

RZ/V2L, RZ/V2M, RZ/V2MA AI IMPLEMENTATION GUIDE MMPose HRNet REV.7.20

SEPTEMBER 2022

RENESAS ELECTRONICS CORPORATION

R11AN0621EJ0720

Overview

This document explains the contents of AI Implementation Guide Get Started document with **HRNet** pre-trained model provided by MMPose framework (hereafter, **MMPose HRNet**).

Please read Get Started document in advance.

This document uses following documents and files.

名称	Filename	Details
Get Started Document	r11an0616ej0720-rzv-ai-imp-getstarted.pdf	Document for guiding how to make AI Implementation Guide environment and how to develop AI application.
Get Started Source Code	rzv_ai-implementation-guide_ver7.20.tar.gz	Source code used throughout the overall AI Implementation Guide.
Document for MMPose HRNet	r11an0621ej0720-rzv-ai-imp-hrnet.pdf	This document. Document for guiding the instruction for MMPose HRNet model.
Source Code for MMPose HRNet	mmpose_hrnet_ver7.20.tar.gz	Source code and example output used in the Document for MMPose HRNet.

MMPose

<https://github.com/open-mmlab>

Flow of Guide

Program made in the Step
Output of previous step

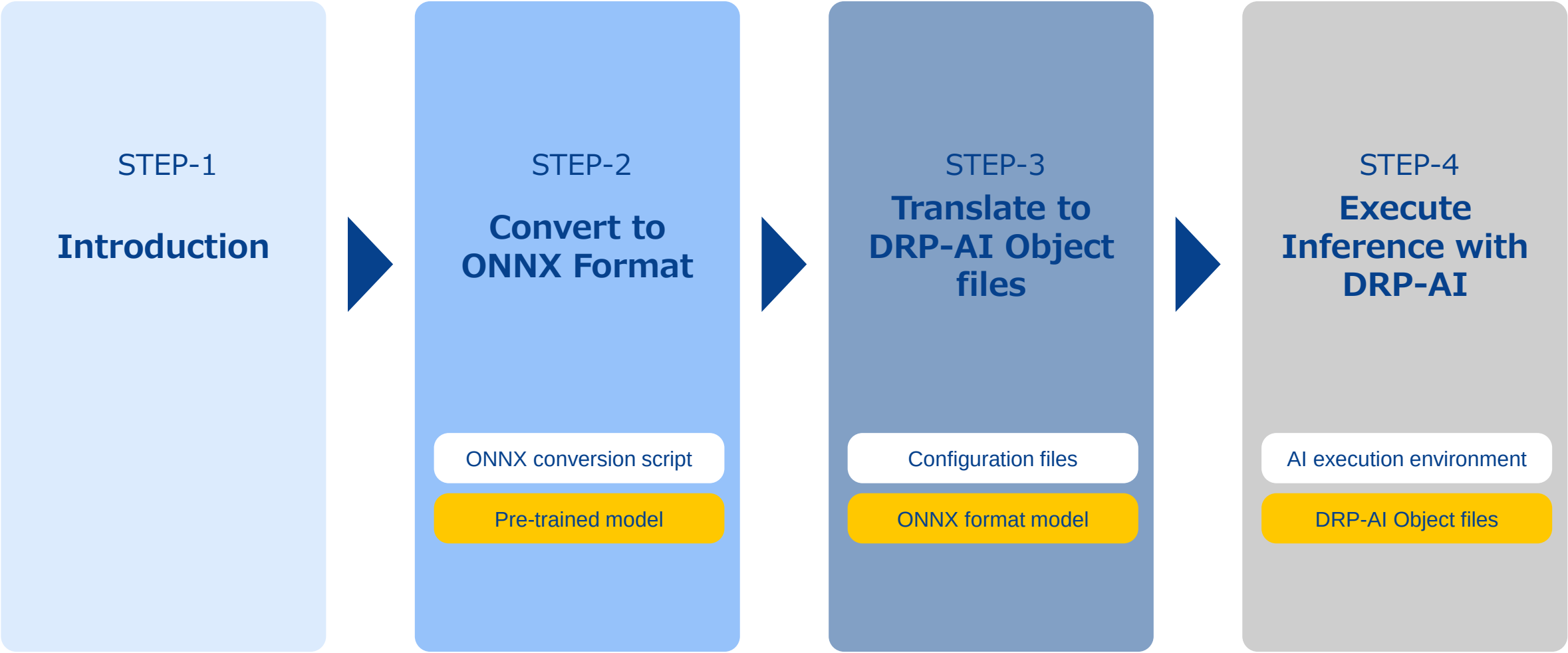


Table of Contents

STEP-1 Introduction

- Neural Network
- AI Framework
- Necessary Environment
- Necessary Files

STEP-2 Convert to ONNX Format

- 2.1: Convert to ONNX Format
- 2.2: Make the ONNX Conversion Environment
- 2.3: Prepare the Necessary Files
- 2.4: Convert AI Model to ONNX Format

STEP-3 Translate to DRP-AI Object files

- 3.1: Make the DRP-AI Translator Environment
- 3.2: Check the File Configuration
- 3.3: Prepare the ONNX File
- 3.4: Prepare the Address Map Definition File
- 3.5: Prepare the Pre/Postprocessing Definition File
- 3.6: Translate the Model Using DRP-AI Translator
- 3.7: Confirm the Translation Result

STEP-4 Execute Inference with DRP-AI

- Execute Inference with DRP-AI

Program made in the Step

Output of previous step

STEP-1

Introduction

STEP-2

Convert to
ONNX Format

ONNX conversion script

Pre-trained model

STEP-3

Translate to
DRP-AI Object
files

Configuration files

ONNX format model

STEP-4

Execute
Inference with
DRP-AI

AI execution
environment

DRP-AI Object files

HRNet

This document uses the HRNet model provided by MMPose framework.

HRNet(**H**igh-**R**esolution **N**etwork) is an image recognition neural network for Human Pose Estimation.

HRNet starts from high-resolution sub-network and adds low-resolution sub-network as the network goes deeper.

By training these sub-network in parallel, HRNet achieves more accurate pose estimation.

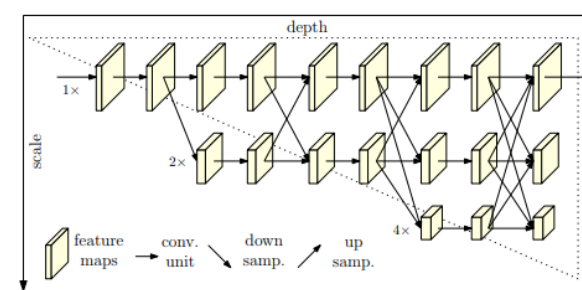
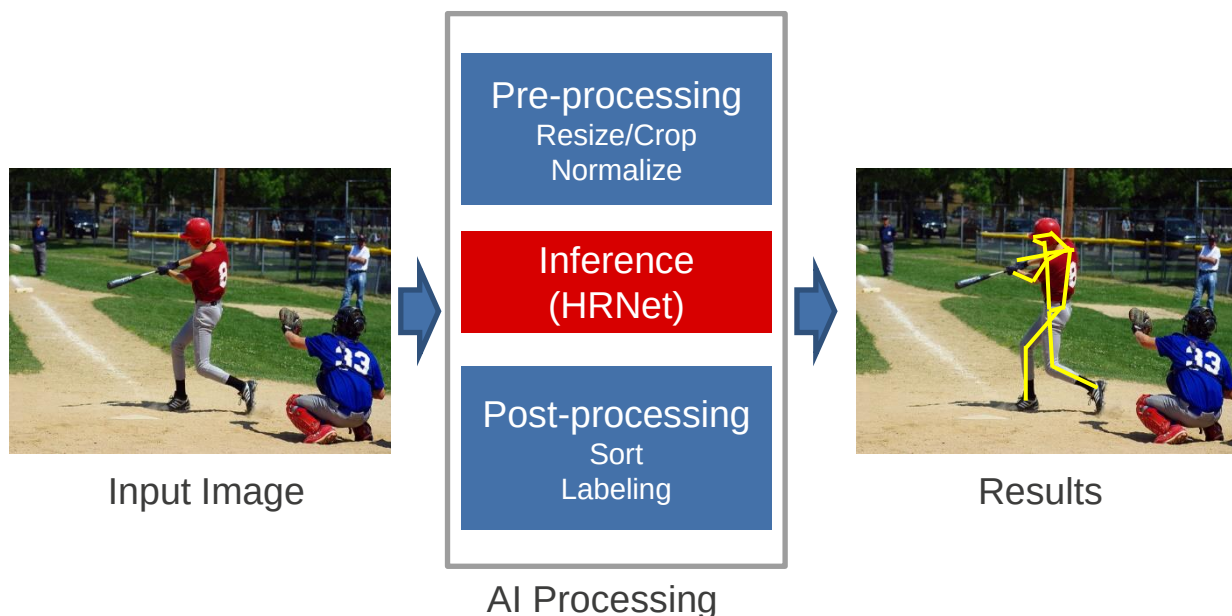


Figure 1. Illustrating the architecture of the proposed HRNet. It consists of parallel high-to-low resolution subnetworks with repeated information exchange across multi-resolution subnetworks (multi-scale fusion). The horizontal and vertical directions correspond to the depth of the network and the scale of the feature maps, respectively.

(Cite) Deep High-Resolution Representation Learning for Human Pose Estimation
<https://arxiv.org/pdf/1902.09212.pdf>

(Paper) Deep High-Resolution Representation Learning for Human Pose Estimation <https://arxiv.org/pdf/1902.09212.pdf>

STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4

Top-down vs Bottom-up

In 2-D Human Pose Estimation for multiple target, there are 2 approaches, which are Top-down approach and Bottom-up approach.

Top-down approach detects the human in the input image first, and then estimate its pose for each human.

