Previous (../../host_lib_0.8.0/host_api/system/)
System API reference (../../host_lib_0.8.0/host_api/…

Next (../yolo_example/)
Yolo Example (../yolo_example/)

Kneron Linux Toolchain Manual

2021 Aug Toolchain v0.15.5

PDF Downloads (manual.pdf)

0. Overview

KDP toolchain is a set of software which provide inputs and simulate the operation in the hardware KDP 520 and KDP 720. For better environment compatibility, we provide a docker which include all the dependencies as well as the toolchain software.

This document is compatible with `kneron/toolchain:v0.15.5`.

Performance simulation result on NPU KDP520:

Model	Size	FPS (npu only)	Time(npu only)	Has CPU node(s)?
Inception v3	224x224	6.3	158 ms	No
Inception v4	299x299	0.48	2068 ms	No
Mobilenet v1	224x224	60.7	16.5 ms	No
Mobilenet v2	224x224	61.3	16.3 ms	No
Mobilenet v2 ssdlite	300x300	30.4	32.9 ms	No
Resnet50 v1.5	224x224	7.02	142.4 ms	No
OpenPose	256x256	0.61	1639 ms	No
SRCNN	384x384	7.04	142 ms	No
Tiny yolo v3	416x416	22.8	43.8 ms	Yes
Yolo v3	416x416	1.5	666.7 ms	Yes

Performance simulation result on NPU KDP720:

Model	Size	FPS (npu only)	Time(npu only)	Has CPU node(s)?
Inception v3	224x224	80.9	12.4 ms	No
Inception v4	299x299	19.9	50.2 ms	No

Model	Size	FPS (npu only)	Time(npu only)	Has CPU node(s)?
Mobilenet v1	224x224	404	2.48 ms	No
Mobilenet v2	224x224	624	1.60 ms	No
Mobilenet v2 ssdlite	300x300	283	3.54 ms	No
Resnet50 v1.5	224x224	52.3	19.1 ms	No
OpenPose	256x256	5.3	189 ms	No
SRCNN	384x384	127	7.87 ms	No
Tiny yolo v3	416x416	148	6.75 ms	No
Yolo v3	416x416	10.5	95.3 ms	No
Centernet res101	512x512	3.02	331 ms	No
Unet	384x384	2.83	354 ms	No

In this document, you'll learn:

1. How to install and use the toolchain docker.
2. What tools are in the toolchain.
3. How to utilize the tools through Python API.

** Major changes of past versions**

- **[v0.15.0]**
    - Document now is written for Python API. The original script document can be found in Command Line Script Tools (http://doc.kneron.com/docs/toolchain/command_line/).
- **[v0.14.0]**
    - ONNX is updated to 1.6.0
    - Pytorch is updated to 1.7.1
    - Introduce toolchain Python API.
- **[v0.13.0]**
    - 520 toolchain and 720 toolchain now is combined into one. But the scripts names and paths are the same as before. You don't need to learn it again.
    - E2E simulator has been updated to a new version. Usage changed. Please check its document.
- **[v0.12.0]** Introduce `convert_model.py` which simplify the conversion process.
- **[v0.11.0]** Batch compile now generate `.nef` files to simplify the output.
- **[v0.10.0]**
    - `input_params.json` and `batch_input_params.json` have been simplified a lot. Please check the document for details.
    - `simulator.sh`, `emulator.sh` and draw yolo image scripts are no longer available from `/workspace/scripts`. They have been moved to E2E simulator. Please check its document.
- **[v0.9.0]** In the example, the mount folder `/docker_mount` is separated from the interactive folder `/data1` to avoid unexpected file changes. Users need to copy data between the mount folder and the interactive folder. Of course you can still mount on `/data1`. But please be careful that the results folder under `/data1` may be overwritten.

1. Installation

Review the system requirements below before start installing and using the toolchain.

1.1 System requirements

1. **Hardware**: Minimum quad-core CPU, 4GB RAM and 6GB free disk space.
2. **Operating system**: Window 10 x64 version 1903 or higher with build 18362 or higher. Ubuntu 16.04 x64 or higher. Other OS which can run docker later than 19.03 may also work. But they are not tested. Please take the risk yourself.
3. **Docker**: Docker Desktop later than 19.03. Here is a link (https://www.docker.com/products/docker-desktop) to download Docker Desktop.

> **TIPS:**
>
> For Windows 10 users, we recommend using docker with wsl2, which is Windows subsystem Linux provided by Microsoft. Here is how to install wsl2 (https://docs.microsoft.com/en-us/windows/wsl/install-win10) and how to install and run docker with wsl2 (https://docs.docker.com/docker-for-windows/wsl/). Also, you might to want to adjust the resources docker use to ensure the tools' normal usage. Please check the FAQ at the end of this document on how to do that.

Please double-check whether the docker is successfully installed and callable from the console before going on to the next section. If there is any problem about the docker installation, please search online or go to the docker community for further support. The questions about the docker is beyond the reach of this document.

1.2 Pull the latest toolchain image

All the following steps are on the command line. Please make sure you have the access to it.

> TIPS:
>
> You may need `sudo` to run the docker commands, which depends on your system configuration.

You can use the following command to pull the latest toolchain docker.

```
docker pull kneron/toolchain:latest
```

Note that this document is compatible with toolchain v0.15.5. You can find the version of the toolchain in `/workspace/version.txt` inside the docker. If you find your toolchain is later than v0.15.5, you may need to find the latest document from the online document center (http://doc.kneron.com/docs).

2. Toolchain Docker Overview

After pulling the desired toolchain, now we can start walking through the process. In all the following sections, we use `kneron/toolchain:latest` as the docker image. Before we actually start the docker, we'd better provide a folder which contains the model files you want to test in our docker, for example, `/mnt/docker`. Then, we can use the following command to start the docker and work in the docker environment:

```
docker run --rm -it -v /mnt/docker:/docker_mount kneron/toolchain:latest
```

> TIPS:
>
> The mount folder path here is recommended to be an absolute path.

Here are the brief explanations for the flags. For detailed explanations, please visit docker documents (https://docs.docker.com/engine/reference/run/).

- `--rm`: the container will be removed after it exists. Each time we use `docker run`, we create a new docker container. Thus, without this flag, the docker will consumes more and more disk space.
- `-it`: enter the interactive mode so we can use the bash.
- `-v`: mount a folder into the docker container. Thus, we can visit the desired files from the host and save the result from the container.

