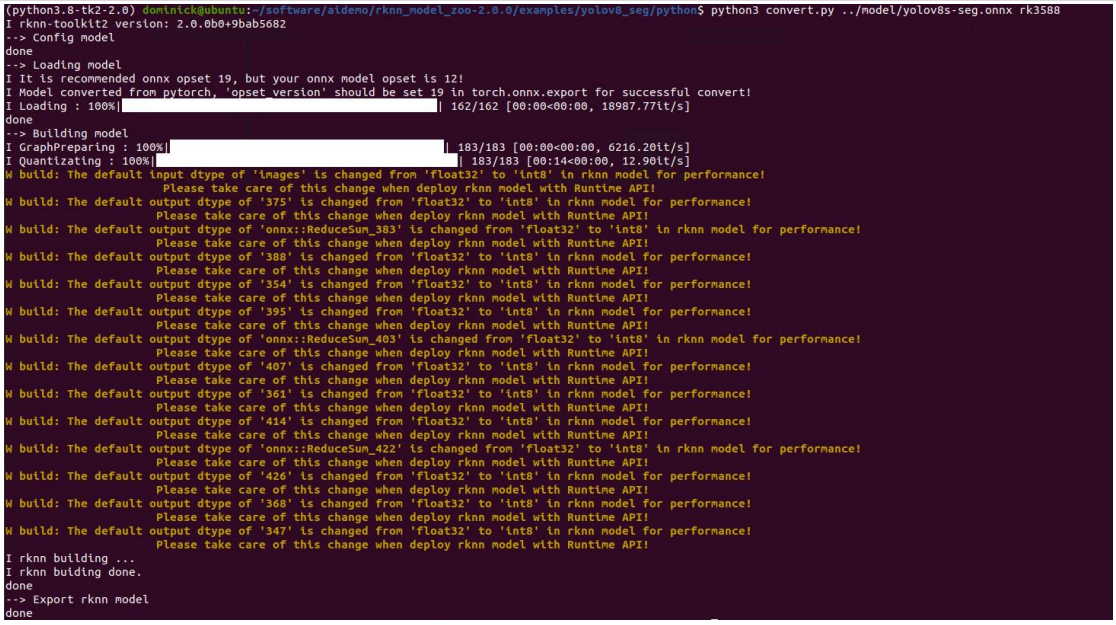


Figure 6.12.1.2 Convert the yolov8_seg model to the rknn format model.

If you are using the rk3568 development board, execute the following command to convert the yolov8-seg.onnx model.

```
python3 convert.py ../model/yolov8-seg.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
./build-linux.sh -t rk3588 -a aarch64 -d yolov8_seg
```

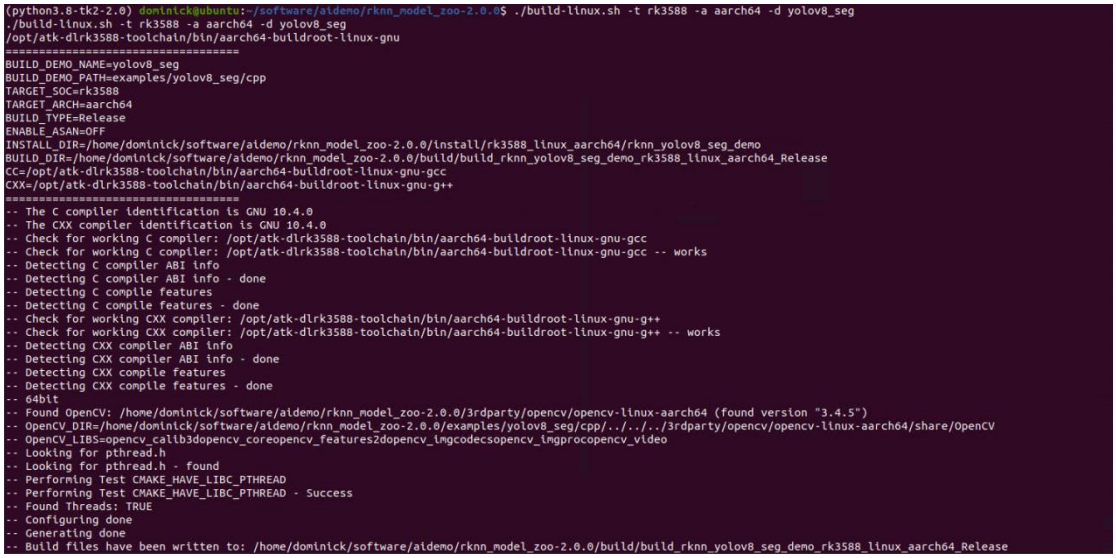


Figure 6.12.1.3 Compile the inference program for the yolov8_seg model.

After the compilation is completed, executable files and packaged models will be generated in the modelzoo directory under the install/rk3588_linux_aarch64/rknn_yolov8_seg_demo folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_yolov8_seg_demo/ /userdata/aidemo
```

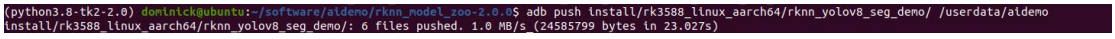


Figure 6.12.1.4 Push the yolov8_seg model and program to the development board.

If you are using the rk3568 development board, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d yolov8_seg
adb push install/rk3568_linux_aarch64/rknn_yolov8_seg_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_yolov8_seg_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use bus.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_yolov8_seg_demo
./rknn_yolov8_seg_demo model/yolov8_seg.rknn model/bus.jpg
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_yolov8_seg_demo# ./rknn_yolov8_seg_demo model/yolov8_seg.rknn model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 13
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=375, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-61, scale=0.115401
  index=1, name=onnx::ReduceSum_383, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003514
  index=2, name=388, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003540
  index=3, name=354, n_dims=4, dims=[1, 32, 80, 80], n_elems=204800, size=204800, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=27, scale=0.019863
  index=4, name=395, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-15, scale=0.099555
  index=5, name=onnx::ReduceSum_403, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003555
  index=6, name=407, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003680
  index=7, name=361, n_dims=4, dims=[1, 32, 40, 40], n_elems=51200, size=51200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=30, scale=0.022367
  index=8, name=414, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-55, scale=0.074253
  index=9, name=onnx::ReduceSum_422, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003813
  index=10, name=426, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=11, name=368, n_dims=4, dims=[1, 32, 20, 20], n_elems=12800, size=12800, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=43, scale=0.019919
  index=12, name=347, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-119, scale=0.032336
model is NHWC input fmt
model input height=640, width=640, channel=3
origin size=640x640 crop size=640x640
input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x558925e100 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x558938e500 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.1_[0]
rknn_run
matmul_by_cpu_uint8 use: 34.214001 ms
resize_by_opencv_uint8 use: 3.149000 ms
crop_mask_uint8 use: 4.949000 ms
seg_reverse use: 0.273000 ms
bus @ (87 137 553 439) 0.911
person @ (109 236 226 534) 0.900
person @ (211 241 283 508) 0.869
person @ (476 234 559 519) 0.866
person @ (79 327 125 514) 0.540
tie @ (248 284 259 310) 0.274
write_image path: out.png width=640 height=640 channel=3 data=0x558925e100
```

Figure 6.12.1.5 Run the yolov8_seg inference program.

An out.png image will be generated in the current directory. You can copy it to your computer via network (scp), USB drive, or other means for viewing.

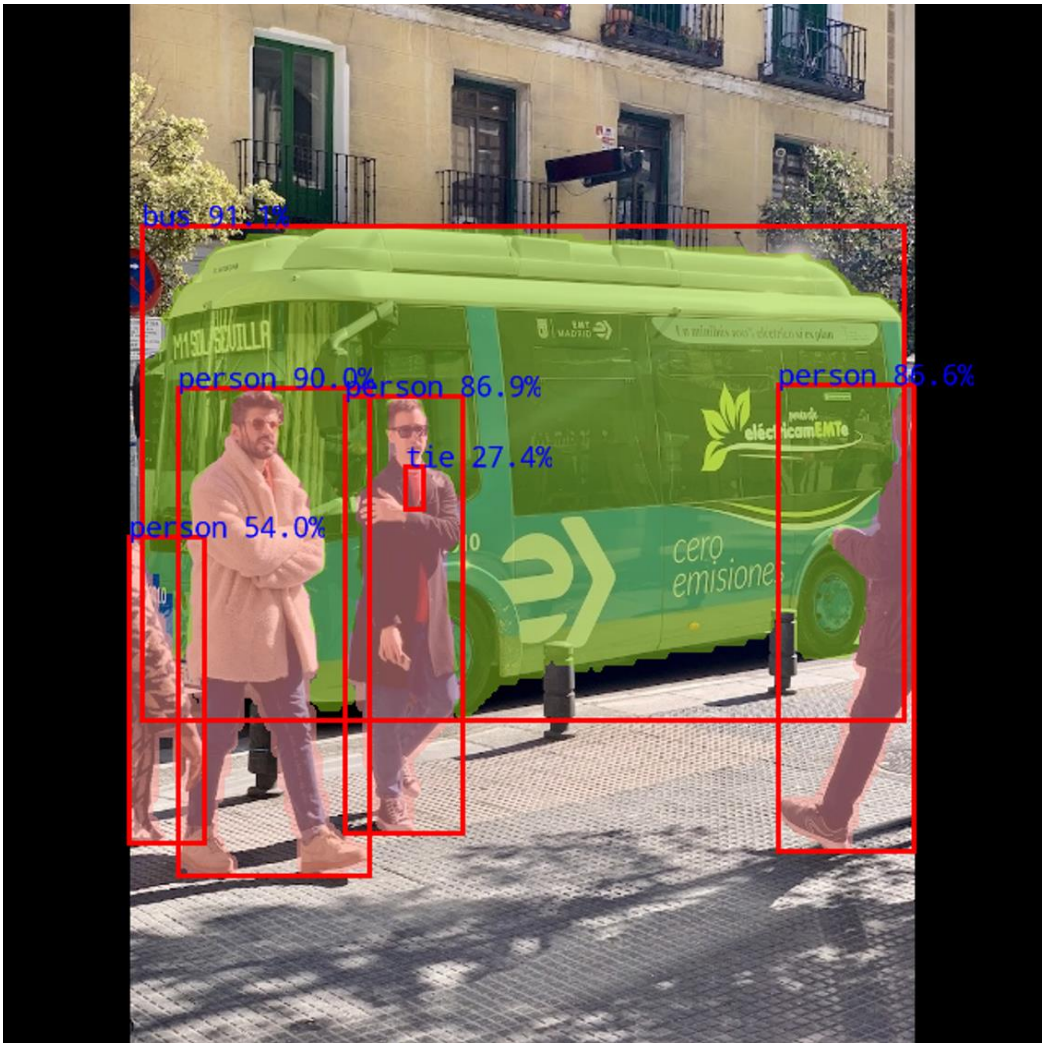


Figure 6.12.1.6 Testing the ppseg Model

6.13 Testing the ppseg Model

To download the model, execute the model download script. Before running the script, make sure the Ubuntu network is functioning properly. Run the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/ppseg/model
sh download_model.sh
```

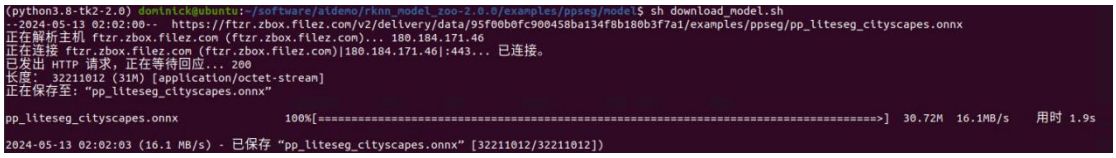


Figure 6.13.1.1 Download the ppseg model.

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we created earlier.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the ppseg routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/ppseg/python
```

If you are using the rk3588 development board, execute the following command to convert the pp_liteseg_cityscapes.onnx model.

```
python3 convert.py ../model/pp_liteseg_cityscapes.onnx rk3588
```

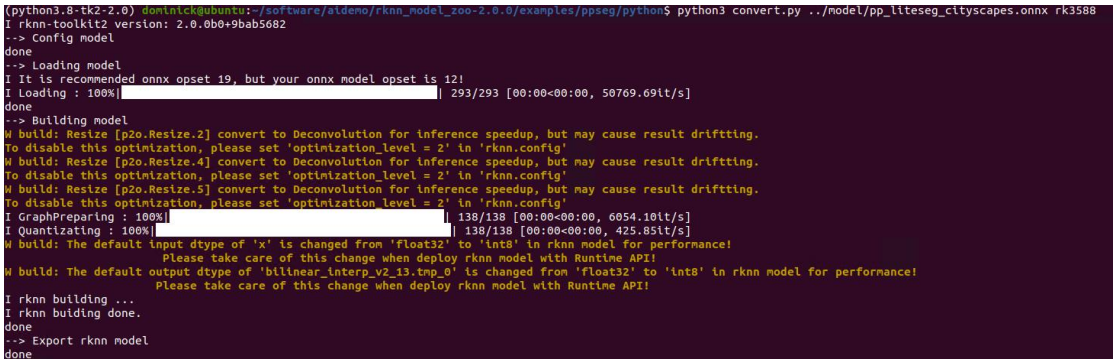


Figure 6.13.1.2 Convert the ppseg model to the rknn model.

If you are using the rk3568 development board, execute the following command to convert the yolov8-seg.onnx model.

```
python3 convert.py ../model/pp_liteseg_cityscapes.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d ppseg
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ./build-linux.sh -t rk3588 -a aarch64 -d ppseg
./build-linux.sh -t rk3588 -a aarch64 -d ppseg
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=ppseg
BUILD_DEMO_PATH=examples/ppseg/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/install/rk3588_linux_aarch64/rknn_ppseg_demo
BUILD_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_ppseg_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_ppseg_demo_rk3588_linux_aarch64_Release
```

Figure 6.13.1.3 Compile the inference program of the ppseg model

After compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_ppseg_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_ppseg_demo/ /userdata/aidemo
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_ppseg_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_ppseg_demo/: 5 files pushed. 1.0 MB/s (19597935 bytes in 18.358s)
```

Figure 6.13.1.4 Push the ppseg model and inference program

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d ppseg
adb push install/rk3568_linux_aarch64/rknn_ppseg_demo/ /userdata/aidemo
```

Open the serial terminal of the development board, enter the directory /userdata/aidemo/rknn_ppseg_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use test.png for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_ppseg_demo
./rknn_ppseg_demo model/pp_liteseg.rknn model/test.png
```

```
root@ATK-DLRK3588:/userdata/aidemo/rknn_ppseg_demo# ./rknn_ppseg_demo model/pp_liteseg.rknn model/test.png
model input num: 1, output num: 1
input tensors:
  index=0, name=x, n_dims=4, dims=[1, 512, 512, 3], n_elems=786432, size=786432, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-19, scale=0.018054
output tensors:
  index=0, name=bilinear_interp_v2_13.tmp_0, n_dims=4, dims=[1, 19, 512, 512], n_elems=4980736, size=4980736, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-32, scale=0.
077175
model is NHWC input fmt
model input height=512, width=512, channel=3
src width=2048 height=1024 fmt=0x1 virAddr=0x0x5583712f00 fd=0
dst width=512 height=512 fmt=0x1 virAddr=0x0x5582d12ae0 fd=0
color=0x0
rga_api version 1.10.1_[0]
rknn_run
rknn_run cost: 67.068 ms
src width=512 height=512 fmt=0x1 virAddr=0x0x5582dd2af0 fd=0
dst width=2048 height=1024 fmt=0x1 virAddr=0x0x5583712f00 fd=0
color=0x0
write_image path: ./result.png width=2048 height=1024 channel=3 data=0x5583712f00
```

Figure 6.13.1.5 Execute the board-end inference program of the ppseg model.

A result.png image will be generated in the current directory. It can be copied to the computer via network scp or USB drive for viewing.

Figure 6.13.1.6 result.png image

6.14 Testing the RetinaFace model

The model can be downloaded by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/RetinaFace/model
sh download_model.sh
```

Figure 6.14.1.1 Download the RetinaFace model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory under the RetinaFace routine in the rknn_model_zoo-2.0.0 folder.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/RetinaFace/python
```

If you are using the rk3588 development board, execute the following command to convert the RetinaFace_mobile320.onnx model.

```
python3 convert.py ../model/RetinaFace_mobile320.onnx rk3588
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0/examples/RetinaFace/python$ python3 convert.py ../model/RetinaFace_mobile320.onnx rk3588
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
I It is recommended onnx opset 19, but your onnx model opset is 9!
I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for successful convert!
I Loading : 100%|██████████████████████████████████████████████| 137/137 [00:00<00:00, 113786.07it/s]
done
--> Building model
W build: Resize [Upsample_66] convert to Deconvolution for inference speedup, but may cause result driftting.
To disable this optimization, please set 'optimization_level = 2' in 'rknn.config'
W build: Resize [Upsample_76] convert to Deconvolution for inference speedup, but may cause result driftting.
To disable this optimization, please set 'optimization_level = 2' in 'rknn.config'
W build: found outlier value, this may affect quantization accuracy
                    const name          abs_mean    abs_std     outlier value
                    onnx::Conv_613      0.69        0.83        21.961
I GraphPreparing : 100%|████████████████████████████████████████| 130/130 [00:00<00:00, 6237.74it/s]
I Quantizating : 100%|██████████████████████████████████████████| 130/130 [00:00<00:00, 1541.76it/s]
W build: The default input dtype of 'input0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'output0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '571' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn building done.
done
--> Export rknn model
done
```

Figure 6.14.1.2 Convert the RetinaFace model to a rknn model.

If you are using the rk3568 development board, execute the following command to convert the RetinaFace_mobile320.onnx model.