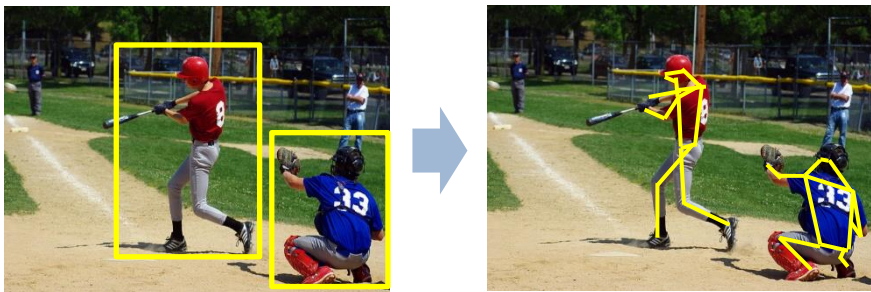
Bottom-up approach detects the all joint (key point) available first, and then groups the key point to each human.

HRNet is Top-down approach Pose Estimation model, which assumes the human detection is already finished.

Therefore, this document assumes the human detection is done and explains how to use the HRNet as a single-person human pose detection model.

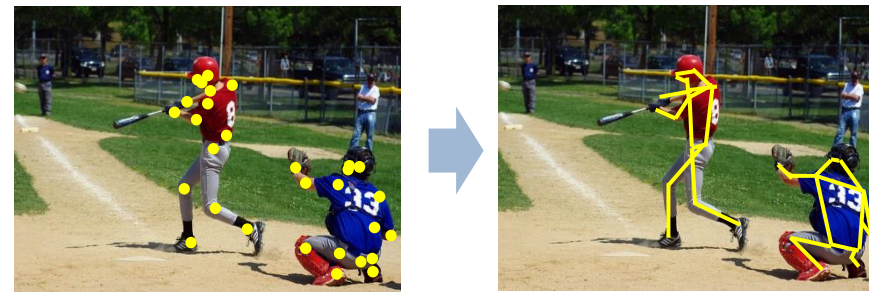
Top-down

Detect human and then detect key points for each human.



Bottom-up

Detect key points and then group them to each human.



(Paper) Deep High-Resolution Representation Learning for Human Pose Estimation <https://arxiv.org/pdf/1902.09212.pdf>

STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

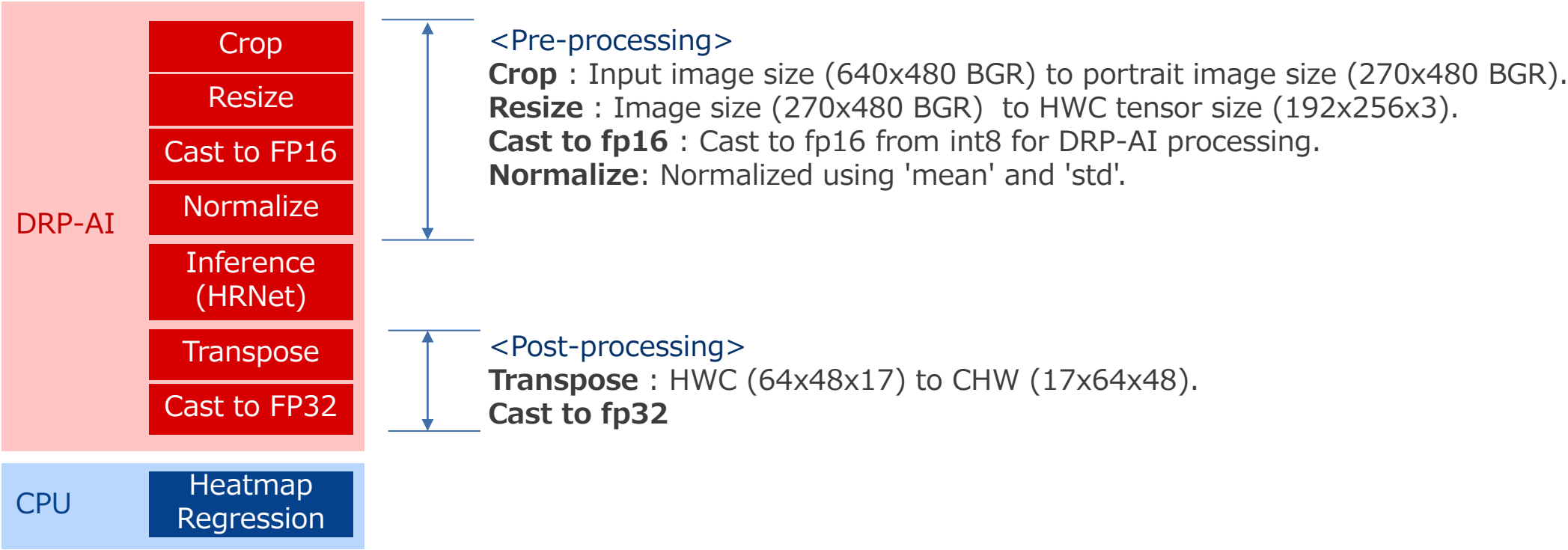
STEP-4

[Reference] HRNet on DRP-AI

Following is the example of HRNet model inference on DRP-AI.

The input image is 640x480 in BGR.

Operators that are not supported by DRP-AI need to be computed by CPU.



STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4

[Reference] MMPose & PyTorch

This document will use the pre-trained model provided by MMPose.

MMPose is a toolbox for pose estimation based on PyTorch framework.

It uses PyTorch training function, onnx conversion function (torch.onnx), etc. to enable the pose estimation model training/onnx conversion.

See PyTorch official document (<https://pytorch.org/docs/1.12/>) to learn more about the PyTorch.

See MMPose official website (<https://github.com/open-mmlab/mmpose>) to learn more about the MMPose.

STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4

MMPose

Pre-trained model

Based on
PyTorch



PyTorch

torch.onnx

Training

PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc. (<https://pytorch.org/>)

Necessary Environment

Following are the necessary environment for each STEP.
See the Get Started Document for how to build the environment.

<ONNX conversion env.>

MMPose  PyTorch^{*1}



Used in the following step.

- STEP-2: Convert to ONNX Format

<DRP-AI Translator env.>

DRP-AI Translator



Used in the following step.

- STEP-3: Translate to DRP-AI Object files

STEP-1
Neural Network
AI Framework
Necessary Environment
Necessary Files
STEP-2
STEP-3
STEP-4

^{*1} PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

Necessary Files

Source codes used in this document are provided in `mmpose_hrnet_ver7.20.tar.gz`.

📁 `mmpose_hrnet_ver7.20`

📁 `drpai_samples`

📁 `onnx`

📁 `hrnet_bmp`

📁 `mmpose`

📁 `hrnet`

} Example of input files of DRP-AI Translator.
Used in STEP-3.

} ONNX conversion script and post-processing script.
Used in STEP-2 and STEP-4.

STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4

[Additional Information] HRNet Training Environment

This guide will not explain about the training procedure of Deep Learning.

Please refer to the MMPose Official Website (<https://github.com/open-mmlab/mmpose>) to see how to train the model by own dataset, or how to customize the neural network.



*1 PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

STEP-1

Neural Network

AI Framework

Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4

STEP-1 Summary

STEP-1 explained the basic knowledge to implement AI model.

Please proceed to "[STEP-2 Convert to ONNX Format](#)".