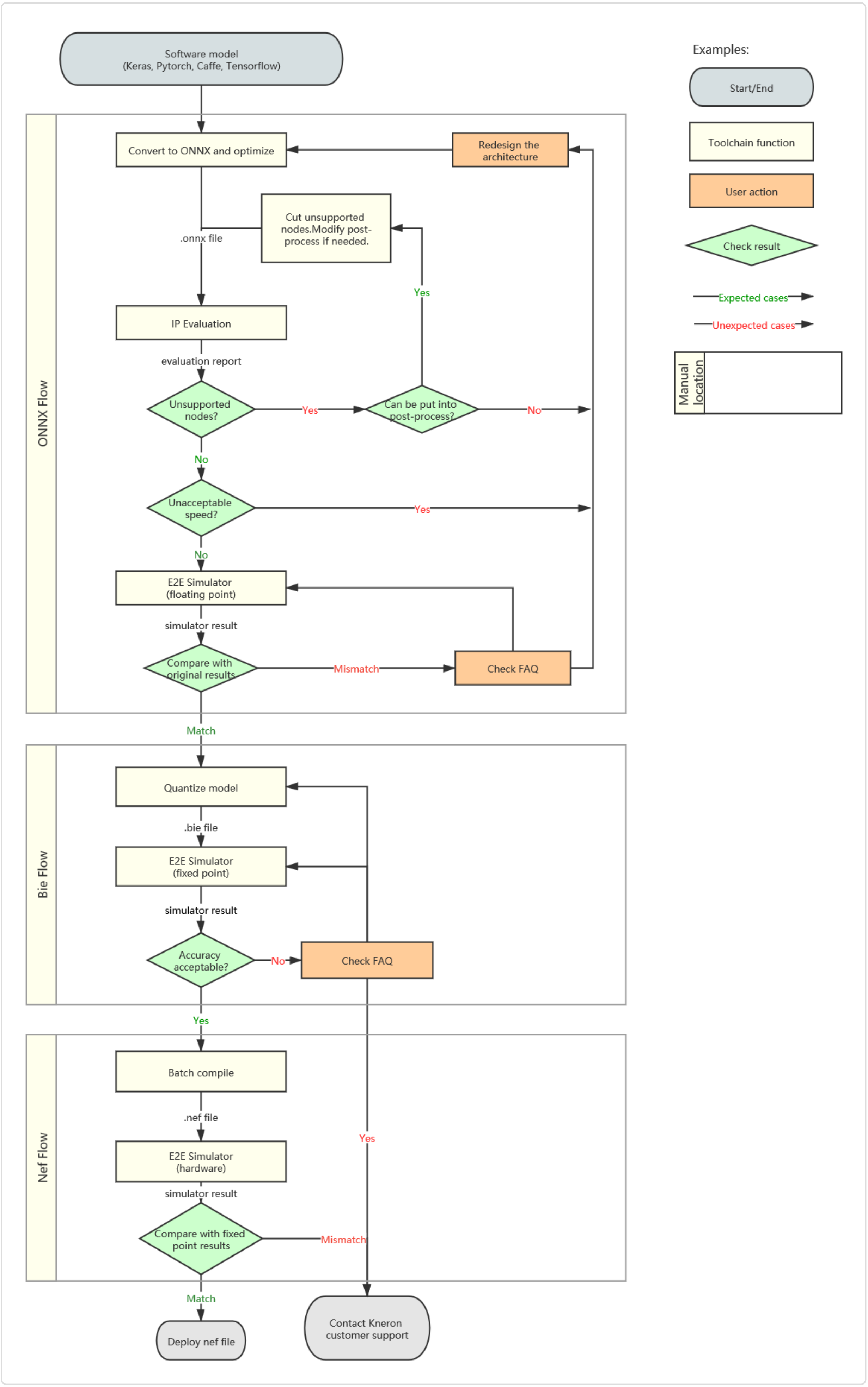
2.1 Folder structure

After logging into the container, you are under `/workspace`, where all the tools are. Here is the folder structure and their usage:

```
/workspace
|-- E2E_Simulator       # End to end simulator
|-- ai_training         # AI training project.
|-- cmake               # Environment
|-- examples            # Example for the workflow, will be used later.
|-- libs                # The libraries
|   |-- ONNX_Convertor  # ONNX Converters and optimizer scripts, will be discussed i
|   |-- compiler        # Compiler for the hardware and the IP evaluator to infer th
|   |-- dynasty         # Simulator which only simulates the calculation.
|   |-- fpAnalyser      # Analyze the model and provide fixed point information.
|   `-- hw_c_sim        # Hardware simulator which simulate all the hardware behavio
|-- miniconda           # Environment
|-- scripts             # Scripts to run the tools, will be discussed in section 3.
`-- version.txt
```

2.2 Work flow

Before we start actually introducing the usage, let us go through the general work flow.



Software model (Keras, Pytorch, Caffe, Tensorflow)
ONNX Flow
Convert to ONNX and optimize
Redesign the architecture
Cut unsupported nodes.Modify post-process if needed.
.onnx file
IP Evaluation
evaluation report
Unsupported nodes?
Yes
Can be put into post-process?
Yes
No
No
Unacceptable speed?
Yes
No
E2E Simulator (floating point)
simulator result
Compare with original results
Mismatch
Check FAQ
Match
Bie Flow
Quantize model
.bie file
E2E Simulator (fixed point)
simulator result
Accuracy acceptable?
No
Check FAQ
Yes
Nef Flow
Batch compile
.nef file
E2E Simulator (hardware)
simulator result
Compare with fixed point results
Mismatch
Yes
Match
Deploy nef file
Contact Kneron customer support
Examples:
Start/End
Toolchain function
User action
Check result
Expected cases
Unexpected cases
Manual location

Figure 1. Diagram of working flow

To keep the diagram as clear as possible, some details are omitted. But it is enough to show the general steps. There are three main sections:

1. ONNX section. Convert the model from different platforms to onnx and optimize the onnx file. Evaluate the onnx the model to check the operator support and the estimate performance. Then, test the onnx model and compare the result with the source.
2. Bie section. Quantize the model and generate bie file. Test the bie file and compare the result with the previous step.
3. Nef section. Batch compile multiple bie models into a bie binary file. Test the nef file and compare the result with the previous step.

> The workflow has been changed a lot since v0.15.0. We recommend Python API instead of the original scritps for a more smooth workflow. But this doesn't means the scripts used before are abandoned. They are still available and the document can be found in Command Line Script Tools (http://doc.kneron.com/docs/toolchain/command_line/). The detailed document of the Python API can be found in Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/)

2.3 Supported operators

Table 1.1 shows the list of functions KDP520 supports base on ONNX 1.6.1.

Table 1.1 The functions KDP520 NPU supports

Type	**Operarots**	**Applicable Subset**	**Spec.**
Convolution	Conv	Kernel dimension	1x1 up to 11x11
		Strides	1,2,4
	Pad		0-15
	Depthwise Conv		Yes
	Deconvolution		Use Upsampling + Conv
Pooling	MaxPool	3x3	stride 1,2,3
	MaxPool	2x2	stride 1,2
	AveragePool	3x3	stride 1,2,3
	AveragePool	2x2	stride 1,2
	GlobalAveragePool		support
	GlobalMaxPool		support
Activation	Relu		support
	LeakyRelu		support
	PRelu		support
Other processing	BatchNormalization		support
	Add		support
	Concat		axis = 1
	Gemm or Dense/Fully Connected		support
	Flatten		support
	Clip		min = 0

Table 1.2 shows the list of functions KDP720 supports base on ONNX 1.6.1.

