Kneron Linux Toolchain Manual

2021 May Toolchain v0.14.2

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:v0.14.2`.

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 use scripts to utilize the tools.

** Major changes of past versions**

- **[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/)

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 v0.14.2. You can find the version of the toolchain in `/workspace/version.txt` inside the docker. If you find your toolchain is later than v0.14.2, 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
|-- cmake               # Environment
|-- examples            # Example for the workflow, will be used later.
|-- libs                # The libraries
|   |-- ONNX_Convertor  # ONNX Converters and optimizer scripts, will be discussed :
|   |-- 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 behavic
|-- 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 scripts, let me introduce the general work flow.

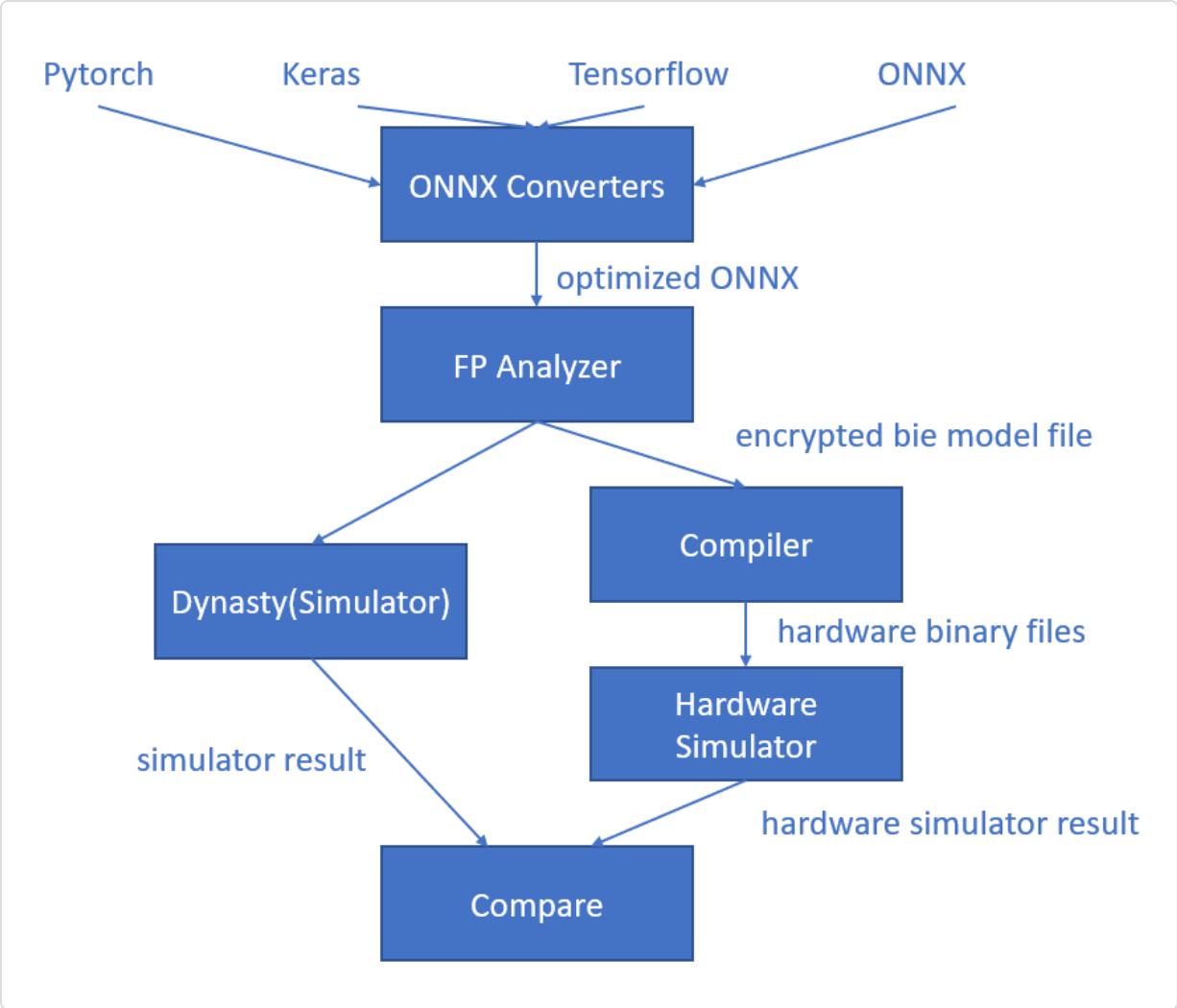

Figure 1. Diagram of working flow

To keep the diagram as clear as possible, some inputs and outputs are omitted. But it is enough to show the general steps:

1. Convert and optimize the models. Generate an optimized onnx file. Details are in section 3.1
2. Use the optimized onnx file and a set of images to do the model analysis. Generate an encrypted bie file.
3. Compile the bie file to generate binaries for hardware simulator. This step also generates the IP evaluation result.
4. Use the binaries generated in step 3 and an image as input to simulate the hardware. Get the results as txt files.
5. Use the bie file and the same image in step 4 as input to simulate the calculation. Get the result as txt files.
6. Compare the txt files generated in step 4 and step 5 to make sure this model can be taken by the kdp720 hardware.