STEP-1

Neural Network

AI Framework

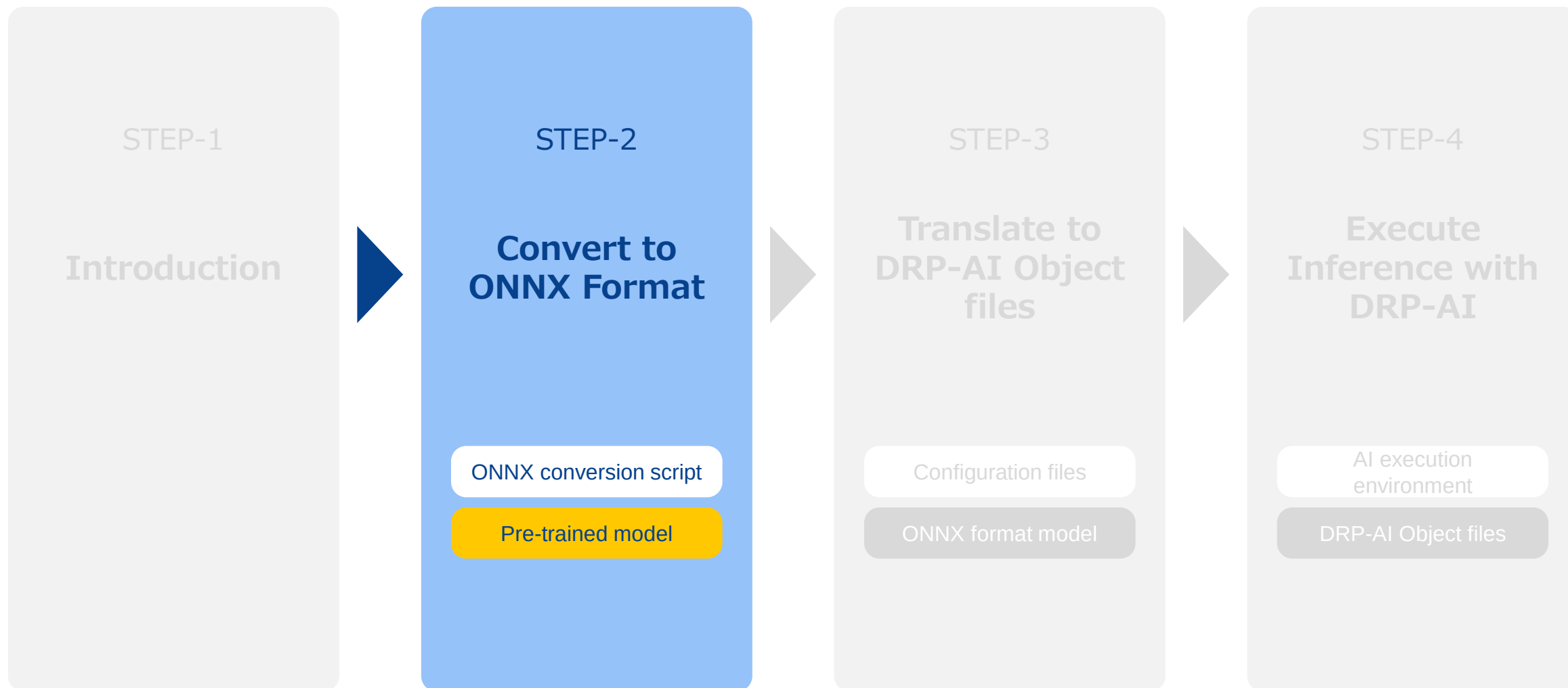
Necessary
Environment

Necessary Files

STEP-2

STEP-3

STEP-4



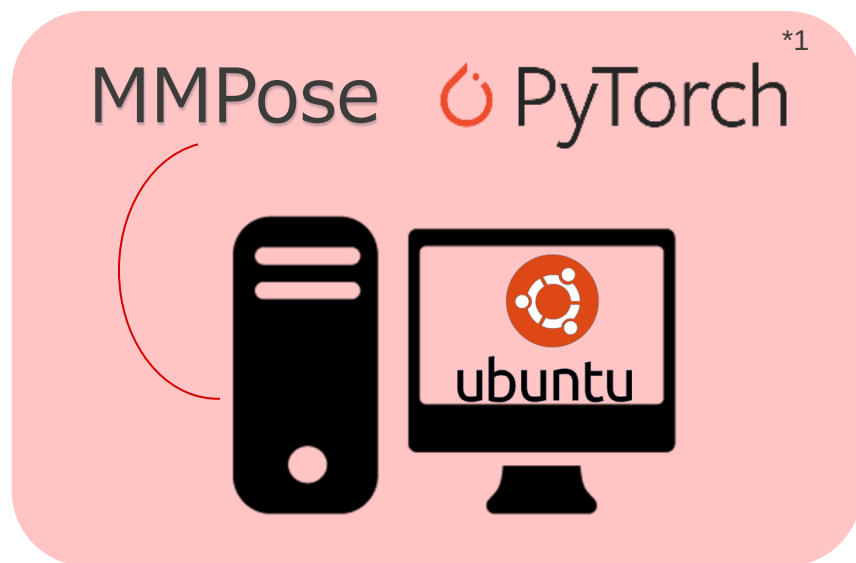
2.1: Convert to the ONNX Format

MMPose provides the pre-trained model structure and its weight parameter.

This STEP will explain the contents of Get Started Document "STEP-2 Convert to the ONNX Format" with HRNet model provided by MMPose.

Following is the necessary environment.

<ONNX conversion env.>



MMPose

Pre-trained HRNet model (COCO Dataset)

HRNet COCO Dataset Model:

<https://mmpose.readthedocs.io/en/latest/papers/algorithms.html#topdown-heatmap-hrnet-on-coco>



ONNX

Conversion from MMPose to ONNX

ONNX tutorials:

<https://github.com/onnx/tutorials>

MMPose official tutorial:

https://mmpose.readthedocs.io/en/latest/tutorials/5_export_model.html

*1 PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

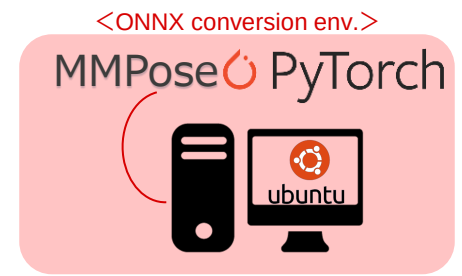
2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.2: Make the ONNX Conversion Environment



This section will explain the instruction with the following assumption

- Constructed the ONNX conversion environment according to Get Started Document "[STEP-2.2: Make the ONNX Conversion Environment](#)".

Please check the following item.

1. Confirm the environment variable is registered properly. **Green is the environment variable.**

```
$ printenv WORK
```

If displayed as follows, the variable is correctly set.

```
<path to the working directory>/rzv_ai_work
```

2. Register the symbolic link. (This command must be executed when new terminal is opened.)

```
$ cd $WORK/mmpose  
$ sudo python3 setup.py develop
```

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.3: Prepare the Necessary Files



This section will explain the instruction with the following assumption

- Extracted the necessary files according to Get Started Document "[STEP-2.3: Prepare the Necessary Files](#)".

Please run the following commands to prepare the necessary files for this document.

1. Move to the working directory. **Green is the environment variable.**

```
$ cd $WORK
```

2. Extract tar.gz file under the working directory.

```
$ tar xvzf <File path>/mmpose_hrnet_ver7.20.tar.gz -C $WORK
```

3. Check the working directory.

```
$ ls $WORK
```

If displayed as follows, the package is correctly extracted.

```
drpai_samples  mmpose
```

"drpai_samples" includes sample codes and example output of DPR-AI Translator.

"mmpose" includes sample code for MMPose.

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.3: Prepare the Necessary Files

Please confirm that each directory configuration is as follows.

<ONNX conversion working directory (\$WORK)>

- 📁 rzv_ai_work
 - 📁 drpai_samples
 - 📁 onnx
 - 📄 hrnet.onnx
 - 📁 hrnet_bmp
 - 📄 addrmap_in_linux.yaml
 - 📁 mmpose
 - 📁 hrnet
 - 📄 postprocess_hrnet.py
 - 📁 configs
 - 📁 demo
 - ...



STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

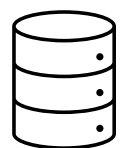
2.4: Convert AI Model to ONNX Format

Since MMPose uses PyTorch onnx conversion function in its script, following NN model structure and its weight parameter are required as explained in the Get Started Document "STEP-2.4: Convert AI Model to ONNX Format".

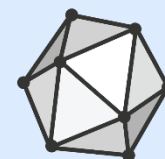
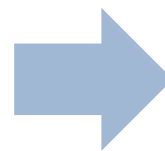
 PyTorch ^{*1}



NN model structure
*.py file



NN model weight parameter
*.pth file



ONNX
*.onnx file

^{*1} PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

MMPose onnx conversion script converts these model structure and weight parameter to ONNX format using the following PyTorch function.

```
torch.onnx.export(model, ...)
```

ONNX tutorials: <https://github.com/onnx/tutorials>

PyTorch official tutorial: https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

MMPose official tutorial: https://mmpose.readthedocs.io/en/latest/tutorials/5_export_model.html



STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

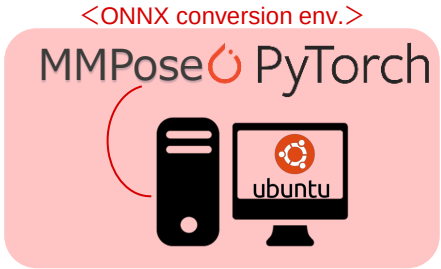
2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.4: Convert AI Model to ONNX Format



This document will use the following script to convert HRNet model provided by MMPose into ONNX format.

These files are already downloaded to MMPose environment when installing the MMPose.

This document will modify the ONNX conversion script in order to make DRP-AI Translator execution easier.

Name	Filename	Usage	Source
ONNX conversion script	pytorch2onnx.py	MMPose-ONNX conversion	Provided by MMPose
HRNet Model Structure	hrnet_w32_coco_256x192.py	MMPose-ONNX conversion	Provided by MMPose
HRNet Weight Parameter	hrnet_w32_coco_256x192-c78dce93_20200708.pth	MMPose-ONNX conversion	Provided by MMPose

STEP-1
STEP-2
2.1 Overview
2.2 Make Environment
2.3 Necessary Files
2.4 Convert to ONNX
STEP-3
STEP-4

2.4: Convert AI Model to ONNX Format



1. Prepare the ONNX conversion script.

Path: `$WORK/mmpose/tools/deployment/pytorch2onnx.py`

```
<Before>
43 ...
44 def pytorch2onnx(model,
45                 input_shape,
46                 opset_version=11,
47                 show=False,
48                 output_file='tmp.onnx',
49                 verify=False):
50 ...

66 register_extra_symbolics(opset_version)
67 torch.onnx.export(
68     model,
69     one_img,
70     output_file,
71     export_params=True,
72     keep_initializers_as_inputs=True,
73     verbose=show,
74     opset_version=opset_version)
```



```
<After>
43 ...
44 def pytorch2onnx(model,
45                 input_shape,
46                 opset_version=11,
47                 show=False,
48                 output_file='tmp.onnx',
49                 verify=False):
50 ...

66     input_names=["input1"]
67     output_names=["output1"]
68
69     register_extra_symbolics(opset_version)
70     torch.onnx.export(
71         model,
72         one_img,
73         output_file,
74         export_params=True,
75         keep_initializers_as_inputs=True,
76         verbose=show,
77         opset_version=opset_version,
78         input_names=input_names,
79         output_names=output_names)
```

Names specified here will be used in STEP-3.

input_names: Input name of first layer of the model

output_names: Output name of first layer of the model

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

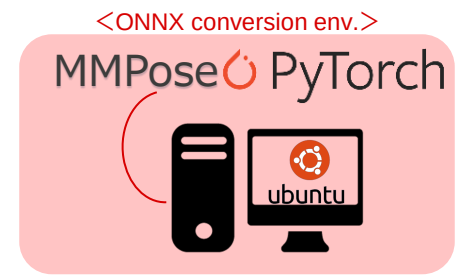
2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.4: Convert AI Model to ONNX Format



2. Register the path to model structure file as an environment variable.

```
$ export NN=configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/hrnet_w32_coco_256x192.py
```

3. Register the URL of weight file as an environment variable.

```
$ export WEIGHT=https://download.openmmlab.com/mmpose/top_down/hrnet/hrnet_w32_coco_256x192-c78dce93_20200708.pth
```

4. Register the output filename as an environment variable.

```
$ export OUTPUT=hrnet.onnx
```

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.4: Convert AI Model to ONNX Format



5. Move to the ONNX conversion working directory.

```
$ cd $WORK/mmpose
```

6. Run the onnx conversion script.

```
$ python3 tools/deployment/pytorch2onnx.py $NN $WEIGHT --opset-version 11 --shape 1 3 256 192 --output-file $OUTPUT
```

Note

Although DRP-AI Translator's supported ONNX opset version is 12, MMPose does not support opset version 12. Therefore, this document specifies opset version as 11.

