RZ/V2L, RZ/V2M, RZ/V2MA
AI IMPLEMENTATION GUIDE
Darknet YOLO

REV.7.20

SEPTEMBER 2022

RENESAS ELECTRONICS CORPORATION

R11AN0620EJ0720

BIG IDEAS FOR EVERY SPACE RENESAS

Overview

This document explains the contents of AI Implementation Guide Get Started document with YOLOv3, YOLOv2, Tiny YOLOv3 and Tiny YOLOv2 pre-trained model provided by Darknet framework (hereafter, Darknet YOLO).

Please read Get Started document in advance.

This document uses following documents and files.

Name	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 Darknet YOLO	r11an0620ej0720-rzv-ai-imp-yolo.pdf	This document. Document for guiding the instruction for Darknet YOLO model.
Source Code for Darknet YOLO	darknet_yolo_ver7.20.tar.gz	Source code and example output used in the Document for Darknet YOLO.

Flow of Guide

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

Table of Contents

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

YOLO

YOLO (**Y**ou **o**nly **l**ook **o**nce) is an image recognition neural network for the "Object detection"

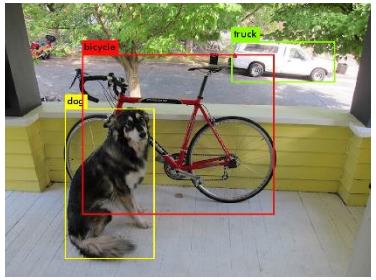
It can recognize what is in the image (Classification) and where it is in the image (Localization).

This document uses YOLOv3, YOLOv2, Tiny YOLOv3 and Tiny YOLOv2 models as examples.

Input Image

AI Processing

- Pre-processing
  - Resize/Crop
  - Normalize
- Inference (YOLO)
- Post-processing
  - Bounding_box
  - Sort
  - Labeling

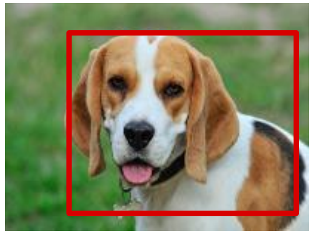


Results

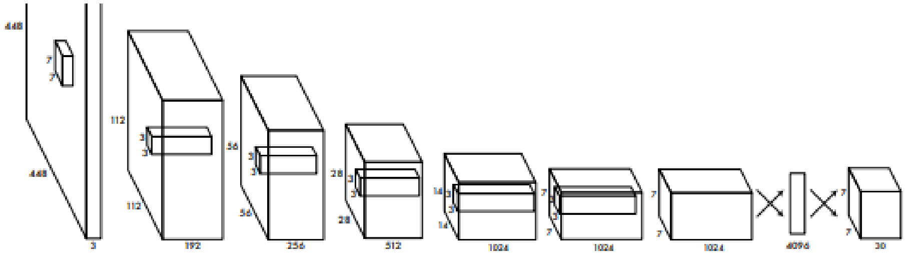
Classification

DOG

Classification + Localization



DOG



(Cites) You Only Look Once: Unified, Real-Time Object Detection
https://arxiv.org/pdf/1506.02640.pdf

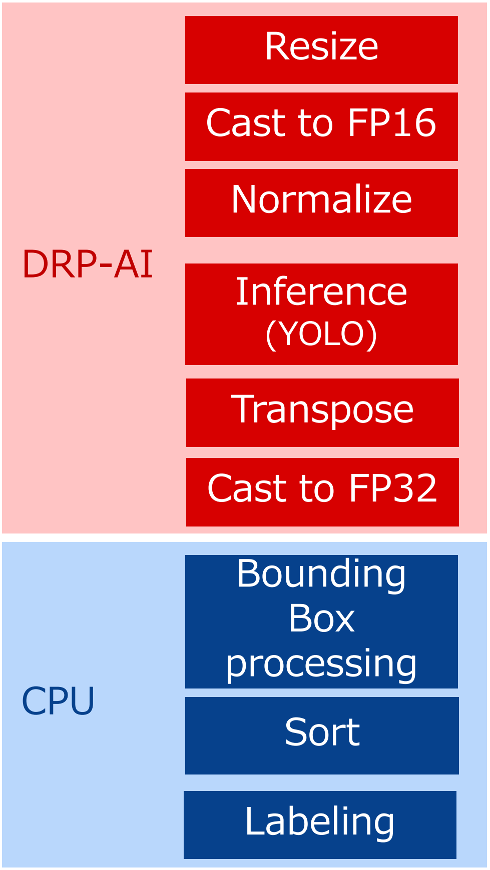
(Paper) You Only Look Once: Unified, Real-Time Object Detection https://arxiv.org/pdf/1506.02640.pdf
(Paper) YOLOv3: An Incremental Improvement https://arxiv.org/pdf/1804.02767.pdf
(Paper) YOLO9000 Better, Faster, Stronger https://arxiv.org/pdf/1612.08242.pdf
(Reference) YOLOv3: Real-Time Object Detection (Joseph Redmon) https://pjreddie.com/darknet/yolo/
(Reference) YOLOv2: Real-Time Object Detection (Joseph Redmon) https://pjreddie.com/darknet/yolov2/

