EMBEDEEP
壹 普 智 慧

EEP-TPU library example program User manual

www.embedeep.com

1. Some instructions on EEP-TPU library files

The library file used by the sample program is: “libeeptpu_pub.so” and the header file is: “eeptpu.h”. The library files are stored in the libs folder, and there are three libraries for different platforms, namely ARM32, AARCH64, and X86. Users can select one of the platforms for testing based on actual conditions. When compiling a neural network algorithm using the EEP-TPU compiler, use the parameter “-- public_bin” to generate a bin file with the suffix “. pub. bin”, and this type of bin file is required for library files.

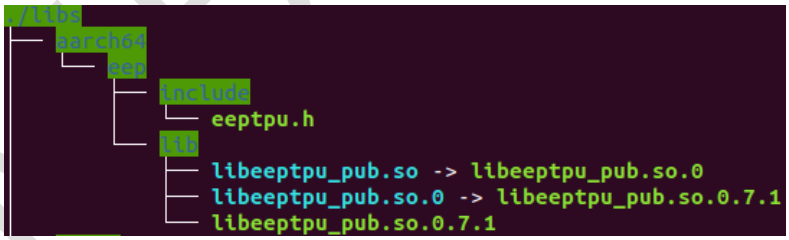
The “Demo/libs” directory is empty by default. Please decompress and copy the latest version of the library file to the “demo/libs” directory.

For example, enter the “libeeptpu_pub” directory and open the command line to run the decompression command:

tar xzvf libeeptpu_pub_v0.7.1.tar.gz

cp -r libeeptpu_pub_v0.7.1/ ../demo/libs/*

Taking the AARCH64 platform as an example, the structure of the “demo/libs” directory is shown in the following figure. Please ensure that the “eep/lib” directory contains three library files, two of which are linked files.



Each sample program has its own folder, such as: classify, icnet, multi_ bins_ test、nntpu_test, yolo, and so on. There are corresponding compilation scripts and running scripts under each sample folder. In addition, the “common" folder contains commonly used code (reading images and reading npy files); and the “eeptpu_bins” folder contains the precompiled algorithm bin files. Please go to the “readme” where this project is located (https://github.com/embedeep) on GitHub to find the corresponding download address and copy it to this folder.

The “EEP-TPU Application Programming Interface (API) User Manual” provides an

introduction to the compilation environment for this library file. Please ensure that the compilation environment under the platform you are using is the same as the library file compilation environment, otherwise compatibility issues may arise.

The EEP-TPU IP configurations used by different hardware platforms may be different. Therefore, when initializing the EEP-TPU in the code, it is necessary to modify the configuration based on the actual situation, such as the register configuration "eeptpu_set_tpu_reg_zones", the memory base address configuration "eeptpu_set_base_address", and so on.

Note that the evaluation version of EEP-TPU on "github" has a usage time limit. After the time limit, the EEP-TPU will not be available (the program gets stuck while running).

2. Sample program compilation:

Copy the entire folder of all sample programs to the target platform, and select the following compilation script based on the platform：

ARM32:	*./compile.sh 32*
AARCH64:	*./compile.sh 64*
X86:	*./compile.sh 86*

3. Sample program running：

Each sample program has a corresponding running script example, test.sh. Before running, please ensure that the "eeptpu_bins" folder contains the corresponding algorithm bin file. You can obtain the download address of the compiled bin file in the introduction to this project on github. After downloading, copy the bin file to this directory.

4. A brief introduction to the functions of the example program:

(1) Sample program: classify

Test program for classified networks. It can test classified networks, such as mobilenet, resnet, vgg, squeezenet, inception and others.

(2) Sample program: icnet

Testing program for ICNet semantic segmentation network. Taking ICNet as an example, the network input is 513×1025. After network reasoning is completed, you can save the inference result picture for viewing.

(3) Sample program: multi_bins_test

Test program for single-core with multi-bin file. Taking the mobilenet and yolov4tiny networks as examples, a single core loads the two networks and infers them in turn.

(4) Sample program: nntpu_test

General test program for neural networks or operators. There is no post processing code. After reasoning, the reasoning results can be output to a "test_output_data. txt" text file for comparison with the software reasoning results of the network or operator. Support Pack mode input data. Support NPY input data or image input data.

(5) Sample program: yolo

Test program for object detection networks. It can be tested using commonly used object detection neural networks, such as: yolov3 series/yolov4 series/mobilenetyolov3/mobilenet-ssd, etc. After network inference is complete, you can save the inference result picture for viewing.

If the bin file is compiled from the Dark-Net framework model, the test program needs to add the parameter: --pixel 'RGB', specifying that the input images are read in RGB order. Because when training a network based on the Dark Net framework, the input images are generally read in RGB order. An example of running a script is as follows:

sudo./demo --bin ../eeptpu_bins/eeptpu_s2_yolov4tiny.pub.bin --input ./input/004545.jpg --pixel 'RGB'

(6) Other Sample programs:

Because the EEP-TPU we open source on github is not fully functional, some features are

not available on this version of TPU. Such as multi-core TPU testing, parameter multiplexing, “Pack” precision conversion mode testing, algorithm jump testing, etc., these functions are described in more detail in the “EEP-TPU Compiler User Manual”. If you need to experience these commercial features, you can contact us.

5. Introduction to some test programs that do not provide code:

Multi-core TPU testing:

A single network can be compiled and deployed on a multi-core TPU for inference, and the inference time will be faster than a single core TPU.

Parameter multiplexing:

Under multi-core TPU, if the same network needs to be used by multiple TPU cores, the algorithm can be loaded in the form of parameter multiplexing, so that algorithm parameters can be used by multiple TPUs only once, in order to reduce memory footprint.

Pack precision conversion mode:

The input data supports FP32/FP16/INT8/INT16/INT32/UINT8 format. For example, when opencv reads images, the data stored by “cv::Mat” can be directly used by the TPU without conversion.

Algorithm jump:

At the TPU level, the algorithm 1 is directly jumped to algorithm 2, and the inference result of algorithm 1 is used as the input of algorithm 2.