NPU programming

Introduction to the use of RK NPU

RK3588s has an NPU with 6T computing power. rk's NPU currently adopts a self-developed architecture and only supports the use of non-open source drivers and libraries to operate.

RK's NPU sdk is divided into two parts, the PC side uses rknn-toolkit2, which can be used for model conversion, inference and performance evaluation on the PC side. Specifically, it converts mainstream models such as Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet, PyTorch, etc. to RKNN models and can use this RKNN model for inference simulation, computation time and memory overhead on the PC side. There is another part on the board side, the rknn runtime environment, which contains a set of C API libraries and driver modules to communicate with the NPU, executable programs, etc. This article describes how to use rk's npu sdk.

Related resource download links

- rknn-toolkit2
- rknpu2

Install docker

The following command is executed on an x86 Ubuntu host instead of YY3568

Since rknn-toolkit2 runtime environment has more dependencies, it is recommended to install rknn PC environment directly by docker, the docker provided by rk already contains all the necessary environment.

1、Uninstall the old docker version

```
apt-get remove docker docker-engine docker.io containerd runc
```

2、Install the dependencies.

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-pro
```

3、Trust docker's gpg public key

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

To verify that the public key was added successfully, you can use the following command

```
apt-key fingerprint 0EBFCD88
```

```
~$ apt-key fingerprint 0EBFCD88
pub   4096R/0EBFCD88 2017-02-22
      密钥指纹 = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid                  Docker Release (CE deb) <docker@docker.com>
sub   4096R/F273FCD8 2017-02-22
```

4. Add the software source and install it

```
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(l
apt-get update
apt-get install docker-ce
```

To verify if the installation is successful, you can use the following command

```
docker -v
```

```
~$ docker -v
Docker version 20.10.7, build f0df350
```

Download and run rknn docker

The following command is executed on an x86 Ubuntu host instead of YY3568

First, download rknn-toolkit2. The address is
https://github.com/rockchip-linux/rknn-toolkit2
Note that there is no docker in this place, there is a link to RK's Baidu website on this page, you can go to this link to get the docker

> ### Download
>
> - You can also download all packages, docker image, examples, docs and platform-tools from baidu cloud: RK_NPU_SDK, fetch code: rknn

Open this link and find

Just download this file.
Then if you want to update rknn-toolkit, you can also find it inside this netdisk. I'm using version 1.3 here, which is the same as the full version of the sdk.

After downloading, open the directory where the docker is located, and execute

```
sudo docker load --input rknn-toolkit2-1.3.0-cp36-docker.tar.gz
```

Then execute

```
sudo docker images
```

You can see that this image has been loaded

```
REPOSITORY      TAG           IMAGE ID       CREATED         SIZE
rknn-toolkit2   1.5.0-cp36    5fe51a25db82   6 months ago    5.89GB
```

Then use the command

```
sudo docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v $(pwd)/rknn-to
```

Where -v is to map the directory into the Docker environment. This is an example in rknn-toolkit2, but other directories can be mapped as well. Also /dev/bus/usb is needed for debugging with adb later. If the board you have has adb service, you can turn it on, if not, you can leave it on

> (pwd)/rknn-toolkit2/ in the above command needs to be replaced with the actual directory of the rknn-toolkit2 project, where there is the examples/onnx/yolov5 folder.

After entering docker, check the files under the root directory with the ls command and you can see that there is indeed a folder rknn_yolov5_demo

Run rknn-toolkit2 to generate the model and reason about it

Go to rknn_yolov5_demo in docker and execute

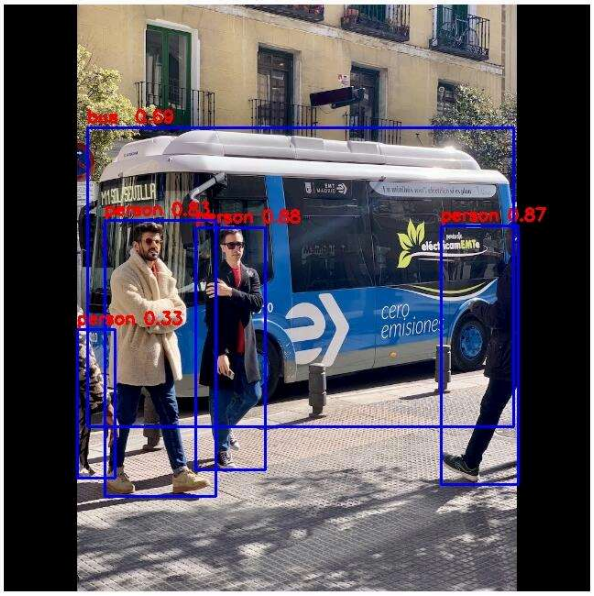
```
python3 ./test.py
```

You can get the following result