[Reference] YOLO on DRP-AI

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

Following is the example of YOLO 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.



<Pre-processing>
Resize : Input image size(640x480 BGR) to HWC tensor size(416x416x3).
Cast to fp16 : Cast to fp16 from int8 for DRP-AI processing.
Normalize: Normalized using 'mean' and 'std'.

<Post-processing>
Transpose: HWC to CHW.
Cast to fp32

Darknet YOLO Transpose Shape

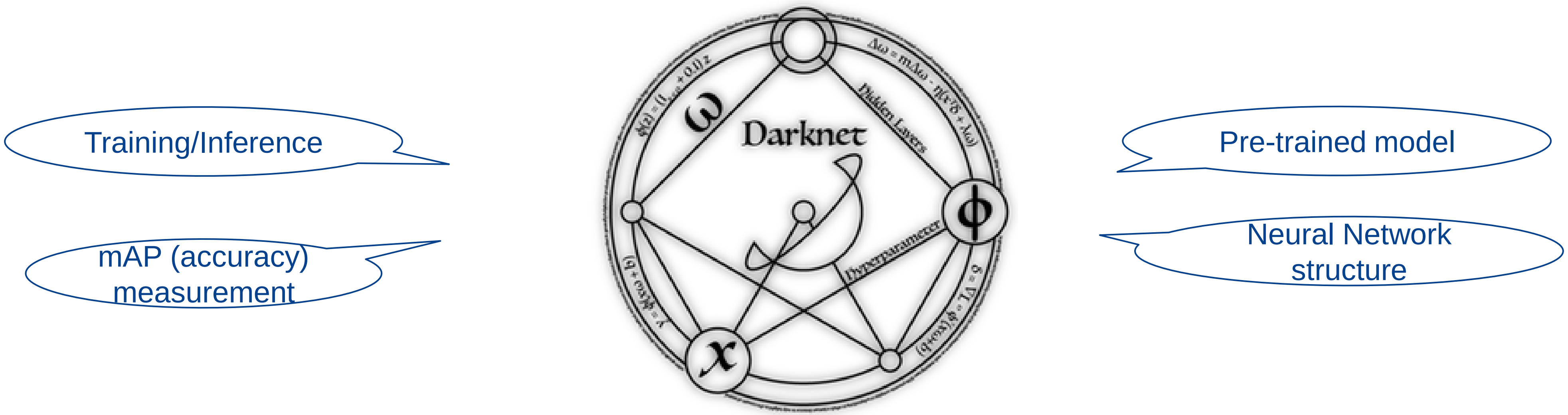
Model	HWC	CHW
YOLOv3	[13, 13, 255] [26, 26, 255] [52, 52, 255]	[255, 13, 13] [255, 26, 26] [255, 52, 52]
YOLOv2	[13, 13, 125]	[125, 13, 13]
Tiny YOLOv3	[13, 13, 255] [26, 26, 255]	[255, 13, 13] [255, 26, 26]
Tiny YOLOv2	[13, 13, 125]	[125, 13, 13]

[Reference] Darknet

This document uses pre-trained model provided by Darknet framework.

Darknet implements the YOLO network by C language and has not only the training functions and inference functions, but also the pre-trained model and mAP (accuracy) functions.

For more details of Darknet, please refer to the Darknet official site (https://pjreddie.com/darknet/yolo).



However, Darknet does not contain ONNX conversion function. Therefore, this guide will prepare the ONNX format model for the DRP-AI Translator from the Darknet Pascal VOC Dataset pre-trained model by using the PyTorch ONNX conversion functionality.

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

[Reference] PyTorch

Since PyTorch has the pre-trained model and the onnx conversion function (torch.onnx) as well as the training functions, it is easy to convert the PyTorch model into the ONNX format model.

