



# Kneron Linux Toolchain 720 Manual

2020 October Toolchain 720 v0.9.0

PDF Downloads (manual\_720.pdf)

## 0. Overview

KDP toolchain is a set of software which provide inputs and simulate the operation in the hardware 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:720_v0.9.0`.**

*Performance simulation result on NPU:*

Model	Size	FPS (npu only)	Time(npu only)	Has CPU node(s)?
Inception v3	224x22480.9		12.4 ms	No
Inception v4	299x29919.9		50.2 ms	No
Mobilenet v1	224x224404		2.48 ms	No
Mobilenet v2	224x224624		1.60 ms	No
Mobilenet v2 ssdlite	300x300283		3.54 ms	No
Resnet50 v1.5	224x22452.3		19.1 ms	No
OpenPose	256x2565.3		189 ms	No
SRCNN	384x384127		7.87 ms	No
Tiny yolo v3	416x416148		6.75 ms	No
Yolo v3	416x41610.5		95.3 ms	No
Centernet res101	512x5123.02		331 ms	No
Unet	384x3842.83		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.

# 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 720.

```
docker pull kneron/toolchain:720
```

Note that this document is compatible with toolchain v0.9.0. You can find the version of the toolchain in `/workspace/version.txt` inside the docker. If you find your toolchain is later than v0.9.0, you may need to visit 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:720` 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:720
```