```
python3 convert.py ../model/RetinaFace_mobile320.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d RetinaFace
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ ./build-linux.sh -t rk3588 -a aarch64 -d RetinaFace
./build-linux.sh -t rk3588 -a aarch64 -d RetinaFace
/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=RetinaFace
BUILD_DEMO_PATH=examples/RetinaFace/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/install/rk3588_linux_aarch64/rknn_RetinaFace_demo
BUILD_DIR=/home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_RetinaFace_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/rknn_model_zoo-2.0.0/build/build_rknn_RetinaFace_demo_rk3588_linux_aarch64_Release
```

Figure 6.14.1.3 Compile the inference program of the RetinaFace model

After the compilation is completed, executable files and packaged models and other files will be generated in the install/rk3588_linux_aarch64/rknn_RetinaFace_demo directory under the modelzoo. Execute the following command to copy the model and executable files and other contents to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_RetinaFace_demo/ /userdata/aidemo
```

```
(python3.8-tk2-2.0) dominick@ubuntu:~/software/aidemo/rknn_model_zoo-2.0.0$ adb push install/rk3588_linux_aarch64/rknn_RetinaFace_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_RetinaFace_demo/: 5 files pushed. 1.0 MB/s (9598362 bytes in 9.133s)
```

Figure 6.14.1.4 Push the RetinaFace model and the inference program

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d RetinaFace
adb push install/rk3568_linux_aarch64/rknn_RetinaFace_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_RetinaFace_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use test.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_RetinaFace_demo
./rknn_retinaface_demo model/RetinaFace.rknn model/test.jpg
```

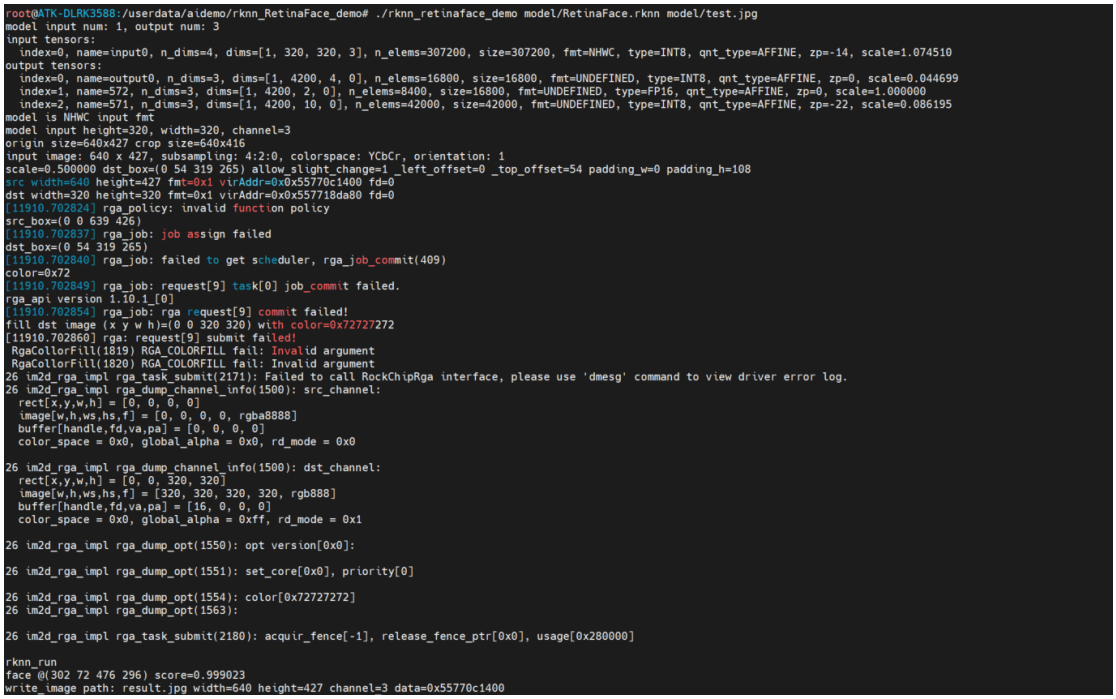


Figure 6.14.1.5 The RetinaFace model inference program is executed at the board end.

A "result.jpg" image will be generated in the current directory. It can be copied to the computer via network scp or USB drive for viewing.

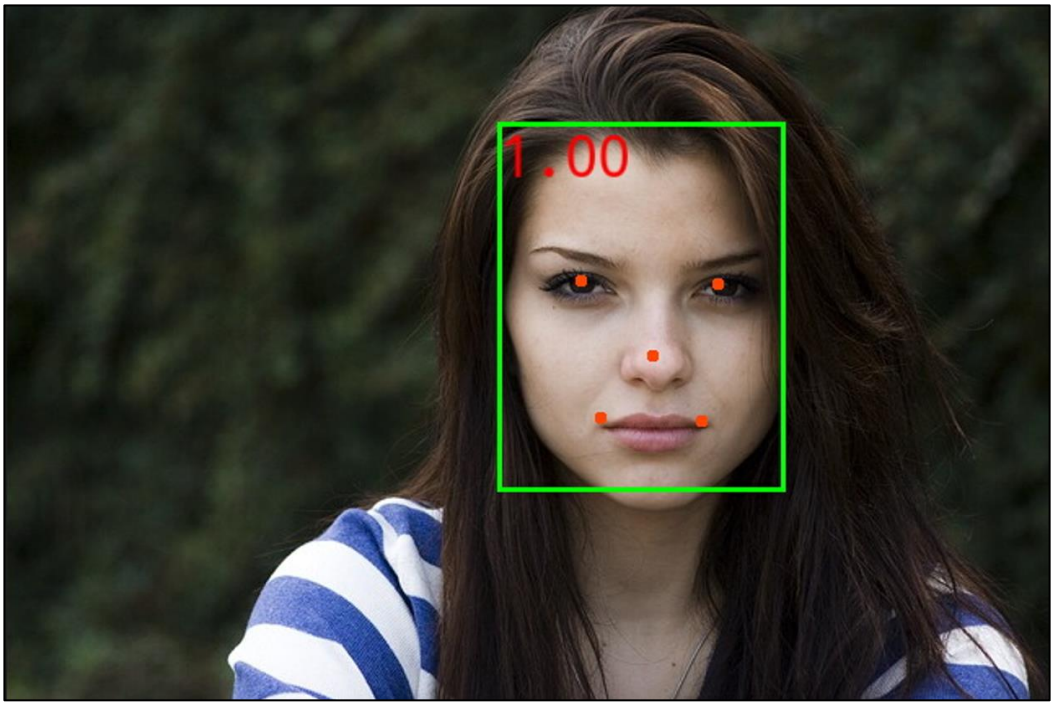


Figure 6.14.1.6 result.jpg image

6.15 Test PPOCR model

To download the model, execute the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following commands in the aidemo directory of Ubuntu to enter the PPOCR-Det and PPOCR-Rec directories and download the model.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/PPOCR/PPOCR-Det/model
sh download_model.sh
```

Figure 6.15.1.1 Download the ppocr-det model

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/PPOCR/PPOCR-Rec/model
sh download_model.sh
```

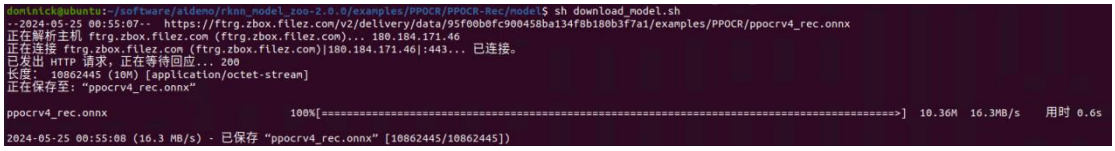


Figure 6.15.1.2 Download the ppocr-rec model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

First, enter the python directory under the PPOCR-Det routine in the rknn_model_zoo-2.0.0 folder.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/PPOCR/PPOCR-Det/python
```

First, convert the OCR detection model. If you are using the rk3588 development board, execute the following command to convert the ppocrv4_det.onnx model.

```
python3 convert.py ../model/ppocrv4_det.onnx rk3588
```

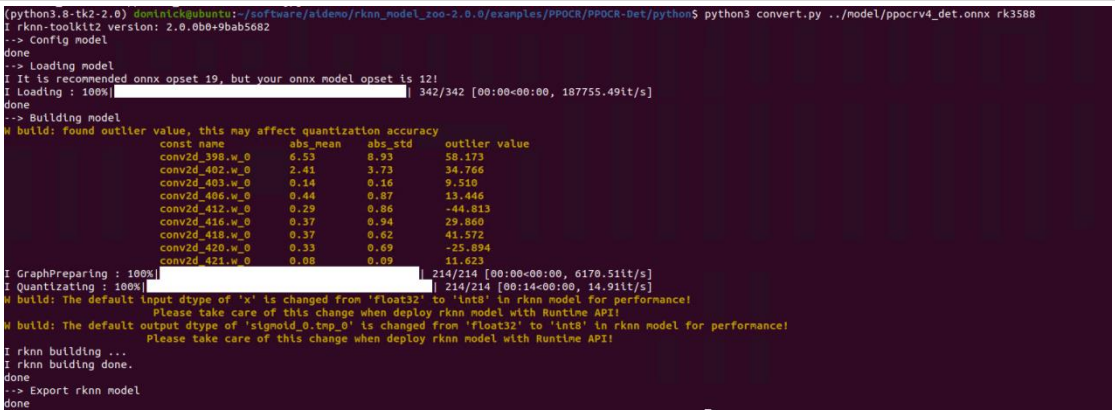


Figure 6.15.1.3 Convert the ppocr-det model to a rknn model.

If you are using the rk3568 development board, execute the following command to convert the ppocrv4_det.onnx model.

```
python3 convert.py ../model/ppocrv4_det.onnx rk3568
```

We also need to convert the OCR recognition model. If you are using the rk3588 development board, execute the following command to enter the directory of the conversion python script, and then convert the ppocrv4_rec.onnx model.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/PPOCR/PPOCR-Rec/python
python3 convert.py ../model/ppocrv4_rec.onnx rk3588
```

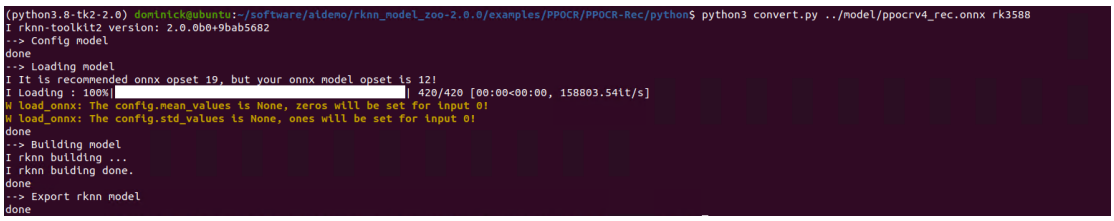


Figure 6.15.1.4 Convert the ppocr-rec model to a rknn model.

If you are using the rk3568 development board, execute the following command to convert the ppocrv4_rec.onnx model.

```
python3 convert.py ../model/ppocrv4_rec.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d PPOCR-System
```

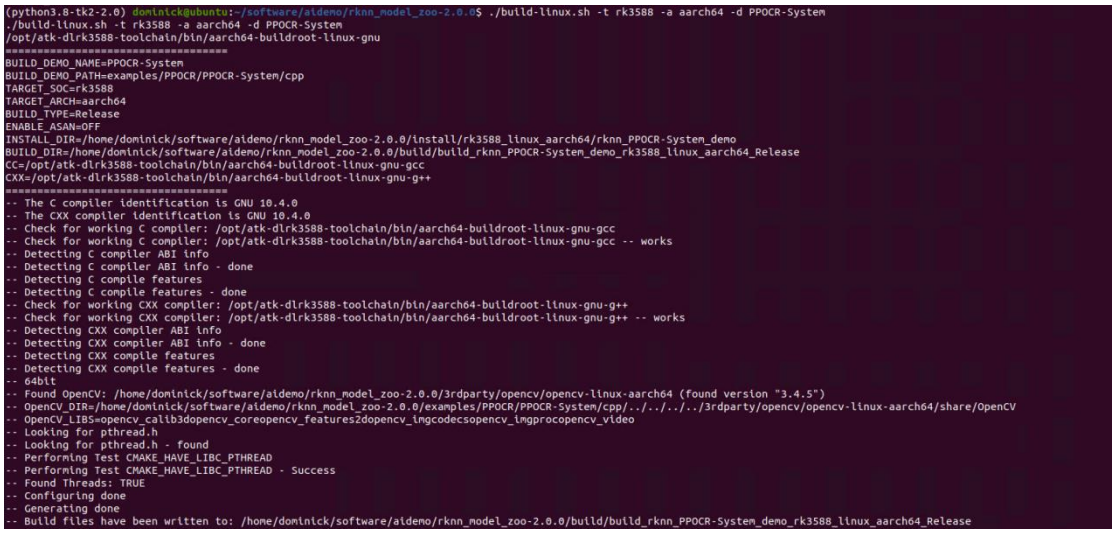


Figure 6.15.1.5 Compile the PPOCR board-end inference program.

After compilation is completed, executable files and packaged models, etc. will be generated in the install/rk3588_linux_aarch64/rknn_PPOCR-System_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_PPOCR-System_demo/ /userdata/aidemo
```

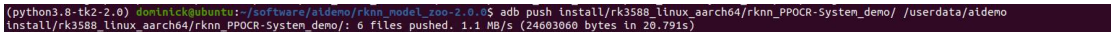


Figure 6.15.1.6 Push the ppocr model and the inference program.

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d PPOCR-System
adb push install/rk3568_linux_aarch64/rknn_PPOCR-System_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_PPOCR-System_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use test.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_PPOCR-System_demo
./rknn_ppocr_system_demo model/ppocrv4_det.rknn model/ppocrv4_rec.rknn model/test.jpg
```

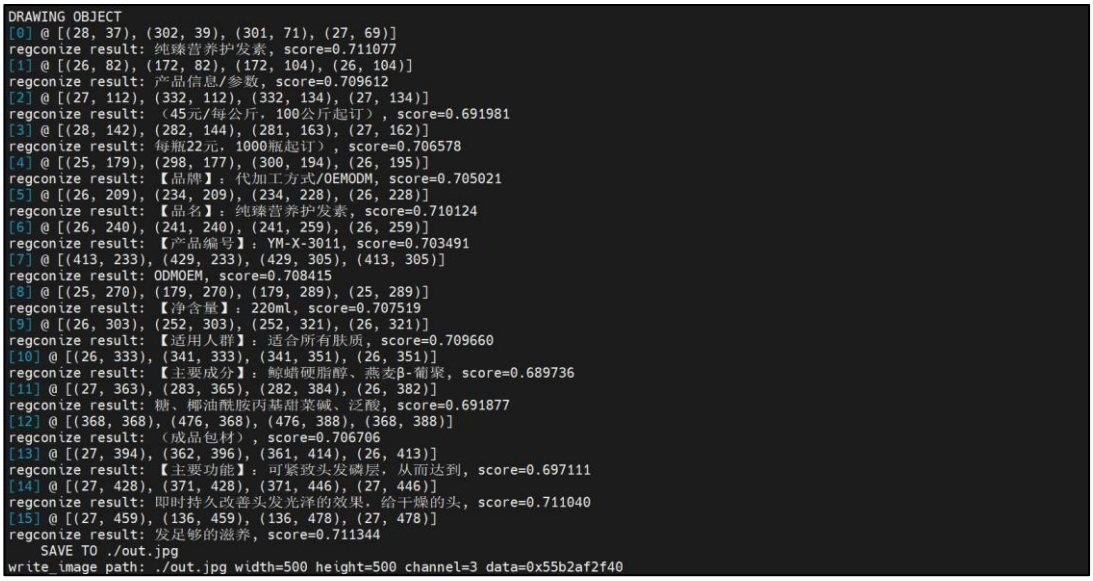


Figure 6.15.1.7 The development board's serial port terminal displays the text recognition information of ppocr.

The serial port information will print out the recognized text and location information. An out.jpg image will be generated in the current directory. It can be copied to a computer via network scp or via a USB drive for viewing.

6.16 Test the LPRNet model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/LPRNet/model
sh download_model.sh
```

Figure 6.16.1.1 Download the LPRNet model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory of the LPRNet routine under the rknn_model_zoo-2.0.0 directory.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/LPRNet/python
```

If you are using the rk3588 development board, execute the following command to convert the onnx model.

```
python3 convert.py ../model/lprnet.onnx rk3588
```

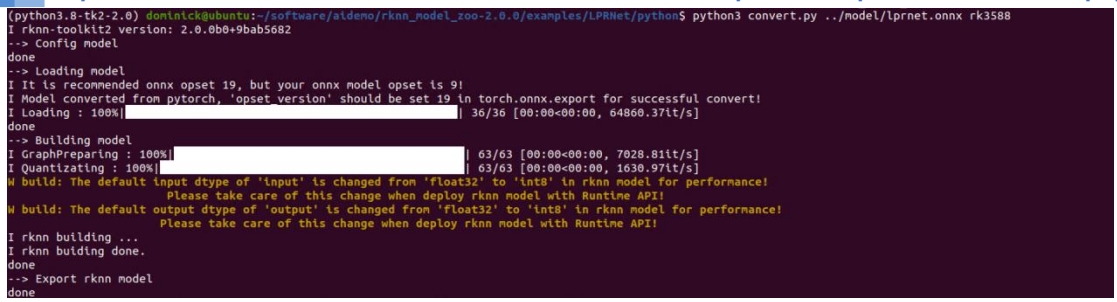


Figure 6.16.1.2 Convert the LPRNet model to a rknn model.

If you are using the rk3568 development board, execute the following command to convert the lprnet.onnx model.

```
python3 convert.py ../model/lprnet.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d LPRNet
```

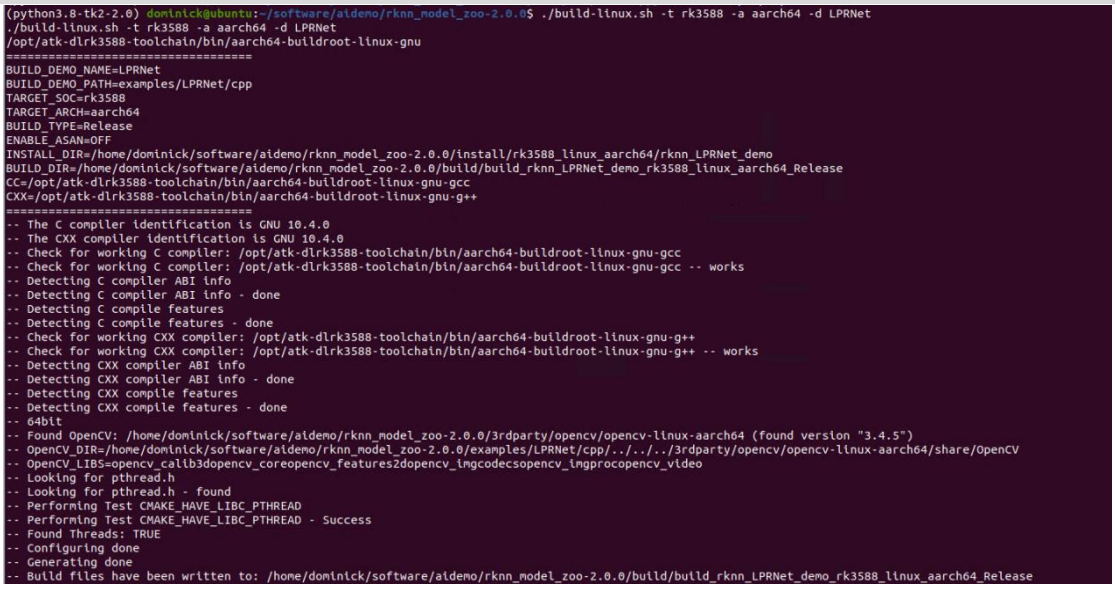


Figure 6.16.1.3 Compile the inference program for the LPRNet model on the board.

After the compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_LPRNet_demo directory under the modelzoo. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_LPRNet_demo/ /userdata/aidemo
```

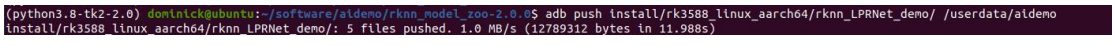


Figure 6.16.1.4 Push the LPRNet model and inference program to the development board.

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
```

```
./build-linux.sh -t rk3568 -a aarch64 -d LPRNet
adb push install/rk3568_linux_aarch64/rknn_LPRNet_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_LPRNet_demo under the directory pushed to the development board, and execute the following command in the development board terminal. Use test.jpg for inference. The execution result is as shown in the following figure.

```
cd /userdata/aidemo/rknn_LPRNet_demo
./rknn_lprnet_demo model/lprnet.rknn model/test.jpg
```

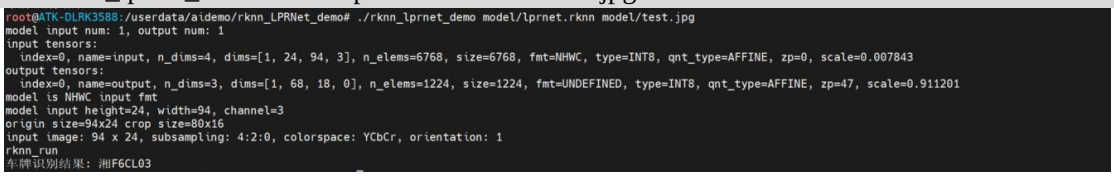


Figure 6.16.1.5 The LPRNet inference program is executed at the board end.

