Previous (../../host_lib_0.9.0/kdp2_api/kdp2_postprocess.h/)
Post Process API (../../host_lib_0.9.0/kdp2_api/kdp…

Next (../manual_720/)
Toolchain 720 Manual (../manual_720/)

Kneron Linux Toolchain 520 Manual

2020 December Toolchain 520 v0.11.0

PDF Downloads (manual_520.pdf)

0. Overview

KDP toolchain is a set of software which provide inputs and simulate the operation in the hardware KDP 520. 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:520_v0.11.0`.

Performance simulation result on NPU:

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

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.

Changes compared to old versions:

1. Now, 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.
2. `input_params.json` and `batch_input_params.json` have been changed a lot. Please check the document for details.
3. `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 for details.

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 for 520.

```
docker pull kneron/toolchain:520
```

Note that this document is compatible with toolchain v0.11.0. You can find the version of the toolchain in `/workspace/version.txt` inside the docker. If you find your toolchain is later than v0.11.0, 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:520` 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:520
```

> 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 kdp520 hardware and the IP evaluator to infe
|   |-- dynasty         # Simulator which only simulates the kdp520 calculation.
|   |-- fpAnalyser      # Analyze the model and provide fixed point information.
|   `-- hw_c_sim        # Hardware simulator which simulate all the kdp520 hardware
|-- 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.

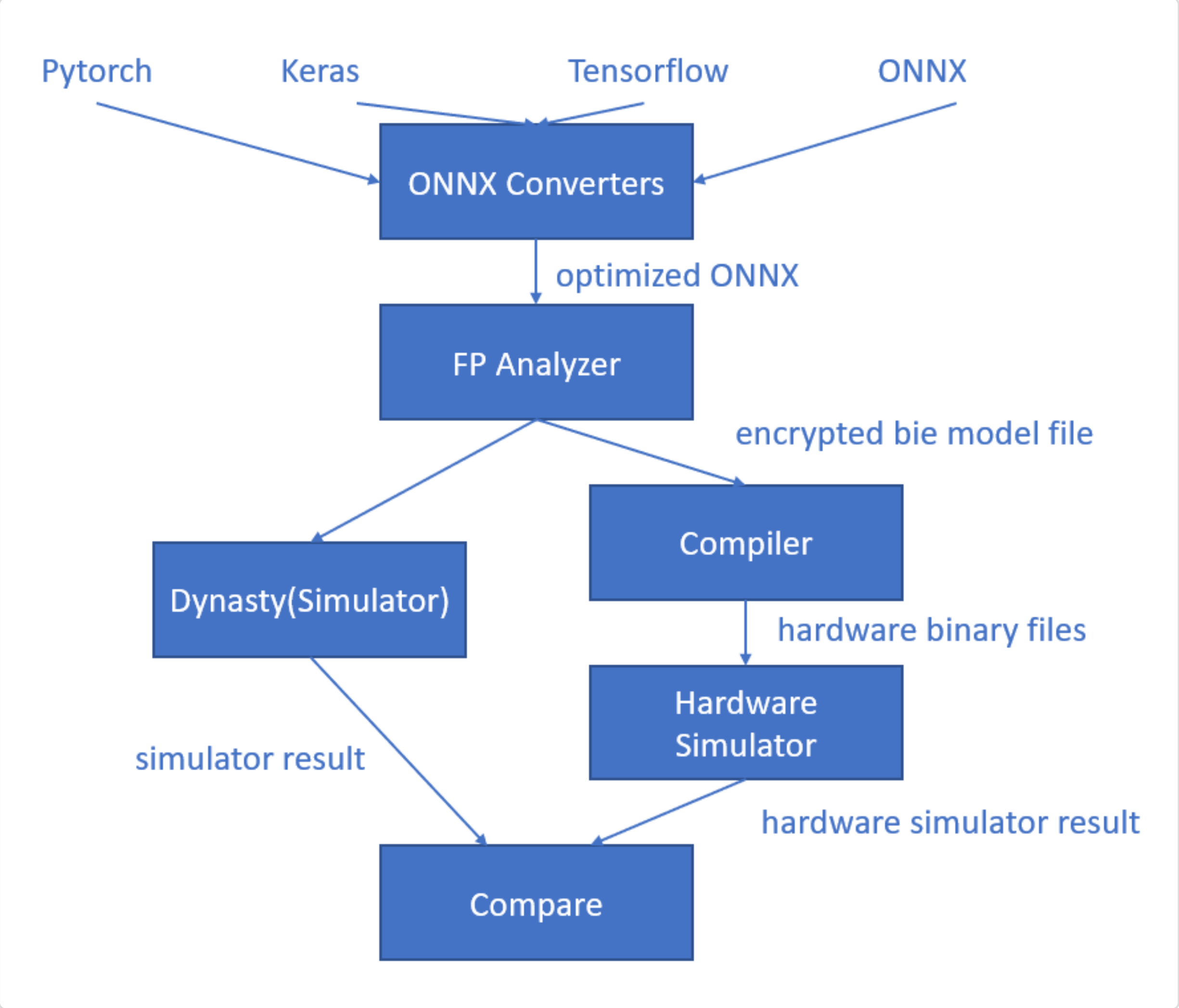

Figure 1. Diagram of working flow

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 kdp520. 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 kdp520 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 1.4.1.

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
	Gemm or Dense/Fully Connected		support
	Flatten		support

3. Toolchain Scripts Usage

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

3.1 Converters

ONNX_Convertor is an open-source project on Github (https://github.com/kneron/ONNX_Convertor). If there is any bugs in the ONNX_Convertor project inside the docker, don't hesitate to try `git pull` under the project folder to get the latest update. And if the problem persists, you can raise an issue there. We also welcome contributions to the project.