### 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.

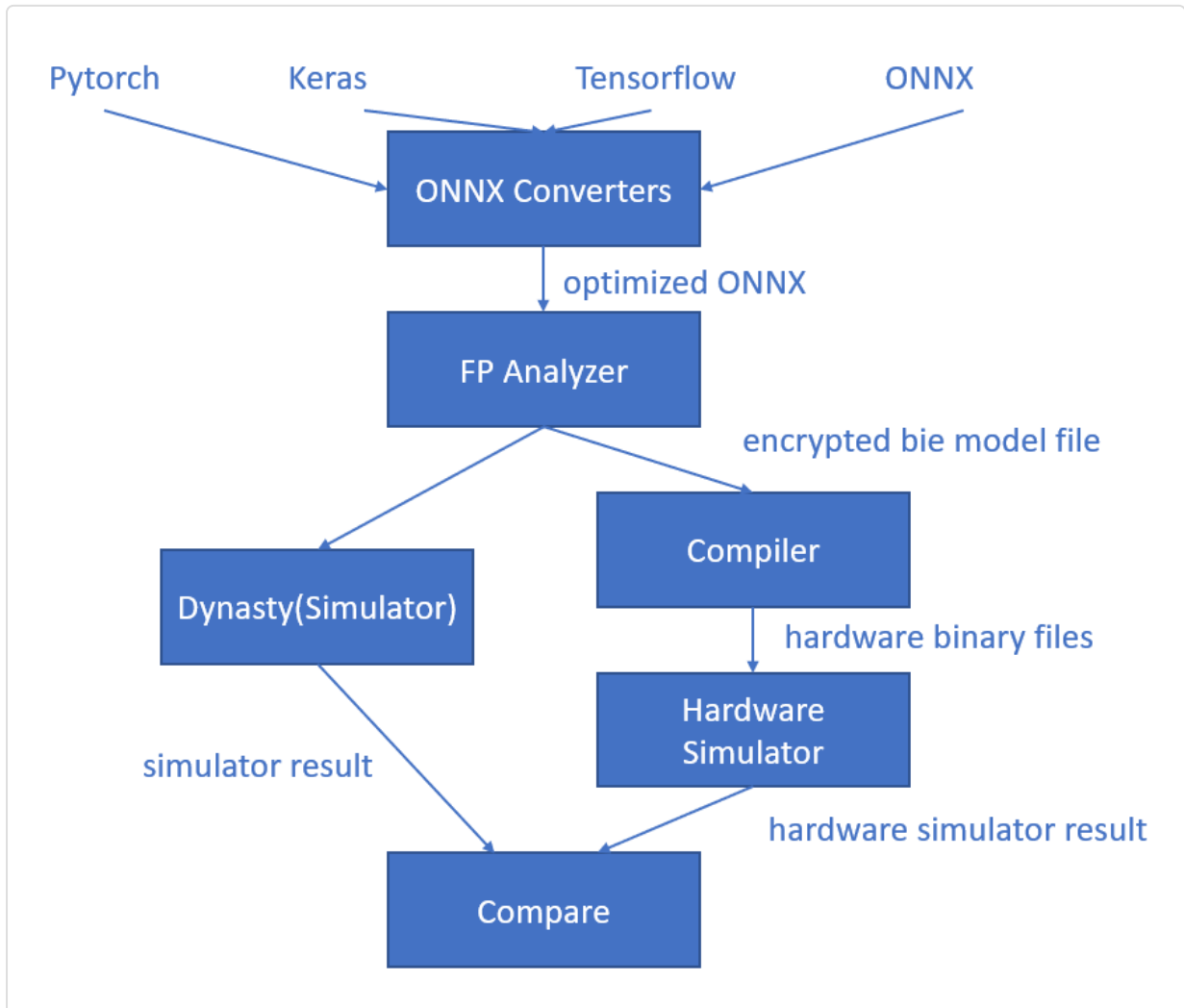
### 2.1 Folder structure

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 kdp720 hardware and the IP evaluator to infer
|   |-- dynasty         # Simulator which only simulates the kdp720 calculation.
|   |-- fpAnalyser      # Analyze the model and provide fixed point information.
|   |-- hw_c_sim        # Hardware simulator which simulate all the kdp720 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 kdp720. 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 step 3.2 using scripts.

## 2.3 Supported operators

Table 1.1 shows the list of functions KDP720 supports base on ONNX 1.4.1.

Table 1.1 The functions KDP720 NPU supports

Node	Applicable Subset	Spec.
Relu		support
PRelu		support
LeakyRelu		support
Sigmoid		support
Clip		min = 0
Tanh		support
BatchNormalization4D	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		support
Mul		support
MaxPool		kernel = [1, 1], [2, 2], [3, 3]
AveragePool	3x3	kernel = [1, 1], [2, 2], [3, 3]
GlobalAveragePool4D	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.

### 3.1 Converters

ONNX\_Convertor is an open-source project on Github ([https://github.com/kneron/ONNX\\_Convertor](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.

### 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_c
```