Table 1.2 The functions KDP720 NPU supports

Node	Applicable Subset	Spec.
Relu		support
PRelu		support
LeakyRelu		support
Sigmoid		support
Clip		min = 0
Tanh		support
BatchNormalization	up to 4D input	support
Conv		strides < [4, 16]
Pad		spacial dimension only
ConvTranspose		strides = [1, 1], [2, 2]
Upsample		support
Gemm	2D input	support
Flatten	Before Gemm	support
Add		support
Concat		axis = 1
Mul		support
MaxPool		kernel = [1, 1], [2, 2], [3, 3]
AveragePool	3x3	kernel = [1, 1], [2, 2], [3, 3]
GlobalAveragePool	4D input	support
GlobalMaxPool		support
MaxRoiPool		support
Slice	input dimension <= 4	support

3 ONNX Workflow

Our toolchain utilities take ONNX files as inputs. The ONNX workflow is mainly about convert models from other platforms to ONNX and prepare the onnx for the quantization and compilation. There are three main steps: conversion, evaluation and testing.

3.0 Preparation

Most of the following codes are using the python API. You may need to import necessary libraries before copying the code in this document.

```
import onnx
import ktc
```

The details of the Python API can be found in Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) document.

3.1 Model Conversion and Optimization

The onnx converter part currently support Keras, TFLite, a subset of Tensorflow, Caffe and Pytorch. Here, we will only briefly introduce some common usage of the python API. This part of python API is based on our converter and optimizer open-source project which can be found on Github https://github.com/kneron/ONNX_Convertor (https://github.com/kneron/ONNX_Convertor). The detailed usage of those converter scripts can be found in ONNX Converter (http://doc.kneron.com/docs/toolchain/converters/). The Tensorflow ktc api is not introduced here. We recommend export the tensorflow model to tflite and convert the tflite model. If you really want to try convert a pb file, please check the onnx converter project.

The example models used in the following converter command are not included in the docker by default. They can be found on Github https://github.com/kneron/ConvertorExamples (https://github.com/kneron/ConvertorExamples). You can download them through these terminal commands:

```
git clone https://github.com/kneron/ConvertorExamples.git
cd ConvertorExamples && git lfs pull
```

3.1.1 Keras to ONNX

For Keras, our converter support models from Keras 2.2.4. **Note that `tf.keras` and Keras 2.3 is not supported.** You may need to export the model as tflite model and see section 3.1.4 for TF Lite model conversion.

Suppose there is an onet hdf5 model exported By Keras, you need to convert it to onnx by the following python code:

```
result_m = ktc.onnx_optimizer.keras2onnx_flow('/data1/ConvertorExamples/keras_examp
```

In this line of python code, `ktc.onnx_optimizer.keras2onnx_flow` is the function that takes an hdf5 file path and convert the hdf5 into an onnx object. The return value `result_m` is the converted onnx object. It need to take one more optimization step (section 3.1.5) before going into the next section.

There might be some warning log printed out by the Tensorflow backend, but we can ignore it since we do not actually run it. You can check whether the conversion succeed by whether there is any exception raised.

This function has more parameters for fine-tuning. Please check Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) if needed.

3.1.2 Pytorch to ONNX

Our Python API do not actually convert the python model. It only takes Pytorch exported onnx object as the input and optimize it. Pleace checkout the tips below on how to export an onnx with Pytorch.

The Pytorch inside the docker is version 1.7.1. We currently only support models exported by Pytorch version >=1.0.0, <=1.7.1. Other versions are not tested.

> TIPS on export onnx using `torch.onnx`:
>
> You can use `torch.onnx` to export your model into onnx object. Here is the Pytorch 1.7.1 version document (https://pytorch.org/docs/1.6.1/onnx.html) for `onnx.torch`. An example code for exporting the model is:

```
import torch.onnx
dummy_input = torch.randn(1, 3, 224, 224)
torch.onnx.export(model, dummy_input, 'output.onnx', opset_version=11)
```

> In the example, `(1, 3, 224, 224)` are batch size, input channel, input height and input width. `model` is the pytorch model object you want to export. `output.onnx` is the output onnx file path.

Pytorch exported onnx needs to pass through a special optimization designed for pytorch exported models first. Suppose the input file is loaded into a onnx object `exported_m`, here is the python code for pytorch exported onnx optimization:

```
result_m = ktc.onnx_optimizer.torch_exported_onnx_flow(exported_m)
```

In this line of python code, `ktc.onnx_optimizer.torch_exported_onnx_flow` is the function that takes an onnx object and optimize it. The return value `result_m` is the optimized onnx object. It need to take one more general onnx optimization step (section 3.1.5) before going into the next section.

For the example in ConvertorExamples project, the whole process would be:

```
import torch
import torch.onnx

# Load the pth saved model
pth_model = torch.load("/data1/ConvertorExamples/keras_example/resnet34.pth", map_lo
# Export the model
dummy_input = torch.randn(1, 3, 224, 224)
torch.onnx.export(pth_model, dummy_input, '/data1/resnet34.onnx', opset_version=11)
# Load the exported onnx model as an onnx object
exported_m = onnx.load('/data1/resnet34.onnx')
# Optimize the exported onnx object
result_m = ktc.onnx_optimizer.torch_exported_onnx_flow(exported_m)
```

There might be some warning log printed out by the Pytorch because our example model is a little old. But we can ignore it since we do not actually run it. You can check whether the conversion succeed by whether there is any exception raised.

Crash due to name conflict

If you meet the errors related to `node not found` or `invalid input`, this might be caused by a bug in the onnx library. Please try set `disable_fuse_bn` to `True`. The code would be:

```
result_m = ktc.onnx_optimizer.torch_exported_onnx_flow(exported_m, disable_fuse_bn=T
```

3.1.3 Caffe to ONNX

For caffe, we only support model which can be loaded by Intel Caffe 1.0 (https://github.com/intel/caffe).

Here we will use the example from ConvertorExamples. You can find two files for the caffe model: the model structure definition file `mobilenetv2.prototxt` and the model weight file `mobilenetv2.caffemodel`. Here is the example python code for model conversion:

```
result_m = ktc.onnx_optimizer.caffe2onnx_flow('/data1/ConvertorExamples/caffe_examp
```

In this line of python code, `ktc.onnx_optimizer.caffe2onnx_flow` is the function that takes caffe model file paths and convert the them into an onnx object. The return value `result_m` is the converted onnx object. It need to take one more optimization step (section 3.1.5) before going into the next section.

3.1.4 TF Lite to ONNX

We only support unquantized TF Lite models for now. Also tensorflow 2 is not supported yet.

Suppose we are using the tflite file `model_unquant.tflite` from the ConvertorExamples, here is the example python code:

```
result_m = ktc.onnx_optimizer.tflite2onnx_flow('/data1/ConvertorExamples/tflite_exam
```

In this line of python code, `ktc.onnx_optimizer.tflite2onnx_flow` is the function that takes an tflite file path and convert the tflite into an onnx object. The return value `result_m` is the converted onnx object. It need to take one more optimization step (section 3.1.5) before going into the next section.