The general process for model conversion is as following:

1. Convert the model from other platforms to onnx using the specific converters. See section 3.1.1 - 3.1.5.
2. Optimize the onnx for Kneron toolchain using `onnx2onnx.py`. See section 3.1.6.
3. Check the model and do further customization using `editor.py` (optional). See section 3.1.7

> TIPS:
>
> ONNX exported by Pytorch **cannot** skip step 1 and directly go into step 2. Please check section 3.1.2 for details.

If you're still confused reading the manual, please try our examples from https://github.com/kneron/ConvertorExamples (https://github.com/kneron/ConvertorExamples)

Table 1.2 shows supported operators conversion mapping table for every platform.

Table 1.2 Operators conversion table for every platform

type	keras	caffe	tflite	tensorflow(*)	pytorc
add	Add	Eltwise	ADD	(*)	(**)
average pooling	AveragePooling2D	Pooling	AVERAGE_POOL_2D	(*)	(**)
batchnormalization	BatchNormalization	BatchNorm		(*)	(**)
concatenate	Concatenate	Concat	CONCATENATION	(*)	(**)
convolution	Conv2D	Convolution	CONV_2D	(*)	(**)
deconvolution	Conv2DTranspose	Deconvolution	TRANSPOSE_CONV	(*)	(**)
dense	Dense	InnerProduct	FULLY_CONNECTED	(*)	(**)
depthwise convolution	DepthwiseConv2D	DepthwiseConvolution	DEPTHWISE_CONV_2D	(*)	(**)
flatten	Flatten	Flatten		(*)	(**)
global average pooling	GlobalAveragePooling2D	Pooling	MEAN	(*)	(**)
global max pooling	GlobalMaxPooling2D	Pooling		(*)	(**)
leaky relu	LeakyReLU		LEAKY_RELU	(*)	(**)
max pooling	MaxPooling2D	Pooling	MAX_POOL_2D	(*)	(**)

type	keras	caffe	tflite	tensorflow(*)	pytorc
multiply	Multiply	Eltwise	MUL	(*)	(**)
padding	ZeroPadding2D		PAD	(*)	(**)
prelu	PReLU	PReLU	PRELU	(*)	(**)
relu	ReLU	ReLU	RELU	(*)	(**)
relu6	ReLU		RELU6	(*)	(**)
separable conv2d	SeparableConv2D			(*)	(**)
squeeze			SQUEEZE	(*)	(**)

(*) our tensorflow conversion tool is based on opensource "tf2onnx", please check "https://github.com/onnx/tensorflow-onnx/blob/r1.6/support_status.md" for the supported op information
(**) our pytorch conversion tool is based on onnx exporting api in torch.onnx, please check "https://pytorch.org/docs/stable/onnx.html#supported-operators" for the supported op information**
(*) some operators will be replaced by onnx2onnx.py in order to fit the kl520 spec.

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/libs/ONNX_Convertor/keras-onnx/generate_onnx.py -o absolute_path_o
```