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/c
```

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 rgba2yyynn.py input_hdf5_file
```

### 3.1.2 Pytorch to ONNX

The `pytorch2onnx.py` script 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  $\geq 1.0.0$ ,  $< 1.6.0$ , 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=True)
```

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/pyto
```

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

### **Run pytorch2onnx with 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/pytorch2onnx.py /data1/pyto
```

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/r
```

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_r
```

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 -tflite
```

For the provided example model: `model_unquant.tflite`

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

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 c
```

#### **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

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 KL720 NPU cannot support well, such as `Softmax` or `Sigmoid`. 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 KL720 NPU can run the model more efficiently.



## General Editor Guideline

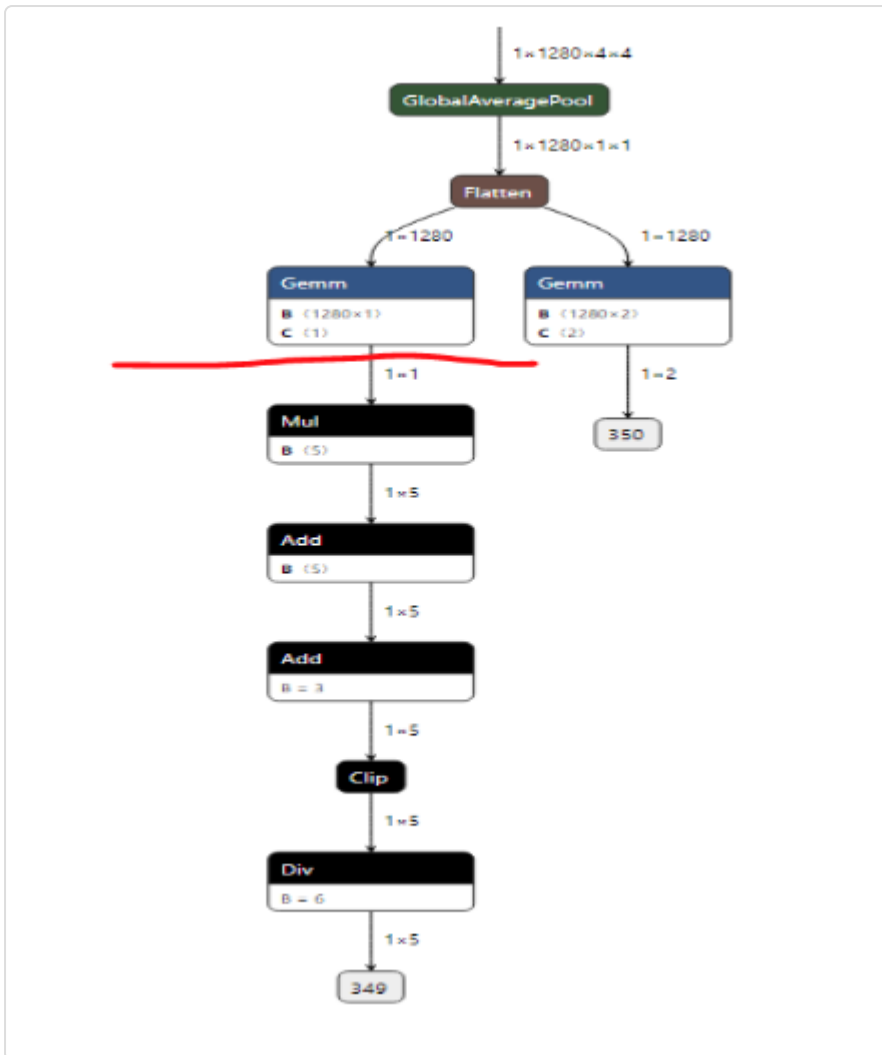
Here are some general guideline to edit a model to take full advantage of KL720 NPU MAC efficiency:

**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 available or invalid, we can use the 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 sequence 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 output 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 Interactive 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 720 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 the part FAQ question 1.

### 3.2.2 Running the program

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

```
# cd /workspace/scripts && ./fpAnalyserCompilerIpevaluator_720.sh thread_number
cd /workspace/scripts && ./fpAnalyserCompilerIpevaluator_720.sh 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 `output.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 `ProfileResult.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.

```
cd /workspace/scripts && ./hardware_validate_720.sh
```

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 `/validate/dynasty` and `/validate/hw_c_sim`.

## 3.3 Simulator and Emulator

`emulator_720.sh` is just the simulator running on a folder of images. It supports multi-process to accelerate the speed.

### 3.3.1 Fill the input parameters

Fill the simulator and emulator input parameters in the `input_params.json` in Interactive Folder. Please refer on the FAQ question 1 to fill the related parameters. And here as an example, we'll also use `LittleNet` mentioned at the beginning of section 3.2.

### 3.3.2 Running the programs

For running the simulator:

```
cd /workspace/scripts && ./simulator_720.sh
```

And a folder called `simulator` will be generated in Interactive Folder, which stores the result of the simulator.

For running the emulator:

```
cd /workspace/scripts && ./emulator_720.sh
```

Those two scripts need the output from the model analyzer as input. Thus, please run

```
fpAnalyserCompilerIpevaluator_720.sh
```

 first.

### 3.3.3 Get the result

In Interactive Folder, you'll find a folder called `simulator`, which contains the output files of simulator; a folder called `emulator`, which contains the output folders of emulator.

In each folder, there are three files: one is the input image file, one whose format is `layer_output_***_f1.txt` is the output of the last layer, and the other one is the preprocess image result and result in binary format.

## 3.4 Compiler and Evaluator

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_720.sh /data1/LittleNet/LittleNet.onnx
```

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_720.sh`.

It uses the default configuration for the kdp720 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_compile_input_params.json`

```
cp /data1/LittleNet/batch_compile_input_params.json /data1
```

### 3.5.1 Fill the input parameters

Fill the simulator and emulator input parameters in the `/data1/batch_compile_input_params.json` in Interactive Folder. Please refer on the FAQ question 7 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:

```
# cd /workspace/scripts && ./fpAnalyserBatchCompile_720.sh thread_number  
cd /workspace/scripts && ./fpAnalyserBatchCompile_720.sh 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 `fpAnalyser` 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 the `fpAnalyser` and `batch compile`. If you have questions for the meaning of the output files, please refer to the FAQ question 8.