Step 1 is in section 3.1. Step 2-6 are automated in section 3.2 using scripts.

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]

Node	Applicable Subset	Spec.
GlobalAveragePool	4D input	support
GlobalMaxPool		support
MaxRoiPool		support
Slice	input dimension <= 4	support

3. Toolchain Scripts Usage

In this section, we'll go through how to run the tools using scripts.

For section 3.1, both KDP520 and KDP 720 share the same scripts. But after section 3.2 (including 3.2), KDP520 and KDP720 are using different scripts. **Please run the commands corresponding to your hardware version.**

3.1 Converters

Here we will introduce scripts which help us easily convert the models from different platforms to ONNX. Note that the script introduced here is a wrapper of `ONNX_Convertor` for ease of usage. If you want to gain more control of the conversion, please check our document `ONNX Converter` from the document center.

You can also find the mapping table for each platfrom under our document `ONNX Converter` from the document center.

The example mentioned in the following document may not be in the docker image to reduce the image size. You can download them yourself from https://github.com/kneron/ConvertorExamples (https://github.com/kneron/ConvertorExamples)

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 3.1.5 for TF Lite model conversion.

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

```
python /workspace/scripts/convert_model.py keras input_model_path output_model_path
```

For example, if the model is `/docker_mount/onet.hdf5`, and you want to convert it to `/docker_mount/onet.onnx`, the detailed command is:

```
python /workspace/scripts/convert_model.py keras /docker_mount/onet.hdf5 /docker_mou
```

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 checking whether the onnx file is generated.

3.1.2 Pytorch to ONNX

Our scripts only takes Pytorch exported `onnx` as the input. Pleace checkout the tips below on how to export an onnx with Pytorch.

We currently only support models exported by Pytorch version >=1.0.0, <1.6.0.

> TIPS
>
> You can use `torch.onnx` to export your model into onnx format. Here is the Pytorch 1.3.0 version document (https://pytorch.org/docs/1.3.1/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', keep_initializers_as_inputs=Tru
```

> In the example, `(1, 3, 224, 224)` are batch size, input channel, input height and input width. `model` is the model object you want to export. `output.onnx` is the output file. For Pytorch version before 1.3.0, `keep_initializers_as_inputs=True` is not needed. Otherwise, it is required.

Suppose the input file is called `/docker_mount/resnet34.onnx` and you want to save the optimized onnx as `/docker_mount/resnet34.opt.onnx`. Here is the example command:

```
python /workspace/scripts/convert_model.py pytorch /docker_mount/resnet34.onnx /docl
```

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 using `--no-bn-fusion` flag.

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).

Suppose you have model structure definition file `/docker_mount/mobilenetv2.prototxt` and model weight file `/docker_mount/mobilenetv2.caffemodel` and you want to output the result as `/docker_mount/mobilenetv2.onnx`, Here is the example command:

```
python /workspace/scripts/convert_model.py caffe /docker_mount/mobilenetv2.prototxt
```

Here `-w` with the `caffemodel` weight is required.

3.1.4 Tensorflow to ONNX

Tensorflow to onnx script only support Tensorflow 1.x and the operator support is very limited. If it cannot work on your model, please try to export the model as tflite and convert it using section 3.1.5.

Suppose you want to convert the model `/docker_mount/mnist.pb` to `/docker_mount/mnist.onnx`, here is the example command:

```
python /workspace/scripts/convert_model.py tf /docker_mount/mnist.pb /docker_mount/m
```

3.1.5 TF Lite to ONNX

```
python /workspace/scripts/convert_model.py tflite path_of_input_tflite_model path_of
```

For the provided example model: `model_unquant.tflite`

```
python /workspace/scripts/convert_model.py tflite /data1/tflite/model/model_unquant
```

3.1.6 ONNX to ONNX (ONNX optimization)

After converting models from other frameworks to onnx format which are not mentioned above, or you download the onnx models directly from an online source, you need to run the following command to optimize the model for Kneron hardware. Suppose your model is `input.onnx` and the output model is called `output.onnx`, the command would be:

```
python /workspace/scripts/convert_model.py onnx input.onnx output.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 using `--no-bn-fusion` flag.