Please refer to the PyTorch official document (https://pytorch.org/docs/1.12/) to learn more about the various features of PyTorch.

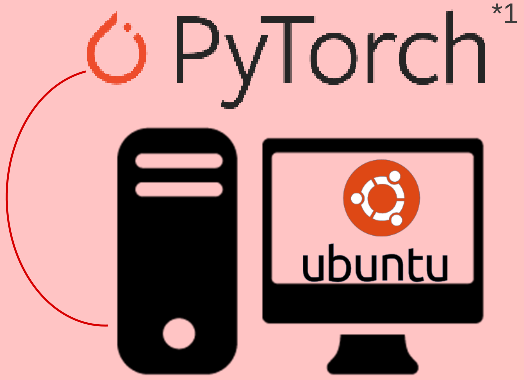
Necessary Environment

Following are the necessary environment for each STEP.

See the Get Started Document for how to build the environment.

Note: This document uses Darknet pre-trained model, which does not require Darknet framework.
Instead, we use PyTorch framework for ONNX conversion.

<ONNX conversion env.>



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

*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 darknet_yolo_ver7.20.tar.gz.

- darknet_yolo_ver7.20
  - drpai_samples
    - onnx
    - yolov3_bmp
    - yolov2_bmp
    - tinyyolov3_bmp
    - tinyyolov2_bmp
  - darknet
    - yolo

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

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

[Additional Information] YOLO Training Environment

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

Please refer to the Darknet Official Website (https://pjreddie.com/darknet/yolo/) to see how to train the model by own dataset, or how to customize the neural network.

<Training Env.>

STEP-1 Summary

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

Please proceed to "STEP-2 Convert to ONNX Format".

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

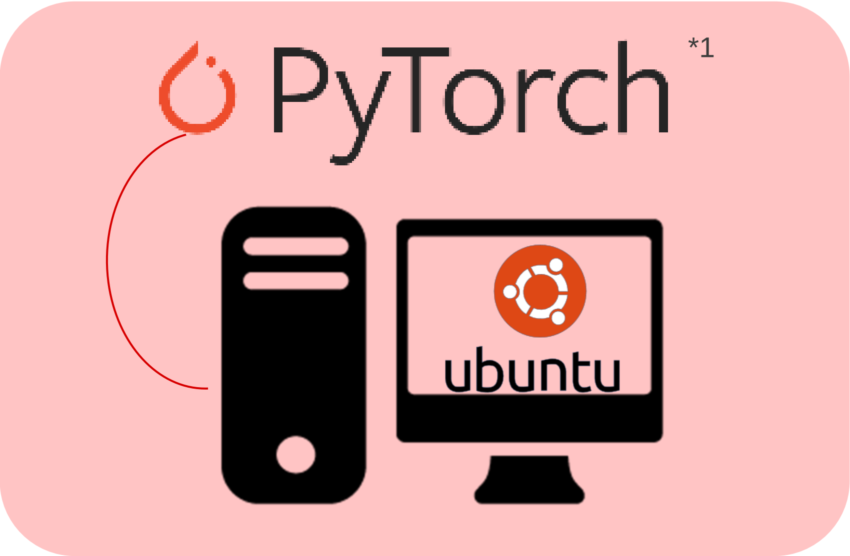
2.1: Convert to the ONNX Format

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

This chapter will explain the contents of Get Started Document "STEP-2 Convert to the ONNX Format" with YOLO model provided by Darknet framework.

Since Darknet does not have ONNX conversion function, this document will convert Darknet into PyTorch format first, and then convert it to the ONNX format.

＜ONNX conversion env.＞



Pre-trained YOLO model

Darknet YOLO Model:
https://pjreddie.com/darknet/yolo/

\+

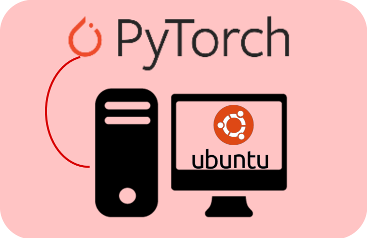
PyTorch *1 ONNX

Convert from PyTorch to ONNX

ONNX tutorials:
https://github.com/onnx/tutorials
PyTorch official tutorial:
https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