Also, to compile the `all_model.bin` and `fw_info.bin` files into `dfu.bin`, please use the following command:

```
cd /workspace/scripts && ./generate_dfu_binary_for_720 -model fw_info.bin all_model
```

## 3.6 Draw Yolo result on images

The toolchain also provides the function of drawing final result on images for yolo model, i.e. drawing the box and class name.

### 3.6.1 Steps

**Step 1:** Follow the part 3.3 Simulator and Emulator, it will generate the result of the emulator for multiply images, and the result folder path is `/data1/emulator`. In this folder, the original image, the preprocess image txt file and the final output of the model are classified in different folders.

**Step 2:** Fill the anchor and class name information. Go to the folder path `/workspace/scripts/utis/yolo/kerasyolo3/model_data`, fill the `anchor.txt` and `classes.txt` based on the information of your model.

**Step 3:** Run the scripts to draw the yolo result.

```
cd /workspace/scripts/utils/yolo && python convert_sim_result_yolo.py --input_size y

# input_size is the height/width of the model's input, e.g. if the input onnx mode
# output_num is the number of the model's output nodes;
# whether_keep_aspect_ratio should be the same when you configured in part 3.7.3;
# your_score is the score threshold you set for the detection box
```

**Step 4:** After step 3, the drawing result will be saved in the subfolders of `/data1/emulator`, with the format `imgname_thresh_xxx.png`, `xxx` means the threshold for the box score, which means only the boxes with the score higher than this threshold will be drawn in this image.

## 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 output
```

### 3.7.2 Post process

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

## 3.8 E2ESimulator workflow

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

The detailed manual of E2ESimulator can be found at [http://doc.kneron.com/docs/#python\\_app/app\\_flow\\_manual/](http://doc.kneron.com/docs/#python_app/app_flow_manual/) ([http://doc.kneron.com/docs/#python\\_app/app\\_flow\\_manual/](http://doc.kneron.com/docs/#python_app/app_flow_manual/)).

### 3.8.1 test case data

The folder `/workspace/scripts/E2E_Simulator/bin/test1` provides the input data for the face detection test case.

### 3.8.2 test case command