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 3.1.8 to update your model first.

3.1.7 Model 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 a model editor which is `editor.py` to help user modify the model so that KL520 NPU can run the model more efficiently.

You can find the detailed description of this tool in the `ONNX Converter` document on our kneron document center website.

3.1.8 Model Updater

From toolchain version 0.14.0, ONNX in the docker has been updated to 1.6. And the opset that converters support is changed from opset 9 into opset 11. Thus, if you have a onnx model with opset 9, you may need to update it with the following command before any other actions:

```
python /workspace/libs/ONNX_Convertor/optimizer_scripts/onnx1_4to1_6.py input.onnx o
```

3.2 FpAnalyser, Compiler and IpEvaluator

In the following part of this section, we'll use the `LittleNet` as an example. You can find the folder under `/workspace/examples`. To use it copy it under `/data1`.

```
cp -r /workspace/examples/LittleNet /data1
cp /data1/LittleNet/input_params.json /data1
```

3.2.1 Prepare the input

Before running the programs, you need to prepare the inputs. In our toolchain, all the outputs are placed under `/data1`. We call it the Interactivate Folder. So, we recommend you create this folder if it is not already there. Then we need to configure the input parameters using `input_params.json` for the toolchain under `/data1`. As an example, `input_params.json` for `LittleNet` model is under `/data1/LittleNet`, if you have already copy the folder as described in the beginning of section 3.2. You already has the `input_params.json` ready. You can see the detailed explanation for the input parameters in section FAQ question 1.

> The `input_params.json` is different in 0.10.0. Please check the FAQ for new config fields or use section 3.7.3 to upgrade your existed `input_params.json`.

3.2.2 Running the program

After preparing `input_params.json`, you can run the programs by the following command:

```
# For KDP520
# python /workspace/scripts/fpAnalyserCompilerIpevaluator_520.py -t thread_number
python /workspace/scripts/fpAnalyserCompilerIpevaluator_520.py -t 8

# For KDP720
# python /workspace/scripts/fpAnalyserCompilerIpevaluator_720.py -t thread_number
python /workspace/scripts/fpAnalyserCompilerIpevaluator_720.py -t 8
```

`thread_number`: the number of thread to use.

3.2.3 Get the result

After running this program, the folders called `compiler` and `fpAnalyser` will be generated under `/data1`. `compiler` stores the result of compiler and ipEvaluator. `fpAnalyser` stores the result of the model analyzer.

We can find the following files after running the script above:

- `/data1/fpAnalyser/<model_name>.quan.wqbi.bie`: the encrypted result of the model analyzer. It includes the quantize information and the model itself. Can be taken by the compiler and the simulator.
- `/data1/compiler/command.bin`: the compiled binary for the hardware.
- `/data1/compiler/setup.bin`: the compiled binary for the hardware.
- `/data1/compiler/weight.bin`: the compiled binary for the hardware.

- `/data1/compiler/ioinfo.csv`: the hardware IO mapping information of the model.
- `/data1/compiler/ip_eval_prof.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 520 version script generate this file.)*
- `/data1/compiler/ProfileResult.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 720 version script generate this file.)*

3.2.4 Hardware validation (optional)

This step will make sure whether the mathematical simulator's result is the same as the hardware simulator's. In other words, this step makes sure the model can run correctly on the hardware.

```
# For KDP520
python /workspace/scripts/hardware_validate_520.py

# For KDP720
python /workspace/scripts/hardware_validate_720.py
```

If it succeeds, you can the command line print out log: `[info] hardware validating successes!`. Otherwise, you might need to check your model to see if it has passed the converter. If the problem persist, please ask our stuff for help and provide the simulator result and the hardware simulator result under `/data1/simulator` and `/data1/c_sim`.

3.3 Simulator and Emulator (Deprecated)

The simulator and the emulator are no longer available for running seperately with the script. Please use end to end simulator instead.

3.4 Compiler and Evaluator

The function of this part is similar with part 3.2, and the difference is that this part does not run FP-analyser, it can be used when your model structure is prepared but hasn't been trained, or when you only want to check your model's performance estimation.

3.4.1 Running the programs

To run the compile and the ip evaluator, you need to have an onnx or a bie file as the input. Here we take the `LittleNet` as the example:

```
# For KDP520
cd /workspace/scripts && ./compilerIpevaluator_520.sh /data1/LittleNet/LittleNet.onn

# For KDP720
cd /workspace/scripts && ./compilerIpevaluator_720.sh /data1/LittleNet/LittleNet.onn
```

This part uses the default configuration for the hardware and not available for fine-tuning.