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/libs/ONNX_Convertor/keras-onnx/generate_onnx.py -o /docker_mount/o
```

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.

You need to run the command in section 3.1.6 after this step.

Input shape change*(optional)*

If there’s customized input shape for the model file, you need to use `-I` flag in the command. Here is an example:

```
python /workspace/libs/ONNX_Convertor/keras-onnx/generate_onnx.py abs_path_of_input_
```

Add RGBN to YYNN layer for Keras model*(optional)*

Some of the models might take gray scale images instead of RGB images as the input. Here is a small script to add an extra layer to let the model take RGB input.

```
cd /workspace/libs/ONNX_Convertor/keras-onnx && python rgba2yynn.py input_hdf5_file
```

3.1.2 Pytorch to ONNX

Our scripts not only takes `pth` file as the input. It also takes Pytorch exported `onnx` as the input. In fact, we recommend using the Pytorch exported `onnx` file instead of the `pth` file, since the Pytorch do not has a very good model save and load API. You can check TIPS below on how to export models to onnx.

We currently only support models exported by Pytorch version >=1.0.0, <1.6.0, no matter it is a `pth` file or an `onnx` file exported by `torch.onnx`.

> 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.

Run pytorch2onnx with pth file

Suppose the input file is called `/docker_mount/resnet34.pth` and you want to save the onnx as `/docker_mount/resnet34.onnx`. The input channel, height, width for the model are (3, 224, 224). Here is the example command:

```
python /workspace/libs/ONNX_Convertor/optimizer_scripts/pytorch2onnx.py /data1/pytor
```

You need to run the command in section 3.1.6 after this step.

Preprocess with Pytorch generated onnx file

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/libs/ONNX_Convertor/optimizer_scripts/pytorch_exported_onnx_prepro
```

You need to run the command in section 3.1.6 after this step.

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/libs/ONNX_Convertor/caffe-onnx/generate_onnx.py -o /docker_mount/
```

You need to run the command in section 3.1.6 after this step.

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/libs/ONNX_Convertor/optimizer_scripts/tensorflow2onnx.py /docker_
```

You need to run the command in section 3.1.6 after this step.

3.1.5 TF Lite to ONNX

```
python /workspace/libs/ONNX_Convertor/tflite-onnx/onnx_tflite/tflite2onnx.py -tflit
```

For the provided example model: `model_unquant.tflite`

```
python /workspace/libs/ONNX_Convertor/tflite-onnx/onnx_tflite/tflite2onnx.py -tflit
```

Then need to run the command in section 3.1.6.

3.1.6 ONNX to ONNX (ONNX optimization)

After converting models from other frameworks to onnx format, 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`:

```
python /workspace/libs/ONNX_Convertor/optimizer_scripts/onnx2onnx.py input.onnx -o
```

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.

3.1.7 Model Editor

KL520 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.

General Editor Guideline

Here are some general guideline to edit a model to take full advantage of KL520 NPU MAC efficiency:

Step 1: Start from each output node of the model, we should trace back to an operator that has significant compute workload, such as `GlobalAveragePool`, `Gemm`(fully connect), or `Conv`. Then, we could cut to the output of that OP. For example, there is a model looks Figure 4, and it has two output nodes: `350` and `349`. From output node `349`, we can trace back to the `Gemm` above the red line because the `Div`, `Clip`, `Add` and `Mul` only have 1x5 dimension, and these OPs are not very heavy computation. Since `Mul` and `Div` are not support in NPU, so it is recommend to cut the rest of the OPs and let the model finish at the output of the `Gemm` (red line). For the other output node `350`, since it is the output of a `Gemm`, there is no need to do any more edition.

Step 2: If both input nodes and output nodes are channel last, and there is a `Transpose` after the input and before the output, then the model is transposed into channel first. We can use the model editor to safely remove the `Transpose`.

Step 3: If the input shape is not availble or invalid, we can usethe editor to give it a valid shape.

Step 4: The model need to pass `onnx2onnx.py` again after running the editor. See section 3.1.6.