If displayed as follows, the model is successfully converted.

```
Successfully exported ONNX model: hrnet.onnx
```

A warning occurs as below, but there is no problem.

```
/mmcv/cnn/bricks/transformer.py:33: UserWarning: Fail to import ``MultiScaleDeformableAttention`` from ``mmcv.ops.multi_scale_deform_attn``, You should install ``mmcv-full`` if you need this module.  
tools/deployment/pytorch2onnx.py:151: UserWarning: DeprecationWarning: This tool will be deprecated in future. Welcome to use the unified model deployment toolbox MMDeploy: https://github.com/open-mmlab/mmdelay  
/mmcv/onnx/symbolic.py:481: UserWarning: DeprecationWarning: This function will be deprecated in future. Welcome to use the unified model deployment toolbox MMDeploy: https://github.com/open-mmlab/mmdelay
```

If error occurred, please register the symbolic link as explained in "[STEP-2.2 Make the ONNX Conversion Environment](#)" and try the onnx conversion again.

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

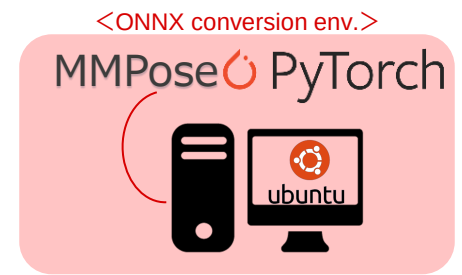
2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

2.4: Convert AI Model to ONNX Format



7. Check that *onnx* file is generated under *mmpose* directory.

```
$ ls $WORK/mmpose
```

Check the *hrnet.onnx* file is generated.

```
hrnet.onnx ...
```

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4

STEP-2 Summary

In this chapter, MMPose HRNet model has been converted to ONNX format model.

Next, we will use the converted ONNX model to run the DRP-AI Translator.

Please proceed to the next step "[STEP-3 Translate to DRP-AI Object files](#)".

STEP-1

STEP-2

2.1 Overview

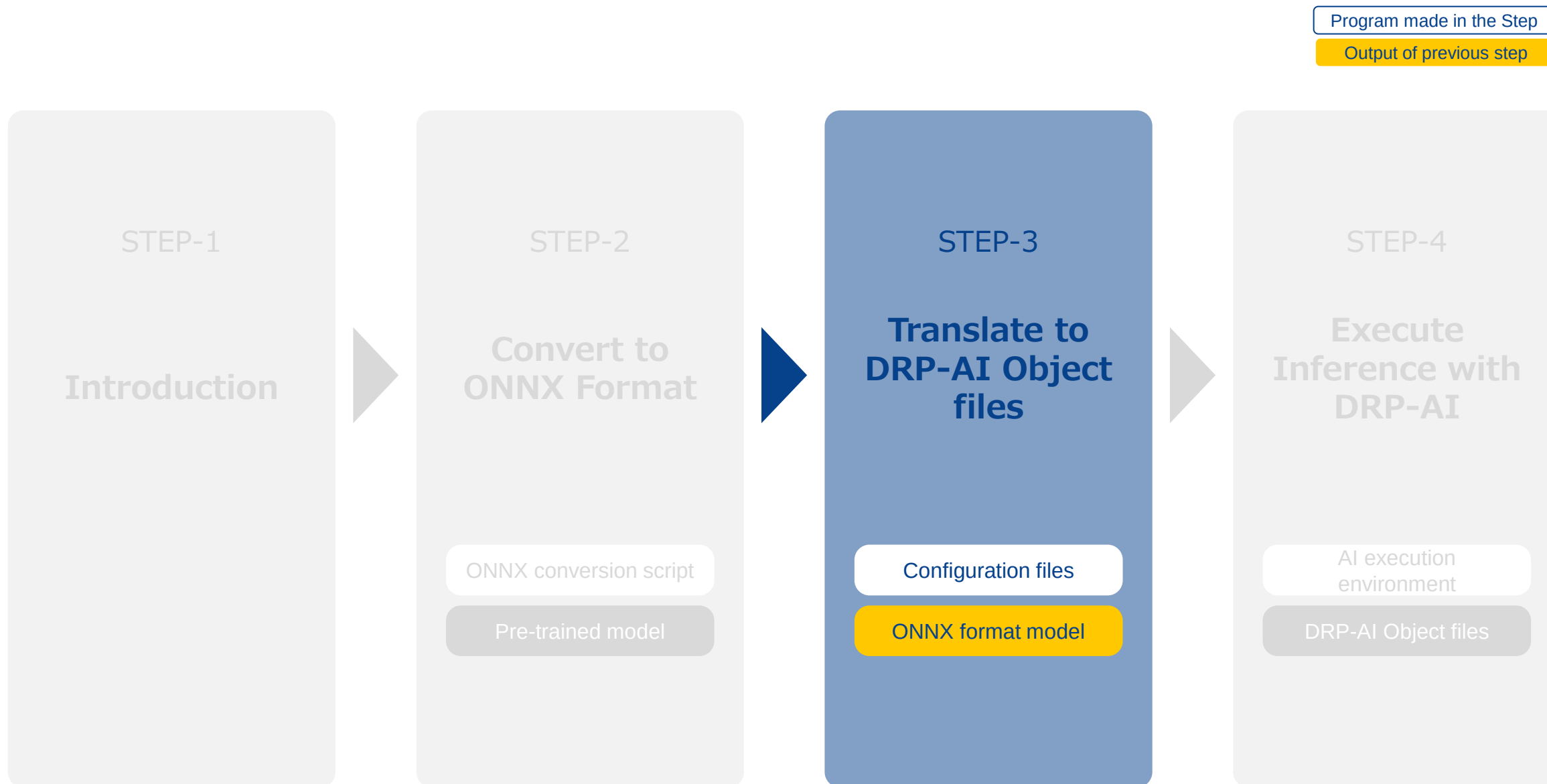
2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

STEP-4



Translate to DRP-AI Object files

This step will explain how to translate the ONNX format model created in STEP-2 to the DRP-AI Object files.

<DRP-AI Translator env.>

DRP-AI Translator



ONNX

ONNX format model
created in STEP-2

STEP-1

STEP-2

STEP-3

3.1 Make
Environment

3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.1: Make the DRP-AI Translator Environment

This section will explain the instruction with the following assumption

- Constructed the DRP-AI Translator environment according to Get Started Document "[STEP-3.1: Make the DRP-AI Translator Environment](#)".

Please check following items.

1. Confirm the environment variable for working directory is registered properly. **Green is the environment variable.**

```
$ printenv WORK
```

If displayed as follows, the variable is correctly set.

```
<path to the working directory>/rzv_ai_work
```

2. Confirm the environment variable for DRP-AI Translator working directory is registered properly.

```
$ printenv DRPAI
```

If displayed as follows, the variable is correctly set.

```
<$WORK Path>/drp-ai_translator_release
```

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

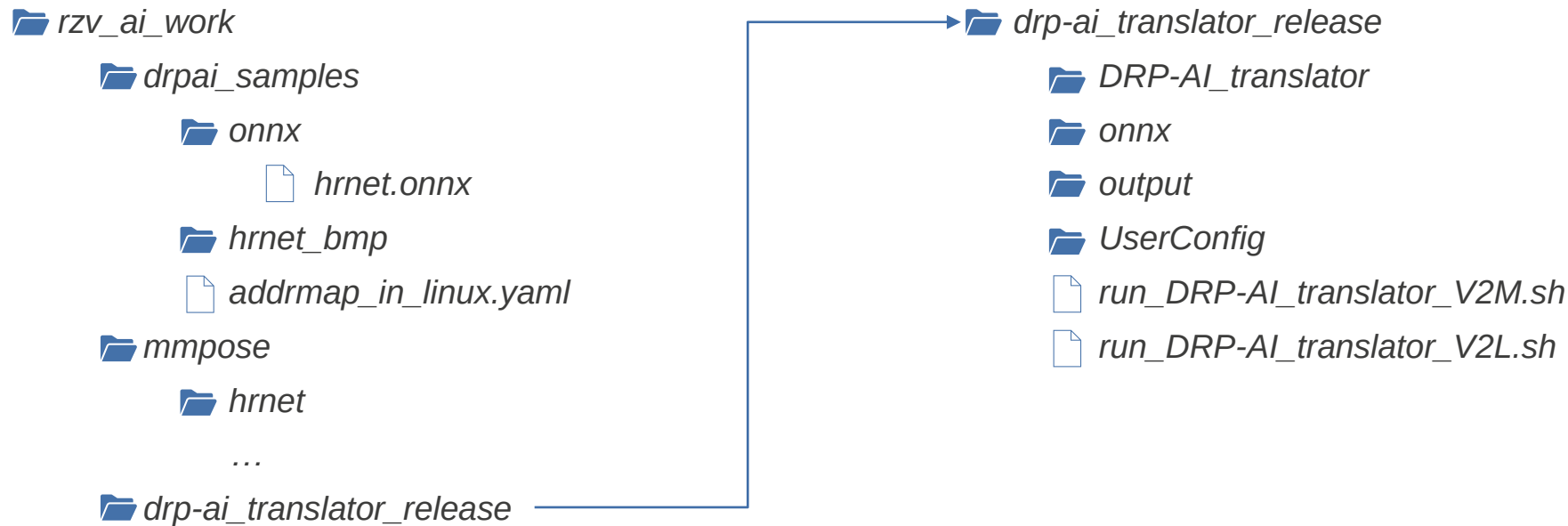
3.7 Result

STEP-4

3.2: Check the File Configuration

Please confirm that each directory configuration is as follows.

<ONNX conversion working directory (\$WORK)> <DRP-AI Translator working directory (\$DRPAI)>



STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

To see the details of DRP-AI Translator directory, please refer to the Get Started Document.

3.3: Prepare the ONNX File

In this section, we will prepare the ONNX file which is necessary for DRP-AI Translator.

```

  drp-ai_translator_release
  ...
  onnx
  UserConfig
  hrnet.onnx
  addrmap_in_hrnet.yaml
  prepost_hrnet.yaml
  
```

1. Copy the *onnx* file created in STEP-2 to the *onnx* directory.

```
$ cp -v $WORK/mmpose/hrnet.onnx $DRPAI/onnx/
```

2. Check the *onnx* directory.

```
$ ls $DRPAI/onnx/
```

Check that there is *hrnet.onnx* file.

```
hrnet.onnx  tiny_yolov2.onnx  yolov2.onnx
```

└─ These two files are sample models of DRP-AI Translator.