There might be some warning log printed out by the Tensorflow backend, but we can ignore it since we do not actually run it. You can check whether the conversion succeed by whether there is any exception raised.

This function has more parameters for fine-tuning. Please check Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) if needed.

3.1.5 ONNX Optimization

We provide a general onnx optimize API. We strongly recommend that all the onnx, including the onnx generated from the previous subsections, shall pass this API before going into the next section. This general onnx optimization API would modify the onnx graph to fit the toolchain and Kneron hardware specification. The optimization includes: inference internal value_info shapes, fuse consecutive operators, eliminate do-nothing operators, replace high-cost operators with low-cost operators, etc..

Suppose we have a onnx object, here is the example python code:

```
optimized_m = ktc.onnx_optimizer.onnx2onnx_flow(result_m, eliminate_tail=True)
```

In this line of python code, `ktc.onnx_optimizer.onnx2onnx_flow` is the function that takes an onnx object and optimize it. The return value `result_m` is the converted onnx object.

Note that for hardware usage, `eliminate_tail` should be set to true as in the example. This option eliminate the no calculation operators and the npu unsupported operators (Reshape, Transpose, ...) at the last of the graph. However, since this changes the graph structure, you may need to check the model yourself and add the related functions into post-process to keep the algorithm consistent. If you only want to use onnx model for software testing, the `eliminate_tail` can be set to false to keep the model same as the input from the mathematics perspective.

This function has more parameters for fine-tuning. Please check Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) if needed.

By the way, to save the model, you can use the following function from the onnx package.

```
# Save the onnx object optimized_m to path /data1/optimized.onnx.
onnx.save(optimized_m, '/data1/optimized.onnx')
```

Crash due to name conflict

If you meet the errors related to `node not found` or `invalid input`, this might be caused by a bug in the onnx library. Please try set `disable_fuse_bn` to `True`. The code would be:

```
optimized_m = ktc.onnx_optimizer.onnx2onnx_flow(result_m, eliminate_tail=True, disal
```

Error due to the opset version

If you have met errors which are related to the opset version or the ir version. Please check section 3.1.6 to update your model first.

3.1.6 ONNX Opset Upgrade

From toolchain version 0.14.0, ONNX in the docker has been updated from 1.4.1 to 1.6. And the default opset that converters support is changed from opset 9 into opset 11. IR version is updated from 4 to 6. Thus, if you have a onnx model with opset 9 or IR version 4, you may need to update it with the following Python API:

```
new_m = ktc.onnx_optimizer.onnx1_4to1_6(old_m)
```

In this line of python code, `ktc.onnx_optimizer.onnx1_4to1_6` is the function that takes an old version onnx object and upgrade it. The return value `new_m` is the converted onnx object. It need to take one more optimization step (section 3.1.5) before going into the next section. Even if you have already passed optimizar before, we still recommend you do it again after this upgrade.

3.1.7 ONNX Editor

KL520/KL720 NPU supports most of the compute extensive OPs, such as Conv, BatchNormalization, Fully Connect/GEMM, in order to speed up the model inference run time. On the other hand, there are some OPs that KL520 NPU cannot support well, such as `Softmax` or `Sigmod`. However, these OPs usually are not compute extensive and they are better to execute in CPU. Therefore, Kneron provides python APIs which help user modify the model so that KL520 NPU can run the model more efficiently.

You can find the detailed description of this tool from Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) for the python API and ONNX Converter (http://doc.kneron.com/docs/toolchain/converters/) for the command usage.

3.2 IP Evaluation

Before we start quantizing the model and try simulating the model, we need to test if the model can be taken by the toolchain structure and estimate the performance. IP evaluator is such a tool which can estimate the performance of your model and check if there is any operator or structure not supported by our toolchain.

From this section, we would use the LittleNet which is included in the docker as an example. You can find it under /workspace/examples/LittleNet.

We need to create a `ktc.ModelConfig` object. The `ktc.ModelConfig` is the class which contains the basic needed information of a model. You can initilize it through the API below.

```
km = ktc.ModelConfig(id, version, platform, onnx_model=None, onnx_path=None, bie_pa
```

- `id` is the identifier of the model. It should be a integer greater than 0. ID before 32768 are reserved for Kneron models. Please use ID greater than 32768 for custom models.
- `version` is the model version. It should be a four digit hex code which is written as string, e.g. '001a'.
- `platform` is the target platform for this model. It should be either '520' or '720'.
- `onnx_model`, `onnx_path` and `bie_path`. User should provide one of those three parameter and only one. It stores the model itself. `onnx_model` takes the onnx object which is generated through the converter APIs or loaded through onnx library. `onnx_path` is the path to a onnx file. `bie_path` is the path to a bie file. The bie file is the file generated by the kneron toolchain after quantization, which is introduced in the later section.

For this example, we create the LittleNet ModelConfig with the following python code:

```
km = ktc.ModelConfig(32769, "0001", "520", onnx_path="/workspace/examples/LittleNet/
```

`evaluate` is class function of the `ktc.ModelConfig`.

```
eval_result = km.evaluate()
```

The evaluation result will be returned as string. User can also find the evaluation result under /data1/compiler/. But the report file names are different for different platforms.

- 520: ip_eval_prof.txt

- 720: ProfileResult.txt

If the model is not supported, there would be warning messages or exceptions. Please modify the model structure referring to the message. Please check the report to see if the performance meets your expectation. Please consider redesign the network structure. Also note that the evaluator report only considers the performance of NPU. Thus, if the model contains many operators that are not supported by NPU but by CPU, the actual performance would be even worse.

3.3 E2E Simulator Check (Floating Point)

Before going into the next section of quantization, we need to ensure the optimized onnx file can produce the same result as the originally designed model.

Here we introduce the E2E simulator which is the abbreviation for end to end simulator. It can inference a model and simulate the calculation of the hardware. The inference function of the E2E simulator is called `ktc.kneron_inference`. Here we are using the onnx as the input model. But it also can take bie file and nef file which would be introduced later.

The python code would be like:

```
inf_results = ktc.kneron_inference(input_data, onnx_file="/workspace/examples/Littl
```

In the code above, `inf_results` is a list of result data. `onnx_file` is the path to the input onnx. `input_data` is a list of input data after preprocess the `input_names` is a list of string mapping the input data to specific input on the graph using the sequence. **Note that the input shoule have the same dimension as the model but in channel last format.**

Here we provide a very simple preprocess function which only do the resize and normalization:

```
from PIL import Image
import numpy as np

def preprocess(input_file):
    image = Image.open(input_file)
    image = image.convert("RGB")
    img_data = np.array(image.resize((112, 96), Image.BILINEAR)) / 255
    img_data = np.transpose(img_data, (1, 0, 2))
    return img_data
```

Then, the full code of this section would be:

```
from PIL import Image
import numpy as np

def preprocess(input_file):
    image = Image.open(input_file)
    image = image.convert("RGB")
    img_data = np.array(image.resize((112, 96), Image.BILINEAR)) / 255
    img_data = np.transpose(img_data, (1, 0, 2))
    return img_data

input_data = [preprocess("/workspace/examples/LittleNet/pytorch_imgs/Abdullah_0001.
inf_results = ktc.kneron_inference(input_data, onnx_file="/workspace/examples/Littl
```

Since we want to focus on the toolchain usage here, we do not provide any postprocess. In reality, you may want to have your own postprocess function in Python, too.