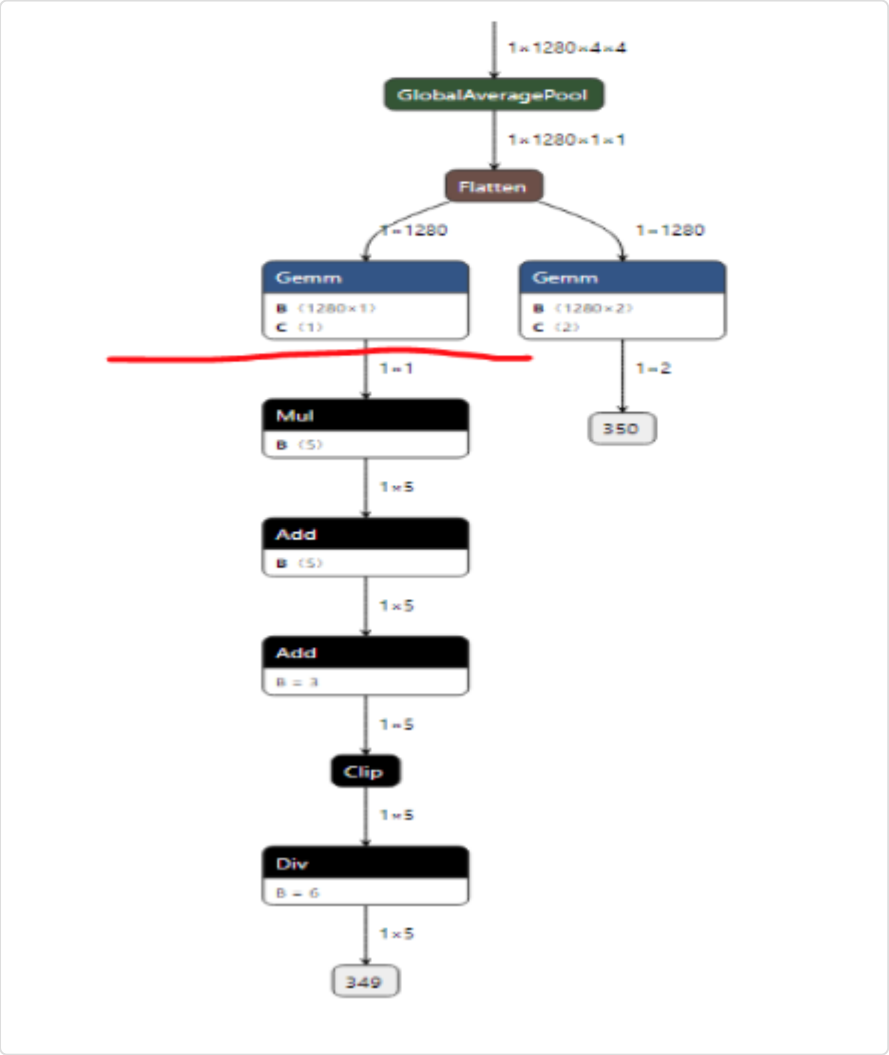

Figure 4. Pre-edited model

Feature The script called `editor.py` is under the folder `/workspace/libs/ONNX_Convertor/optimizer_scripts`. It is a simple ONNX editor which achieves the following functions:

1. Add nop `BN` or `Conv` nodes.
2. Delete specific nodes or inputs.
3. Cut the graph from certain node (Delete all the nodes following the node).
4. Reshape inputs and outputs
5. Rename the output.