3.4.2 Get the result

A folder called `compiler` will be generated in `/data1`, which stores the result of the compiler and IP Evaluator. If you are using this script with an `onnx` file, please ignore the binary files since they are not quantized and unable to be used.

We can find the following files after running the script above:

- `/data1/compiler/command.bin`: the compiled binary for the hardware.
- `/data1/compiler/setup.bin`: the compiled binary for the hardware.
- `/data1/compiler/weight.bin`: the compiled binary for the hardware.
- `/data1/compiler/ioinfo.csv`: the hardware IO mapping information of the model.
- `/data1/compiler/ip_eval_prof.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 520 version script generate this file.)*
- `/data1/compiler/ProfileResult.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 720 version script generate this file.)*

3.5 Batch-Compile

This part is the instructions for batch-compile, which will generate the binary file requested by firmware with the `bie` files given.

3.5.1 Fill the input parameters

Fill the input parameters in the `/data1/batch_input_params.json` under `/data1`. Please refer on the FAQ question 6 to fill the related parameters.

For the LittleNet example, if you already follow the instructions in the section 3.2. You should already have `LittleNet.quan.wqbi.bie` under `/data1/fpAnalyser/`

Please make sure you are generated the bie file with the correct hardware version. For example, if you want to batch-compile for the 520 hardware, you need to generate the bie with the 520 FP-analyzer

Here is the config `/data1/batch_input_params.json` we need to batch compile the LittleNet:

```
{
    "models": [
        {
            "id": 19,
            "version": "1",
            "path": "/data1/fpAnalyser/LittleNet.quan.wqbi.bie"
        }
    ]
}
```

We can also do batch compile with multiple models and with encryption. Please check the FAQ 6 for more details about the config fields.

3.5.2 Running the programs

For running the compiler and ip evaluator:

```
# For KDP520
# python /workspace/scripts/batchCompile_520.py
python /workspace/scripts/batchCompile_520.py

# For KDP720
# python /workspace/scripts/batchCompile_720.py
python /workspace/scripts/batchCompile_720.py
```

3.5.3 Get the result

Under `/data1`, you'll find a folder called `batch_compile`, which contains the output files of batch compile. There so many outputs but not all of them are useful for the user. Here are the main outputs you may need:

- `/data1/batch_compile/<model_name>.command.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.setup.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.weight.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.ioinfo.csv`: the hardware IO mapping information of the model. You might find one for every model you list in the json.
- `/data1/batch_compile/models_520.nef`: the compiled binary with all the models. This file is designed to be loaded by the firmware. *(Only the 520 version script generate this file.)*
- `/data1/batch_compile/models_720.nef`: the compiled binary with all the models. This file is designed to be loaded by the firmware. *(Only the 720 version script generate this file.)*

> **Difference between `*.bin` and `models_*.nef`**
>
> `weight.bin`, `setup.bin` and `weight.bin` are for hardware simulator. They only contains information for a single model. They are mostly for debugging and testing usages.
>
> `model_*.nef` is for the firmware to load onto the chip. It could contain information for multiple models. It's mainly used for firmware testing and deployment.

3.6 FpAnalyser and Batch-Compile (Optional)

This part is the instructions for FP-analysis and batch-compile. The script introduced in this part is actually an combination of section 3.2 and section 3.5.

We recommend doing FP-analysis (section 3.2) and batch-compile (section 3.5) separately. The fP-Analysis is very time-consuming. Thus, saving the `bie` file of each model for the future usage is more convinient.

Back to this section, again, we'll use the `LittleNet` as an example. Just like in the section 3.5, we need `batch_input_params.json`. But this time, **the fields we need to prepare are different**. Please check the next section.

3.6.1 Fill the input parameters

As said in section 3.5, the details of the config can be found in the FAQ question 6. You can use the following code to create the `batch_input_params.json` for a walk through.

```
{
    "encryption": {
        "whether_encryption": false,
        "encryption mode": 1,
        "encryption_key": "0x12345678",
        "key_file": "",
        "encryption_efuse_key": "0x12345678"
    },
    "models": [
        {
            "id": 19,
            "version": "1",
            "path": "/data1/LittleNet/LittleNet.onnx",
            "input_params": "/data1/LittleNet/input_params.json"
        }
    ]
}
```

We can take a look on what are the differences. Ignore the encryption section which is set to false. The real differences here are that we are giving `onnx` instead of `bie` in the model's `path`, and giving an extra `input_params.json` in a field called `input_params`. This `input_params.json` is the one that we use in section 3.2. With the onnx and the json, it could run the FP-analysis just like what we do in secion 3.2, uses the generated `bie` to do batch-compile right after all the FP-analysis is finished.