```
cd /workspace/E2E_Simulator/python_flow && python example.py -d ../bin/test1 -i bin/
```

And the final result will be saved at the path: `/workspace/scripts/E2E_Simulator/bin/`.

## 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.

The parameters in `input_params.json` are:

1. `input_image_folder`  
The absolute path of input image folder for fpAnalyser.
2. `img_channel`  
Options: L, RGB  
The channel information after the input image is preprocessed. L means single channel.  
Input for fpAnalyser.
3. `model_input_width`  
The width of the model input size. Input for fpAnalyser.
4. `model_input_height`  
The height of the model input size.
5. `img_preprocess_method`  
Options: `kneron`, `tensorflow`, `yolo`, `caffe`, `pytorch`  
The image preprocess methods, input for fpAnalyser, and the related formats are following:  
`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
6. `input_onnx_file`  
The absolute path of the onnx file, which works as the input file for fpAnalyser.
7. `keep_aspect_ratio`  
Options: True, False  
Indicates whether or not to keep the aspect ratio.
8. `command_addr`  
Address for command, input for compiler.
9. `weight_addr`  
Address for weight, input for compiler.
10. `sram_addr`  
Address for sram, input for compiler.
11. `dram_addr`  
Address for dram, input for compiler.
12. `whether_encryption`  
Option: Yes, No  
Whether add encryption on the bin files generated by compiler, input for compiler.
13. `encryption_key`  
Encryption key for bin files, input for compiler.
14. `simulator_img_file`  
Input for simulator.  
The absolute path of the image you want to inferenced by simulator.
15. `emulator_img_folder`  
The absolute path of the image folder you want to inferenced by emulator.
16. `cmd_bin`  
The absolute path of command binary file, which is the input file for simulator or emulator.
17. `weight_bin`  
The absolute path of weight binary file, which is the input file for simulator or emulator.



18. `setup_bin`  
The absolute path of setup binary file, which is the input file for simulator or emulator.
19. `whether_npu_preprocess`  
The option for whether simulator or emulator using the same image processing as the npu uses.  
If false, the parameters (20) - (25) will not be utilized.  
Parameters (20) - (25) is for npu image preprocessing.
20. `raw_img_fmt`  
The input image format for simulator and emulator  
Options: IMG, RGB565, NIR888  
IMG: jpg/png/jpeg/bmp image files  
RGB565: binary file with rgb565 format;  
NIR888: binary file with nir888 format.
21. `radix`  
The radix information for the npu image process.  
The formula for radix is  $7 - \text{ceil}(\log_2(\text{abs\_max}))$   
For example, if the image processing method we utilize is "kneron", which is introduced in the parameter (5). So the related image processing formula is "kneron":  $\text{RGB}/256 - 0.5$ , and the processed value range will be  $(-0.5, 0.5)$ , and then  
 $\text{abs\_max} = \max(\text{abs}(-0.5), \text{abs}(0.5)) = 0.5$   
 $\text{Radix} = 7 - \text{ceil}(\log_2(\text{abs\_max})) = 7 - (-1) = 8$