Usage

```
editor.py [-h] [-c CUT_NODE [CUT_NODE ...]]
             [--cut-type CUT_TYPE [CUT_TYPE ...]]
             [-d DELETE_NODE [DELETE_NODE ...]]
             [--delete-input DELETE_INPUT [DELETE_INPUT ...]]
             [-i INPUT_CHANGE [INPUT_CHANGE ...]]
             [-o OUTPUT_CHANGE [OUTPUT_CHANGE ...]]
             [--add-conv ADD_CONV [ADD_CONV ...]]
             [--add-bn ADD_BN [ADD_BN ...]]
             in_file out_file

Edit an ONNX model. The processing sequense is 'delete nodes/values' -> 'add nodes'
Cutting is not recommended to be done with other operations together.

positional arguments:

in_file   input ONNX FILE
out_file  ouput ONNX FILE

optional arguments:

        -h, --help            show this help message and exit
        -c CUT_NODE [CUT_NODE ...], --cut CUT_NODE [CUT_NODE ...]
        remove nodes from the given nodes(inclusive)

        --cut-type CUT_TYPE [CUT_TYPE ...]
        remove nodes by type from the given nodes(inclusive)

        -d DELETE_NODE [DELETE_NODE ...], --delete DELETE_NODE [DELETE_NODE ...]
        delete nodes by names and only those nodes

        --delete-input DELETE_INPUT [DELETE_INPUT ...]
        delete inputs by names

        -i INPUT_CHANGE [INPUT_CHANGE ...], --input INPUT_CHANGE [INPUT_CHANGE ...]
        change input shape (e.g. -i 'input_0 1 3 224 224')

        -o OUTPUT_CHANGE [OUTPUT_CHANGE ...], --output OUTPUT_CHANGE [OUTPUT_CHANGE
        change output shape (e.g. -o 'input_0 1 3 224 224')

        --add-conv ADD_CONV [ADD_CONV ...]
        add nop conv using specific input

        --add-bn ADD_BN [ADD_BN ...]
        add nop bn using specific input
```

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`.

```
mkdir /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 toolchain 520 in Interactive Folder. 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:

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

`thread_number`: the number of thread to use.

After running this program, the folders called `compiler` and `fpAnalyser` will be generated in the Interactive Folder. `compiler` stores the result of compiler and ipEvaluator. `fpAnalyser` stores the result of the model analyzer.

3.2.3 Get the result

Under `/data1/fpAnalyser`, we can find `<model_name>.quan.bias.bie` which is the encrypted result of the model analyzer.

Under `/data1/compiler`, we can find three binary files generated by the compiler which are the inputs of the simulator. We can also find `ip_eval_prof.txt` which is the evaluation result of ipEvaluator. You can check the estimate performance of your model from it.

3.2.4 Hardware validation

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.

```
python /workspace/scripts/hardware_validate_520.py
```

If it succeeds, you can the command line print out log: `[info] hardware validating successes!`. And you can find 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 fpAnalyser, it can be used when your model structure is prepared but hasn’t been trained.

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:

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

And a folder called `compiler` will be generated in Interactive Folder, which stores the result of the compiler and ipEvaluator. The files should be the same name as the result from `fpAnalyserCompilerIpevaluator_520.sh`.

It uses the default configuration for the kdp520 and not available for fine-tuning.

3.5 FpAnalyser and Batch-Compile

This part is the instructions for batch-compile, which will generate the binary file requested by firmware.

Again, we'll use the `LittleNet` as an example. But this time, we need to copy the config `batch_input_params.json`

> The `batch_input_params.json` is different in 0.10.0. Please check the FAQ for new config fields.