3.6.2 Running the programs

For running the FP-analysis and batch-compiler:

```
# For KDP520
# python /workspace/scripts/fpAnalyserBatchCompile_520.py -t thread_number
python /workspace/scripts/fpAnalyserBatchCompile_520.py -t 8

# For KDP720
# python /workspace/scripts/fpAnalyserBatchCompile_720.py -t thread_number
python /workspace/scripts/fpAnalyserBatchCompile_720.py -t 8
```

`thread_number`: the number of thread to use

3.6.3 Get the result

Under `/data1`, you'll find a folder called `batch_compile`, which contains the output files of batch compile and FP-analysis. There so many outputs but not all of them are useful for the user. Here are the main outputs you may need:

- `/data1/batch_compile/<model_name>.quan.wqbi.bie`: the encrypted result of the model analyzer. It includes the quantize information and the model itself. Can be taken by the compiler and the simulator.

- `/data1/batch_compile/<model_name>.command.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.setup.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.weight.bin`: the compiled binary for the hardware of a single model. You might find one for every model you list in the json.
- `/data1/batch_compile/<model_name>.ioinfo.csv`: the hardware IO mapping information of the model. You might find one for every model you list in the json.
- `/data1/batch_compile/models_520.nef`: the compiled binary with all the models. This file is designed to be loaded by the firmware. *(Only the 520 version script generate this file.)*
- `/data1/batch_compile/models_720.nef`: the compiled binary with all the models. This file is designed to be loaded by the firmware. *(Only the 720 version script generate this file.)*

3.7 Other utilities

3.7.1 Convert bin file to png

```
cd /workspace/scripts/utils && python bintoPng.py -i input_rgb565_file_path –o outpu
```

3.7.2 Post process

```
cd /workspace/scripts/utils && python post_process.py -i emulator_result_folder -m r
```

3.7.3 Upgrade `input_params.json`

If you have the configuration file from toolchain v0.9.0, you can use the following script to convert the old `input_params.json` into a new one.

```
python /workspace/scripts/upgrade_input_params.py old_input_params.json new_input_pa
```

3.7.4 Draw Yolo result on images (Deprecated)

This function has been moved to end to end simulator.

3.8 E2ESimulator workflow

E2ESimulator workflow is implemented in C, which will get the exactly same result as the hardware platforms.

The detailed manual of E2ESimulator can be found at http://doc.kneron.com/docs/#toolchain/python_app/app_flow_manual/ (http://doc.kneron.com/docs/#toolchain/python_app/app_flow_manual/).

4. Python API

We provide a Python package in the docker toolchain. The package name is `ktc`. You can simply start by having `import ktc` in your Python script. We hope this python API could help you simplify the workflow. You may find the detailed description in the Python API document (http://doc.kneron.com/docs/#toolchain/python_api/). You can also find the usage using the python `help()` function.

There is also an simple example called `/workspace/examples/test_python_api.py` in the docker. This might helps you understand the Python API usage.

Note that this package is only available in the docker due to the dependency issue.

FAQ

1. How to configure the `input_params.json`?

By following the above instructions, the `input_params.json` will be saved in `/data1`. Please do not change the parameters' names.