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

<ONNX conversion env.>



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
```

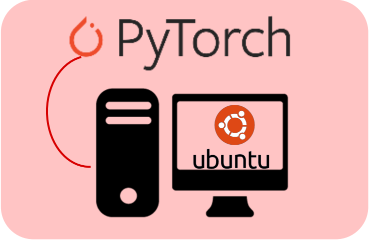
If displayed as follows, the variable is correctly set.

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

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

<ONNX conversion env.>



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>/darknet_yolo_ver7.20.tar.gz -C $WORK
```

3. Check the working directory.

```
$ ls $WORK
```

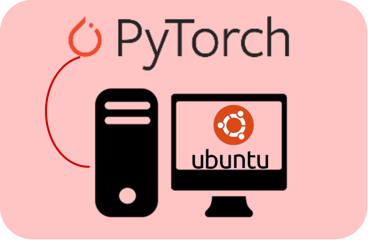
If displayed as follows, the package is correctly extracted.

```
drpai_samples    darknet
```

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

"darknet" includes PyTorch sample code for Darknet model.

<ONNX conversion env.>

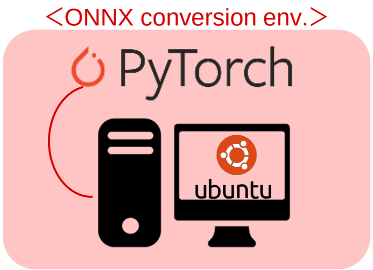


2.3: Prepare the Necessary Files

Please confirm that each directory configuration is as follows.

<ONNX conversion working directory ($WORK)>

- *rzv_ai_work*
  - *drpai_samples*
    - *onnx*
      - *d-yolov3.onnx*
      - *d-yolov2.onnx*
      - *d-tinyyolov3.onnx*
      - *d-tinyyolov2.onnx*
    - *yolov3_bmp*
    - *yolov2_bmp*
    - *tinyyolov3_bmp*
    - *tinyyolov2_bmp*
    - *addrmap_in_linux.yaml*
  - *darknet*
    - *yolo*
      - *convert_to_onnx.py*
      - *convert_to_pytorch.py*
      - …
      - *postprocess_yolo.py*



2.4: Convert AI Model to ONNX Format

File format of Neural Network model differs depending on Deep Learning framework.

Since Darknet does not have ONNX conversion function, this document will convert Darknet into PyTorch format first, and then convert it to the ONNX format.

The flow of ONNX conversion for Darknet Neural Network model will be as follows.

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

Darknet provides pre-trained Neural Network structure and its weight parameter.

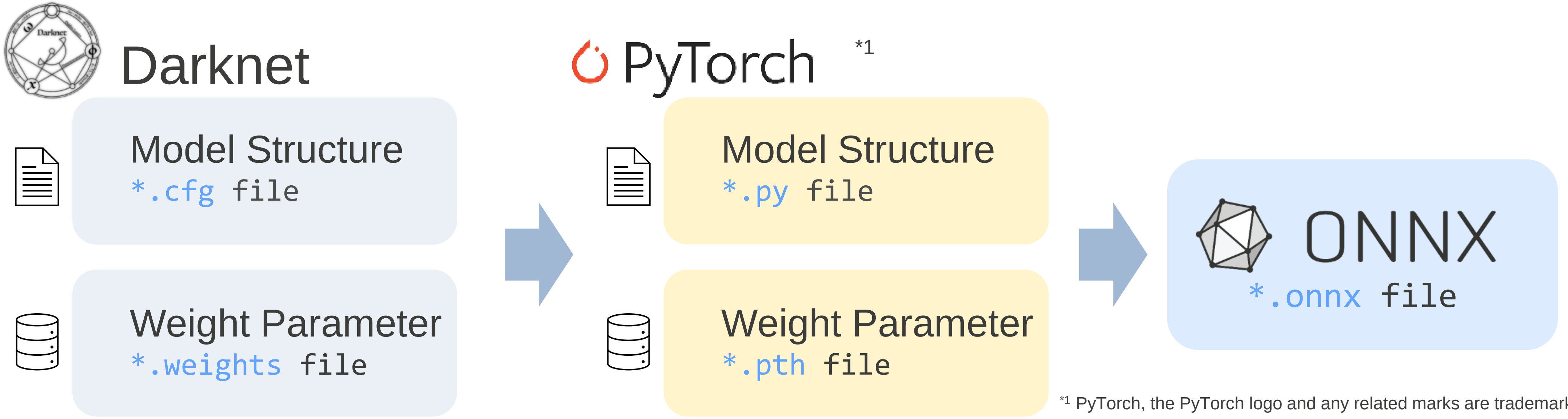
We will convert these files to PyTorch format, and then convert it to ONNX format.