```
cp /data1/LittleNet/batch_input_params.json /data1
```

3.5.1 Fill the input parameters

Fill the input parameters in the `/data1/batch_input_params.json` in Interactive Folder. Please refer on the FAQ question 6 to fill the related parameters. We already have this file under `/data1` if we followed the example.

3.5.2 Running the programs

For running the compiler and ip evaluator:

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

`thread_number`: the number of thread to use

And a folder called batch_compile will be generated in Interactive Folder, which stores the result of the fpAnalyer and batch-compile.

3.5.3 Get the result

In Interactive Folder, you'll find a folder called `batch_compile`, which contains the output files of batch compile. The main output are `all_model.bin`, `fw_info.bin` and `model_520.nef`. If you have questions for the meaning of the output files, please refer to the FAQ question 7.

3.6 Batch-Compile

This part is the instructions for batch-compile, which will generate the binary file requested by firmware. It is similar with section 3.5, but with already existed FP-analyser result models.

3.6.1 Fill the input parameters

Fill the input parameters in the `/data1/batch_input_params.json` in Interactive Folder. Please refer on the FAQ question 6 to fill the related parameters. We already have this file under `/data1` if we followed the example. But to run this part. we need some modifications.

In the `batch_input_params.json` mentioned above, replaced the path of onnx model with the path to the bie model. And `input_params` fields are no longer needed. In the example, replace `/data1/LittleNet/LittleNet.onnx` with `/data1/fpAnalyser/LittleNet.quan.bias.bie`.

Please check the FAQ 6 for more details about the new config fields.

3.6.2 Running the programs

For running the compiler and ip evaluator:

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

And a folder called `batch_compile` will be generated in Interactive Folder, which stores the result of the batch-compile.

3.6.3 Get the result

In Interactive Folder, you'll find a folder called `batch_compile`, which contains the output files of batch compile. The main output are `all_model.bin`, `fw_info.bin` and `model_520.nef`. If you have questions for the meaning of the output files, please refer to the FAQ question 7.

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/#python_app/app_flow_manual/.