Here is an example JSON with comments. **Please remove all the comments in the real configuration file.**

```
{
    // The basic information of the model needed by the toolchain.
    "model_info": {
        // The input model file. If you are not sure about the input names, you can
        // check it through model visualize tool like [netron](https://netron.app/)
        // If the configuration is referred by `batch_input_params.json`, this field
        // will be ignored.
        "input_onnx_file": "/data1/yolov5s_e.onnx",
        // A list of the model inputs name with the absolute path of their correspon
        // inputs image folders. It is required by FP-analysis.
        "model_inputs": [{
            "model_input_name": "data_out_0" ,
            "input_image_folder": "/data1/100_image/yolov5",
        }],
        // Special mode for fp-analysis. Currently available mode:
        // - default: for most of the models.
        // - post_sigmoid: recommand for yolo models.
        // If this option is not present, it uses the 'default' mode.
        "quantize_mode": "default",
        // For fp-analysis, remove outliers when calculating max & min. It should be
        // If not given, default value is 0.999.
        "outlier": 0.999
    },
    // The preprocess method of the input images.
    "preprocess": {
        // The image preprocess methods.
        // Options: `kneron`, `tensorflow`, `yolo`, `caffe`, `pytorch`
        // `kneron`: RGB/256 - 0.5,
        // `tensorflow`: RGB/127.5 - 1.0,
        // `yolo`: RGB/255.0
        // `pytorch`: (RGB/255. -[0.485, 0.456, 0.406]) / [0.229, 0.224, 0.225]
        // `caffe`(BGR format) BGR  - [103.939, 116.779, 123.68]
        // `customized”`: please refer to FAQ question 8
        "img_preprocess_method": "kneron",
        // The channel information after the input image is preprocessed.
        // Options: RGB, BGR, L
        // L means single channel.
        "img_channel": "RGB",
        // The radix information for the npu image process.
        // The formula for radix is 7 – ceil(log2 (abs_max)).
        // For example, if the image processing method we utilize is "kneron",
        // the related image processing formula is "kneron": RGB/256 - 0.5,
        // and the processed value range will be (-0.5, 0.5).
        // abs_max = max(abs(-0.5), abs(0.5)) = 0.5
        // radix = 7 – ceil(log2(abs_max)) = 7 - (-1) = 8
        "radix": 8,
        // [optional]
        // Indicates whether or not to keep the aspect ratio. Default is true.
        "keep_aspect_ratio": true,
        // [optioanl]
        // This is the option for the mode of adding paddings, and it will be utili
```

```
        // when `keep_aspect_ratio` is true. Default is 1.
        // Options: 0, 1
        // 0 — If the original width is too small, the padding will be added at both
        //     and left sides equally; if the original height is too small, the pad
        //     will be added at both up and down sides equally.
        // 1 — If the original width is too small, the padding will be added at the
        //     side only, if the original height is too small, the padding will be
        //     added at the down side.
        "pad_mode": 1,
        // [optional]
        // The parameters for cropping image. And it has four sub parameters. Defau
        // -crop_x, cropy, the left cropping point coordinate.
        // -crop_y, cropy, the up cropping point coordinate.
        // -crop_h, the width of the cropped image.
        // -crop_w, the height of the cropped image.
        "p_crop": {
            "crop_x": 0,
            "crop_y": 0,
            "crop_w": 0,
            "crop_h": 0
        }
    },
    // [optional]
    // A list of the model inputs name with the absolute path of their correspondin
    // input images. It is used by the hardware validator. If this field is not give
    // a random image for each input will be picked up from the `input_image_folder`
    // in the `model_info`.
    "simulator_img_files": [{
            "model_input_name": "data_out" ,
            "input_image": "/data1/100_image/yolov5/a.jpg",
    }]
}
```

2. Fails when implement models with SSD structure.

Currently, our NPU does not support SSD like network since it has Reshape and Concat operations in the end of the model, but we do offer a work around solution to this situation.

The reason we do not support Reshape and Concat operation is that we do offer Reshape operation capability, However, we only support regular shape transportation. Which means you could flatten your data or extract some channels form the feature map. However, NPU does not expect complex transportation. For example, in Figure 13, you could notice there is a 1x12x4x5 feature map reshapes to 1x40x6.

For Concat operation, the NPU also supports channel-based feature map concatenation. However, it does not support Concate operation based on another axis. For example, in Figure 14, The concatenation on based on axis 1 and the following concatenation is based on axis 2.

The workaround we will offer is that deleting these Reshape and Concat operations and enable make the model to a multiple outputs model since they are in the end of the models and they do not change the output feature map data. So converted model should look like this as in Figure 15.