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

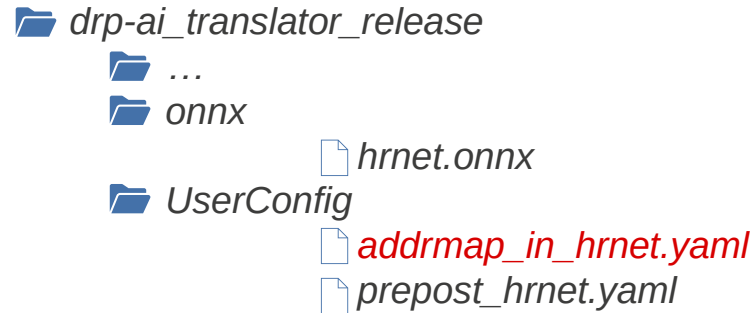
3.6 Conversion

3.7 Result

STEP-4

3.4: Prepare the Address Map Definition File

In this section, we will prepare the address map definition file which is necessary for DRP-AI Translator.



This section explains the actual commands only.

The start address need to be changed.

To see more details of the address map definition file,
 please refer to "STEP-3.4: Prepare the Address Map Definition File" in the Get Started Document.

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4



3.4: Prepare the Address Map Definition File

For address map definition file, we will use the *addrmap_in_linux.yaml* provided in *rzv_ai-implementation-guide_ver7.20.tar.gz*.

We need to rename it according to the address map definition file naming rule.

hrnet.onnx

addrmap_in_hrnet.yaml

1. Copy *addrmap_in_linux.yaml* to the *drp-ai_translator_release/UserConfig* directory.

```
$ cp -v $WORK/drpai_samples/addrmap_in_linux.yaml $DRPAI/UserConfig
```

2. Rename *addrmap_in_linux.yaml* to *addrmap_in_hrnet.yaml*.

```
$ cd $DRPAI/UserConfig
```

```
$ mv -v ./addrmap_in_linux.yaml ./addrmap_in_hrnet.yaml
```

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

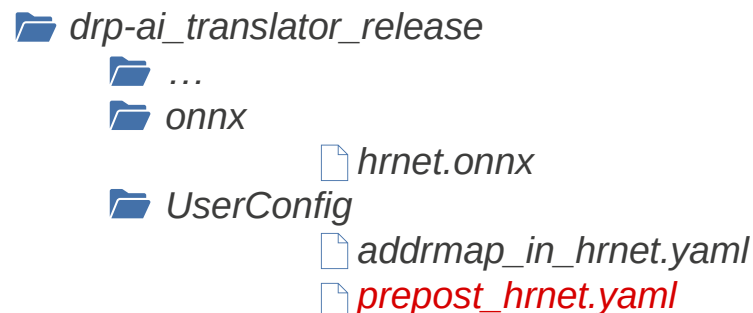
3.7 Result

STEP-4



3.5: Prepare the Pre/Postprocessing Definition File

In this section, we will prepare the pre/postprocessing definition file which is necessary for DRP-AI Translator.



This section explains the actual commands only.

To see more details of the pre/postprocessing definition file,
please refer to "[STEP-3.5: Prepare the Pre/Postprocessing Definition File](#)" in the Get Started Document.

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

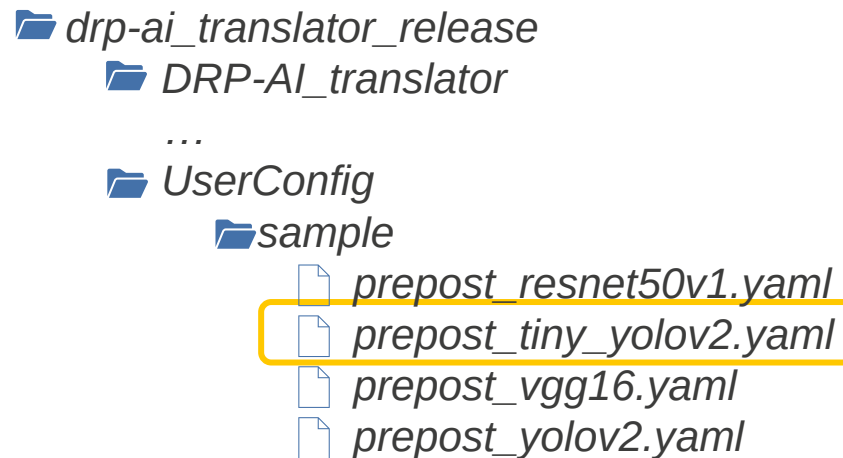
STEP-4



3.5: Prepare the Pre/Postprocessing Definition File

It is easier to make the pre/postprocessing definition file based on the sample file included in DRP-AI Translator.

The sample file is under the *UserConfig/sample* directory.



Postprocessing for MMPose HRNet is similar to the definition stated in the *prepost_tiny_yolov2.yaml*. Modify this file to prepare the pre/postprocessing definition file for MMPose HRNet.

NOTE:

Please store all customized yaml files under the *UserConfig* directory

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

1. Copy *prepost_tiny_yolov2.yaml* to the *UserConfig* directory.

```
$ cd $DRPAI/UserConfig
```

```
$ cp -v ./sample/prepost_tiny_yolov2.yaml ./
```



3.5: Prepare the Pre/Postprocessing Definition File

Since DRP-AI Translator finds the pre/postprocessing definition file based on the ONNX file name, rename the sample pre/postprocessing definition file.

hrnet.onnx
→
prepost_hrnet.yaml

2. Rename *prepost_tiny_yolov2.yaml* to *prepost_hrnet.yaml*.

```
$ cd $DRPAI/UserConfig
```

```
$ mv -v ./prepost_tiny_yolov2.yaml ./prepost_hrnet.yaml
```

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

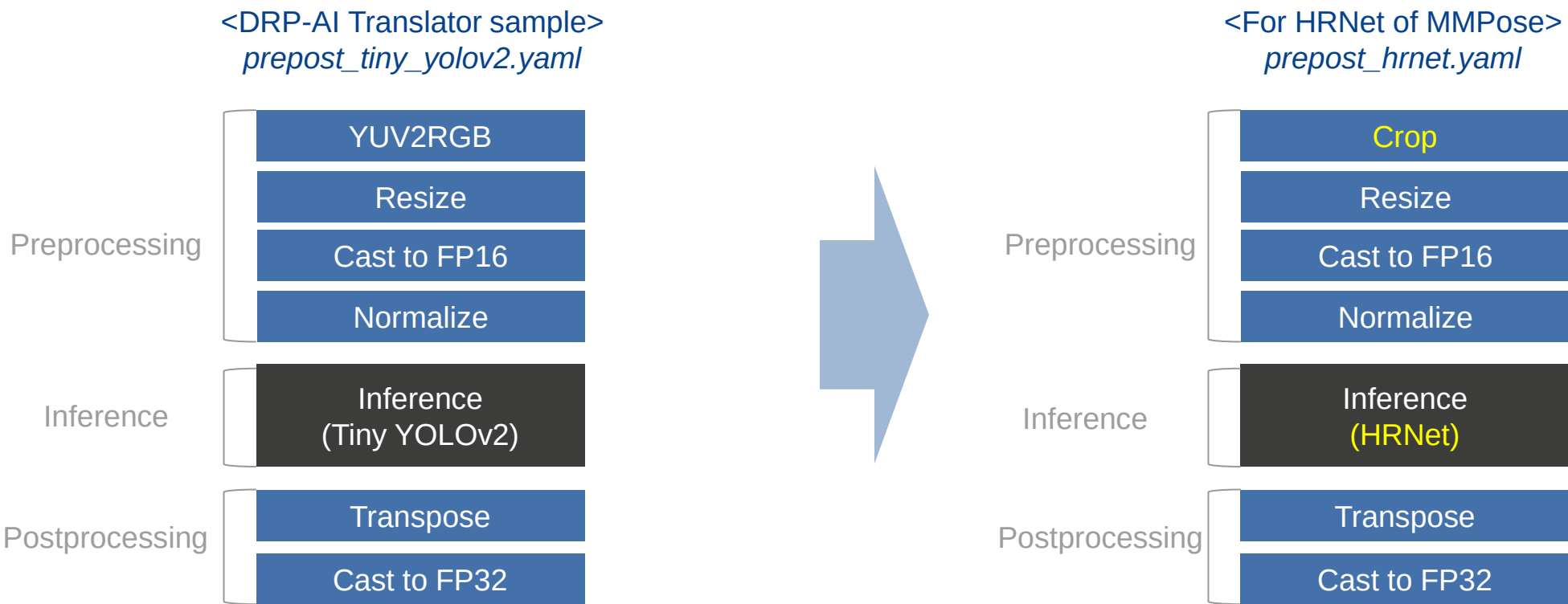
3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

prepost_tiny_yolov2.yaml included in DRP-AI Translator is defined as shown in the left figure.

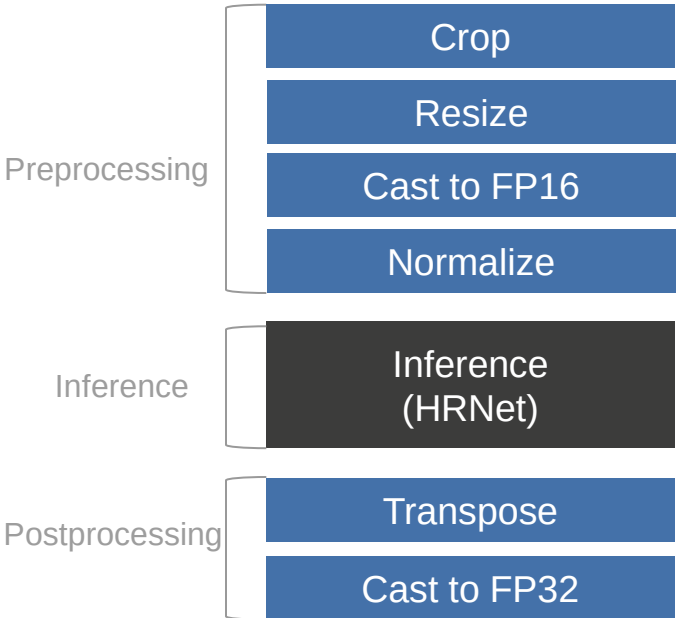
This section will explain how to rewrite the pre/postprocessing definition file for HRNet of MMPose shown in the right figure, which includes the modification of input data format from YUV to BGR.