```
I rknn buiding done.
done
--> Export rknn model
done
--> Init runtime environment
W init_runtime: Target is None, use simulator!
done
--> Running model
W inference: The 'data_format' has not been set and defaults is nhwc!
Analysing : 100%|################################################| 146/146 [00:00<00:00, 3963.81it/s]
Preparing : 100%|################################################| 146/146 [00:00<00:00, 475.35it/s]
W inference: The dims of input(ndarray) shape (640, 640, 3) is wrong, expect dims is 4! Try expand dims to (1, 640, 640, 3)!
done
class: person, score: 0.8838784694671631
box coordinate left,top,right,down: [209.6862335205078, 243.11955797672272, 285.13685607910156, 507.7035621404648]
class: person, score: 0.8669421076774597
box coordinate left,top,right,down: [477.6677174568176, 241.00597953796387, 561.1506419181824, 523.3208637237549]
class: person, score: 0.826057493686676
box coordinate left,top,right,down: [110.24830067157745, 235.76190769672394, 231.76915538311005, 536.1012514829636]
class: person, score: 0.32633310556411743
box coordinate left,top,right,down: [80.75779604911804, 354.98213291168213, 121.49669003486633, 516.5315389633179]
class: bus , score: 0.6890695095062256
box coordinate left,top,right,down: [91.16828817129135, 134.78936767578125, 556.8909769654274, 460.78936767578125]
root@e30975b8e840:/rknn_yolov5_demo#
```

The converted model is then stored in rknn-toolkit2/examples/onnx/yolov5 on the host, where yolov5s.rknn is the model that supports the rknn format.

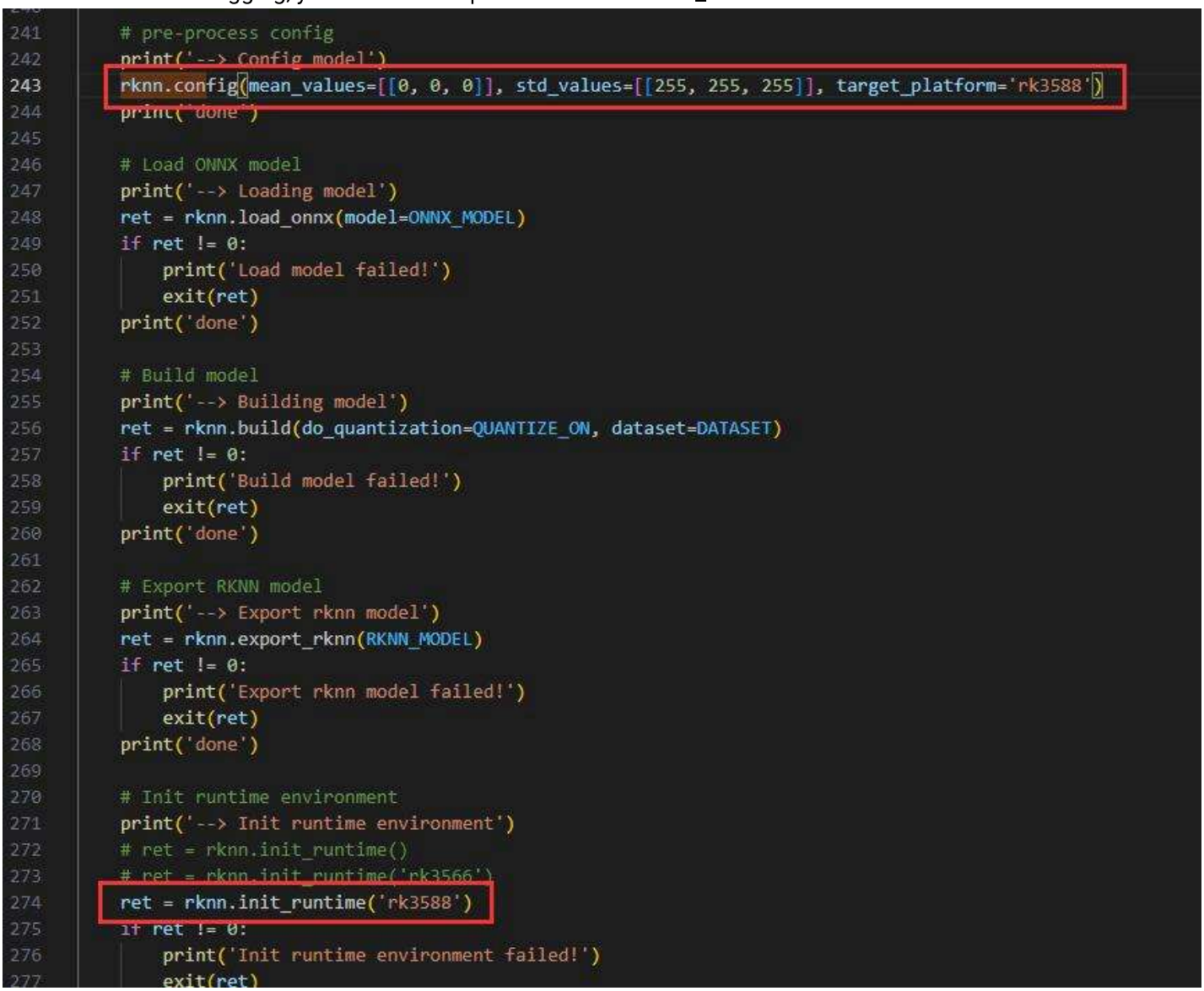
The inference results are saved in result.jpg in this directory, as follows (the original image on the left and the inference results on the right)



Board-side execution

First, open rknn-toolkit2, you need to re-generate the rknn model files suitable for board-side execution. The rknn libraries generated in the way described in the previous section are simulated on a PC. To generate them for the target platform, you need to change test.py .
If there is no adb, then just add the parameter target_platform='rk3588' to rknn.config. If you have adb and need to

connect to it for debugging, you need to add parameter to rknn.init_runtime.