The recognized license plate number will be printed in the last line.

```
License plate recognition result: 湘 F6CL03
```

6.17 Test the lite_transformer model

Download the model by executing the model download script. Before execution, it is necessary to first check if the network of Ubuntu is normal. Execute the following command in the aidemo directory of Ubuntu.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/lite_transformer/model
sh download_model.sh
```

Figure 6.17.1.1 Download the lite_transformer model

If the current terminal is not in the conda environment, open the Ubuntu terminal and execute the following command to enter the conda environment we previously created.

```
conda activate python3.8-tk2-2.0
```

Enter the python directory under the lite_transformer routine in the rknn_model_zoo-2.0.0 folder.

```
cd ~/software/aidemo/rknn_model_zoo-2.0.0/examples/lite_transformer/python
```

If you are using the rk3588 development board, execute the following commands to convert the lite-transformer-decoder-16.onnx model and the lite-transformer-encoder-16.onnx model.

```
python3 convert.py ../model/lite-transformer-decoder-16.onnx rk3588
python3 convert.py ../model/lite-transformer-encoder-16.onnx rk3588
```

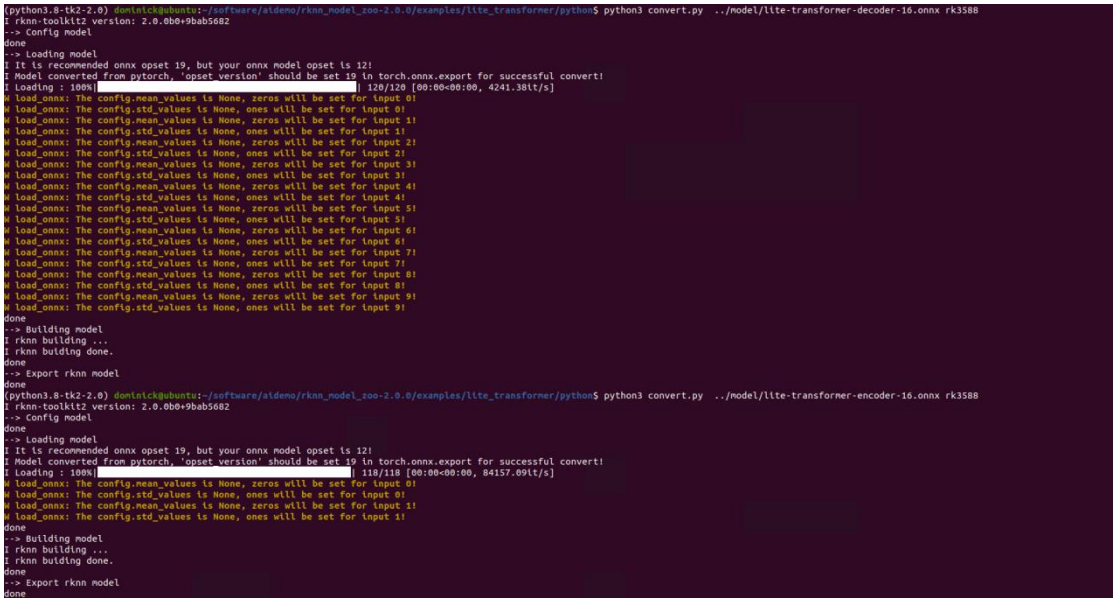


Figure 6.17.1.2 Convert the model to the rknn format

If you are using the rk3568 development board, execute the following commands to convert the lite-transformer-decoder-16.onnx model and the lite-transformer-encoder-16.onnx model.

```
python3 convert.py ../model/lite-transformer-decoder-16.onnx rk3568
python3 convert.py ../model/lite-transformer-encoder-16.onnx rk3568
```

After the conversion is completed, you need to first compile the routines in the C++ directory, and then push the compiled folder to the development board. Enter the initial directory of modelzoo, and first enable the compiler. Execute the following command to compile the routines.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3588 -a aarch64 -d lite_transformer
```

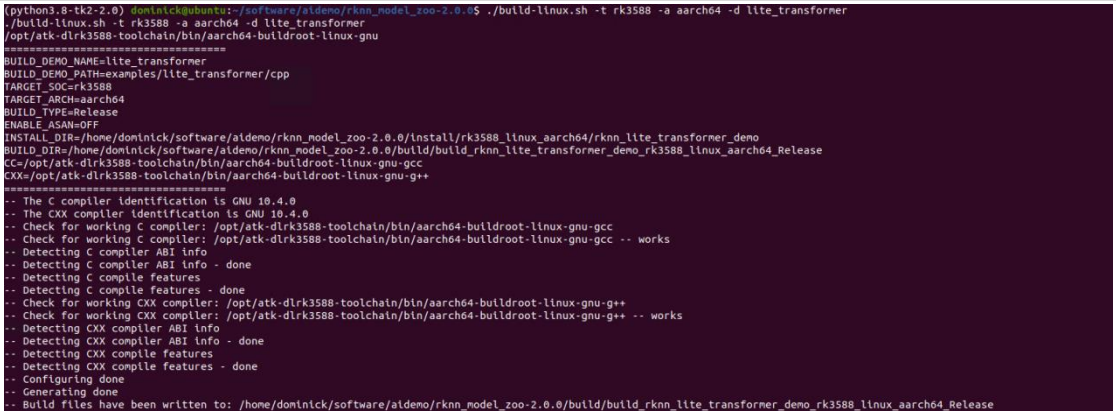


Figure 6.17.1.3 Compile the inference program on the board

After compilation is completed, executable files and packaged models will be generated in the install/rk3588_linux_aarch64/rknn_lite_transformer_demo directory under the modelzoo folder. Execute the following command to copy the model and executable files, etc. to the development board.

```
adb push install/rk3588_linux_aarch64/rknn_lite_transformer_demo/ /userdata/aidemo
```

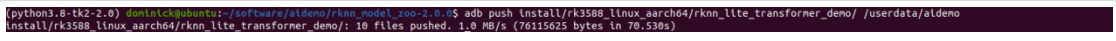


Figure 6.17.1.4 Push the board-end reasoning program and model

If the rk3568 development board is used, execute the following command to compile the routine and push it to the development board.

```
cd ../../../
export GCC_COMPILER=/opt/atk-dlrk3568-5_10_sdk-toolchain/bin/aarch64-buildroot-linux-gnu
./build-linux.sh -t rk3568 -a aarch64 -d lite_transformer
adb push install/rk3568_linux_aarch64/rknn_lite_transformer_demo/ /userdata/aidemo
```

Open the serial port terminal of the development board, enter the directory /userdata/aidemo/rknn_lite_transformer_demo under the directory pushed to the development board, and execute the following commands in the development board terminal for inference. The execution result is as shown in the figure below.

```
cd /userdata/aidemo/rknn_lite_transformer_demo
./rknn_lite_transformer_demo model/lite-transformer-encoder-16.rknn model/lite-transformer-decoder-16.rknn
```

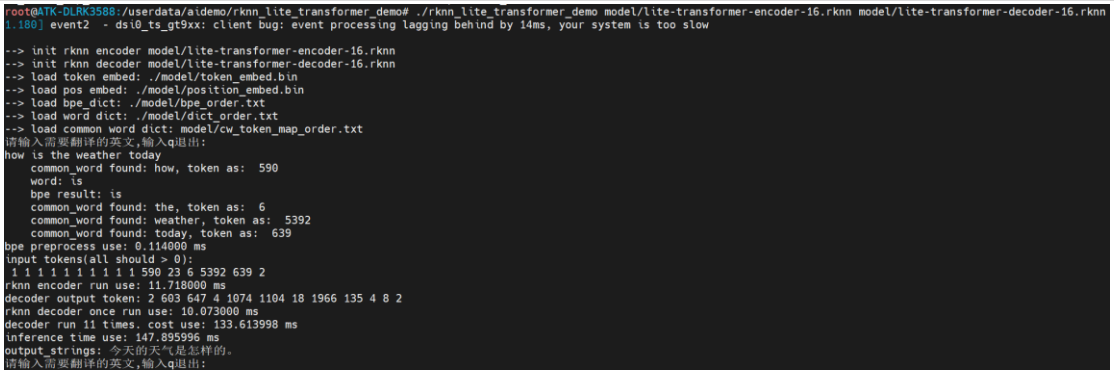


Figure 6.17.1.5 Execute the board-end reasoning program

After testing, it was found that the translation of some words or phrases was not very good. Some words and phrases were translated incorrectly. The explanation of the RENESAS micro's model zoo is that the training dataset is too small. If you are interested, you can try using a larger dataset to train the model and then convert it to RKNN for reasoning to see if there are any different effects.

Chapter 7. Classic classification network model + RKNN AI routine

This chapter will explain the routines for testing various classification models trained using the animal dataset on the development board. The following routine is adopted.

This chapter will be divided into the following sections:

1. Routine introduction
2. Routine test for self-training alexnet model
3. Routine test for self-training vggnet model
4. Routine test for resnet model trained using transfer learning
5. Routine test for mobilenet_v1 model trained using transfer learning

7.1 Routine Overview

AlexNet is a classic convolutional neural network in the field of deep learning, proposed by Alex Krizhevsky et al. in 2012. It made a significant breakthrough in the ImageNet image classification competition, marking the rise of deep learning. AlexNet utilized deep convolutional and pooling layers to extract image features and introduced the ReLU activation function to alleviate the problem of gradient disappearance. Its depth and parameter quantity were innovative at that time, stimulating the research craze of deep neural networks. It also was the first to use GPU for training, accelerating the deep learning training process. The success of AlexNet inspired the development of subsequent network architectures, such as VGG and ResNet.

Paper Link: http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf

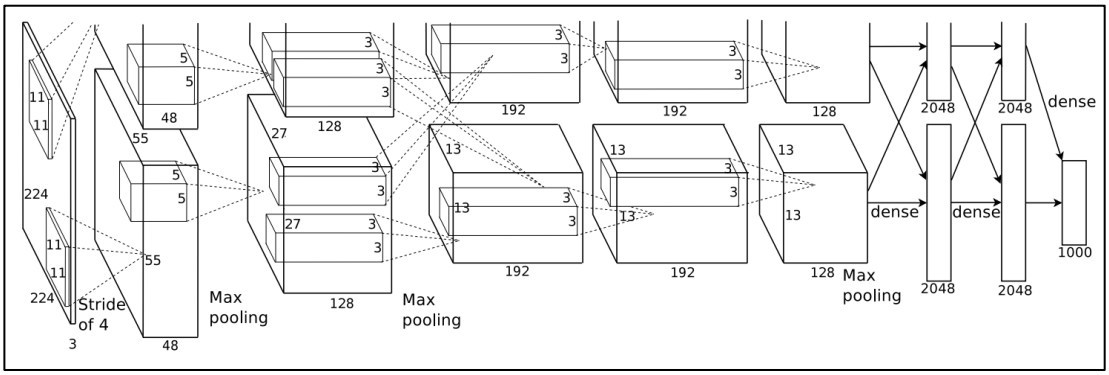


Figure 7.1.1.1 AlexNet framework

VGGNet is a classic convolutional neural network architecture, proposed by the research team from the University of Oxford in 2014. VGGNet is renowned for its simple yet profound design principles, constructing deep networks by stacking multiple smaller convolutional layers and pooling layers. Its core idea is to extract higher-level features of images by increasing the depth of the network, thereby achieving better performance. The architecture of VGGNet is very regular, with all convolutional layers using the same-sized convolution kernels, making the design of the network more consistent. Although VGGNet achieved excellent results in the ImageNet image classification competition, its large number of parameters led to high computational costs for training and inference.

Paper link: https://arxiv.org/pdf/1409.1556.pdf

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Figure 7.1.1.2 The VGGNet framework

ResNet (Residual Network) is a significant milestone in the field of deep learning, proposed by Kaiming He et al. in 2015. Its core innovation lies in introducing residual blocks to address the problems

of vanishing gradients and training difficulties, allowing the construction of extremely deep neural networks. Residual blocks transmit information through skip connections, making the network more amenable to optimization. ResNet can flexibly stack multiple residual blocks, creating models of different depths. Its performance surpasses previous networks, achieving remarkable achievements in tasks such as image classification, object detection, and segmentation. The idea of ResNet also influenced subsequent network designs, providing a powerful solution to address the challenges of deep learning.

MobileNet-v1 is a lightweight convolutional neural network designed to achieve efficient image classification and feature extraction on resource-constrained mobile devices and embedded systems. By employing depthwise separable convolution and a carefully designed structure, MobileNet-v1 significantly reduces computational complexity and model size while maintaining reasonable performance. It allows users to adjust the network width and input image resolution to adapt to different application scenarios. MobileNet-v1 is applicable to tasks such as image classification, object detection, and face recognition, providing an effective solution for deep learning applications on embedded systems and mobile devices, and promoting the development of lightweight neural networks.

We trained these deep learning introductory must-know classification network models using TensorFlow and converted them into rknn models that can be used for inference on the ALIENTEK rk3588 development board. All four networks were trained using the 10-classification of the animal dataset, and the categories of the animal dataset from 0 to 9 are butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, and squirrel.

The animal dataset can be downloaded on Kaggle:

https://www.kaggle.com/datasets/alessiocorrado99/animals10

7.2 Testing the self-training AlexNet model routine

Next, I will introduce to you how the self-trained Alexnet model can run on the board. This section explains how to transfer the routine to the development board for running. If you are using the rk3588 development board, we will copy the files at the A drive of the development board disc - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 03_alexnet (including atk_rknn_alexnet_demo.zip) to Ubuntu. If you are using the rk3568 development board, we will copy the files at the A drive of the development board disc - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 03_alexnet (including atk_rknn_alexnet_demo.zip) to Ubuntu. Unzip the atk_rknn_alexnet_demo.zip archive, and enter the corresponding directory. Before compiling, please make sure that your routine has been installed and updated the cross-compiler and the NPU service as described in Chapter 2 and Chapter 3. Open the terminal and execute the following commands to compile the program.

```
./build.sh
```

After the compilation is completed, execute the following command to push the compiled program and model to the development board via adb.

```
adb push install/atk_alexnet_demo/ /userdata/aidemo
```

Open the serial port terminal, and then go to the directory where the corresponding routine executable file is located and execute the following command.

```
./atk_alexnet ./model/animal-alexnet.rknn
```

Align the printed test image with the camera and make it display as much as possible to cover all the screen areas. You will find that the recognition effect of AlexNet is relatively average. Using other models will yield better results, such as MobileNet-v1.

7.3 Self-training VggNet Model Routine Test

This section explains how to transfer the routine to the development board for running. If you are using the rk3588 development board, we will copy the atk_rknn_vggnet_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 01_Source_Code → 04_vggnet to Ubuntu. If you are using the rk3568 development board, we will copy the atk_rknn_vggnet_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → **01_codes** → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 04_vggnet to Ubuntu. Decompress the atk_rknn_vggnet_demo.zip archive, and enter the corresponding directory. Before compiling, be sure to check that your routine has been installed with the cross-compiler and updated the NPU service as described in Chapter 1 and Chapter 2. Open the terminal and execute the following commands to compile the program.

```
./build.sh
```

After the compilation is completed, execute the following command to push the compiled program and model to the development board via adb.

```
adb push install/atk_vggnet_demo/ /userdata/aidemo
```

Open the serial port terminal, and then go to the directory where the corresponding routine executable file is located and execute the following command.

```
./atk_vggnet ./model/animal-vggnet16.rknn
```

It can be observed that the performance will be slightly better than that of Alexnet, but it will be slower than Alexnet.

7.4 Example routine testing of the resnet model trained using transfer learning

This section explains how to transfer the routine to the development board for running. If you are using the rk3588 development board, we will copy the atk_rknn_resnet_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 05_transfer_resnet into Ubuntu. If you are using the rk3568 development board, we will copy the atk_rknn_resnet_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 05_transfer_resnet into Ubuntu. Decompress the atk_rknn_resnet_demo.zip archive, and enter the corresponding directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service. Open the terminal and execute the following commands to compile the program.

```
./build.sh
```

After the compilation is completed, execute the following command to push the compiled program and model to the development board via adb.

```
adb push install/atk_resnet_demo/ /userdata/aidemo
```

Open the serial port terminal, and then go to the directory where the corresponding routine executable file is located and execute the following command.

```
./atk_resnet ./model/transferlearn-resnet50.rknn
```

7.5 Testing the mobilenet_v1 model routine trained by transfer learning

This section explains how to transfer the routine to the development board for running. If you are using the rk3588 development board, we will copy the atk_rknn_mobilenet_v1_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 06_transfer_mobilenet_v1 into Ubuntu. If you are using the rk3568 development board, then copy the atk_rknn_mobilenet_v1_demo.zip file from the A drive of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 06_transfer_mobilenet_v1 into Ubuntu. Decompress the atk_rknn_mobilenet_v1_demo.zip archive, and enter the corresponding directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service. Open the terminal and execute the following commands to compile the program.

```
./build.sh
```

After the compilation is completed, execute the following command to push the compiled program and model to the development board via adb.

```
adb push install/atk_mobilenet_v1_demo/ /userdata/aidemo
```

Open the serial port terminal, and then go to the directory where the corresponding routine executable file is located and execute the following command.

```
./atk_mobilenet_v1 ./model/transferlearn-mobilnetv1.rknn
```

This classification network routine is supposed to be the best. It can quickly identify the categories, and the model is also relatively small, fast, and highly recommended for everyone to use this classification model for related applications.

Figure 7.5.1.1 Identified the puppy

Chapter 8. YOLO series object detection model + RKNN AI routine

The task of object detection is to identify all the interesting targets or objects in an image, determine their categories and positions. It is one of the core issues in the field of computer vision. Due to the different appearances, shapes, and postures of various objects, as well as the interference of factors such as lighting and occlusion during imaging, object detection has always been the most challenging problem in the field of computer vision. This chapter explains how to run the most commonly used and popular yolo series model on the ALIENTEK RK3568/RK3588 development board through the camera.

This chapter will be divided into the following sections:

1. Introduction to the yolo series routine
2. yolov5 routine for camera inference
3. yolov6 routine for camera inference
4. yolov7 routine for camera inference
5. yolov8 routine for camera inference
6. yolox routine for camera inference
7. ppyoloe routine for camera inference

8.1 Introduction to the YOLO series routines

YOLOv5 is one of the latest versions of the YOLO object detection algorithm series. As an important advancement in the field of computer vision, YOLOv5 has achieved significant performance improvements in object detection. Compared to previous versions, YOLOv5 adopts a single-stage detection strategy, enabling the localization and classification of objects through a single forward pass. It incorporates the "CSPDarknet53" architecture as the base network to extract rich feature information, thereby effectively enhancing the accuracy of detection. YOLOv5 has also made improvements in multi-scale detection, capable of detecting objects of different scales, ranging from small to large. This performs well in handling diverse scenarios. Additionally, data augmentation techniques are widely applied, through random cropping, color transformation, etc., to increase the diversity of training data and enhance the generalization ability of the model. In terms of computational efficiency, YOLOv5 maintains high detection performance while optimizing the size and speed of the model, making it suitable for embedded devices, mobile terminals, and real-time applications. Its open-source nature enables researchers and developers to freely access the code and pre-trained models for research and customization. In summary, YOLOv5 through single-stage detection, efficient network architecture, multi-scale strategy, and data augmentation technologies, significantly improves the performance of object detection. It has broad application prospects in computer vision, autonomous driving, security monitoring, etc., bringing new solutions to real-time object detection tasks.