22. `pad_mode`  
This is the option for the mode of adding paddings, and it will be utilized only when (7) `keep_aspect_ratio` is True.  
And it has two options: 0 and 1.  
0 – If the original width is too small, the padding will be added at both right and left sides equally; if the original height is too small, the padding 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 right side only, if the original height is too small, the padding will be only added at the down side.
23. `roate`  
It is actually a typo of rotate. But current examples and scripts are adapted to this typo. It will be fixed later.  
It has three options:  
0 – no rotating operation  
1 – rotate 90 degrees in clockwise direction  
2 – rotate 90 degrees in counter-clockwise direction
24. `pCrop`  
The parameters for cropping image.  
And it has four sub parameters.  
-bCropFirstly, whether cropping the image firstly, if false, the following parameters won't be utilized, and there won't be any cropping operations.  
-crop\_x, crop\_x, the left-up cropping point coordinate.  
-crop\_w, the width of the cropped image.  
-crop\_h, the height of the cropped image.
25. `imgSize`  
-width: input image width  
-height: input image height

26. `add_norm` (deprecated)

Option: `True` / `False`

This option is whether add an extra conv layer at the beginning of the model, it will help when dealing with the image preprocess. For example, when you choose `yolo` as the image preprocess method, the preprocess's output range would be 0~1, in this case, all the computation is finished in CPU, which is time-consuming; when you choose `add_norm` as `True`, the image preprocess will be split into two parts: firstly, the CPU will finish the part of the preprocess, and the internal output range is -0.5 ~ 0.5, and then the NPU (the extra norm layer) will finish the remaining adding computation (+0.5), and the final will be 0~1.

27. `model_input_name`

The input name of the model. If you are not sure about it, you can check it through model visualize tool like netron (<https://netron.app/>)

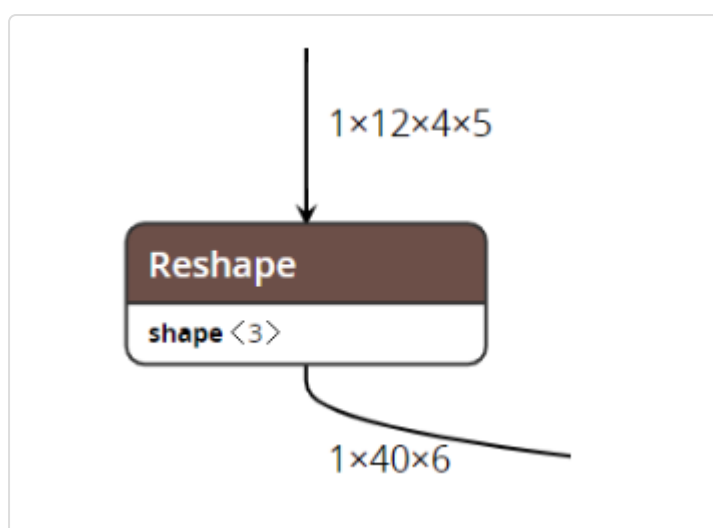
## 2. Fails when implement models with SSD structure.

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.

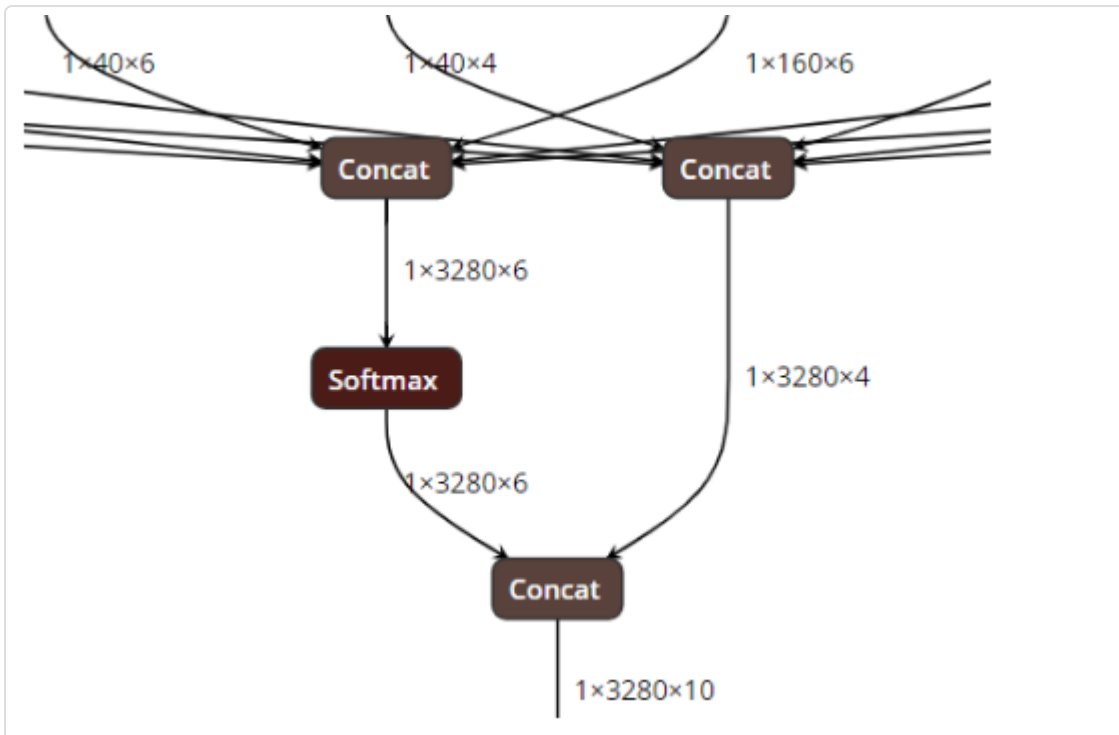
The reason we do not support Reshape and Concat operation is that we do offer Reshape operation capability, However, we only support regular shape transportation. Which means you could flatten your data or extract some channels form the feature map. However, NPU does not expect complex transportation. For example, in Figure 13, you could notice there is a  $1 \times 12 \times 4 \times 5$  feature map reshapes to  $1 \times 40 \times 6$ .

For Concat operation, the NPU also supports channel-based feature map concatenation. However, it does not support Concat 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.

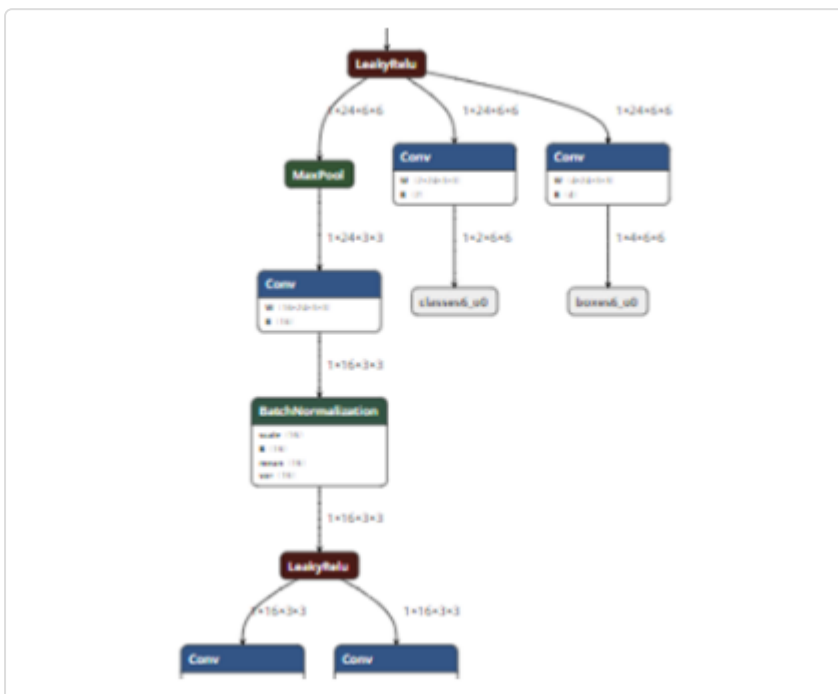
The workaround we will offer is that deleting these Reshape and Concat operations and enable make the model to a multiple outputs model since they are in the end of the models and they do not change the output feature map data. So converted model should look like this as in Figure 15.