After getting the `inf_results` and post-process it, you may want to compare the result with the one generated by the source model. For example, if the source model is from pytorch, you may want to try inference the source model using Pytorch with the same input image to see if the results match. If the result mismatch, please check FAQ 1 for possible solution.

Since we do not actually has any source model here for the simplicity of example, we would skip the step of comparing result.

4 BIE Workflow

As mentioned briefly in the previous section, the bie file is the model file which is usually generated after quantization. It is encrpyted and not available for visuanlization. In this chapter, we would go through the steps of quantization.

4.1 Quantization

Quantization is the step where the floating-point weight are quantized into fixed-point to reduce the size and the calculation complexity. The Python API for this step is called `analysis`. It is also a class function of `ktc.ModelConfig`. It takes a dictionary as input.

```
analysis(input_mapping, output_bie = None, threads = 4)
```

`input_mapping` is the a dictionary which maps a list of input data to a specific input name. Generally speaking, the quantization would be preciser with more input data. `output_bie` is the path where you want your bie generated. By default, it is under /data1/fpAnalyser. `threads` is the threads number you want to utilize. Please note more threads would lead to more RAM usage as well. The return value is the generated bie path.

Please also note that this step would be very time-consuming since it analysis the model with every input data you provide.

Here as a simple example, we only use four input image as exmaple and run it with the `ktc.ModelConfig` object `km` created in section 3.2:

```
# Preprocess images and create the input mapping
input_images = [
    preprocess("/workspace/examples/LittleNet/pytorch_imgs/Abdullah_0001.png"),
    preprocess("/workspace/examples/LittleNet/pytorch_imgs/Abdullah_0002.png"),
    preprocess("/workspace/examples/LittleNet/pytorch_imgs/Abdullah_0003.png"),
    preprocess("/workspace/examples/LittleNet/pytorch_imgs/Abdullah_0004.png"),
]
input_mapping = {"data_out": input_images}

# Quantization
bie_path = km.analysis(input_mapping, output_bie = None, threads = 4)
```

This function has more parameters for fine-tuning. Please check Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) if needed.

4.2 E2E Simulator Check (Fixed Point)

Before going into the next section of compilation, we need to ensure the quantized model do not lose too much precision.

We would use `ktc.kneron_inference` here, too. But here we are using the generated bie file as the input.

The python code would be like:

```
fixed_results = ktc.kneron_inference(input_data, bie_file=bie_path, input_names=["da
```

The usage is almost the same as using onnx. In the code above, `inf_results` is a list of result data. `bie_file` is the path to the input bie. `input_data` is a list of input data after preprocess the `input_names` is a list of model input name. The requirement is the same as in section 3.3. We also need to provide the radix value which is a special value can be obtained by `ktc.get_radix`. If your platform is not 520, you may need an extra parameter `platform`, e.g. `platform=720`.

`ktc.get_radix` takes the preprocessed input images as input and generate the radix value needed by hardware E2E simulation.

Here we use the same input `input_data` which we used in section 3.3. And the `bie_path` is the return value in section 4.1.

The full code would be:

```
radix = ktc.get_radix(input_images)
fixed_results = ktc.kneron_inference(input_data, bie_file=bie_path, input_names=["da
```

As mentioned above, we do not provide any postprocess. In reality, you may want to have your own postprocess function in Python, too.

After getting the `fixed_results` and post-process it, you may want to compare the result with the `inf_results` which is generated in section 3.3 to see if the precision lose too much. If the result is unacceptable, please check FAQ 2 for possible solutions.

5 NEF Workflow

The nef file is the binary file after compiling and can be taken by the Kneron hardware. But unlike the utilities mentioned above, the process in this chapter can compile multiple models into one nef file. This process is call batch process. In this chapter, we would go through how to batch compile and how to verify the nef file.

5.1 Batch Compile

Batch compile turns multiple models into a single binary file. But, we would start with single model first.

The Python API is very simple:

```
compile_result = ktc.compile(model_list)
```

The `compile_result` is the path for the generated nef file. By default, it is under /data1/batch_compile. It takes a list of `ktc.ModelConfig` object as the input `model_list`. The usage of `kt.ModelConfig` can be found in section 3.2. Note that the ModelConfig onject must have bie file inside. In details, it must be under either of the following status: the ModelConfig is initialized with `bie_path`, the ModelConfig is initialized with `onnx_model` or `onnx_path` but it have successfully run `analysis` function.

For the LittleNet example, please check the code below. Note that `km` is the `ktc.ModelConfig` object we generate in section 3.2 and use in the section 4.

```
compile_result = ktc.compile([km])
```

For multiple models, we can simply extend the model list.

```
# dummy.bie is not a real example bie which is available in the docker. Just for co
# Please adjust the parameters according to your actual input.
km2 = ktc.ModelConfig(32770, "0001", "520", bie_path="dummy.bie")
compile_result = ktc.compile([km, km2])
```

Note that for multiple models, all the models should share the same hardware platform and the model ID should be different.

This function has more parameters for fine-tuning. Please check Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) if needed.

5.2 E2E Simulator Check (Hardware)

After compilation, we need to check if the nef can work as expected.

We would use `ktc.kneron_inference` here again. And we are using the generated nef file this time.

For the batch compile with only LittleNet, the python code would be like:

```
hw_results = ktc.kneron_inference(input_data, nef_file=compile_result, radix=radix)
```

The usage is a little different here. In the code above, `hw_results` is a list of result data. `nef_file` is the path to the input nef. `input_data` is a list of input data after preprocess. The requirement is the same as in section 3.3. `radix` is the radix value of inputs. If your platform is not 520, you may need an extra parameter `platform`, e.g. `platform=720`.

Here we use the same input `input_data` which we used in section 3.3 and the same `radix` in section 4.2. And the `compile_result` is the one that generated with only LittleNet model.

As mentioned above, we do not provide any postprocess. In reality, you may want to have your own postprocess function in Python, too.

For nef file with mutiple models, we can specify the model with model ID:

```
hw_results = ktc.kneron_inference(input_data, nef_file=compile_result, model_id=327
```

After getting the `hw_results` and post-process it, you may want to compare the result with the `fixed_results` which is generated in section 4.2 to see if the results match. If the results mismatch, please contact us direcly through forum https://www.kneron.com/forum/ (https://www.kneron.com/forum/).

5.3 NEF Combine

This section is not part of the normal workflow. But it would be very useful when you already have multiple nef files, some of which might from different versions of the toolchain, and you want to combine them into one. Here we provide a python API to achieve it.

```
ktc.combine_nef(nef_path_list, output_path = "/data1/combined")
```

Here the `nef_path_list` shall be a list of the `str` which are the path to the nef files you want to combine. It should not be empty. And the second argument is optional. It should be the output folder path of the combined nef. By default, it should be `/data1/combined`. The return value is the output folder path. The combined nef file would be under the output folder and be named as `models_<target>.def`. For example, if your target platform is 520, the result file name would be `models_520.nef` inside the output folder.