For more details on training and other information, please refer to the source code link of YOLOv5: https://github.com/ultralytics/yolov5.

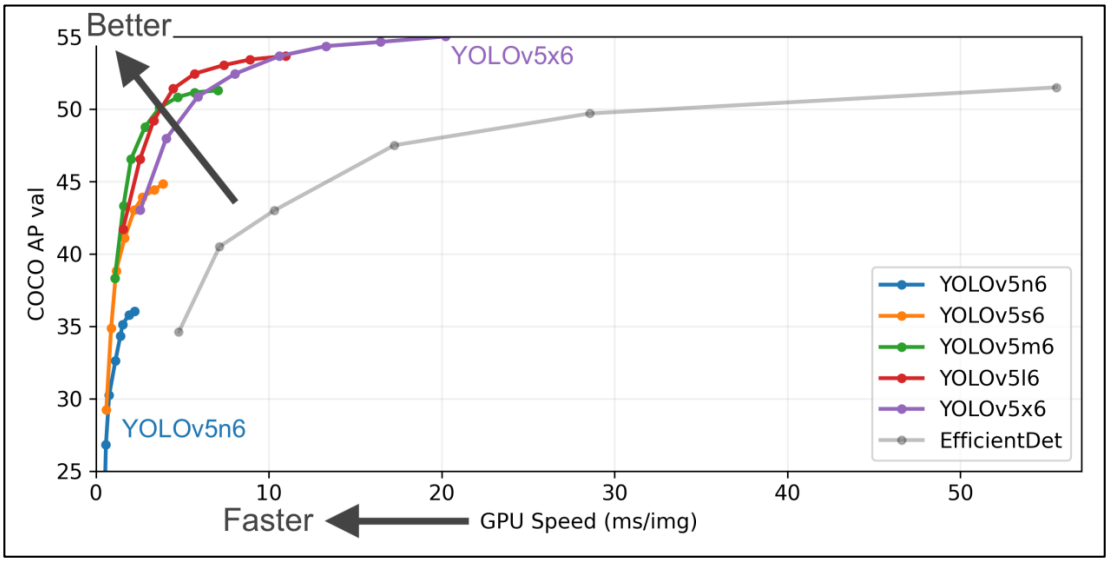


Figure 8.1.1.1 Performance Comparison of yolov5 Inference

YOLOv6 is a target detection framework developed by the Visual Intelligence Department of Meituan, focusing on industrial applications. This framework maintains high detection accuracy while also considering inference efficiency. Among them, YOLOv6-nano achieves an accuracy of 35.0% AP on the COCO dataset and an inference speed of 1242 FPS; while YOLOv6-s achieves an accuracy of 43.1% AP and an inference speed of 520 FPS. YOLOv6 supports deployment on various platforms, including GPU, CPU, and ARM, greatly simplifying the adaptation work for engineering deployment. Its unique network structure and training strategy enable YOLOv6 to outperform other algorithms of the same scale in both accuracy and speed, and is suitable for various real-time application scenarios.

YOLOv6, as an advanced object detection framework, has the following notable advantages:

High Performance: YOLOv6 maintains high detection accuracy while also considering inference efficiency. In terms of both accuracy and speed, YOLOv6 demonstrates outstanding performance and can meet the requirements of various real-time application scenarios. By optimizing the network structure and algorithm, YOLOv6 achieves faster inference speed while maintaining high accuracy, enabling the model to respond and process data more quickly in practical applications.

Cross-platform compatibility: YOLOv6 supports deployment on multiple platforms, including GPU, CPU, and ARM, etc. This enables it to run on different hardware devices and perform exceptionally well.

This cross-platform compatibility greatly simplifies the adaptation work for engineering deployment, allowing users to more conveniently deploy the model in various application scenarios.

Easy to use: The code structure of YOLOv6 is clear, making it easy to understand and reproduce. This provides users with a good user experience.

It supports the PyTorch framework, which enables users to more easily utilize the tools and libraries in the PyTorch ecosystem for model development and deployment. YOLOv6 also provides abundant documentation and sample code to help users quickly get started and solve practical problems.

Advanced algorithm design: YOLOv6 adopts a variety of advanced algorithm designs, such as the improvement of anchor boxes and the optimization of feature fusion technology, etc. These designs enable the model to better adapt to various complex scene detection tasks. YOLOv6 also introduces multiple training strategies and techniques, such as data augmentation and model pruning, to further enhance the performance and generalization ability of the model.

Continuous updates and maintenance: As an open-source project, YOLOv6 has been continuously updated and maintained by the AI team of Meituan and other developers. This ensures that users can obtain the latest optimizations and improvements, keeping the model in the best condition at all times.

The AI team of Meituan will also continuously improve and optimize the functions and performance of YOLOv6 based on user feedback and the requirements of actual application scenarios.

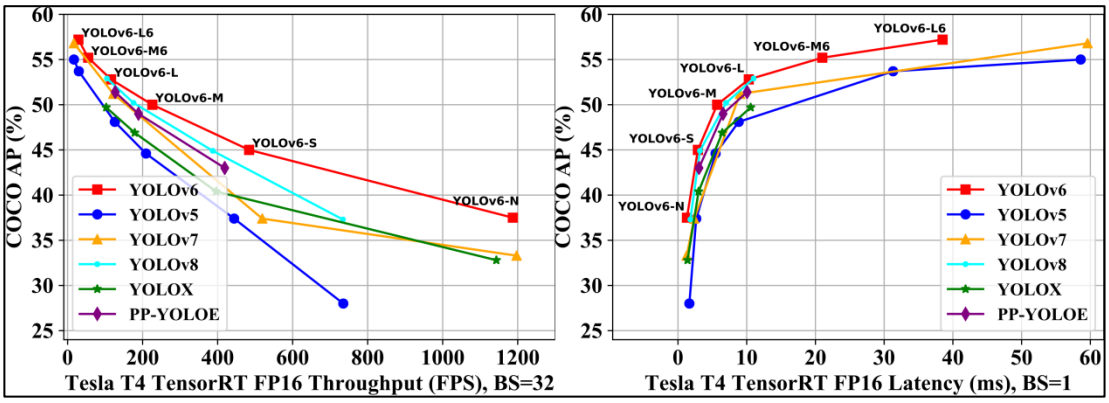


Figure 8.1.1.2 Comparison of Inference Performance of Different Sizes of Models in YOLOv6

Paper address of YOLOv6:

https://arxiv.org/abs/2301.05586

https://arxiv.org/abs/2209.02976

YOLOv6 Chinese Manual: https://yolov6-docs.readthedocs.io/zh-cn/latest/

YOLOv7 is a target detection algorithm, being the latest version of the YOLO series. YOLOv7 is based on the YOLOv5 algorithm and incorporates some new technologies and optimizations, further enhancing the detection performance and speed. Compared to previous versions, YOLOv7 has better accuracy and faster inference speed, and also supports more application scenarios.

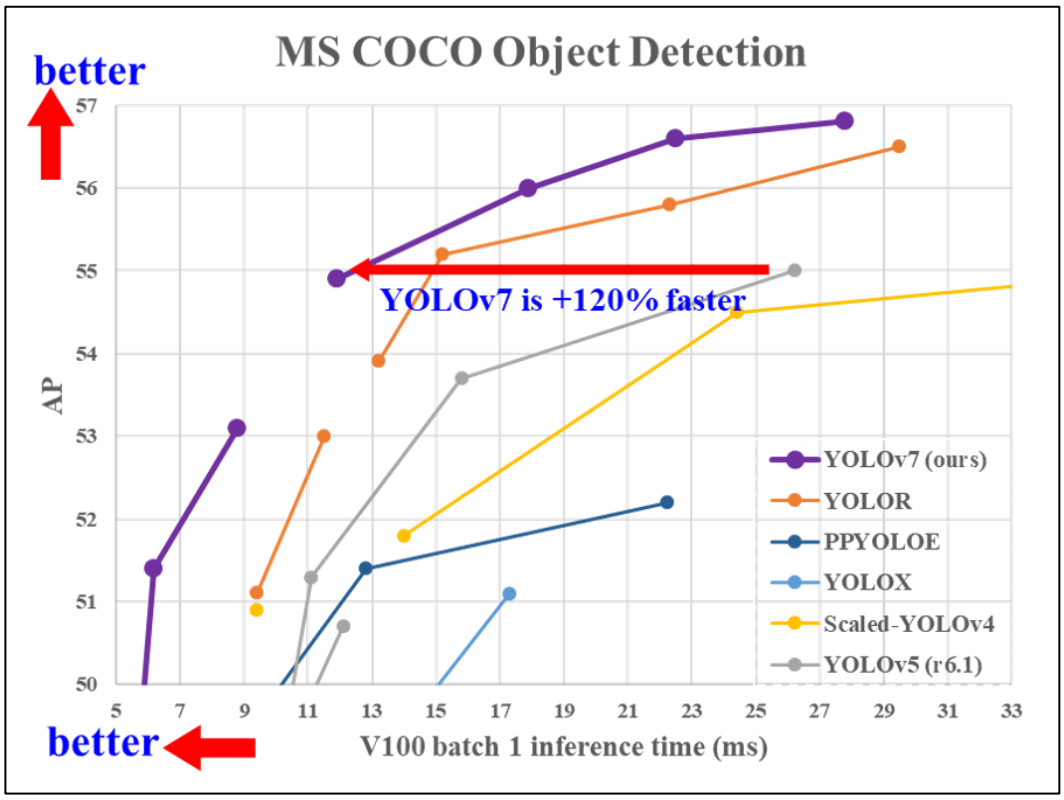


Figure 8.1.1.3 Performance Comparison of YOLOv7 Inference

Paper Link: https://arxiv.org/abs/2207.02696

Code Link: https://github.com/WongKinYiu/yolov7

YOLOv8 is the latest iteration of the YOLO (You Only Look Once) series. It performs exceptionally well in the field of real-time object detection, incorporating multiple innovations and improvements. YOLOv8 is the next major update version of YOLOv5, which was open-sourced by Ultralytics on January 10, 2023. It aims to achieve real-time object detection while maintaining high accuracy and speed. Compared to previous models, YOLOv8 further improves detection accuracy by introducing new features and optimizations while maintaining real-time performance.

Key Features and Advantages:

Real-time performance: YOLOv8 inherits the real-time detection feature of the YOLO series. Even on lower hardware configurations, it can achieve a high frame rate (FPS).

High accuracy: Through a deeper and more complex network structure and improved training techniques, YOLOv8 not only maintains high speed but also significantly improves the accuracy of detection.

Multi-scale prediction: An improved multi-scale prediction technology has been introduced, which can better detect objects of different sizes.

Adaptive anchor boxes: The new version has optimized the adaptive adjustment of anchor boxes, enabling more accurate prediction of the position and size of objects.

Powerful feature extractor: A more advanced feature extraction network has been used, which helps extract richer and more distinguishable features from the image.

YOLOv8 adopts a new backbone network architecture to enhance feature extraction and object detection performance. Compared with the anchor-based methods, the anchor-free detection head used in YOLOv8 helps improve the accuracy and efficiency of the detection process. YOLOv8 introduces a new loss function to better optimize the model parameters during training. Currently, YOLOv8 supports image classification, object detection, and instance segmentation tasks. It can run on various hardware platforms ranging from CPU to GPU, performing exceptionally well. Ultralytics positions this library as an algorithm framework rather than a specific algorithm, supporting non-YOLO models as well as various tasks such as classification, segmentation, and pose estimation. Test results on the COCO Val 2017 dataset show that YOLOv8 has a significant improvement in accuracy compared to YOLOv5, but the corresponding parameter size and FLOPs have also increased. YOLOv8 provides pre-trained models of different sizes (such as YOLOv8n/s/m/l/x) to meet various task and performance requirements.

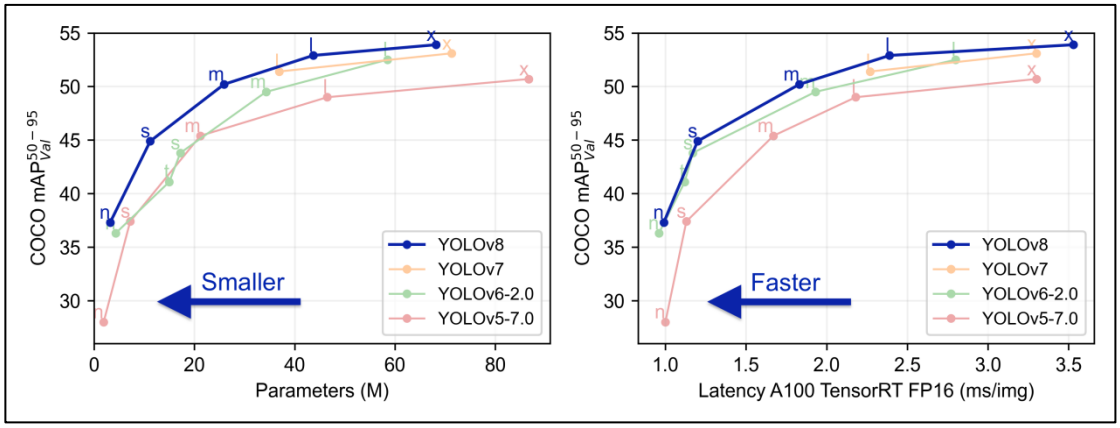


Figure 8.1.1.4 Performance Comparison of YOLOv8 Inference

GitHub website of YOLOV8: https://github.com/ultralytics/ultralytics

Documentation of YOLOV8: https://docs.ultralytics.com/zh#yolo-licenses-how-is-ultralytics-yolo-licensed

YOLOX is a high-performance and real-time object detection algorithm, dedicated to pushing the performance limit in the field of object detection. Its innovative design and optimization have achieved a better balance between speed and accuracy, making it a new generation leader in the field of object detection. YOLOX performs well in multi-scale detection and small-scale object detection, and is suitable for various scenarios such as intelligent transportation, security monitoring, industrial vision, etc. Its outstanding performance makes it an ideal choice for real-time object detection tasks. As an open-source project, YOLOX provides code and pre-trained models, allowing researchers and

developers to freely access and use them. It brings innovation to the field of object detection and makes significant contributions to the development of computer vision. Whether in terms of speed, accuracy, or flexibility, YOLOX demonstrates remarkable potential and brings new possibilities for real-time object detection tasks.

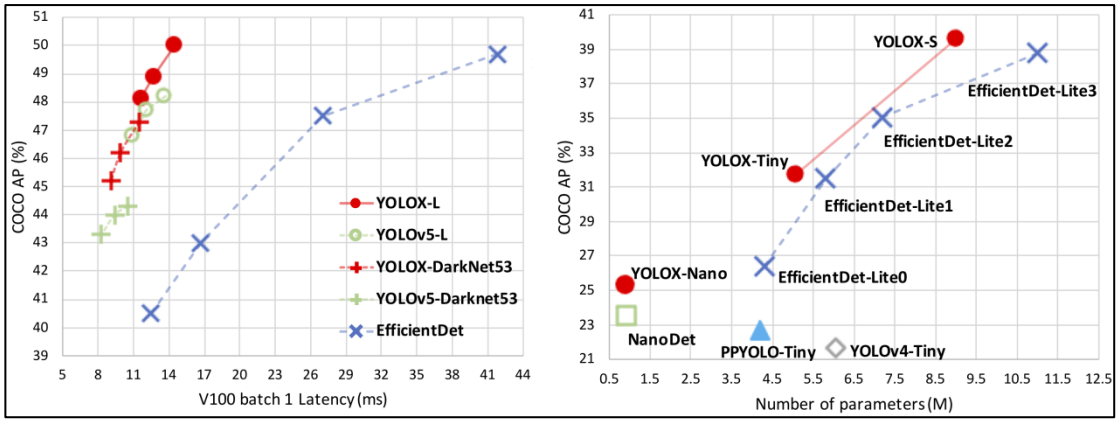


Figure 8.1.1.5 YOLOX Performance Data Comparison

The GitHub website of YOLOX: https://github.com/Megvii-BaseDetection/YOLOX

The documentation of YOLOX: https://yolox.readthedocs.io/en/latest/

Ricewave's Modelzoo provides performance and accuracy test results data for some YOLO models on the RKNN platform, as well as related information such as the data sets used in the routines. Please refer to the link: https://github.com/airockchip/rknn_model_zoo/tree/main.

PP-YOLOE is an outstanding single-stage Anchor-free model based on PP-YOLOv2, surpassing many popular YOLO models. PP-YOLOE has a series of models, namely s/m/l/x, which can be configured through width multiplier and depth multiplier. PP-YOLOE avoids using special operators such as Deformable Convolution or Matrix NMS to enable its easy deployment on various diverse hardware. PP-YOLOE+_l achieved an mAP of 53.3 on the COCO test-dev2017, and its speed reached 78.1 FPS on Tesla V100.