<ONNX conversion env.>
PyTorch
ubuntu

2.4: Convert AI Model to ONNX Format

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

As explained in the Get Started Document "STEP-2.4: Convert AI Model to ONNX Format", to convert AI model to ONNX format, NN model structure and its weight parameter are required.



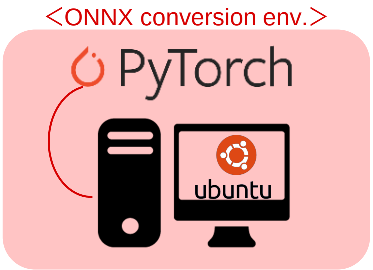
*1 PyTorch, the PyTorch logo and any related marks are trademarks of Facebook, Inc.

This document will convert Darknet model structure and weight parameter to PyTorch format and convert it to ONNX format using the following PyTorch function.

```
torch.onnx.export(model, …)
```

For more details, please refer to the PyTorch and ONNX official website.

ONNX tutorials: https://github.com/onnx/tutorials
PyTorch official tutorial:
https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html



2.4: Convert AI Model to ONNX Format

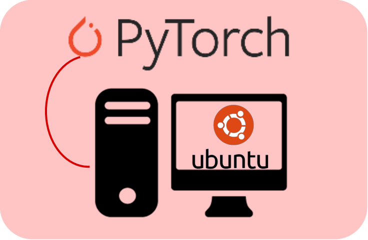
This guide will use the following scripts to convert YOLO model provided by Darknet into ONNX format.

Note: All scripts are included in the darknet_yolo_ver7.20.tar.gz.

Name	Filename	Source	Usage
Darknet YOLO model structure and configuration	*.cfg	Darknet yolov3: https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg yolov2: https://github.com/pjreddie/darknet/blob/master/cfg/yolov2-voc.cfg tinyyolov3: https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg tinyyolov2: https://github.com/pjreddie/darknet/blob/master/cfg/yolov2-tiny-voc.cfg	Darknet-PyTorch conversion
Darknet YOLO model parameter (weight)	*.weights	Darknet yolov3: https://pjreddie.com/media/files/yolov3.weights yolov2: https://pjreddie.com/media/files/yolov2-voc.weights tinyyolov3: https://pjreddie.com/media/files/yolov3-tiny.weights tinyyolov2: https://pjreddie.com/media/files/yolov2-tiny-voc.weights	
Darknet cfg file parser	darknet_cfg.py	Provided by Renesas	
Darknet-PyTorch conversion script	convert_to_pytorch.py	Provided by Renesas	
PyTorch YOLO model structure	yolo.py	Provided by Renesas Reference: PyTorch-YOLOv3 https://github.com/eriklindernoren/PyTorch-YOLOv3	Darknet-PyTorch conversion PyTorch-ONNX conversion
Conversion configuration file	yolo.ini	Provided by Renesas	
Conversion configuration file parser	read_ini.py	Provided by Renesas	
PyTorch-ONNX conversion script	convert_to_onnx.py	Provided by Renesas	PyTorch-ONNX conversion

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

<ONNX conversion env.>



2.4: Convert AI Model to ONNX Format

When running the conversion script, please specify the name of model as shown below with BLUE.

Note: If not specified, YOLOv3 model will be selected by default.

```
$ python3 convert_to_*.py yolov3
```

Parameter	Model
yolov3	YOLOv3 (Default)
yolov2	YOLOv2
tinyyolov3	Tiny YOLOv3
tinyyolov2	Tiny YOLOv2

The conversion script load the filename and parameters listed in *yolo.ini* file and use them in the conversion. Please write the parameters to the *yolo.ini* file as shown below.

Parameter	Details
cfg	Darknet YOLO model structure and configuration filename (*.cfg)
weights	Darknet YOLO model parameter (weight) filename (*.weights)
pth	Intermediate PyTorch format weight filename. Specify any arbitrary name.
input	Input layer name of ONNX model (Will be used in STEP-3)
output	Output layer name of ONNX model (Will be used in STEP-3)
onnx	ONNX model filename, which will be generated after conversion. Specify any arbitrary name.