6 What's Next

- Check `/workspace/examples/LittleNet/python_api_workflow.py`. All the example Python API usages are inside this runnable script.
- Try it with your own models.
- Check the YOLO Example (http://doc.kneron.com/docs/toolchain/yolo_example/) for a step-by-step walk through using YOLOv3 as the example.
- Check the Toolchain Python API (http://doc.kneron.com/docs/toolchain/python_api/) document for more detailed Python API usage.
- Check the Command Line Script Tools (http://doc.kneron.com/docs/toolchain/command_line/) for command line script usage.

- Check the ONNX Converter (http://doc.kneron.com/docs/toolchain/converters/) for the usage of underlying project https://github.com/kneron/ONNX_Convertor (https://github.com/kneron/ONNX_Convertor).

FAQ

1. What if the E2E simulator results from the original model and the optimized onnx mismatch?

Please double check if the final layers are cut due to unsupported by NPU. If so, please add the deleted operator as part of the E2E simulater post process and test again. Otherwise, please search on forum https://www.kneron.com/forum/categories/ai-model-migration (https://www.kneron.com/forum/categories/ai-model-migration). You can also contact us through the forum if no match issue found. The technical support would reply directly to your post.

2. What if the E2E simulator results of floating-point and fixed-point lost too match accuracy?

Please try the following solutions:

1. Try redoing the analysis with more image that are the expected input of the network.
2. Double check if the cut final CPU nodes are added in post-process.
3. Fine tuning the analysis with outlier and quantize mode.

If none of the above works, please search on forum https://www.kneron.com/forum/categories/ai-model-migration (https://www.kneron.com/forum/categories/ai-model-migration). You can also contact us through the forum if no match issue found. The technical support would reply directly to your post.

3. How to adjust the system resources usage of the docker?

To ensure the quantization tool can work, we recommend the docker has at least 4GB of memory. The actual required size depends on your model size and the image number of quantization.

For Linux uses, by default, docker can share all the CPU and memory resouces of their host machine. So, this isn't a problem. But for Windows users, not like Linux, the system resources are not shared. User might want to adjust the resources usage by themselves.

For the docker based on wsl2, as we recommended in the section 1 of this document, it can use update to 50% of your total system memory and all the CPU resources. And here is a artical introduce how to manage the system resources used by wsl2 (https://ryanharrison.co.uk/2021/05/13/wsl2-better-managing-system-resources.html#:~:text=1%20Setting%20a%20WSL2%20Memory%20Limit.%20By%20default,the%20WSL2%20Virtual%20Disk.%20...%204%20Docker.%20).

For the docker based on wsl, users can find the management of the system resouces directly in the setting of the docker.

For the docker toolbox, it is actually based on the VirtualBox virtual machine. So, user need find which virtual machine the docker is using first. User need to start the docker terminal to ensure the docker is running before we start. And here is following precedue

• Open the VirtualBox management tool.

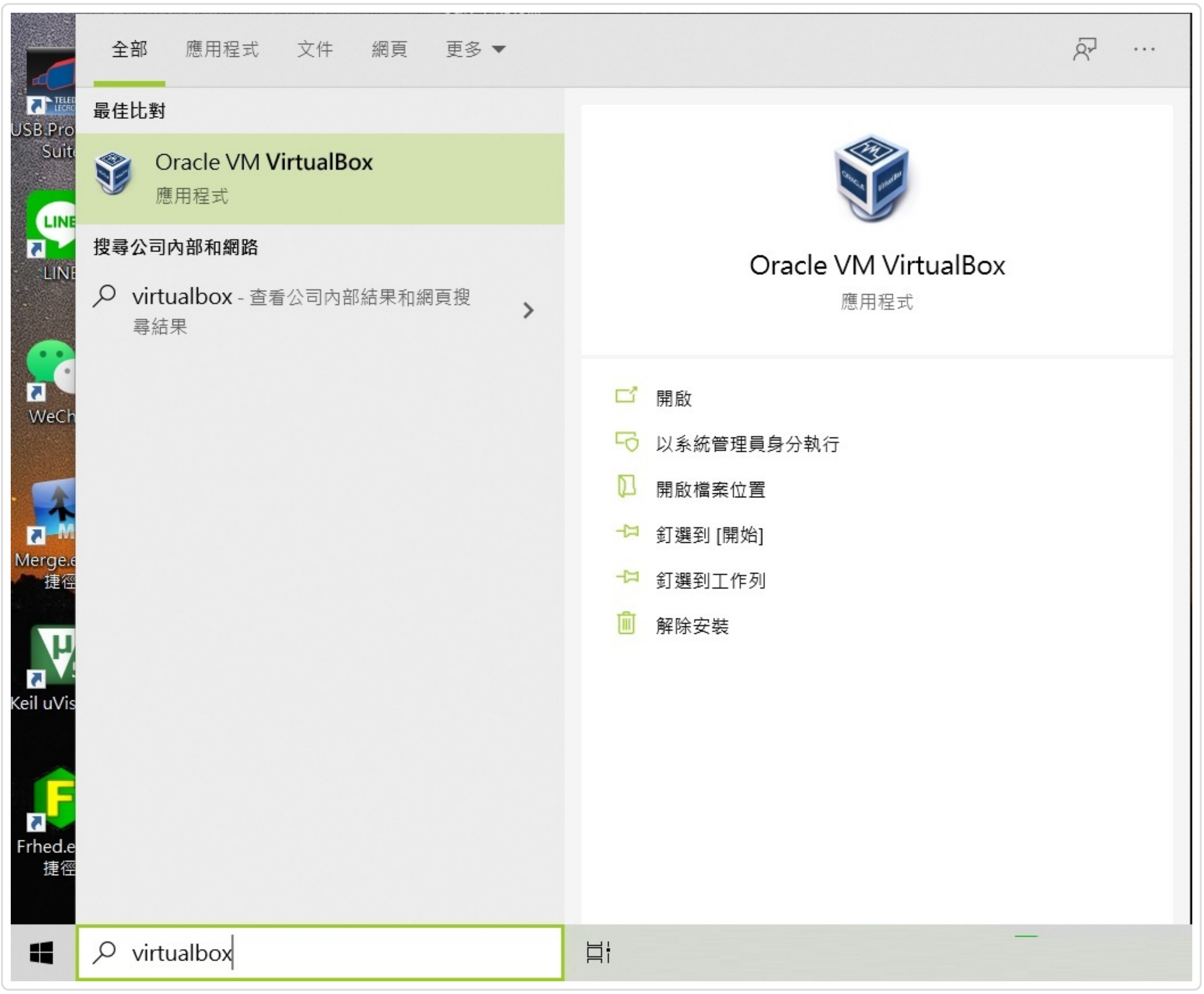
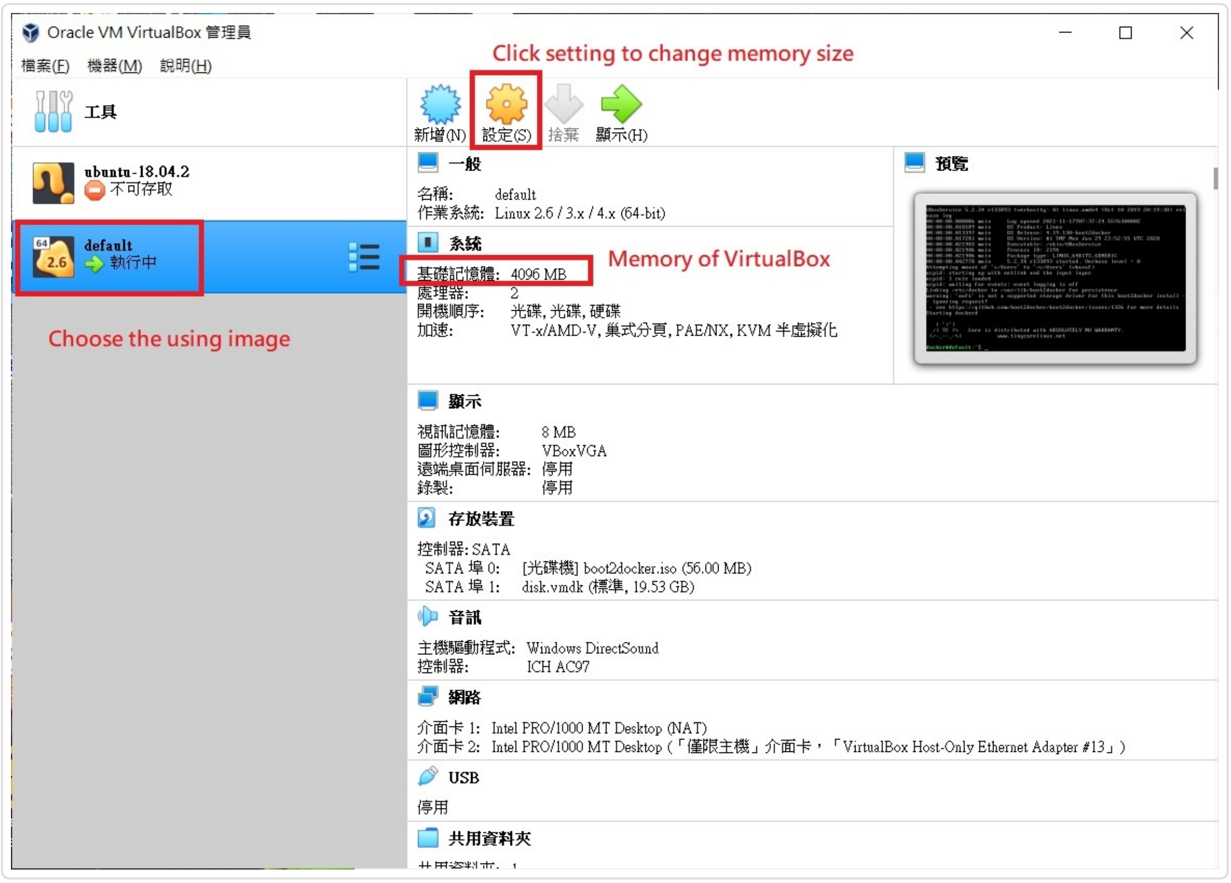