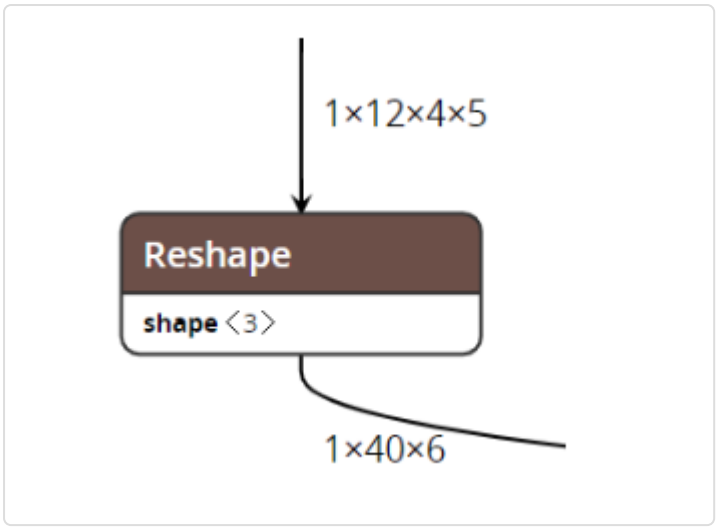

Figure 13. Invalid Reshape operation

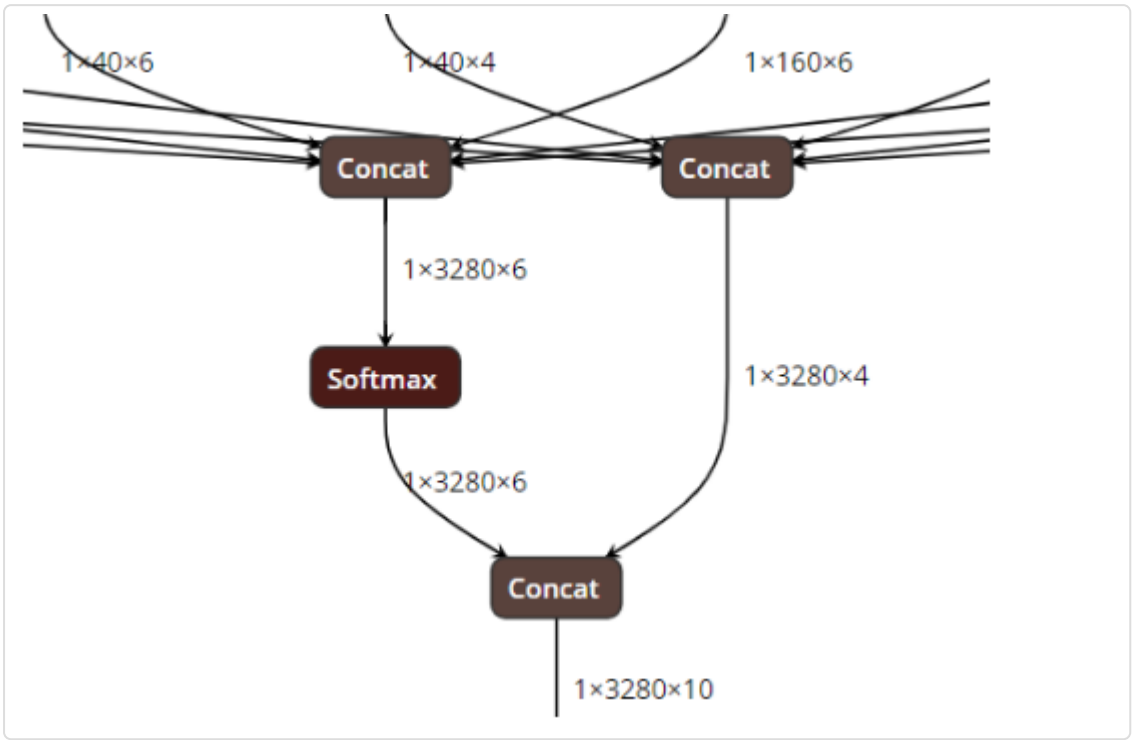

Figure 14. Invalid Concat operation

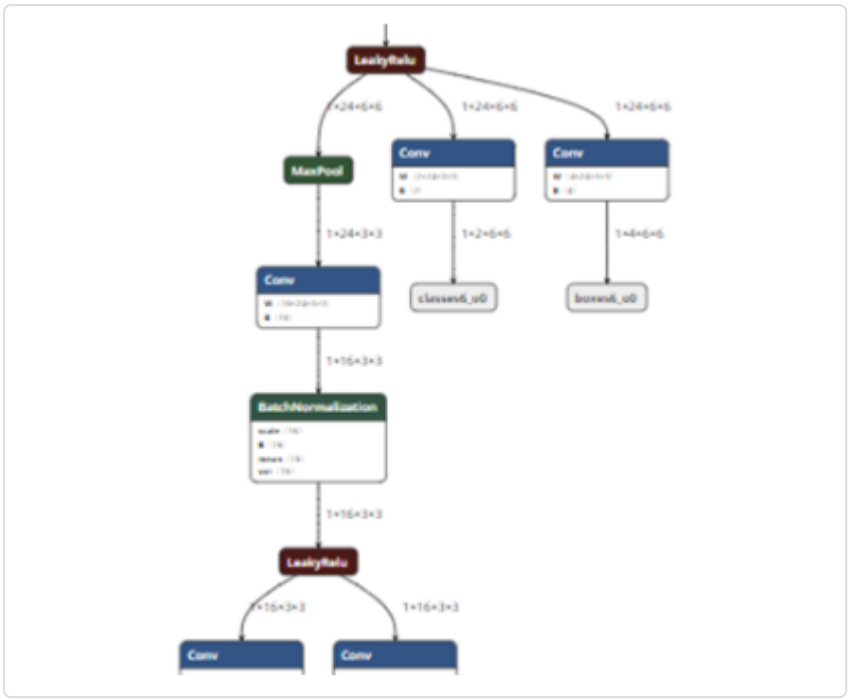

Figure 15. Valid SSD operation

3. Fails in the step of FpAnalyser

When it shows the log `mystery`, it means there are some customized layers in the model you input, which are not support now.