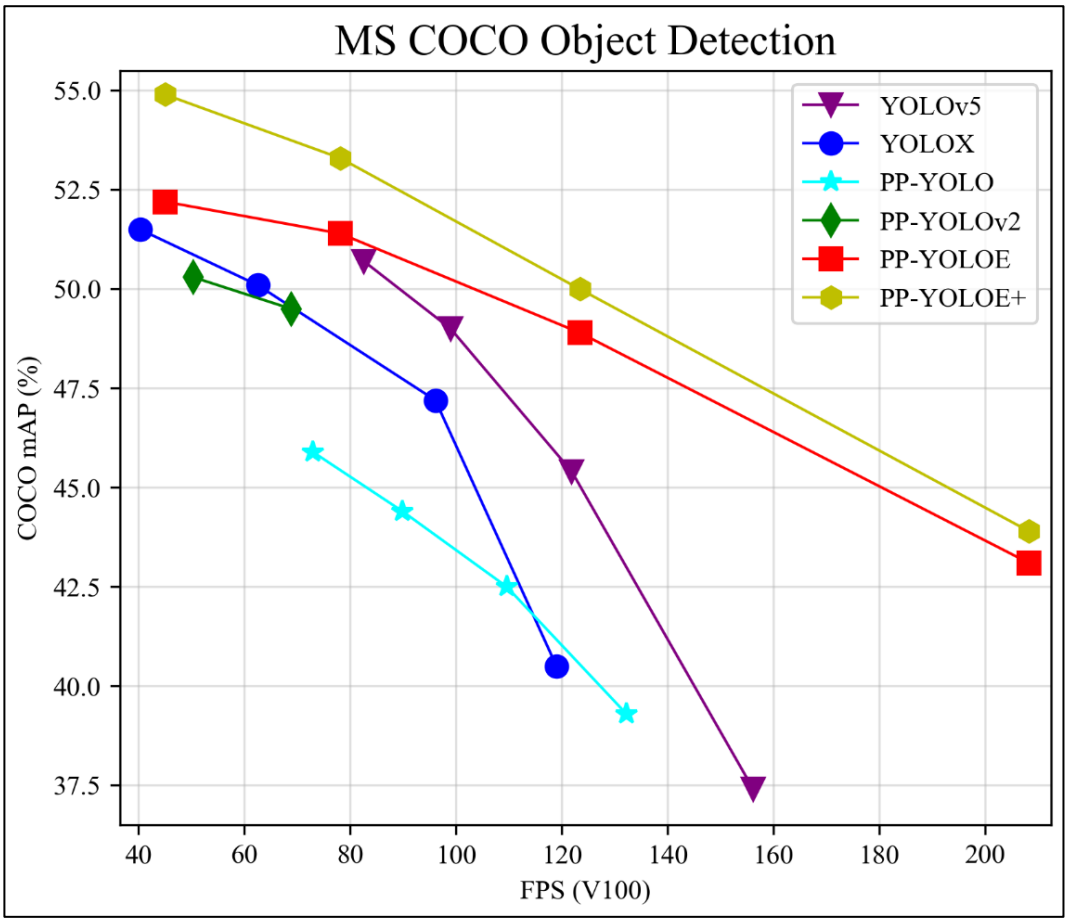


Figure 8.1.1.5 Performance Data Comparison of PPYOLOE

The Chinese introduction of PP-YOLOE on the official website: https://github.com/PaddlePaddle/PaddleDetection/blob/release/2.5/configs/ppyoloe/README_cn.md

8.2 YOLOv5 Routine Camera Inference

Before conducting the test, if the serial port terminal of the development board is not closed and the Qt interface is not shut down, it is recommended to first close the Qt interface. Then, execute the following commands in the development board terminal.

```
/etc/init.d/S50systemui stop
```

This section explains how to run the YOLOv5 camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the RK3588 development board, we will copy the yolov5.zip file from the 01 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 07_yolov5 folder to Ubuntu. If you are using the RK3568 development board, we will copy the yolov5.zip file from the 04 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 07_yolov5 folder to Ubuntu. Unzip the yolov5.zip archive and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service

on the development board. Open the terminal and execute the following commands to compile the program.

```
./build-linux.sh
```

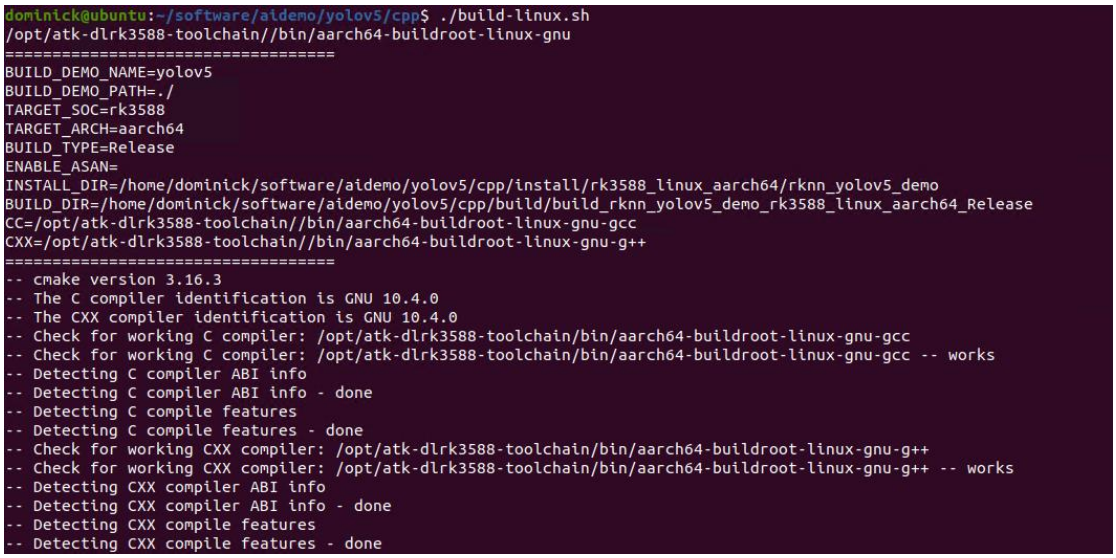


Figure 8.2.1.1 Compile the yolov5 routine

If it is the rk3588 development board, the compiled program and model should be placed in the "install" directory. Push the compiled program and model to the development board via adb. On the RK3588 development board, execute the following command.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov5_cam/ /userdata/aidemo
```

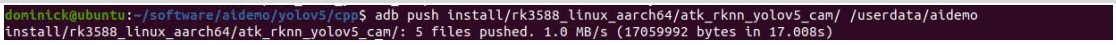


Figure 8.2.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov5_cam/ /userdata/aidemo
```

On the development board, open the serial terminal and navigate to the directory /userdata/aidemo/atk_rknn_yolov5_demo, then execute the following command.

```
cd /userdata/aidemo/atk_rknn_yolov5_cam
./rknn_yolov5_cam model/yolov5.rknn
```

The serial port information for the normal execution of the reasoning program is shown in the following figure.

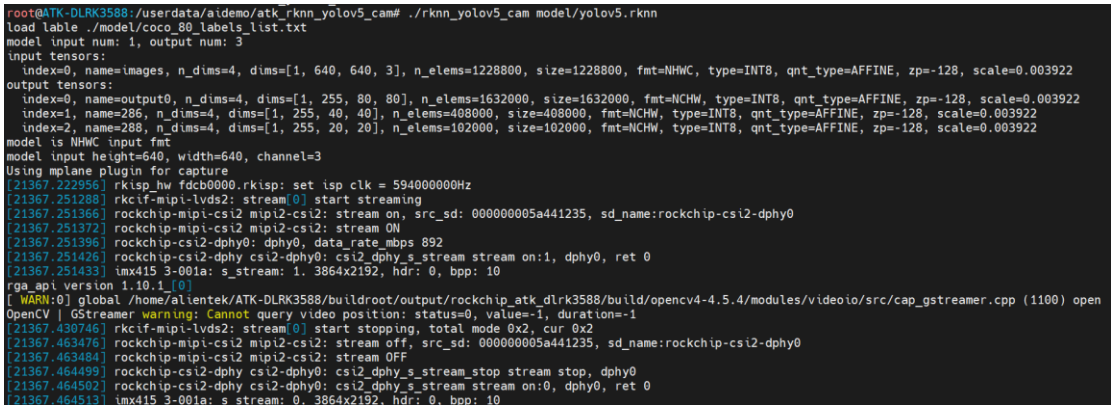


Figure 8.2.1.3 Execute the command to run the YOLOv5 inference program.

At this point, you can see on the screen that the objects have been highlighted, and the corresponding category and its confidence level are displayed at the top of the box, as shown in the following figure.

Note: If the RK3588 development board is running on a core board with more than 4GB of RAM, it will report an error. As shown in the figure below, this is a bug in the current version of the RGA module. It needs to be fixed by Renesas Microelectronics in the future. For now, you can simply ignore it.

Figure 8.2.1.4 Error occurred during rga operation

8.3 YOLOv6 routine for camera inference

This section explains how to run the YOLOv6 camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the RK3588 development board, we will copy the yolov6.zip file from the 01 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routines → 01, Linux 5.10 AI Routine Source Code → 08_yolov6 folder to Ubuntu. If you are using the RK3568 development board, we will copy the yolov6.zip file from the 04 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routines → 04, Linux 5.10 AI Routine Source Code → 08_yolov6 folder to Ubuntu. Unzip the yolov6.zip archive and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following commands to compile the program.

```
./build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/yolov6/cpp$ ./build-linux.sh
/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=yolov6
BUILD_DEMO_PATH=./
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=
INSTALL_DIR=/home/dominick/software/aidemo/yolov6/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov6_cam
BUILD_DIR=/home/dominick/software/aidemo/yolov6/cpp/build/build_atk_rknn_yolov6_cam_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-g++
==================================
-- cmake version 3.16.3
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
```

Figure 8.3.1.1 Compile the yolov6 routine

If it is the rk3588 development board, the compiled program and model should be placed in the "install" directory. Then, push the compiled program and model to the development board via adb.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov6_cam/ /userdata/aidemo
```

```
dominick@ubuntu:~/software/aidemo/yolov6/cpp$ adb push install/rk3588_linux_aarch64/atk_rknn_yolov6_cam/ /userdata/aidemo
install/rk3588_linux_aarch64/atk_rknn_yolov6_cam/: 5 files pushed. 1.0 MB/s (14288290 bytes in 13.525s)
```

Figure 8.3.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov6_cam/ /userdata/aidemo
```

On the development board, open the serial terminal and navigate to the directory /userdata/aidemo/atk_rknn_yolov6_demo, then execute the following command.

```
cd /userdata/aidemo/atk_rknn_yolov6_cam
./rknn_yolov6_cam model/yolov6.rknn
```

The serial port information of the normal execution of the reasoning program is shown in the following figure. At this time, the effect can be seen on the screen, which is similar to that of yolov5.

```
root@ATK-DLRK3588:/userdata/aidemo/atk_rknn_yolov6_cam# ./rknn_yolov6_cam model/yolov6.rknn
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 9
input tensors:
  index=0, name=image_arrays, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.00392
2
output tensors:
  index=0, name=outputs, n_dims=4, dims=[1, 4, 80, 80], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-127, scale=0.047491
  index=1, name=281, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003009
  index=2, name=287, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=294, n_dims=4, dims=[1, 4, 40, 40], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-127, scale=0.059930
  index=4, name=299, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003610
  index=5, name=305, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=6, name=312, n_dims=4, dims=[1, 4, 20, 20], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.059661
  index=7, name=317, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003851
  index=8, name=323, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
model is NHWC input fmt
model input height=640, width=640, channel=3
Using mplane plugin for capture
[25830.444748] rkisp_hw fdcb0000.rkisp: set isp clk = 594000000Hz
[25830.451948] rkcif-mipi-lvds2: stream[0] start streaming
[25830.451988] rockchip-mipi-csi2 mipi2-csi2: stream on, src_sd: 000000005a441235, sd_name:rockchip-csi2-dphy0
[25830.451991] rockchip-mipi-csi2 mipi2-csi2: stream ON
[25830.452008] rockchip-csi2-dphy0: dphy0, data_rate_mbps 892
[25830.452031] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream stream on:1, dphy0, ret 0
[25830.452036] imx415 3-001a: s_stream: 1. 3864x2192, hdr: 0, bpp: 10
```

Figure 8.3.1.3 Execute the command to run the yolov6 inference program

8.4 yolov7 routine camera inference

This section explains how to run the yolov7 camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the rk3588 development board, we will copy the yolov7.zip file from the 01 → AI routines → 01 → Source code →

09_yolov7 folder in the A drive of the development board optical disc to Ubuntu. If you are using the rk3568 development board, we will copy the yolov7.zip file from the 04 → Linux 5.10 AI routine source code → 09_yolov7 folder in the A drive of the development board optical disc to Ubuntu. Unzip the yolov7.zip archive, and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following command to compile the program.

```
./build-linux.sh
```

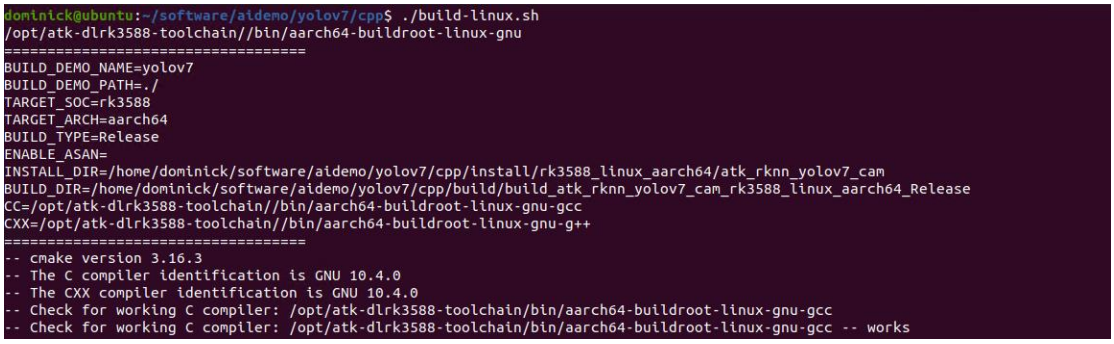


Figure 8.4.1.1 Compile the yolov7 routine

If it is the rk3588 development board, the compiled program and model should be placed in the "install" directory. Then, push the compiled program and model to the development board via adb.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov7_cam/ /userdata/aidemo
```

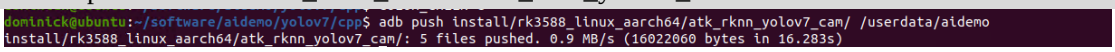


Figure 8.4.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov7_cam/ /userdata/aidemo
```

On the development board, open the serial terminal and navigate to the directory /userdata/aidemo/atk_rknn_yolov7_demo, then execute the following command.

```
cd /userdata/aidemo/atk_rknn_yolov7_cam
./rknn_yolov7_cam model/yolov7.rknn
```

The serial port information for the normal execution of the reasoning program is shown in the following figure. At this time, the effect can be seen on the screen, which is similar to other YOLO routines.

```
root@ATK-DLRK3588:/userdata/aidemo/atk_rknn_yolov7_cam# ./rknn_yolov7_cam model/yolov7.rknn
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 3
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=271, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=1, name=273, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=2, name=275, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
model is NHWC input fmt
model input height=640, width=640, channel=3
init_yolov7_model use: 24.006001 ms
Using mplane plugin for capture
[26871.941901] rkisp_hw fdcb0000.rkisp: set isp clk = 594000000Hz
[26871.957412] rkcif-mipi-lvds2: stream[0] start streaming
[26871.957474] rockchip-mipi-csi2 mipi2-csi2: stream on, src_sd: 000000005a441235, sd_name:rockchip-csi2-dphy0
[26871.957480] rockchip-mipi-csi2 mipi2-csi2: stream ON
[26871.957501] rockchip-csi2-dphy0: dphy0, data_rate_mbps 892
[26871.957529] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream stream on:1, dphy0, ret 0
[26871.957537] imx415 3-001a: s_stream: 1. 3864x2192, hdr: 0, bpp: 10
rga_api version 1.10.1_[0]
[ WARN:0] global /home/alientek/ATK-DLRK3588/buildroot/output/rockchip_atk_dlrk3588/build/opencv4-4.5.4/modules/videoio/src/cap_gstreamer.cpp (1100) open
OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
[26872.194982] rkcif-mipi-lvds2: stream[0] start stopping, total mode 0x2, cur 0x2
[26872.227749] rockchip-mipi-csi2 mipi2-csi2: stream off, src_sd: 000000005a441235, sd_name:rockchip-csi2-dphy0
[26872.227811] rockchip-mipi-csi2 mipi2-csi2: stream OFF
[26872.228895] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream_stop stream stop, dphy0
[26872.228922] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream stream on:0, dphy0, ret 0
[26872.228956] imx415 3-001a: s_stream: 0. 3864x2192, hdr: 0, bpp: 10
[26872.229540] rkcif-mipi-lvds2: stream[0] stopping finished, dma_en 0x0
[26872.243042] rkisp rkisp0-vir0: first params buf queue
Using mplane plugin for capture
[26872.279290] rkisp_hw fdcb0000.rkisp: set isp clk = 594000000Hz
[26872.292744] rkcif-mipi-lvds2: stream[0] start streaming
[26872.292815] rockchip-mipi-csi2 mipi2-csi2: stream on, src_sd: 000000005a441235, sd_name:rockchip-csi2-dphy0
[26872.292821] rockchip-mipi-csi2 mipi2-csi2: stream ON
[26872.292843] rockchip-csi2-dphy0: dphy0, data_rate_mbps 892
[26872.292873] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream stream on:1, dphy0, ret 0
[26872.292880] imx415 3-001a: s_stream: 1. 3864x2192, hdr: 0, bpp: 10
[26872.462953] rkcif-mipi-lvds2: stream[0] start stopping, total mode 0x2, cur 0x2
```

Figure 8.4.1.3 Execute the command to run the yolov7 inference program

8.5 yolov8 routine camera inference

This section explains how to run the yolov8 camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the rk3588 development board, we will copy the yolov8.zip file from the 10_yolov8 folder in the 01 - Basic Materials → 01_codes → 01_AI_Routines → 01_Foundation_Materials → A Disk of the development board to Ubuntu. If you are using the rk3568 development board, we will copy the yolov8.zip file from the 10_yolov8 folder in the 04 - Linux 5.10 AI Routine Source Code → 01_AI_Routines → 01 - Foundation Materials → A Disk of the development board to Ubuntu. Unzip the yolov8.zip archive and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following command to compile the program.

```
./build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/yolov8/cpp$ ./build-linux.sh
/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu
============================
BUILD_DEMO_NAME=yolov8
BUILD_DEMO_PATH=./
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=
INSTALL_DIR=/home/dominick/software/aidemo/yolov8/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov8_cam
BUILD_DIR=/home/dominick/software/aidemo/yolov8/cpp/build/build_atk_rknn_yolov8_cam_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-g++
==================================
-- cmake version 3.16.3
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/yolov8/cpp/build/build_atk_rknn_yolov8_cam_rk3588_linux_aarch64_Release
```

Figure 8.5.1.1 Compile the yolov8 routine

If it is the rk3588 development board, the compiled program and model should be placed in the "install" directory. Then, push the compiled program and model to the development board via adb.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov8_cam/ /userdata/aidemo
```

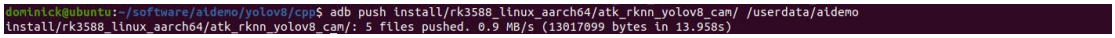


Figure 8.5.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov8_cam/ /userdata/aidemo
```

On the development board, open the serial terminal and navigate to the directory /userdata/aidemo/atk_rknn_yolov8_demo, then execute the following command.

```
cd /userdata/aidemo/atk_rknn_yolov8_cam
./rknn_yolov8_cam model/yolov8.rknn
```

The serial port information for the normal execution of the reasoning program is shown in the following figure. At this time, the effect can be seen on the screen, which is similar to other YOLO routines.