Figure FAQ3.1 VirtualBox

* Check the status. There should be only one virtual machine running if there is no other virtual machines started manually by the user.


Oracle VM VirtualBox 管理員
檔案(F) 機器(M) 說明(H)
Click setting to change memory size
工具
新增(N) 設定(S) 捨棄 顯示(H)
Memory of VirtualBox
Choose the using image
預覽

ubuntu-18.04.2
不可存取

default
執行中

一般

名稱: default
作業系統: Linux 2.6 / 3.x / 4.x (64-bit)

系統

基礎記憶體: 4096 MB
處理器: 2
開機順序: 光碟, 光碟, 硬碟
加速: VT-x/AMD-V, 巢式分頁, PAE/NX, KVM 半虛擬化

顯示

視訊記憶體: 8 MB
圖形控制器: VBoxVGA
遠端桌面伺服器: 停用
錄製: 停用

存放裝置

控制器: SATA
SATA 埠 0: [光碟機] boot2docker.iso (56.00 MB)
SATA 埠 1: disk.vmdk (標準, 19.53 GB)

音訊

主機驅動程式: Windows DirectSound
控制器: ICH AC97

網路

介面卡 1: Intel PRO/1000 MT Desktop (NAT)
介面卡 2: Intel PRO/1000 MT Desktop (「僅限主機」介面卡，「VirtualBox Host-Only Ethernet Adapter #13」)