When it shows the log `start datapath analysis`, you need to check whether you input the proper image pre-process parameters.

4. Other unsupported models

This version of SDK doesn't support the models in the following situations:

1. Have customized layers.

5. What's the meaning of IP Evaluator's output?

- estimate FPS float => average Frame Per Second
- total time => total time duration for single image inference on NPU
- MAC idle time => time duration when NPU MAC engine is waiting for weight loading or data loading
- MAC running time => time duration when NPU MAC engine is running
- average DRAM bandwidth => average DRAM bandwidth used by NPU to complete inference
- total theoretical convolution time => theoretically minimum total run time of the model when MAC efficiency is 100%
- MAC efficiency to total time => time ratio of the theoretical convolution time to the total time

6. How to configure the `batch_input_params.json`?

By following the above instructions, the `batch_input_params.json` will be saved under `/data1`. Please do not change the parameters' names.

Here is an example JSON with comments. **Please remove all the comments in the real configuration file.**

```
{
    // [optional]
    // The encryption setting for the batch compiler. Default is not enabled.
    "encryption": {
        // Whether enable encrytion
        "whether_encryption": false,
        // Encrytion mode selection
        // Options: 1, 2
        "encryption mode": 1,
        // Encrytion key. A hex string. Required in mode 1.
        "encryption_key": "0x12345678",
        // Encrytion file. An absolute path. Required in mode 1.
        "key_file": "/data1/enc.txt",
        // Encrytion key. A hex string. Required in mode 2, optional in mode 1.
        "encryption_efuse_key": "0x12345678"
    },
    // [optional]
    // Whether seperate buffers for each model output. Default is true.
    "dedicated_output_buffer": true,
    // Batch compile model list
    "models": [
        {
            // Model ID
            "id": 19,
            // Model version. should be an hex code at most 4 digit.
            "version": "1",
            // The path to the model.
            // If you are running fpAnalyserBatchCompile, this field should be an or
            // If you are running batchCompile, this field could be an onnx or a bie
            // If onnx is provided in the second case, you also need to provide the
            // `radix_json`.
            "path": "models/output_0.onnx",
            // [optional]
            // Only needed when you are running batchCompile_*.py with an onnx.
            "radix_json": ".json",
            // Required when you are running fpAnalyserBatchCompile. This configurat
            // file is used to provide information for FP-analysis. Please check FAQ
            // for details.
            "input_params": "/data1/input_params.json"
        }
    ]
}
```

7. What's the meaning of the output files?

- `command.bin`: the compiled binary for the hardware.
- `setup.bin`: the compiled binary for the hardware.
- `weight.bin`: the compiled binary for the hardware.
- `ioinfo.csv`: the hardware IO mapping information of the model.
- `ip_eval_prof.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 520 version script generate this file.)*
- `ProfileResult.txt`: the IP evaluation report, which is the estimate performance of your model on the NPU. *(Only the 720 version script generate this file.)*
- `<model_name>.quan.wqbi.bie`: the encrypted result of the model analyzer. It includes the quantize information and the model itself. Can be taken by the compiler and the simulator.
- `models_520/720.nef`: the compiled binary with all the models. This file is designed to be loaded by the firmware.

If you find the cpu node in `ioinfo.csv`, whose format is `c,\**,**` "`, you need to implement and register this function in SDK.

8. How to use customized methods for image preprocess?

1. Configure the `input_params.json`, and fill the value of `img_preprocess_method` as `customized`;
2. edit the file `/workspace/scripts/utils/img_preprocess.py`, search for the text `#this is the customized part` and add your customized image preprocess method there.

9. Can I install packages in the toolchain?

Of course you can.

To install ubuntu packages, for example `vim`, you can use the following command `apt-get update && apt-get install vim`.

To install python packages, for example `tornado`, you can use either `conda install -c tornado` or `pip install tornado`. **Not that `pip3` links the system python instead of the conda python. Do not use it.**

But the environment will be reset each time you create a container from the image. You may need to remove `--rm` flag in the `docker run` command and learn how to start an existed docker container from the docker document (https://docs.docker.com/engine/reference/commandline/start/).

❮ Previous (../../plus/api_reference_1.0.x/kp_inference.h/)
Inference API (../../plus/api_reference_1.0.x/kp_infe…

Next ❯ (../python_app/app_flow_manual/)
End to End Simulator (../python_app/app_flow_man…