STEP-1
STEP-2
STEP-3
3.1 Make Environment
3.2 File Configuration
3.3 ONNX File
3.4 Address Map
3.5 Pre/Post-processing
3.6 Conversion
3.7 Result
STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

<For HRNet of MMPose>
prepost_hrnet.yaml



Following items need to be rewritten to change the format based on the left figure.

- (1) Set the input/output name of the model to the same name as the input/output layer name of the model named in STEP-2.
- (2) Change the input data format for the preprocessing from YUY2 to BGR.
- (3) Set the parameters of normalize to the training value.
- (4) Change the input/output size of the model based on HRNet.
- (5) Add the crop operator.

The correspondence of rewriting information and each definition is shown below.

Definitions of <i>prepost_hrnet.yaml</i>	
Input data definition	(1), (2), (4)
Output data definition	(1), (4)
Preprocessing definition	(1), (2), (3), (5)
Postprocessing definition	(1)

STEP-1
STEP-2
STEP-3
3.1 Make Environment
3.2 File Configuration
3.3 ONNX File
3.4 Address Map
3.5 Pre/Post-processing
3.6 Conversion
3.7 Result
STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

First, set the input/output name of the model to the same name as the input/output layer name of the model named in STEP-2.

The input name to the first layer of the model and the output name from the final layer of the model were defined by *pytorch2onnx.py* in "STEP-2.4: Convert AI Model to ONNX Format".

We will set this input name (*input1*) and the output name (*output1*) in each definition.

```
<pytorch2onnx.py After Modification>
42 ...
43 def pytorch2onnx(model,
44     input_shape,
45     opset_version=11,
46     show=False,
47     output_file='tmp.onnx',
48     verify=False):
49 ...
65     input_names=["input1"]
66     output_names=["output1"]
67
68     register_extra_symbolics(opset_version)
69     torch.onnx.export(
70         model,
71         one_img,
72         output_file,
73         export_params=True,
74         keep_initializers_as_inputs=True,
75         verbose=show,
76         opset_version=opset_version,
77         input_names=input_names,
78         output_names=output_names)
```

Defined here.

- Input name : *input1*
- Output name : *output1*

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

Definitions of <i>prepost_hrnet.yaml</i>
Input data definition
Output data definition
Preprocessing definition
Postprocessing definition

3.5: Prepare the Pre/Postprocessing Definition File

3. Open the *prepost_hrnet.yaml* with the editor.

```
$ vi $DRPAI/UserConfig/prepost_hrnet.yaml
```

4. Rewrite the input name to the model in the input data definition.

<pre> 12 input_to_body: 13 - 14 name: "image2" 15 format: "RGB" 16 order: "HWC" 17 shape: [416, 416, 3] 18 type: "fp16" </pre>	Input name to model 	<pre> 12 input_to_body: 13 - 14 name: "input1" 15 format: "RGB" 16 order: "HWC" 17 shape: [416, 416, 3] 18 type: "fp16" </pre>
--	--	--

5. Rewrite the output name from the model in the output data definition.

<pre> 23 output_from_body: 24 - 25 name: "grid" 26 shape: [13, 13, 125] 27 order: "HWC" 28 type: "fp16" </pre>	Output name from model 	<pre> 23 output_from_body: 24 - 25 name: "output1" 26 shape: [13, 13, 125] 27 order: "HWC" 28 type: "fp16" </pre>
--	---	---

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

6. Rewrite the input name to the model in preprocessing definition.

```
40 preprocess:
41   -
42     src      : ["camera_data"]
43     Input name to model
44     dest     : ["image2"]
```

```
40 preprocess:
41   -
42     src      : ["camera_data"]
43
44     → dest   : ["input1"]
```

7. Rewrite the output name to the model in the postprocessing definition.

```
74 postprocess:
75   - Output name from model
76     src: ["grid"]
77
78     dest: ["post_out"]
```

```
74 postprocess:
75   -
76     → src: ["output1"]
77
78     dest: ["post_out"]
```

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

Next, change the input data format for the preprocessing from YUY2 to BGR.

8. Rewrite the input data format to the preprocessing in the input data definition.

<pre> 4 input_to_pre: 5 - 6 name: "camera_data" 7 format: "YUY2" 8 order: "HWC" 9 shape: [480, 640, 2] 10 type: "uint8" </pre>		<pre> 4 input_to_pre: 5 - 6 name: "bgr_data" 7 format: "BGR" 8 order: "HWC" 9 shape: [480, 640, 3] 10 type: "uint8" </pre>
---	--	---

Annotations for the transformation:

- Input name:** "camera_data" → "bgr_data"
- Format name:** "YUY2" → "BGR"
- Number of channels:** 2 → 3

9. Rewrite the input data to the preprocessing in the preprocessing definition.

<pre> 40 preprocess: 41 - 42 src : ["camera_data"] </pre>		<pre> 40 preprocess: 41 - 42 src : ["bgr_data"] </pre>
--	--	---

Annotation for the transformation:

- Input name:** "camera_data" → "bgr_data"

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

10. Comment out the *conv_yuv2rgb* operation (YUV to RGB conversion) in the preprocessing definition.

<pre> 47 - 48 op: conv_yuv2rgb 49 param: 50 DOUT_RGB_FORMAT: 0 # "RGB" </pre>	→	<pre> 47 # - 48 # 49 # 50 # op: conv_yuv2rgb param: DOUT_RGB_FORMAT: 0 # "RGB" </pre>
---	---	--

11. Rewrite the parameter of *normalize* operation in the preprocessing definition.

<pre> 64 - 65 op: normalize 66 param: 67 DOUT_RGB_ORDER: 0 # Output RGB order = Input RGB order 68 cof_add: [0.0, 0.0, 0.0] 69 cof_mul: [0.00392157, 0.00392157, 0.00392157] </pre>	→	<pre> 64 - 65 op: normalize 66 param: 67 DOUT_RGB_ORDER: 1 68 cof_add: [0.0, 0.0, 0.0] 69 cof_mul: [0.00392157, 0.00392157, 0.00392157] </pre>
---	---	--

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

Next, set the parameters of normalize to the training value.

Since this guide uses the pre-trained model provided by MMPose, the training value of normalization can be confirmed in the Neural Network model structure file (*hrnet_w32_coco_256x192.py*).

Path: `$WORK/mmpose/configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/hrnet_w32_coco_256x192.py`

Quotes from *hrnet_w32_coco_256x192.py*

```
104 train_pipeline = [
105     dict(type='LoadImageFromFile'),
106     dict(type='TopDownRandomFlip', flip_prob=0.5),
107     dict(
108         .
109         .
110     ),
111     dict(
112         type='NormalizeTensor',
113         mean=[0.485, 0.456, 0.406],
114         std=[0.229, 0.224, 0.225]),
115     dict(type='TopDownGenerateTarget', sigma=2),
```

Use these values, ***mean=[0.485, 0.456, 0.406]*** and ***std=[0.229, 0.224, 0.225]*** to set the configuration of DRP-AI Translator.

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

Convert the Normalize parameter, *mean*=[0.485, 0.456, 0.406] and *std*=[0.229, 0.224, 0.225] to *add* and *mul*.

To compute, use the formulae explained in

"STEP-3.5: Prepare the Pre/Postprocessing Definition File" in the Get Started Document.

$$\text{add} = - (\text{mean} \times \text{range})$$

$$\text{mul} = 1 / (\text{std} \times \text{range})$$

To normalize the input data to the model between 0 and 1 as in PyTorch, use the maximum value of the input image dynamic range as "*range*", which is 255.

12. Compute *cof_add* and *cof_mul* in the *normalize* operation.

Change the pre/postprocessing definition file based on the following result

cof_add=[-123.675, -116.28, -103.53]

cof_mul=[0.01712475, 0.017507, 0.01742919].

64	-	64	-
65	op: normalize	65	op: normalize
66	param:	66	param:
67	DOUT_RGB_ORDER: 1	67	DOUT_RGB_ORDER: 1
68	cof_add: [0.0, 0.0, 0.0]	68	cof_add: [-123.675, -116.28, -103.53]
69	cof_mul: [0.00392157, 0.00392157, 0.00392157]	69	cof_mul: [0.01712475, 0.017507, 0.01742919]

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

Next, Change the input/output size of the model based on HRNet.

Since this guide uses the pre-trained model provided by MMPose, the input/output size can be confirmed in the Neural Network model structure file (*hrnet_w32_coco_256x192.py*).

Path: `$WORK/mmpose/configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/hrnet_w32_coco_256x192.py`

Quotes from *hrnet_w32_coco_256x192.py*

```
87 data_cfg = dict(
88     image_size=[192, 256],
89     heatmap_size=[48, 64],
90     num_output_channels=channel_cfg['num_output_channels'],
```

```
29 channel_cfg = dict(
30     num_output_channels=17,
31     dataset_joints=17,
```

Input Size (HWC) = [256, 192, 3]

Output Size (HWC) = [64, 48, 17]

Use these values, **Input Size = [256, 192, 3]** and **Output Size = [64, 48, 17]**, to set the configuration of DRP-AI Translator.

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

13. Rewrite the input size to the model in the input data definition.

```
12 input_to_body:
13     -
14       name: "input1"
15       format: "RGB"
16       order: "HWC" Input size to model
17       shape: [416, 416, 3]
18       type: "fp16"
```

```
12 input_to_body:
13     -
14       name: "input1"
15       format: "RGB"
16       order: "HWC"
17       shape: [256, 192, 3]
18       type: "fp16"
```

14. Rewrite the parameters in resize operator in the preprocessing definition.

```
53     op: resize_hwc
54     param:
55         RESIZE_ALG: 1 # "Bilinear"
56         DATA_TYPE: 0 # "uint8"
57         shape_out: [416, 416] Input size to model
58
```

```
53     op: resize_hwc
54     param:
55         RESIZE_ALG: 1 # "Bilinear"
56         DATA_TYPE: 0 # "uint8"
57         shape_out: [256, 192]
58
```

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

15. Rewrite the output size from the model in output data definition.

```

23 output_from_body:
24     -           Output size from model
25       name: "grid"
26       shape: [13, 13, 125]
27       order: "HWC"
28       type: "fp16"

```

```

23 output_from_body:
24     -
25       name: "output1"
26       shape: [64, 48, 17]
27       order: "HWC"
28       type: "fp16"

```

16. Rewrite the output size from the postprocessing in output data definition.

```

30 output_from_post:   Output size from
31     -               postprocessing
32       name: "post_out"
33       shape: [125, 13, 13]
34       order: "CHW"
35       type: "fp32"

```

```

30 output_from_post:
31     -
32       name: "post_out"
33       shape: [17, 64, 48]
34       order: "CHW"
35       type: "fp32"

```

STEP-1

STEP-2

STEP-3

3.1 Make
Environment

3.2 File
Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4

3.5: Prepare the Pre/Postprocessing Definition File

Last, **Add the crop operator.**

In this guide, the input image is size of width 640px and height 480px.