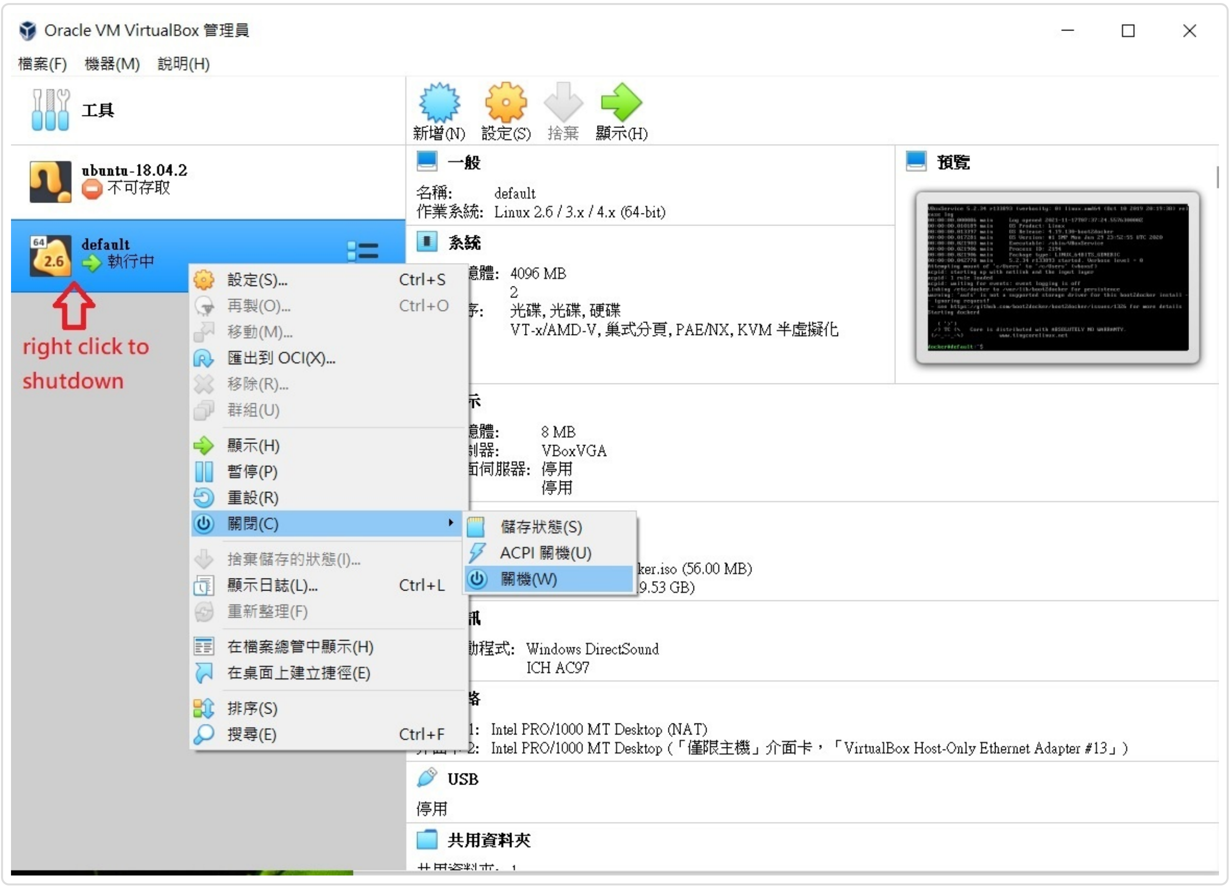
USB

停用

共用資料夾

Figure FAQ3.2 VM status

- Close the docker terminal and shutdown the virtual machine before we adjust the resources usage.



名稱: default
作業系統: Linux 2.6 / 3.x / 4.x (64-bit)

憶體: 4096 MB
2
序: 光碟, 光碟, 硬碟
VT-x/AMD-V, 巢式分頁, PAE/NX, KVM 半虛擬化

憶體: 8 MB
制器: VBoxVGA
面伺服器: 停用
停用

ker.iso (56.00 MB)
9.53 GB)

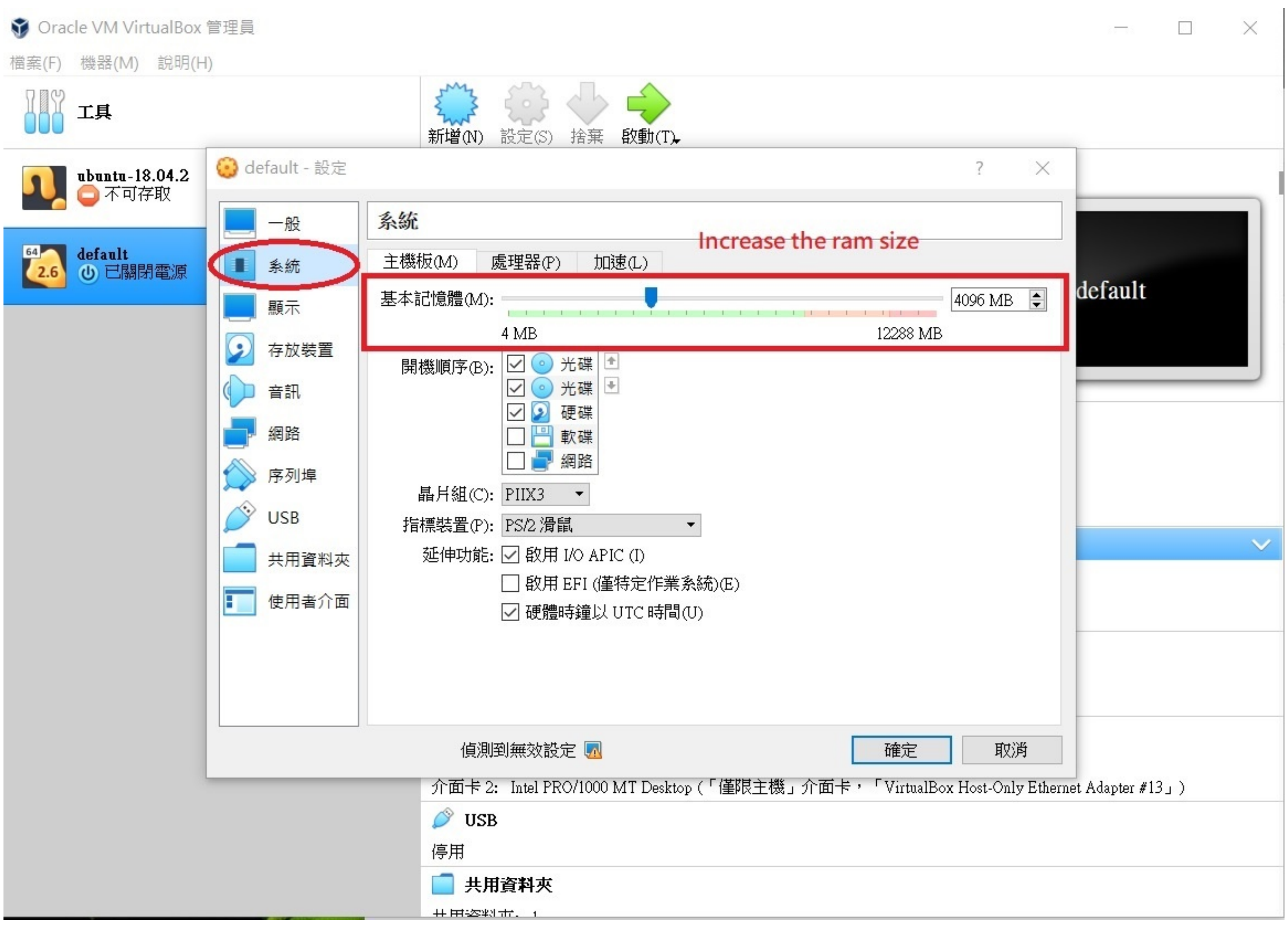
動程式: Windows DirectSound
ICH AC97

1: Intel PRO/1000 MT Desktop (NAT)
2: Intel PRO/1000 MT Desktop (「僅限主機」介面卡, 「VirtualBox Host-Only Ethernet Adapter #13」)

停用

Figure FAQ3.3 VM shutdown

- Adjust the memory usage in the virtual machine settings. You can also change the cpu count here as well.



系統

基本記憶體(M): 4096 MB (4 MB – 12288 MB)

開機順序(B): ☑ 光碟 ☑ 光碟 ☑ 硬碟 ☐ 軟碟 ☐ 網路

晶片組(C): PIIX3

指標裝置(P): PS/2 滑鼠

延伸功能: ☑ 啟用 I/O APIC (I)
☐ 啟用 EFI (僅特定作業系統)(E)
☑ 硬體時鐘以 UTC 時間(U)

介面卡 2: Intel PRO/1000 MT Desktop (「僅限主機」介面卡，「VirtualBox Host-Only Ethernet Adapter #13」)

USB

停用

共用資料夾

Figure FAQ3.4 VM settings

- Save the setting and restart the docker terminal. Now you can use more memory in your docker container.

❮ Previous (../../host_lib_0.8.0/host_api/system/)
System API reference (../../host_lib_0.8.0/host_api/sys…

Next ❯ (../yolo_example/)
Yolo Example (../yolo_example/)