**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_compile_input_params.json`?

By following the above instructions, the `batch_compile_input_params.json` will be saved in Interactive Folder. Please do not change the parameters' names. The parameters in `batch_compile_input_params.json` are:

1. `input_image_folder`

The absolute path of input image folder for fpAnalyser. Since that batch-compile can compile more than one models together, the order of the `input_imgae_folder` is related to the order of `input_onnx_file`.

2. `img_channel`

Options: L, RGB

The channel information after the input image is preprocessed. L means single channel. Input for fpAnalyser. Same as (1), the order of the `img_channel` is related to the order of `input_onnx_file`.

3. `model_input_width`

The width of the model input size. Input for fpAnalyser. Same as (1), the order of the `model_input_width` is related to the order of `input_onnx_file`.

4. `model_input_height`

The height of the model input size. Same as (1), the order of the `model_input_height` is related to the order of `input_onnx_file`.

5. `img_preprocess_method`

Options: kneron, tensorflow, yolo, caffe, pytorch. Same as (1), the order of the `img_preprocess_method` is related to the order of `input_onnx_file`.

The image preprocess methods, input for fpAnlayer, and the related formats are following:

`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]
- 6. `input_onnx_file`  
The absolute path of the onnx file, which works as the input file for fpAnalyser. Since that the batch-compile can compile more than one models at a time. The order of the model in the array of `input_onnx_file` decides the model binary's order in `all_models.bin`
- 7. `keep_aspect_ratio`  
Options: `True` , `False`  
Indicates whether or not to keep the aspect ratio.
- 8. `command_addr`  
Address for command, input for compiler.
- 9. `weight_addr`  
Address for weight, input for compiler.
- 10. `sram_addr`  
Address for sram, input for compiler.
- 11. `dram_addr`  
Address for dram, input for compiler.
- 12. `whether_encryption`  
Option: Yes, No  
Whether add encryption on the bin files generated by compiler, input for compiler.
- 13. `encryption_key`  
Encryption key for bin files, input for compiler.
- 14. `model_id_list`  
The list of model id information
- 15. `model_version_list`  
The list of model version information.
- 16. `add_norm_list` (deprecated) The list of whether add norm layer on each model.
- 17. `dedicated_output_buffer`  
Default: false  
If the option is chosen as true, the output nodes' results would be saved together, which will then be good for multiple output node models or parallel processing.

## 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 \(../manual\\_520/\)](#)

[Toolchain 520 Manual \(../manual\\_520/\)](#)

[Next ▶ \(../python\\_app/app\\_flow\\_manual/\)](#)

[End to End Simulator \(../python\\_app/app\\_flow\\_man...\)](#)