FAQ

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

By following the above instructions, the `input_params.json` will be saved in Interactive Folder. 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 fiel
        // will be ignored.
        "input_onnx_file": "/data1/yolov5s_e.onnx",
        // A list of the model inputs name with the absolute path of their correspo
        // inputs image folders. It is required by FP-analysis.
        "model_inputs": [{
            "model_input_name": "data_out_0" ,
            "input_image_folder": "/data1/100_image/yolov5",
        }]
    },
    // 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 bot
        //     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. Defaul
        // -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.

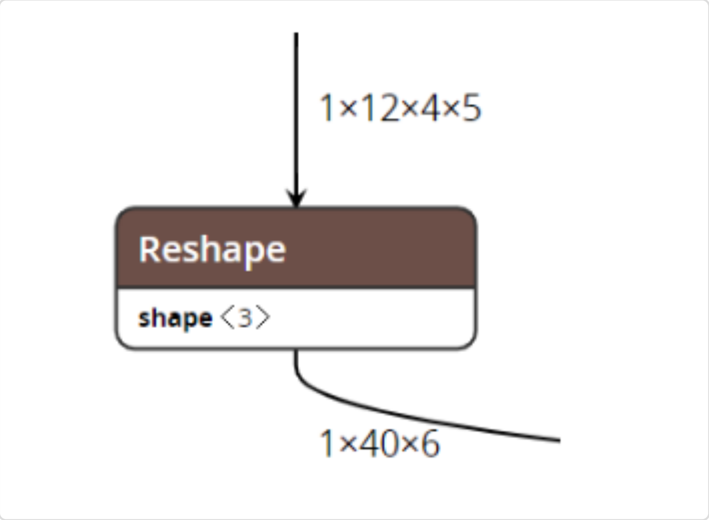

Figure 13. Invalid Reshape operation

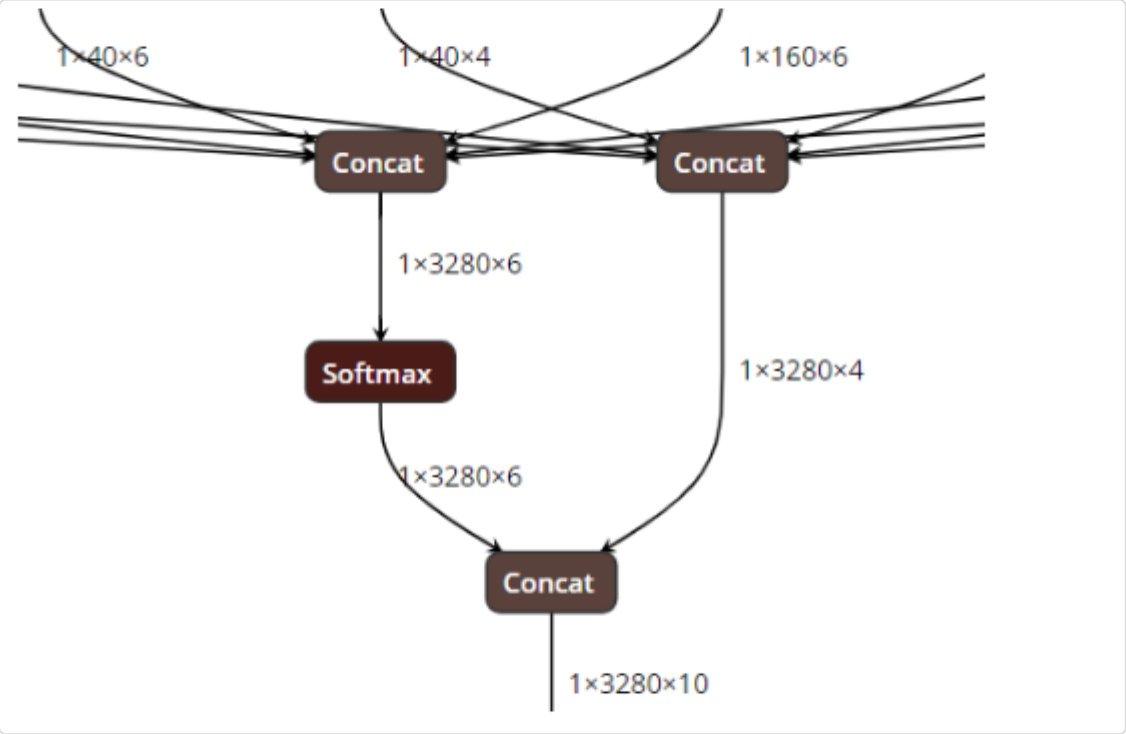

Figure 14. Invalid Concat operation

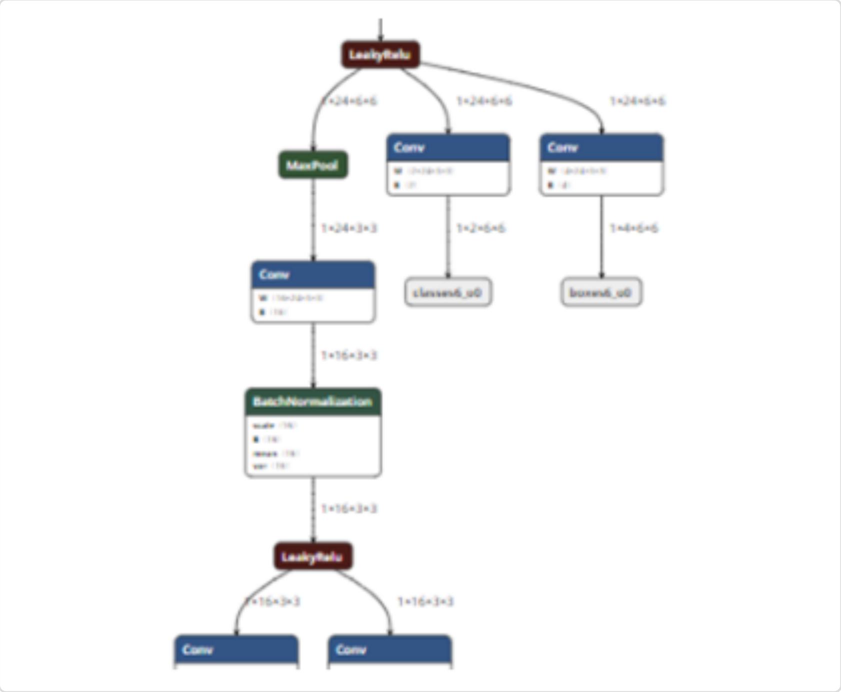

Figure 15. Valid SSD operation

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 in Interactive Folder. 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": false,
    // Batch compile model list
    "models": [
        {
            // Model ID
            "id": 19,
            // Model version
            "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 of batch-compile?

The result of 3.7 FpAnalyser and Batch-Compile is generated at a folder called batch_compile at Interactive Folder, and it has two sub-folders called fpAnalyser and compiler.

In fpAnalyser subfolder, it has the folders with name format as `input_img_txt_X`, which contains the .txt files after image preprocessing. The index X is the related to the order of model file in FAQ question 7 (6), which means the folder `input_img_txt_X` is number X model’s preprocess image text files.

In compiler subfolder, it will have the following files: `all_model.bin`, `fw_info.bin`, `temp_X_ioinfo.csv`. The X is still the order of the models.

-`all_model.bin` and `fw_info.bin` is for firmware to use.

-`temp_X_ioinfo.csv` contains the information that cpu node and output node.

If you find the cpu node in `temp_X_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 (../../host_lib_0.9.0/kdp2_api/kdp2_postprocess.h/)
Post Process API (../../host_lib_0.9.0/kdp2_api/kdp…

Next ❯ (../manual_720/)
Toolchain 720 Manual (../manual_720/)