Figure 8.5.1.3 Execute the command to run the YOLOv8 inference program

8.6 Yolox routine camera inference

This section explains how to run the yolox camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the RK3588 development board, we will copy the yolox.zip file from the 11_yolox folder in the 01_program_source_code_AI_routine_01_basic_data_01_A_disk of the development board to Ubuntu. If you are using the RK3568 development board, we will copy the yolox.zip file from the 11_yolox folder in the 04_Linux5_10_Ai_routine_source_code_01_A_disk of the development board to Ubuntu. Unzip the yolox.zip archive and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following commands to compile the program.

```
./build-linux.sh
```

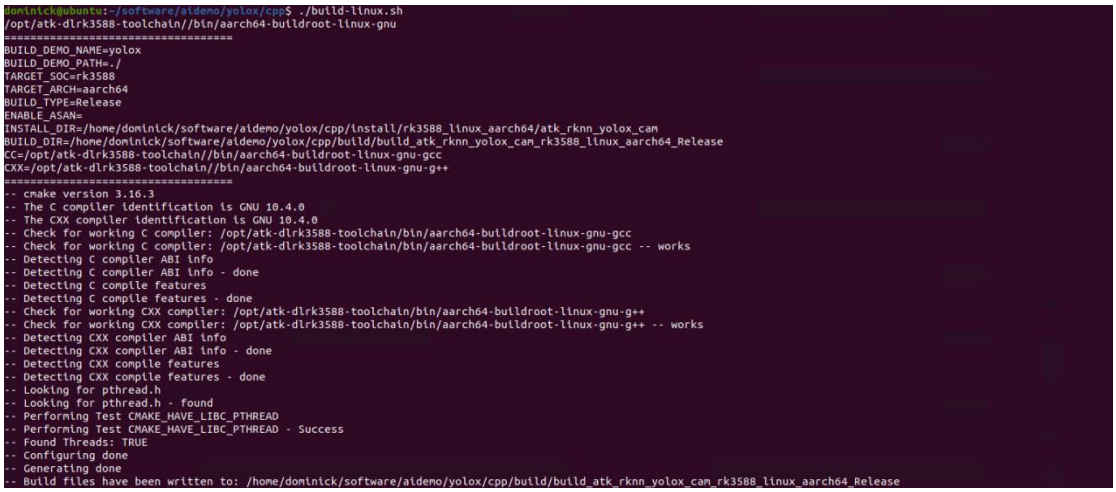


Figure 8.6.1.1 Compile the yolox routine

If it is the rk3588 development board, the compiled program and model should be placed in the "install" directory. Then, push the compiled program and model to the development board via adb.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolox_cam/ /userdata/aidemo
```

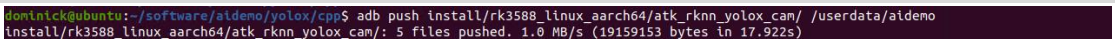


Figure 8.6.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolox_cam/ /userdata/aidemo
```

On the development board, enter the serial port terminal and go to the directory /userdata/aidemo/atk_rknn_yolox_demo, then execute the following commands.

```
cd /userdata/aidemo/atk_rknn_yolox_cam
./rknn_yolox_cam model/yolox.rknn
```

The serial port information for the normal execution of the reasoning program is shown in the following figure. At this time, the effect can be seen on the screen, which is similar to other YOLO routines.

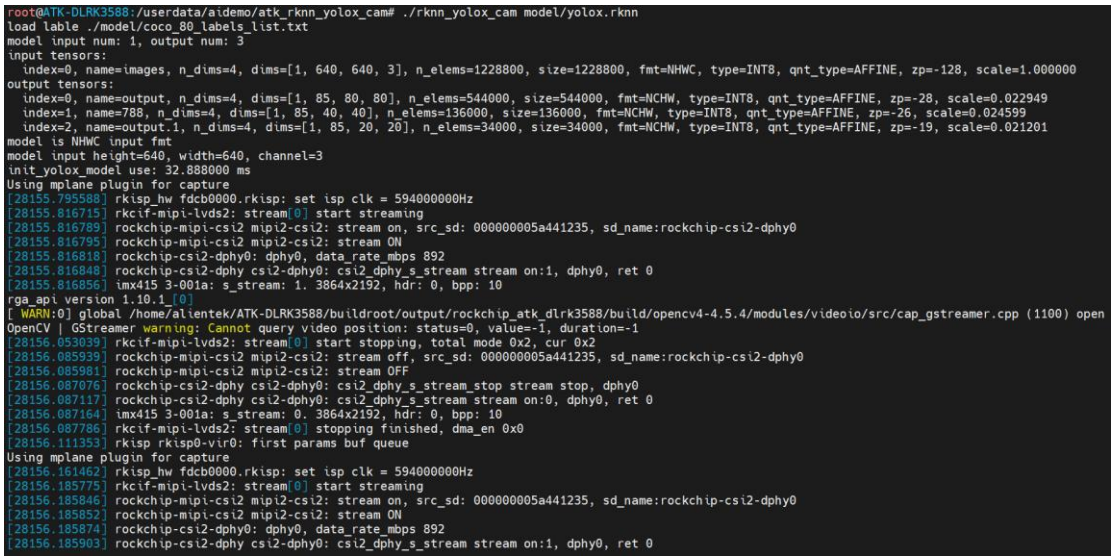


Figure 8.6.1.3 Execute the command to run the Yolox inference program

8.7 ppyoloe routine camera inference

This section explains how to run the ppyoloe camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the rk3588 development board, we will copy the ppyoloe.zip file from the 01 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 12, ppyoloe folder to Ubuntu. If you are using the rk3568 development board, we will copy the **ppyoloe.zip** file from the 04 folder under the A disk of the development board - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 12, ppyoloe folder to Ubuntu. After extracting the ppyoloe.zip archive, go to the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following commands to compile the program.

```
./build-linux.sh
```

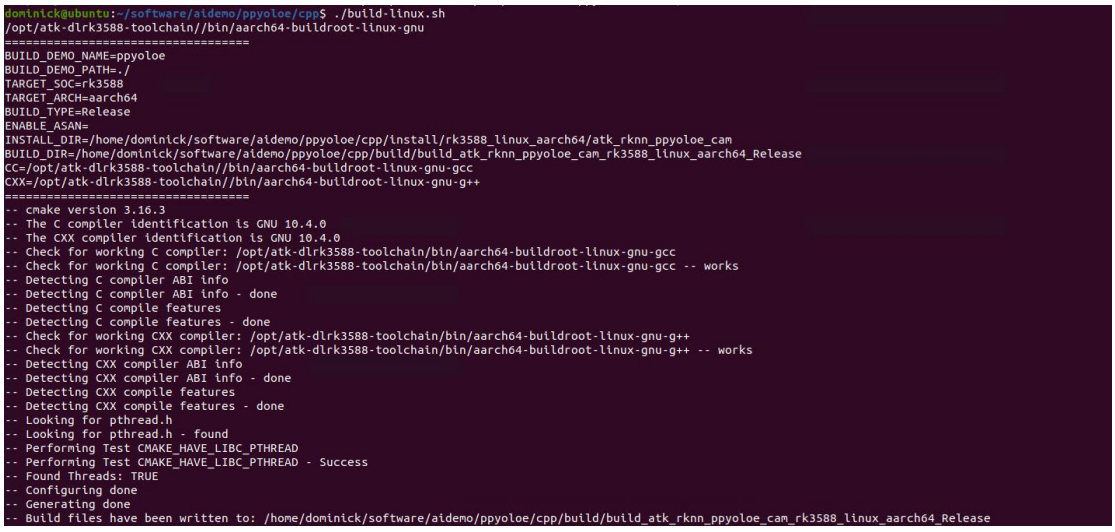


Figure 8.7.1.1 Compile the ppyoloe program

If it is the rk3568 development board, the compiled program and model should be placed in the "install" directory. Then, push the compiled program and model to the development board via adb.

```
adb push install/rk3588_linux_aarch64/atk_rknn_ppyoloe_cam/ /userdata/aidemo
```

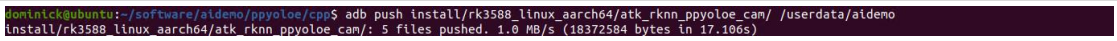


Figure 8.7.1.2 Push the routine to the development board

If it is a rk3568 development board, connect the OTG interface of the rk3568 development board to the Ubuntu system. Then, execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_ppyoloe_cam/ /userdata/aidemo
```

On the development board, open the serial terminal and navigate to the directory /userdata/aidemo/atk_rknn_ppyoloe_demo, then execute the following command.

```
cd /userdata/aidemo/atk_rknn_ppyoloe_cam
./rknn_ppyoloe_cam model/ppyoloe.rknn
```

The serial port information for the normal execution of the reasoning program is shown in the following figure. At this time, the effect can be seen on the screen, which is similar to other YOLO routines.

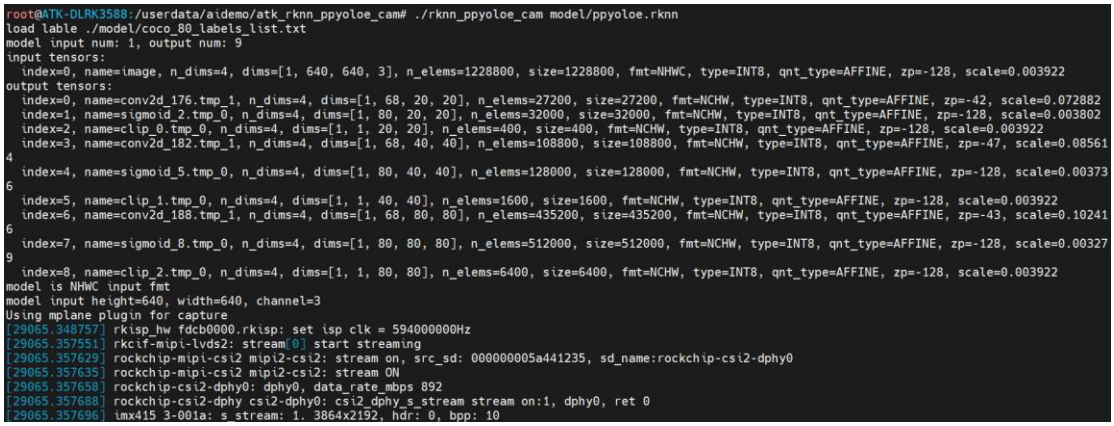


Figure 8.7.1.3 Execute the command to run the ppyoloe inference program

Chapter 9. Target Segmentation Routine

In this chapter, we test the target segmentation task and provide multiple routines for inference using cameras, making it convenient for everyone to learn and use. It is divided into the following sections:

9.1, Introduction to Target Segmentation Routine

9.2, Test of yolov5_seg Routine

9.3, Test of yolov8_seg Routine

9.4, Test of pp_seg Routine

9.1 Introduction to the target segmentation routine

The development board documentation provides several segmentation routines that perform inference using cameras. These are the yolov5_seg, yolov8_seg, and pp_seg routines.

1、YOLOv5_seg

YOLOv5_seg is an image segmentation model based on YOLOv5. By improving and expanding it, it enables the simultaneous execution of target detection and image segmentation tasks. This model maintains the lightweight network structure and efficient inference algorithm of YOLOv5 while adding a segmentation head, thereby achieving the function of image segmentation. In terms of the loss function, YOLOv5_seg has also been optimized, introducing a loss function suitable for image segmentation tasks to enhance the accuracy of segmentation. Additionally, this model supports running on various hardware platforms, including CPU, GPU, and FPGA, etc., demonstrating good compatibility and scalability.

2、YOLOv8_seg

YOLOv8_seg is a target detection and semantic segmentation model based on YOLOv8. It combines the characteristics of YOLOv4 and DeepLabV3+ models, aiming to achieve accurate target detection and fine semantic segmentation. Compared with YOLOv5_seg, YOLOv8_seg has improved in both accuracy and speed, and introduces new technologies and strategies, such as model fusion and data augmentation, to optimize the model performance. When performing target detection, this model can accurately identify objects in the image and mark their bounding boxes and category information; when performing semantic segmentation, it can classify each pixel in the image, achieving pixel-level semantic segmentation. This function of simultaneously achieving target detection and semantic segmentation enables YOLOv8_seg to have better comprehensive performance when processing complex images.

3、ppseg

PaddleSeg (PPSeg) is an end-to-end image segmentation development kit developed based on PaddlePaddle. It offers high-quality segmentation models ranging from high precision to lightweight. PaddleSeg, through its modular design, helps developers easily complete the entire process of image segmentation applications from training to deployment. One of the features of PaddleSeg is the provision of a newly upgraded portrait segmentation function, as well as an ultra-lightweight portrait segmentation model that supports real-time segmentation in web and mobile scenarios. This enables PaddleSeg to have wide applications in scenarios such as video conferences and online education. For example, in video conferences, PaddleSeg's virtual background function can achieve real-time background replacement and background blurring, protecting user privacy while increasing the fun of the meeting. In addition, PaddleSeg has launched the refined segmentation solution PaddleSeg-Matting and open-sourced the panoramic segmentation algorithm Panoptic-DeepLab, further enriching the model types. At the same time, PaddleSeg has also released the intelligent annotation tool EISeg for interactive segmentation, significantly improving the annotation efficiency. PaddleSeg is a powerful and user-friendly image segmentation development kit that provides a complete solution from training to deployment for developers.

9.2 Yolov5 Segmentation Routine Test

If you are using the RK3588 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 01_Source_Code → 13_yolov5_seg in the atk_yolov5_seg_cam.zip provided routine under the home directory of ubuntu, copy it to the software/aidemo directory under the home directory of ubuntu, decompress it for compilation and then conduct the test. If you are using the RK3568 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 13_yolov5_seg in the atk_yolov5_seg_cam.zip provided routine under the home directory of ubuntu, copy it to the software/aidemo directory under the home directory of ubuntu, decompress it for compilation and then conduct the test. After decompression, open the terminal in the current directory and execute the following commands to enter the corresponding directory's cpp directory for compilation.

```
cd ~/software/aidemo/atk_yolov5_seg_cam/cpp
./build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/atk_yolov5_seg_cam/cpp$ ./build-linux.sh
/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu
================================
BUILD_DEMO_NAME=yolov5_seg
BUILD_DEMO_PATH=./
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=
INSTALL_DIR=/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam
BUILD_DIR=/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/build/build_atk_rknn_yolov5_seg_cam_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-g++
================================
-- cmake version 3.16.3
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/build/build_atk_rknn_yolov5_seg_cam_rk3588_linux_aarch64_Release
```

Figure 9.2.1.1 Start compiling

After the compilation is complete, it will look like the picture below.

```
[ 10%] Building C object utils.out/CMakeFiles/imagedrawing.dir/image_drawing.c.o
[ 20%] Building C object utils.out/CMakeFiles/fileutils.dir/file_utils.c.o
[ 30%] Building C object utils.out/CMakeFiles/imageutils.dir/image_utils.c.o
[ 40%] Linking C static library libfileutils.a
/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.c: 在函数'write_image'中:
/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.c:282:35: 警告: passing argument 1 of 'get_image_size' discards 'const' qualifier from pointer target ty
pe [-Wdiscarded-qualifiers]
  282 |         int size = get_image_size(img);
      |                                   ^~~
In file included from /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.c:21:
/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.h:67:36: 附注: 需要类型'image_buffer_t *', 但实参的类型为'const image_buffer_t *'
   67 | int get_image_size(image_buffer_t* image);
      |                    ~~~~~~~~~~~~~~~~^~~~~
/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.c: 在函数'convert_image_rga'中:
/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/utils/image_utils.c:626:27: 警告: initialization of 'char *' from incompatible pointer type 'int *' [-Wincompatible-pointe
r-types]
  626 |         char* p_imcolor = &imcolor;
      |                           ^
[ 40%] Built target fileutils
[ 50%] Linking C static library libimagedrawing.a
[ 50%] Built target imagedrawing
[ 60%] Linking C static library libimageutils.a
[ 60%] Built target imageutils
Scanning dependencies of target rknn_yolov5_seg_cam
[ 80%] Building CXX object CMakeFiles/rknn_yolov5_seg_cam.dir/rknpu2/postprocess.cc.o
[ 80%] Building CXX object CMakeFiles/rknn_yolov5_seg_cam.dir/main.cc.o
[ 90%] Building CXX object CMakeFiles/rknn_yolov5_seg_cam.dir/rknpu2/yolov5_seg.cc.o
[100%] Linking CXX executable rknn_yolov5_seg_cam
[100%] Built target rknn_yolov5_seg_cam
[ 20%] Built target fileutils
[ 40%] Built target imageutils
[ 60%] Built target imagedrawing
[100%] Built target rknn_yolov5_seg_cam
Install the project...
-- Install configuration: "Release"
-- Installing: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/./rknn_yolov5_seg_cam
-- Set runtime path of "/home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/./rknn_yolov5_seg_cam" to "$ORIGIN/../lib"
-- Installing: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/lib/librknnrt.so
-- Installing: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/model/coco_80_labels_list.txt
-- Installing: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/model/yolov5_seg.rknn
-- Up-to-date: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/lib/librknnrt.so
-- Installing: /home/dominick/software/aidemo/atk_yolov5_seg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/lib/librga.so
```

Figure 9.2.1.2 Compilation completed.

Connect the OTG interface of the rk3588 development board to Ubuntu. Execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov5_seg_cam/ /userdata/aidemo
```

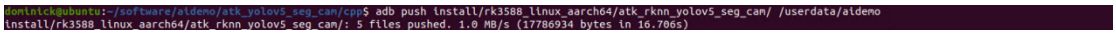


Figure 9.2.1.3 Push the model and executable files to the development board.

Check if the mipi csi3 on the development board has properly connected to the mipi interface camera, and check if the mipi dsi0 has properly connected to the mipi interface screen.

If it is a rk3568 development board, connect the otg interface of the rk3568 development board to the ubuntu system, and execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov5_seg_cam/ /userdata/aidemo
```

Open the serial terminal of the development board and execute the following commands (if the Qt interface has not been closed, it is best to close the Qt interface first).

```
/etc/init.d/S50systemui stop
cd /userdata/aidemo/atk_rknn_yolov5_seg_cam/
./rknn_yolov5_seg_cam model/yolov5_seg.rknn
```

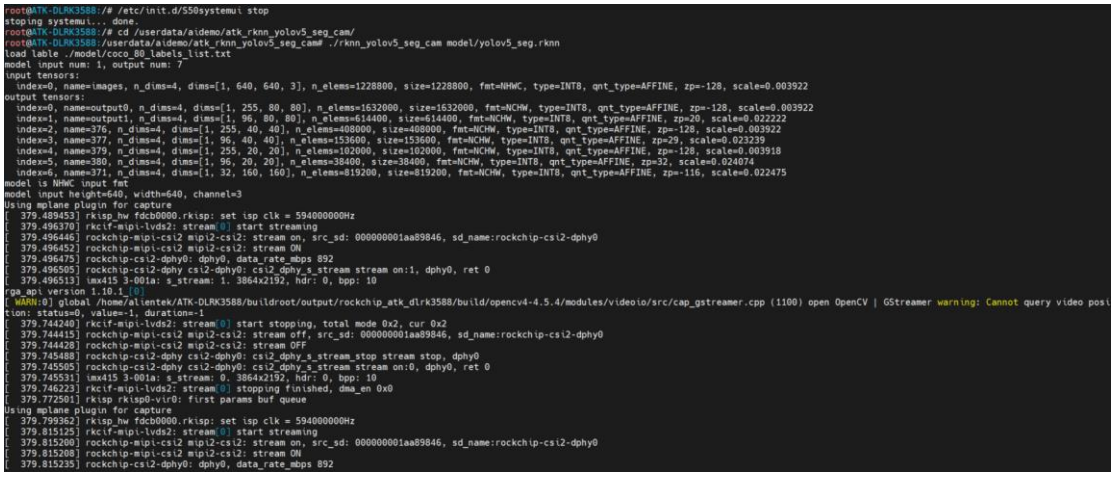


Figure 9.2.1.4 Execute the executable program for reasoning.

You can see the real-time reasoning process displayed on the MIPI screen. Pressing the Ctrl+C key on the keyboard will exit the program.

Figure 9.2.1.5 Identified skis and people

9.3 yolov8_seg routine test

If using the rk3588 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 01, Source Code → 14_yolov8_seg in atk_yolov8_seg_cam.zip provided by the routine, copy it to the home directory of ubuntu under the software/aidemo directory, decompress for compilation and then test. If using the rk3568 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 14_yolov8_seg in atk_yolov8_seg_cam.zip provided

by the routine, copy it to the home directory of ubuntu under the software/aidemo directory, decompress for compilation and then test. After decompression, open the terminal in the current directory and execute the following commands to enter the corresponding directory's cpp directory for compilation.