However, input data size to HRNet model must be width 192px and height 256px, which is vertical image.

Resizing the original image will change the aspect ratio significantly and change the recognition result.

In this document, we will crop the image and resize to fit the input data size to HRNet model.

STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

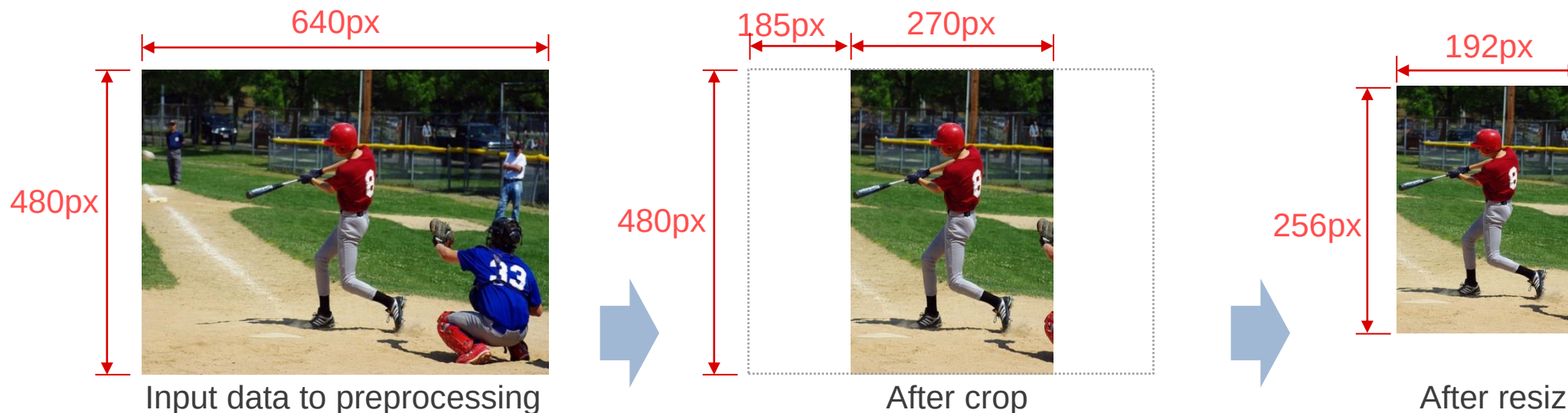
3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4



Definitions of <i>prepost_hrnet.yaml</i>
Input data definition
Output data definition
Preprocessing definition
Postprocessing definition

3.5: Prepare the Pre/Postprocessing Definition File

17. Add the crop operator in preprocessing definition.

```

48 #      op: conv_yuv2rgb
49 #      param:
50 #      DOUT_RGB_FORMAT: 0 # "RGB"
51 -
52      op: resize_hwc Add crop operator
53
54
```

```

48 #      op: conv_yuv2rgb
49 #      param:
50 #      DOUT_RGB_FORMAT: 0 # "RGB"
51 -
52      op: crop
53      param:
54          CROP_POS_X : 185
55          CROP_POS_Y : 0
56          DATA_TYPE : 0
57          DATA_FORMAT: 0 # 0 : HWC
58          shape_out: [480, 270] # [H, W]
59 -
60
61      op: resize_hwc

```

Major parameters are as follows.

param > CROP_POS_X : X coordinate where start cropping (Top-left)

param > CROP_POS_Y : Y coordinate where start cropping (Top-left)

param > DATA_TYPE : Data type ("0" if uint8)

param > DATA_FORMAT : Data format("0" if HWC)

param > shape_out : Output data size from crop operator

For more details of crop operator, please refer to DRP-AI Translator User's Manual.

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

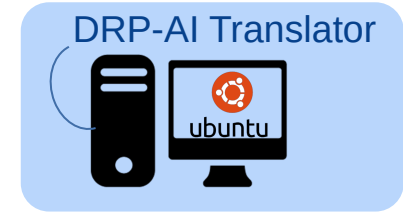
3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

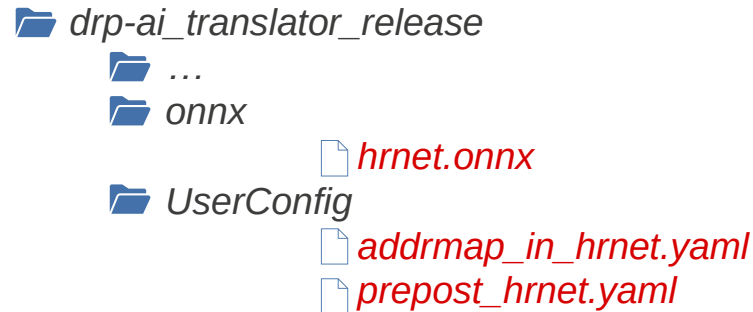
STEP-4

3.6: Translate the Model Using DRP-AI Translator



This section will explain how to run the DRP-AI Translator.

1. please confirm that there are following three files under *drp-ai_translator_release* directory.



STEP-1

STEP-2

STEP-3

3.1 Make
Environment3.2 File
Configuration

3.3 ONNX File

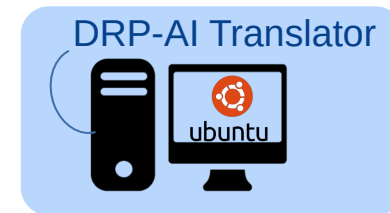
3.4 Address Map

3.5 Pre/Post-
processing

3.6 Conversion

3.7 Result

STEP-4



3.6: Translate the Model Using DRP-AI Translator

2. Move to the DRP-AI Translator working directory.

```
$ cd $DRPAI
```

3. Translate the model with the following commands. (Please execute it under the *drp-ai_translator_release* directory.)

Blue is PREFIX (output directory) name. Any name is available.

For RZ/V2M, RZ/V2MA:

```
$ ./run_DRP-AI_translator_V2M.sh hrnet -onnx ./onnx/hrnet.onnx
```

For RZ/V2L:

```
$ ./run_DRP-AI_translator_V2L.sh hrnet -onnx ./onnx/hrnet.onnx
```

If displayed as follows without any errors, translation is successful.

```
[Run DRP-AI Translator] Ver. 1.80
[Input file information]
PREFIX                : hrnet
ONNX Model             : ./onnx/hrnet.onnx
Prepost file           : ./UserConfig/prepost_hrnet.yaml
Address mapping file   : ./UserConfig/addrmap_in_hrnet.yaml
...
```

```
-----
[Converter for DRP] Finish
```

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

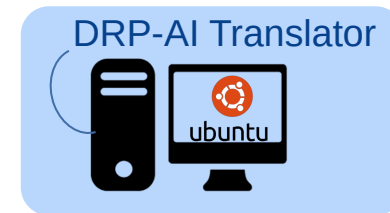
3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4



3.7: Confirm the Translation Result

1. The translation result is stored in the output directory with **PREFIX** named at the time of translation.

```
$ ls -l $DRPAI/output/hrnet/
```

If displayed as follows, the model is correctly translated.

```
aimac_desc.bin
drp_desc.bin
drp_lib_info.txt
drp_param.bin
drp_param.txt
drp_param_info.txt
hrnet.json
hrnet_addrmap_intm.txt
hrnet_addrmap_intm.yaml
hrnet_data_in_list.txt
hrnet_data_out_list.txt
hrnet_drpcfg.mem
hrnet_prepost_opt.yaml
hrnet_summary.xlsx
hrnet_tbl_addr_data.txt
hrnet_tbl_addr_data_in.txt
hrnet_tbl_addr_data_out.txt
hrnet_tbl_addr_drp_config.txt
hrnet_tbl_addr_merge.txt
hrnet_tbl_addr_weight.txt
hrnet_tbl_addr_work.txt
hrnet_weight.dat
```

Yellow files are DRP-AI Object files required for actual operation.

Green file is a summary file for estimating the processing time.

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

STEP-3 Summary

In STEP-3, DRP-AI Object files are generated from the ONNX model.

Please proceed to "[STEP-4 Execute Inference with DRP-AI](#)".