```
    # pre-process config
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform='rk3588')
    print('done')

    # Load ONNX model
    print('--> Loading model')
    ret = rknn.load_onnx(model=ONNX_MODEL)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export RKNN model
    print('--> Export rknn model')
    ret = rknn.export_rknn(RKNN_MODEL)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    # ret = rknn.init_runtime()
    # ret = rknn.init_runtime('rk3566')
    ret = rknn.init_runtime('rk3588')
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
```

Then run python3 again in docker . /test.py to get the yolov5s.rknn file, which is ready to run on the rk3588 device.

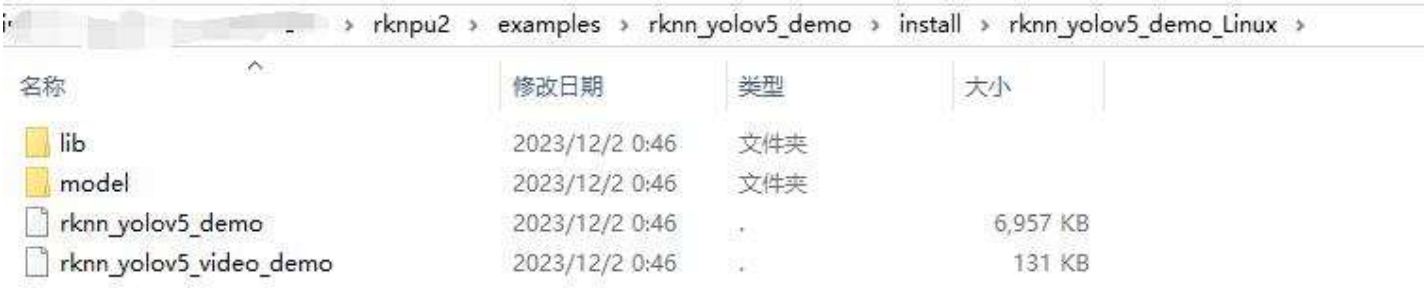
Then on top of the host, open the rknpu2 folder, and here you still have to choose the yolov5 demo.