```
cd ~/software/aidemo/atk_yolov8_seg_cam/cpp
./build-linux.sh
```

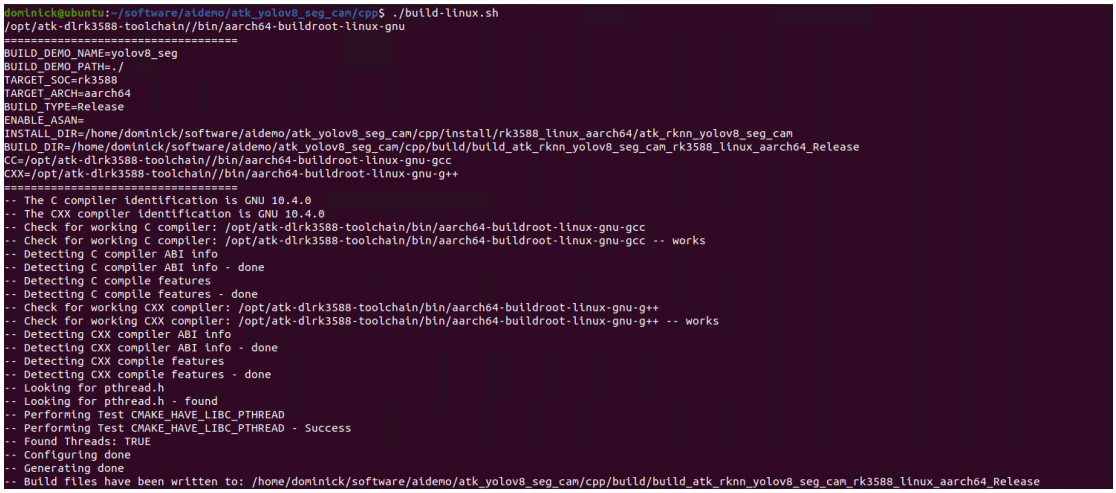


Figure 9.3.1.1 Start compiling

After the compilation is complete, it will look like the picture below.

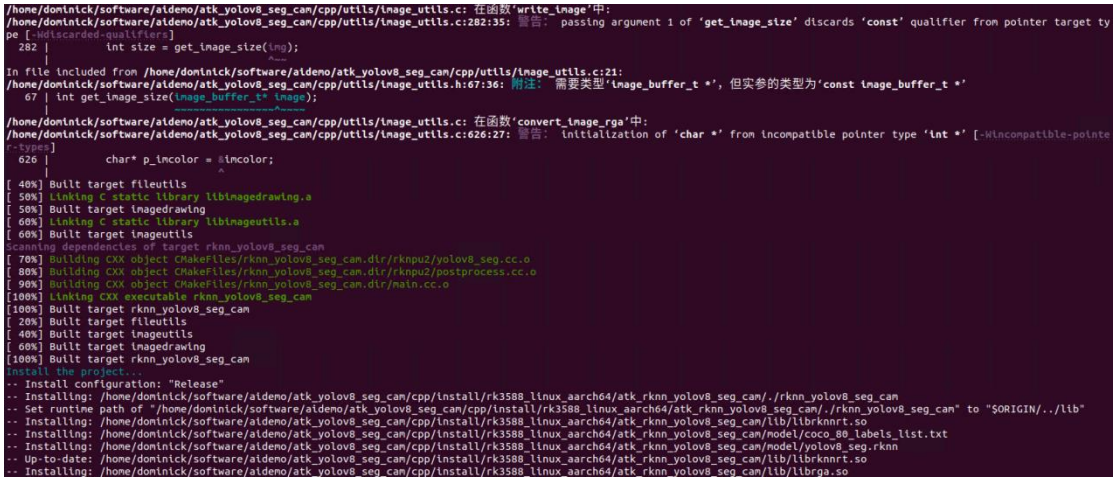


Figure 9.3.1.2 Compilation completed.

Connect the OTG interface of the rk3588 development board to Ubuntu. Execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3588_linux_aarch64/atk_rknn_yolov8_seg_cam/ /userdata/aidemo
```

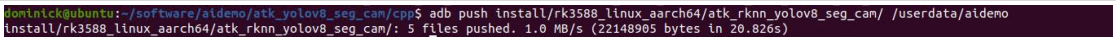


Figure 9.3.1.3 Push the executable program and model to the development board.

Check whether the mipi csi3 on the development board has connected the mipi interface camera, and check whether the mipi dsi0 has connected the mipi interface screen.

If it is a rk3568 development board, connect the otg interface of the rk3568 development board to the ubuntu, and execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_yolov8_seg_cam/ /userdata/aidemo
```

Open the serial terminal of the development board and execute the following commands (if the Qt interface has not been closed, it is best to close the Qt interface first, then execute /etc/init.d/S50systemui stop in the development board terminal).

```
cd /userdata/aidemo/atk_rknn_yolov8_seg_cam/
./rknn_yolov8_seg_cam model/yolov8_seg.rknn
```

Figure 9.3.1.4 Execute the executable program for reasoning.

You can see the real-time inference pictures displayed on the MIPI screen. Pressing Ctrl+C on the keyboard will exit the program.

9.4 ppseg routine test

If you are using the RK3588 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 01, Source Code → 15_ppseg, where atk_ppseg_cam.zip provides the routine. Copy it to the home directory of Ubuntu under the software/aidemo directory, decompress it for compilation, and then conduct the test. If you are using the RK3568 development board, open the A disk of the development board CD - Basic Data → 01_codes → 01_AI_Routine → 04, Linux5_10 AI Routine Source Code → 15_ppseg, where atk_ppseg_cam.zip provides the routine. Copy it to the home directory of Ubuntu under the software/aidemo directory, decompress it for compilation, and then conduct the test. After decompression, open the terminal in the current directory and execute the following command to enter the corresponding directory's cpp directory for compilation.

```
cd ~/software/aidemo/atk_ppseg_cam/cpp
./build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/atk_ppseg_cam/cpp$ ./build-linux.sh
/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu
==================================
BUILD_DEMO_NAME=ppseg
BUILD_DEMO_PATH=./
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=
INSTALL_DIR=/home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam
BUILD_DIR=/home/dominick/software/aidemo/atk_ppseg_cam/cpp/build/build_atk_rknn_ppseg_cam_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-g++
==================================
-- cmake version 3.16.3
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/atk_ppseg_cam/cpp/build/build_atk_rknn_ppseg_cam_rk3588_linux_aarch64_Release
```

Figure 9.4.1.1 Start compiling

After the compilation is complete, it will look like the picture below.

```
/home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.c: 在函数'write_image'中:
/home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.c:282:35: 警告:  passing argument 1 of 'get_image_size' discards 'const' qualifier from pointer target type [-
Wdiscarded-qualifiers]
  282 |         int size = get_image_size(img);
      |                                   ^~~
In file included from /home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.c:21:
/home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.h:67:36: 附注:  需要类型'image_buffer_t *', 但实参的类型为'const image_buffer_t *'
   67 | int get_image_size(image_buffer_t* image);
      |                    ~~~~~~~~~~~~~~~~^~~~~
/home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.c: 在函数'convert_image_rga'中:
/home/dominick/software/aidemo/atk_ppseg_cam/cpp/utils/image_utils.c:626:27: 警告:  initialization of 'char *' from incompatible pointer type 'int *' [-Wincompatible-pointer-typ
es]
  626 |         char* p_imcolor = &imcolor;
      |                           ^
[ 55%] Linking C static library libimagedrawing.a
[ 55%] Built target imagedrawing
[ 66%] Linking C static library libimageutils.a
[ 66%] Built target imageutils
Scanning dependencies of target rknn_ppseg_cam
[ 88%] Building CXX object CMakeFiles/rknn_ppseg_cam.dir/main.cc.o
[ 88%] Building CXX object CMakeFiles/rknn_ppseg_cam.dir/ppseg.cc.o
[100%] Linking CXX executable rknn_ppseg_cam
[100%] Built target rknn_ppseg_cam
[ 22%] Built target fileutils
[ 44%] Built target imageutils
[ 66%] Built target imagedrawing
[100%] Built target rknn_ppseg_cam
Install the project...
-- Install configuration: "Release"
-- Installing: /home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/./rknn_ppseg_cam
-- Set runtime path of "/home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/./rknn_ppseg_cam" to "$ORIGIN/../lib"
-- Installing: /home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/model/pp_liteseg.rknn
-- Installing: /home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/lib/librknnrt.so
-- Installing: /home/dominick/software/aidemo/atk_ppseg_cam/cpp/install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/lib/librga.so
```

Figure 9.4.1.2 Compilation completed.

Connect the OTG interface of the rk3588 development board to Ubuntu. Execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/ /userdata/aidemo
```

```
dominick@ubuntu:~/software/aidemo/atk_ppseg_cam/cpp$ adb push install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/ /userdata/aidemo
install/rk3588_linux_aarch64/atk_rknn_ppseg_cam/: 4 files pushed. 1.0 MB/s (17910517 bytes in 17.792s)
```

Figure 9.4.1.3 Push the executable program and model to the development board.

Check whether the mipi csi3 on the development board has connected the mipi interface camera, and check whether the mipi dsi0 has connected the mipi interface screen.

If it is a rk3568 development board, connect the otg interface of the rk3568 development board to the ubuntu, and execute the following commands in the serial terminal to copy the routine to the development board.

```
adb push install/rk3568_linux_aarch64/atk_rknn_ppseg_cam/ /userdata/aidemo
```

Open the serial terminal of the development board and execute the following commands (if the Qt interface has not been closed, it is best to close the Qt interface first, then execute /etc/init.d/S50systemui stop in the development board terminal).

```
cd /userdata/aidemo/atk_rknn_ppseg_cam/
./rknn_ppseg_cam model/pp_liteseg.rknn
```

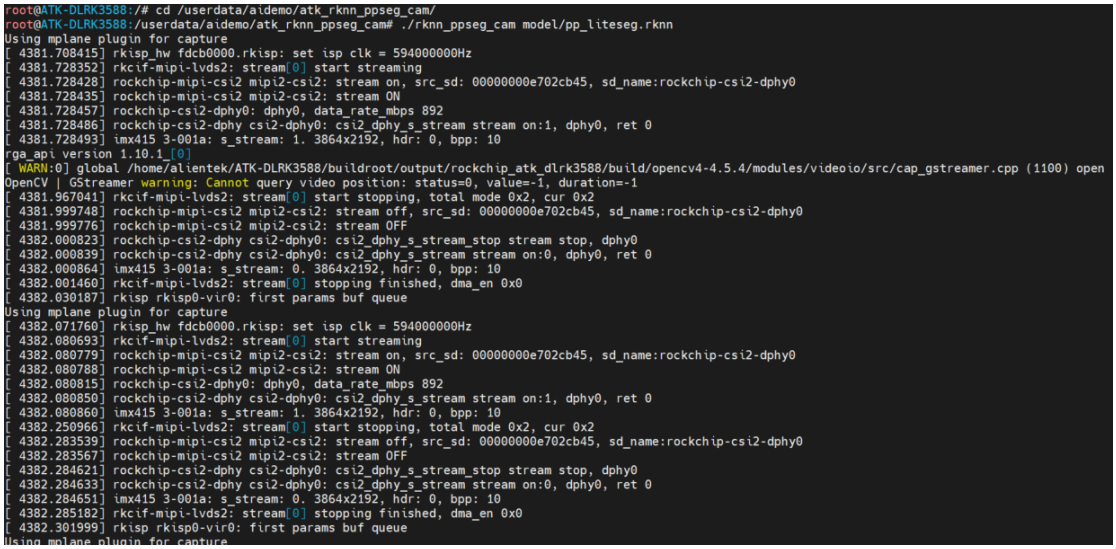


Figure 9.4.1.4 Execute the executable program for reasoning.

You can see two preview boxes displayed on the MIPI screen showing real-time reasoning. One shows the original image and the other shows the image segmentation map. Pressing Ctrl+C on the keyboard will exit the program.

Figure 9.4.1.5 The performance of ppseg

From the above figure, it can be seen that the performance of some other categories is still relatively average. You can try to use some new categories for transfer learning to train the model on your own.

Chapter 10. Face Detection Video Inference Routine

In this chapter, we introduce the routine tests for face detection and the five key points of the face. Face detection is an important technology in the field of computer vision, aiming to identify the face regions in images or videos. This technology provides fundamental support for numerous applications, such as face recognition, expression analysis, human-computer interaction, security monitoring, etc. The goal of face detection is to automatically locate and mark all the faces that appear in the image, providing basic data for subsequent analysis and processing. In daily life, there are many places where it is necessary to detect the face first before performing recognition, comparison, or analysis functions on the face. Face detection is an indispensable and important part of this process.

This chapter will be divided into the following sections:

1. Routine Introduction
2. Routine Testing

10.1 Routine Overview

Traditional face detection methods usually involve multiple stages of processing procedures, including image preprocessing, feature extraction, and classifier. However, this approach faces some challenges in terms of computational cost and accuracy. In recent years, with the rise of deep learning technology, face detection methods based on convolutional neural networks (CNN) have made significant progress.

In the field of face detection, researchers have proposed many innovative methods. One of them is the Single-Stage Context Reasoning Face Detector (SCRFD), which was proposed by InsightFace in 2021. SCRFD is a face detection model. It adopts a single-stage architecture, combining localization and classification into one step, thereby improving the speed and efficiency of detection.

The working principle of SCRFD is based on deep neural networks. This network learns the feature representations and detection patterns of faces by training on a large amount of labeled data. During the testing stage, this network can quickly and accurately identify faces in images and output their positions and confidence scores. In summary, face detection technology, by using advanced computer vision and deep learning techniques, achieves the goal of automatically recognizing faces in images and videos. SCRFD, as one of these methods, brings higher efficiency and accuracy to face detection through its single-stage context reasoning architecture, promoting the development of face recognition and other applications. With the continuous progress of technology, face detection will continue to play an important role in various fields.

For more details about SCRFD, please refer to the following links.

1. Paper: https://arxiv.org/abs/2105.04714
2. Source code: https://github.com/deepinsight/insightface/tree/master/detection/scrfd

10.2 Routine test

Execute the following command in the directory where the scaling_frequency.sh script is located on the RK3588 development board terminal to perform frequency setting and enable the performance mode.

```
sh scaling_frequency.sh -c rk3588
```

If the RK3568 development board is used, the following command can be executed to set the frequency.

```
sh scaling_frequency.sh -c rk3568
```

Before conducting the test of this routine, please ensure that the relevant reasoning environment and libraries have been installed on the development board as per Chapters 2 and 5, and that your development board already has the ALIENTEK -compatible camera IMX415 as well as the 5.5-inch 1080p MIPI screen or the 720p MIPI screen available from ALIENTEK. The supported hardware includes:

- MIPI CSI interface camera (IMX415), RK3588 Buildroot system does not support IMX13850 and IMX335
- MIPI DSI interface screen (5.5-inch 720p and 1080p, 10.1-inch 1280x800)

If using the RK3588 development board, transfer the compressed file "scrfd.zip" from the A drive of the data disk of the development board's A disk - Basic Data → 01_codes → 01_AI_Routine →

01, Source Code → 16_facedet_scrfd_npu routine to the development board's /userdata/aidemo directory via adb. If using the RK3568 development board, transfer the compressed file "scrfd.zip" from the A drive of the data disk of the development board's A disk - Basic Data → 01_codes → 01_AI_Routine → 04, Linux 5.10 AI Routine Source Code → 16_facedet_scrfd_npu routine to the development board's /userdata/aidemo directory via adb. Open the terminal in the directory where the routine is located under Ubuntu, and enter the following commands.

```
adb push scrfd.zip /userdata/aidemo
```

After the transmission is completed, return to the development board terminal and enter the corresponding directory to execute the following commands.

```
cd /userdata/aidemo/
unzip scrfd.zip
cd scrfd
python main.py
```

When the operation is successful, by aligning a face or a picture with a face towards the camera, you can see that our MIPI screen can draw a box around the position of the face and display the confidence level, as well as mark points at two points on the eyes, two points on the mouth, and the nose (a total of 5 points). The operation effect is as shown in the following picture.

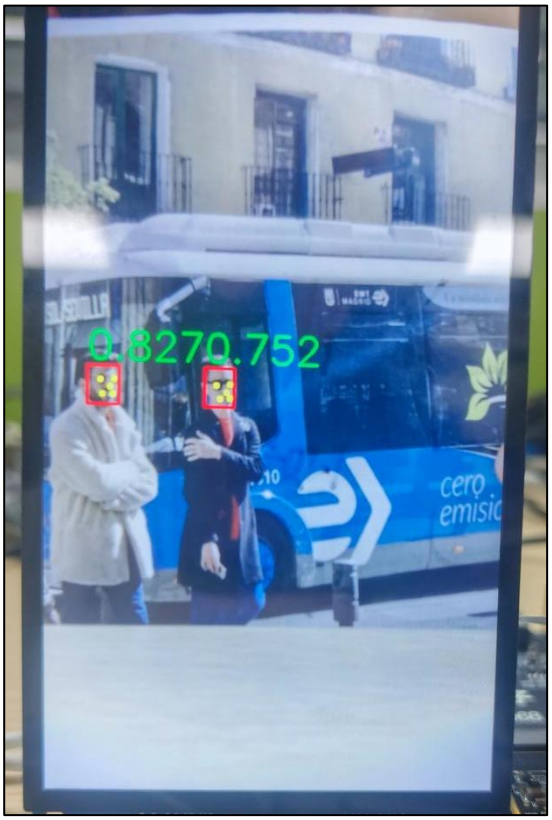


Figure 10.2.1.1 Face detection inference phenomenon

Pressing Ctrl+C on the serial port terminal will exit the program.

Chapter 11. OCR Text Detection and Recognition

In this chapter, we will introduce the routine tests for OCR text detection and recognition. OCR (Optical Character Recognition) text detection and recognition technology is a technique that converts the text information in paper documents or images into editable and processable digital text format. This technology is based on image processing and pattern recognition algorithms. By capturing the character features in the document, such as strokes, shapes, size, spacing, etc., and comparing them with the preset character library, the corresponding text information can be identified. OCR text detection and recognition technology has advantages such as high efficiency, convenience, cost savings, easy storage and transmission, and strong editability. This chapter will introduce how to deploy it on the rk3568/rk3588 development board.

This chapter will be divided into the following sections:

1. Introduction to OCR Text Detection and Recognition
2. Testing the OCR Camera Inference Routine

11.1 Introduction to OCR (Optical Character Recognition)

OCR (Optical Character Recognition) is a technology that converts handwritten, printed, or machine-generated text in images into machine-editable and searchable text format. OCR technology has wide applications in many fields, including document processing, automation, accessibility (such as helping visually impaired people read printed text), and data analysis, etc. OCR (Optical Character Recognition) is one of the important directions in computer vision. The traditional definition of OCR is generally for scanned document objects. Now, what we commonly refer to as OCR generally refers to Scene Text Recognition (STR), mainly targeting natural scenes, such as vehicle recognition and other natural scene visible text as shown in the following figure.