STEP-1

STEP-2

STEP-3

3.1 Make Environment

3.2 File Configuration

3.3 ONNX File

3.4 Address Map

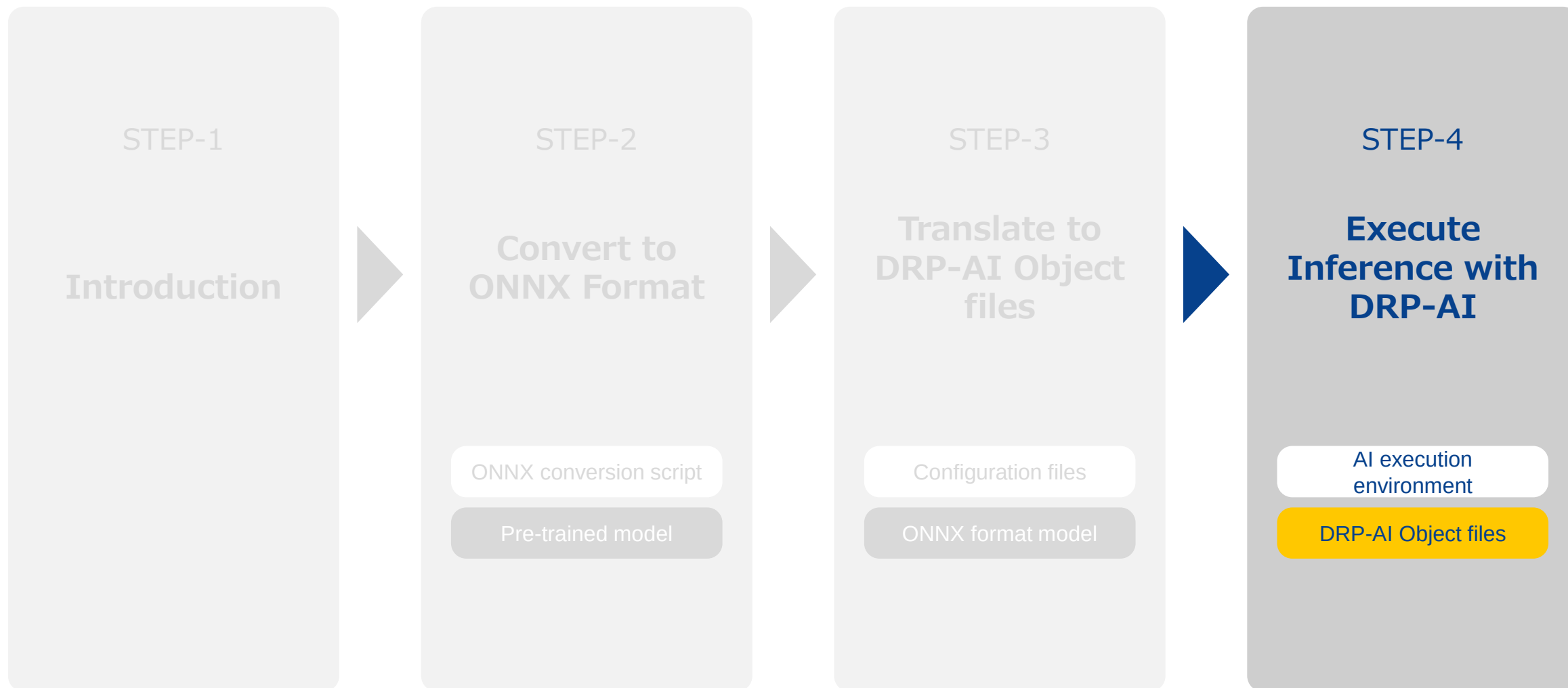
3.5 Pre/Post-processing

3.6 Conversion

3.7 Result

STEP-4

Program made in the Step
Output of previous step



Execute Inference with DRP-AI

This section will explain the instruction with the following assumption

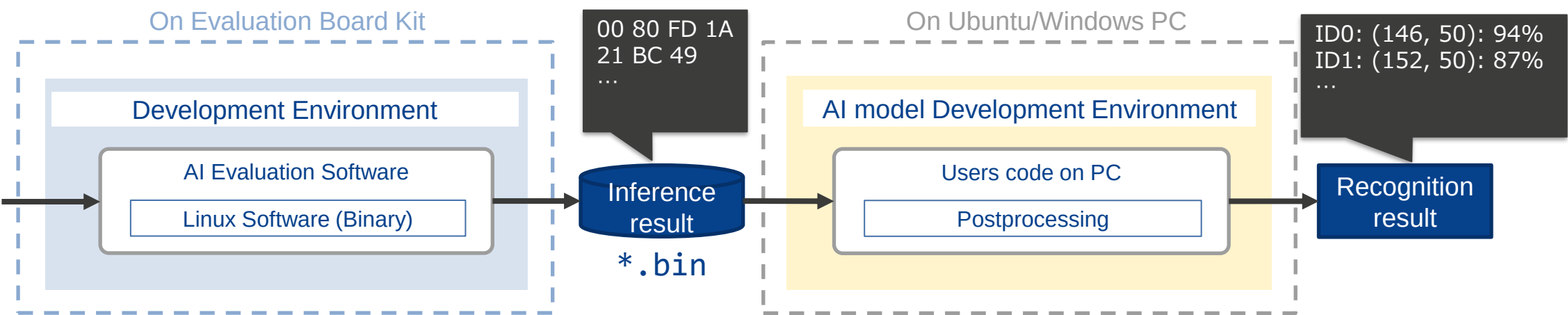
- Read "STEP-4 Execute Inference with DRP-AI" in the Get Started Document.

STEP-1
STEP-2
STEP-3
STEP-4

The DRP-AI Object files for HRNet created in STEP-3 can be executed by AI Evaluation Software.

AI Evaluation Software will generate a binary file that contains the DRP-AI inference result, which is not the recognition result.

To obtain recognition result, we need to apply the CPU post-processing specialized for HRNet.



Note: To see how to use AI Evaluation Software, please refer to AI Evaluation Software Guide.

Execute Inference with DRP-AI

This document provides the post-processing script for MMPose HRNet, which runs on Linux PC.

Path: `$WORK/mmpose/hrnet/postprocess_hrnet.py`

Details of the script are as follow.

STEP-1
STEP-2
STEP-3
STEP-4

Notes

- ✓ Assumed to have run following post-processing on DRP-AI
 - transpose
 - castFP16toFP32
- ✓ Install the necessary packages listed in Confirmed Operational Environment below appropriately.
- ✓ To see the details of algorithm, refer to HRNet paper (<https://arxiv.org/pdf/1902.09212.pdf>).
- ✓ The script uses the AI Evaluation Software output of `$WORK/mmpose/hrnet/sample.bmp` image.
 - ✓ The script draws the estimated skeleton on the input image. Please specify the name of input image used when executing the AI Evaluation Software.

Confirmed Operational Environment

Ubuntu 20.04 LTS	Python	== 3.8.10
	pip	== 22.2.2
	torch	== 1.12.1+cpu
	numpy	== 1.23.1
	opencv-python	== 4.6.0.66
	mmcv	== 1.6.1

DRP-AI Inference Result Binary Data

Details of the DRP-AI inference result binary file are as follow.

- Number of data : 52224 (Depends on the model output size. e.g., (1x17x64x48))
- Data width : 4byte (Because of castFP16toFP32, width is FP32=4byte)
- Byte order : Little endian

Execute Inference with DRP-AI

From `postprocess_hrnet.py`

154	...		
155	if __name__ == '__main__':		
156	output_shape = [1, 17, 64, 48] # [N, C, H, W]		
157	input_size_x = 192 # Width of model input		
158	input_size_y = 256 # Height of model input		
159	# Drawing parameters		
160	kpt_score_thr=0.3		
161	thickness=1		
162	radius=4		
163			
164	# dataset: 'TopDownCocoDataset'		
165	palette = np.array([[255, 128, 0], [255, 153, 51], [255, 178, 102],]		
166	...		
182	# Label from COCO dataset "https://arxiv.org/abs/1405.0312"		
183	label = {		
184	...		
202	# Load DRP-AI output binary		
203	result_bin = open("sample.bmp.bin", 'rb')		
204	data = np.zeros(output_shape, dtype=float)		
205	for c in range(output_shape[1]):		
206	for h in range(output_shape[2]):		
207	for w in range(output_shape[3]):		
208	a = struct.unpack('<f', result_bin.read(4))		
209	data[0, c, h, w] = a[0]		
210	...		
211	# Postprocess		
212	...		
234	# Image processing		
235	img = mmcv.imread("sample.bmp")		
236	...		
261	# Print out result		
262	for i in range(len(all_preds[0])):		
263	print('ID {0:2} {1:14}: ({2:3.0f}, {3:3.0f}): {4:5.1%}'.format(i, label[i], all_preds[0,i,0],all_preds[0,i,1],all_preds[0,i,2]))		
264	...		
265	# Draw skeleton		
266	imshow_keypoints(img, all_preds, skeleton, kpt_score_thr,		
267	...		
269	imwrite(img, "result.jpg")		

STEP-1

STEP-2

STEP-3

STEP-4

Input/Output size of model

Drawing Parameters

Drawing parameters
for COCO Dataset

Label for key point (joint)

Load inference result binary file

Read FP32 data for [64x48x17]
Specify little endian

Load input image for drawing

Print the postprocessing
result on console

Draw skeleton

Save skeleton image

Execute Inference with DRP-AI

Running the `postprocess_hrnet.py` with sample input image (`$WORK/mmpose/hrnet/sample.bmp`) will generate following results.

- STEP-1
- STEP-2
- STEP-3
- STEP-4

Terminal Log

ID	0	nose	:	(82, 66):	92.8%
ID	1	left_eye	:	(82, 64):	94.8%
ID	2	right_eye	:	(82, 62):	85.4%
ID	3	left_ear	:	(92, 64):	92.1%
ID	4	right_ear	:	(100, 60):	72.1%
ID	5	left_shoulder	:	(96, 84):	85.0%
ID	6	right_shoulder	:	(120, 68):	87.4%
ID	7	left_elbow	:	(76, 100):	91.0%
ID	8	right_elbow	:	(88, 80):	71.7%
ID	9	left_wrist	:	(56, 90):	91.9%
ID	10	right_wrist	:	(64, 84):	81.1%
ID	11	left_hip	:	(102, 128):	76.9%
ID	12	right_hip	:	(112, 124):	63.3%
ID	13	left_knee	:	(98, 176):	92.0%
ID	14	right_knee	:	(66, 158):	88.1%
ID	15	left_ankle	:	(146, 194):	87.7%
ID	16	right_ankle	:	(72, 204):	87.4%

(X, Y) Score

Input image(sample.bmp)



Output image (result.jpg)



STEP-4 Summary

STEP-4 explained how to execute the AI inference using DRP-AI.

In the DRP-AI Sample Application, we provide the sample application that runs inference through CPU post-processing on the board with MMPose HRNet model, which is explained in this document.

To see how to use the application, please refer to the DPR-AI Sample Application Note.

STEP-1

STEP-2

STEP-3

STEP-4

Conclusion

Throughout STEP-1 to STEP-4, we have explained how to run the HRNet model provided by MMPose on the RZ/V2x.

If you have any questions, please contact us.

Thank you for reading to the end.

Renesas.com

Version History

Date	Version	Chapter	Contents
Sep. 29, 2022	7.20	-	Issued. (Unified AI Implementation Guide for RZ/V2L, RZ/V2M, RZ/V2MA)