```
cd rknpu2/examples/rknn_yolov5_demo
```

Modify the application build script build-linux_RK3588.sh. The build toolchain is in prebuilts/gcc/linux-x86/aarch64 in the sdk directory. Users need to modify the PATH and TOOL_CHAIN variables according to the actual sdk installation path.

```
set -e

TARGET_SOC="rk3588"
GCC_COMPILER="aarch64-none-linux-gnu"
TOOL_CHAIN="/media/chen/CRUCIAL/rk3588_linux/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu"

export LD_LIBRARY_PATH=${TOOL_CHAIN}/lib64:$LD_LIBRARY_PATH
export CC=${GCC_COMPILER}-gcc
export CXX=${GCC_COMPILER}-g++

ROOT_PWD=$( cd "$( dirname $0 )" && cd -P "$( dirname "$SOURCE" )" && pwd )

# build
BUILD_DIR=${ROOT_PWD}/build/build_linux_aarch64

if [[ ! -d "${BUILD_DIR}" ]]; then
  mkdir -p ${BUILD_DIR}
fi

cd ${BUILD_DIR}
cmake ../.. -DCMAKE_SYSTEM_NAME=Linux -DTARGET_SOC=${TARGET_SOC}
make -j4
make install
cd -
```

Execute build-linux_RK3588.sh. After that, the following file is generated



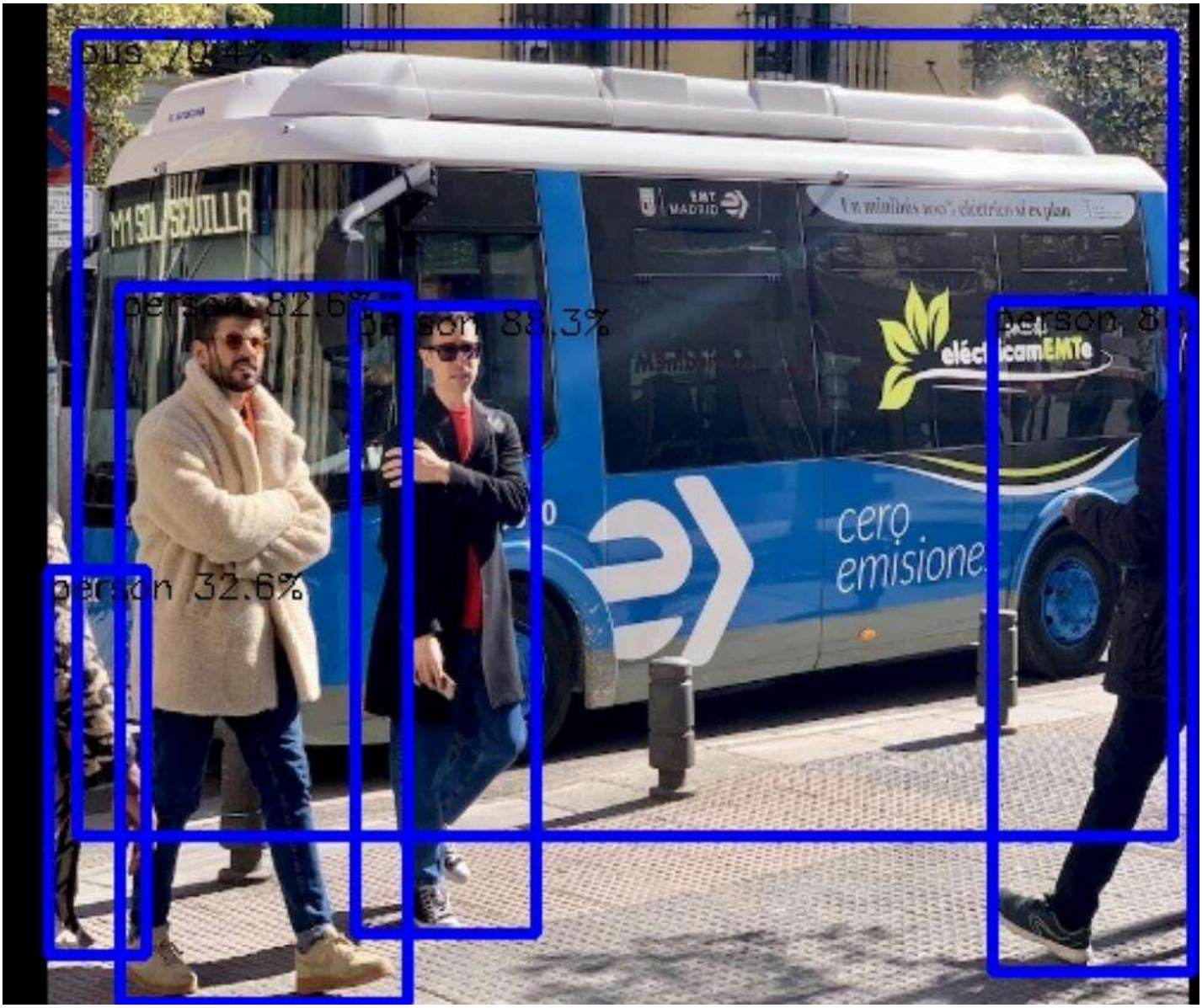
Put this executable, the rknn model, and the bus.jpg and coco_80_labels_list.txt under the model folder together under the same path on the board, you can use ssh,adb etc.

Then run

```
./rknn_yolov5_demo ./model/yolov5s.rknn ./model/bus.jpg
```

```
root@linaro-alip:~# ./rknn_yolov5_demo ./model/yolov5s.rknn ./model/bus.jpg
post process config: box_conf_threshold = 0.25, nms_threshold = 0.45
Read ./model/bus.jpg ...
img width = 640, img height = 640
Loading mode...
sdk version: 1.4.0 (a10f100eb@2022-09-09T09:07:14) driver version: 0.8.2
model input num: 1, output num: 3
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride = 640, size_with_stride=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.0
03922
  index=0, name=269, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, w_stride = 0, size_with_stride=1632000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=83, scale=0.093136
  index=1, name=271, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, w_stride = 0, size_with_stride=408000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=48, scale=0.089854
  index=2, name=273, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, w_stride = 0, size_with_stride=102000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=46, scale=0.078630
model is NHWC input fmt
model input height=640, width=640, channel=3
once run use 33.540000 ms
loadLabelName ./model/coco_80_labels_list.txt
person @ (209 243 285 507) 0.883131
person @ (477 241 561 523) 0.866942
person @ (110 235 231 536) 0.825886
bus @ (92 129 553 466) 0.703667
person @ (80 354 121 516) 0.326333
loop count = 10 , average run  38.056200 ms
```

Will generate out.jpg under this directory, view this file



It is basically the same as the PC analog reasoning, with not much difference in confidence level
By this point, the entire process of reasoning from PC to board execution of an rknn demo is complete.