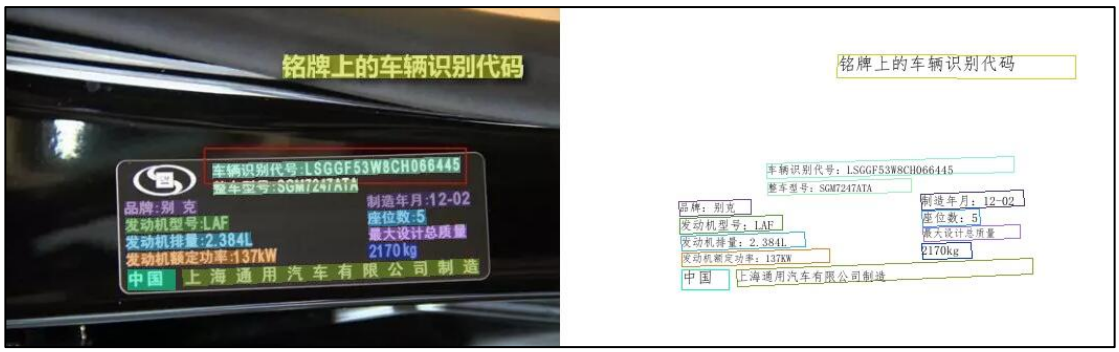


Figure 11.1.1.1 ppocrv4 vehicle model number recognition

Figure 11.1.1.2 ppocrv4 boarding pass recognition

The main steps of OCR text detection and recognition can be divided into the following:

Preprocessing: Before character recognition, the image needs to go through a series of preprocessing steps to improve the accuracy of recognition. These steps may include noise reduction, binarization, skew correction, image enhancement, etc.

Layout analysis: Layout analysis is an important step in OCR, used to determine the layout and arrangement of text on the page. It usually includes determining the position and size of text blocks, lines, and words.

Character segmentation: After determining the text blocks, the characters within the blocks need to be segmented for subsequent recognition. This is usually achieved by finding the blank spaces or separators between characters.

Feature extraction: For each segmented character, its features need to be extracted for use by the recognition algorithm. These features may include the contour, pixel distribution, structural information, etc.

Character recognition: Using machine learning or deep learning algorithms (such as neural networks), the extracted features are classified to determine the identity of each character. This step may be affected by factors such as character font, size, color, and background.

Post-processing: Finally, some post-processing steps may be needed to correct errors in recognition. For example, a dictionary can be used to check whether the recognized text conforms to grammar and semantic rules, or a language model can be used to improve the accuracy of recognition.

With the continuous development of deep learning technology, OCR systems based on neural networks have significantly improved their performance in text detection and recognition. These systems can handle more complex images and text layouts, and have better adaptability to different fonts, sizes, and backgrounds.

It should be noted that although OCR technology can achieve good results in many cases, it still faces some challenges, such as the recognition of low-quality images, the recognition of handwritten text, and the recognition of non-standard fonts. Therefore, in practical applications, it may be necessary to combine multiple technologies and methods to improve the performance and accuracy of the OCR system.

In this example, we have implemented the use of a camera on the rk3588 to perform rknn inference and display it on the screen. We use ppocrv4 to implement the key text detection and text recognition functions, meeting the requirements of basic text detection and recognition applications.

PaddleOCR v4 is a text recognition (OCR) tool developed based on the Baidu PaddlePaddle deep learning framework. It inherits the advantages of the PaddleOCR series of models and has undergone comprehensive upgrades in terms of performance and functionality. Here is a detailed introduction to PaddleOCR v4:

Technical upgrades:

Compared with PP-OCRv3, PaddleOCR v4 has significantly improved the recognition accuracy in various scenarios. Among them, the end-to-end Hmean metric for the Chinese scenario has increased by 4.25%, the text recognition accuracy for the English numerical scenario has increased by 6%, and the recognition accuracy of over 80 supported languages has increased by more than 5% on average in the four language families with evaluation sets. In addition to updating the Chinese model, PaddleOCR v4 has also optimized the English numerical model and upgraded and updated the recognition models of multiple supported languages.

Performance optimization:

PaddleOCR v4 adopts a lightweight model design, balancing recognition accuracy and running speed, enabling smooth operation on devices with limited resources.

Functional features:

1. Multi-language support: Besides the basic Chinese and English, PaddleOCR v4 also supports text recognition in various other languages to meet global requirements.

2. Real-time recognition: Integrated with real-time image processing capabilities, it can perform text detection and recognition in real time when capturing video streams from the mobile phone camera.

3. Customized training: Provides a custom training function, allowing users to fine-tune the model according to specific needs to improve recognition effects in specific scenarios.

Application scenarios:

PaddleOCR v4 can be widely applied in various fields such as finance, education, healthcare, and the printing and publishing industry, achieving functions like document digitization, bill recognition, license plate recognition, and image text extraction.

Prospects:

With the continuous advancement of artificial intelligence technology, the application of OCR technology in various fields will become increasingly widespread. PaddleOCR v4, as an advanced OCR tool, will continue to play an important role in promoting the development and application of OCR technology. In summary, PaddleOCR v4 is a powerful and high-performing OCR tool, demonstrating outstanding performance in terms of technology, performance, and functionality, and will play a significant role in the future application of OCR technology.

More information about ppocrv4 can be learned through the following website.

ppocrv4 project address: https://github.com/PaddlePaddle/PaddleOCR

11.2 Test the OCR camera inference routine

This section explains how to run the yolov8 camera inference routine on the development board. First, you need to copy the routine to Ubuntu for compilation. If you are using the rk3588 development board, we will copy the ppocr.zip file from the 01 folder under the 01 folder of the program source code, the 01 folder of the AI routine, and the 18_ppocr folder under the AI routine source code of the A disk of the development board CD-ROM to the software/aidemo directory in the home directory of Ubuntu. If you are using the rk3568 development board, we will copy the ppocr.zip file from the 04 folder under the 01 folder of the program source code, the 01 folder of the AI routine, and the 18_ppocr folder under the Linux5_10 AI routine source code of the A disk of the development board CD-ROM to the software/aidemo directory in the home directory of Ubuntu. Unzip the ppocr.zip archive and enter the corresponding cpp directory. Before compiling, please make sure that your routine has been installed with the cross-compiler as per Chapter 1 and Chapter 2 and updated the NPU service on the development board. Open the terminal and execute the following commands to compile the program.

```
./build-linux.sh
```

```
dominick@ubuntu:~/software/aidemo/ppocr/cpp$ ./build-linux.sh
/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu
================================
BUILD_DEMO_NAME=PPOCR_CAMERA
BUILD_DEMO_PATH=./
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
INSTALL_DIR=/home/dominick/software/aidemo/ppocr/cpp/install/rk3588_linux_aarch64/rknn_PPOCR_CAMERA_demo
BUILD_DIR=/home/dominick/software/aidemo/ppocr/cpp/build/build_rknn_PPOCR_CAMERA_demo_rk3588_linux_aarch64_Release
CC=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-gcc
CXX=/opt/atk-dlrk3588-toolchain//bin/aarch64-buildroot-linux-gnu-g++
================================
-- cmake version 3.16.3
-- The C compiler identification is GNU 10.4.0
-- The CXX compiler identification is GNU 10.4.0
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/atk-dlrk3588-toolchain/bin/aarch64-buildroot-linux-gnu-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dominick/software/aidemo/ppocr/cpp/build/build_rknn_PPOCR_CAMERA_demo_rk3588_linux_aarch64_Release
```

Figure 11.2.1.1 Compile the ppocr text recognition program

Transfer it to the /userdata/aidemo directory on the development board via adb. Open the terminal in the directory where the routine is located on Ubuntu. If you are using the rk3588 development board, execute the following command.

```
adb push install/rk3588_linux_aarch64/rknn_PPOCR_CAMERA_demo/ /userdata/aidemo
```

```
dominick@ubuntu:~/software/aidemo/ppocr/cpp$ adb push install/rk3588_linux_aarch64/rknn_PPOCR_CAMERA_demo/ /userdata/aidemo
install/rk3588_linux_aarch64/rknn_PPOCR_CAMERA_demo/: 5 files pushed. 0.9 MB/s (22135556 bytes in 22.625s)
```

Figure 11.2.1.2 Transfer the model and program to the development board

If you are using the rk3568 development board, execute the following command.

```
adb push install/rk3568_linux_aarch64/rknn_PPOCR_CAMERA_demo/ /userdata/aidemo
```

After the transmission is completed, return to the development board terminal and enter the corresponding directory to execute the following commands for inference.

```
cd /userdata/aidemo/rknn_PPOCR_CAMERA_demo/
./rknn_ppocr_system_demo model/ppocrv4_det.rknn model/ppocrv4_rec.rknn
```

```
root@ATK-DLRK3588:/# cd /userdata/aidemo/rknn_PPOCR_CAMERA_demo/
root@ATK-DLRK3588:/userdata/aidemo/rknn_PPOCR_CAMERA_demo# ls
lib  model  rknn_ppocr_system_demo
root@ATK-DLRK3588:/userdata/aidemo/rknn_PPOCR_CAMERA_demo# ./rknn_ppocr_system_demo model/ppocrv4_det.rknn model/ppocrv4_rec.rknn
model input num: 1, output num: 1
input tensors:
  index=0, name=x, n_dims=4, dims=[1, 480, 480, 3], n_elems=691200, size=691200, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-14, scale=0.018658
output tensors:
  index=0, name=sigmoid_0.tmp_0, n_dims=4, dims=[1, 1, 480, 480], n_elems=230400, size=230400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.0039
22
model is NHWC input fmt
model input height=480, width=480, channel=3
model input num: 1, output num: 1
input tensors:
  index=0, name=x, n_dims=4, dims=[1, 48, 320, 3], n_elems=46080, size=92160, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
output tensors:
  index=0, name=softmax_11.tmp_0, n_dims=3, dims=[1, 40, 6625, 0], n_elems=265000, size=530000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.0
00000
model is NHWC input fmt
model input height=48, width=320, channel=3
Using mplane plugin for capture
[  886.151596] rkisp_hw fdcb0000.rkisp: set isp clk = 594000000Hz
[  886.165822] rkcif-mipi-lvds2: stream[0] start streaming
[  886.165899] rockchip-mipi-csi2 mipi2-csi2: stream on, src_sd: 00000000c0b859fe, sd_name:rockchip-csi2-dphy0
[  886.165906] rockchip-mipi-csi2 mipi2-csi2: stream ON
[  886.165930] rockchip-csi2-dphy0: dphy0, data_rate_mbps 892
[  886.165960] rockchip-csi2-dphy csi2-dphy0: csi2_dphy_s_stream stream on:1, dphy0, ret 0
[  886.165968] imx415 3-001a: s_stream: 1. 3864x2192, hdr: 0, bpp: 10
```

Figure 11.2.1.3 Execute the ppocr detection and recognition program.

You can see the real-time inference pictures displayed on the mipi screen. By pressing the Ctrl+C key on the keyboard in the serial port terminal, you can exit the program.

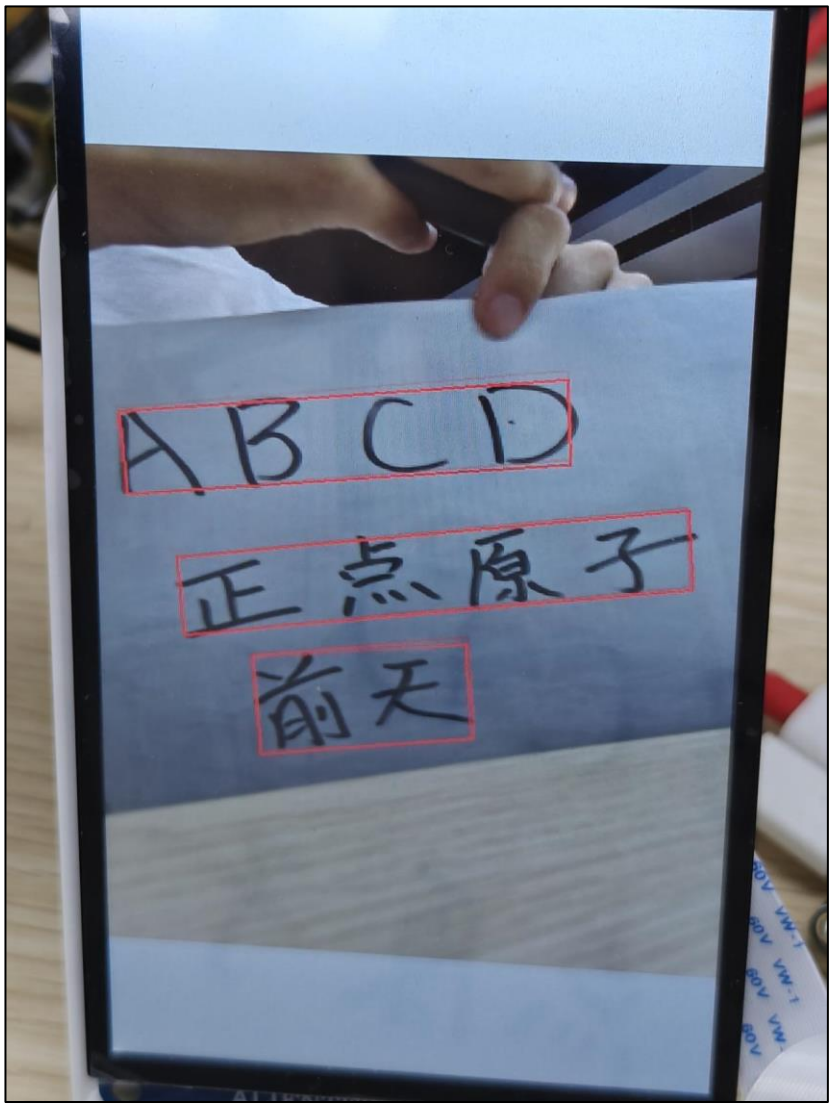


Figure 11.2.1.4 The text detected by the camera

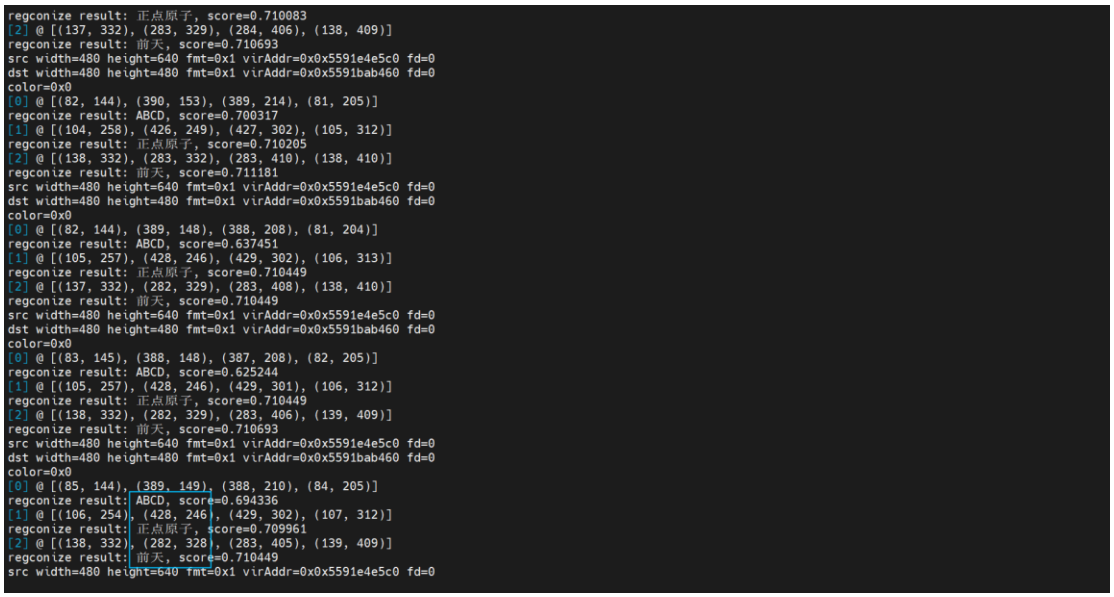


Figure 11.2.1.5 Text recognized by the serial port terminal

Chapter 12. Person Segmentation Routine Test

The person segmentation tool "pp-humanseg" is an open-source project of Baidu PaddlePaddle. We converted it into a RKNN model and successfully deployed it on the RK3568/RK3588 board. The deployment effect was quite good. This chapter will introduce to you how to deploy it on the development board.

This chapter is divided into the following sections:

1. Routine Introduction
2. Routine Test

12.1 Routine Introduction

Distinguishing characters and backgrounds at the pixel level is a classic task in image segmentation and has wide applications. Generally speaking, this task can be divided into two categories: segmentation for half-body portraits, which is referred to as portrait segmentation; and segmentation for full-body and half-body portraits, which is referred to as general portrait segmentation. For portrait segmentation and general portrait segmentation, PaddleSeg has released the PP-HumanSeg series of models, which have the advantages of high segmentation accuracy, fast inference speed, and strong generality. Moreover, the PP-HumanSeg series models can be used out of the box, deployed to products at zero cost, and also support fine-tuning for specific scene data to achieve better segmentation results. This routine adopts general portrait segmentation for full-body images.

For the general human image segmentation task, the Flying Team first constructed a large-scale human image dataset, and then used the state-of-the-art model of PaddleSeg. Finally, multiple PP-HumanSeg general human image segmentation models were released. The PP-HumanSegV2-Lite general human image segmentation model uses the ultra-lightweight segmentation model launched by PaddleSeg, and its mIoU accuracy is improved by 6.5% compared to the V1 model, while the inference time on mobile devices increases by 3ms. The PP-HumanSegV2-Mobile general segmentation model uses the PP-LiteSeg model developed by PaddleSeg, and its mIoU accuracy is improved by 1.49% compared to the V1 model, and the inference time on the server side is reduced by 5.7%. For more information about the PP-HumanSeg models, you can refer to the following link: https://github.com/PaddlePaddle/PaddleSeg/tree/release/2.8/contrib/PP-HumanSeg.

Model name	Optimal input size	Precision mIou(%)	Time taken for reasoning on mobile device (ms)	Server-side inference time (ms)
PP-HumanSegV1-Lite	192x192	86.02	12.3	-
PP-HumanSegV2-Lite	192x192	92.52	15.3	-
PP-HumanSegV1-Mobile	192x192	91.64	-	2.83
PP-HumanSegV2-Mobile	192x192	93.13	-	2.67
PP-HumanSegV1-Server	512x512	96.47	-	24.9

12.2 Routine Testing

Before conducting the test for this routine, please ensure that the relevant reasoning environment and libraries have been installed on the development board as per Chapters 2 and 5, and that the

development board has been connected to the accompanying IMX415 from ALIENTEK and the 5.5-inch 1080p MIPI screen or 720p MIPI screen available from ALIENTEK.

If using the rk3588 development board, we will transfer all the files in the folders of the A disk of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 01, Source Code → 17_pp_human_seg_npu from the A disk to the /userdata/aidemo directory on the development board. If using the rk3568 development board, we will transfer all the files in the folders of the A disk of the development board's CD-ROM - Basic Materials → 01_codes → 01_AI_Routine → 04, Linux 5.10 AI Routine Source Code → 17_pp_human_seg_npu from the A disk to the /userdata/aidemo directory on the development board. Open the terminal in the directory where the routine is located on Ubuntu, and input the following commands.

```
adb push human_seg.zip /userdata/aidemo
```

After the transmission is completed, return to the development board terminal and enter the corresponding directory to execute the following commands.

```
cd /userdata/aidemo/
unzip human_seg.zip
cd human_seg/
python main.py
```

After the operation is successful, have a person or a picture of a person face the camera. Then you can see the area where the person is located being separated out and painted green on the screen. The operation effect is shown as follows.

Figure 12.2.1.1 The actual effect of face segmentation