<yolo.ini>

```
[yolov3]
cfg     =yolov3.cfg
weights =yolov3.weights
pth     =yolov3.pth
input   =["input1"]
output  =["output1", "output2", "output3"]
onnx    =d-yolov3.onnx

[yolov2]
cfg     =yolov2-voc.cfg
weights =yolov2-voc.weights
 :
```


BIG IDEAS FOR EVERY SPACE RENESAS

2.4: Convert AI Model to ONNX Format

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

1. Convert from Darknet to PyTorch

   2. Prepare the Darknet-PyTorch conversion script. If STEP-2.3 are executed, the script is in the following location

      Path: *$WORK/darknet/yolo/convert_to_pytorch.py*

```
<convert_to_pytorch.py>
    ⋮
def convert(darknet_cfg_path, darknet_weights_path, save_model_path, model):
    # Load cfg file
    cfg = DarknetConfig(darknet_cfg_path)
    ⋮
    # Save pth file
    torch.save(model.state_dict(), save_model_path)

if __name__ == '__main__':
    # Load YOLO neural network parameters
    ini = IniFile("yolo.ini")
    model_dict = ini.model_dict
    ⋮
    # Get the model information
    model_params = ini.architecture[model_dict[model_name]].params

    # Load YOLO neural network structure
    model = yolo.Yolo(model_params.get('cfg'))
    # Convert Darknet to PyTorch
    convert(model_params.get('cfg'), model_params.get('weights'), model_params.get('pth'), model)
```

Get the model structure

Model structure & configuration

Model parameter (weight)

pth filename after the conversion

STEP-1

STEP-2

2.1 Overview

2.2 Make Environment

2.3 Necessary Files

2.4 Convert to ONNX

STEP-3

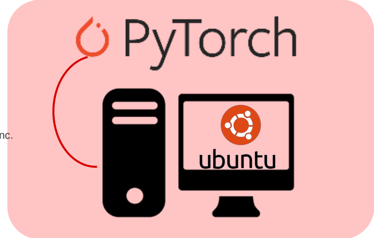
STEP-4

2.4: Convert AI Model to ONNX Format

Darknet-PyTorch

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

<ONNX conversion env.>



2. Move to the ONNX conversion working directory.

```
$ cd $WORK/darknet/yolo
```

3. Run the script to convert from Darknet YOLO model to the *pth* file (PyTorch format).

   Following command is for YOLOv3 model conversion. BLUE is the conversion target model name.

```
$ python3 convert_to_pytorch.py yolov3
```

If displayed as follows without any errors, the script is succeeded.

```
...
REST 0
```

4. Check that *pth* file is generated under *darknet* directory.

```
$ ls $WORK/darknet/yolo
```

Please confirm that *pth* file exists as follows.

Here, YOLOv3 model has been converted, therefore *yolov3.pth* is generated.

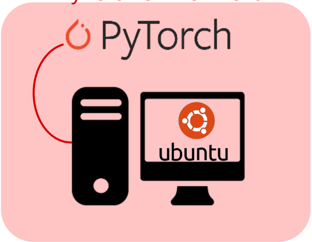
```
convert_to_pytorch.py convert_to_onnx.py yolov3.pth
```

2.4: Convert AI Model to ONNX Format

PyTorch[*1] → PyTorch-ONNX → ONNX

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

<PyTorch environment>



2. Convert from PyTorch to ONNX format.

   1. Prepare the PyTorch-ONNX conversion script. If STEP-2.3 are executed, the script is in the following location.

      Path: *$WORK/darknet/yolo/convert_to_onnx.py*

<convert_to_onnx.py>

```
    ⋮
if __name__ == "__main__":
    # Load YOLO neural network parameters
    ini = IniFile("yolo.ini")
    model_dict = ini.model_dict
    ⋮
    # Get the model information
    model_params = ini.architecture[model_dict[model_name]].params

    # Initialise one random value filled input tensor of an image size 3x416x416 (CHW)
    dummy_input = torch.randn(1, 3, 416, 416)

    # Loads from YOLO neural network structure
    model = yolo.Yolo(model_params.get('cfg'))
    model.load_state_dict(torch.load(model_params.get('pth')))


    # Define the input tensor and output tensor name of the converted onnx neural network
    input_names = model_params.get('input')
    output_names = model_params.get('output')

    # Starts the pythorch to onnx conversion
    torch.onnx.export(model, dummy_input, model_params.get('onnx'), verbose=True, opset_version=12, input_names=input_names, output_names=output_names)
```

- Input size of the model (N, C, H, W)
- Specify the model
- Specify the model parameter (weight)
- Input name of first layer of the model (Will be used in STEP-3)
- Output name of last layer of the model (Will be used in STEP-3)
- Set the ONNX file name

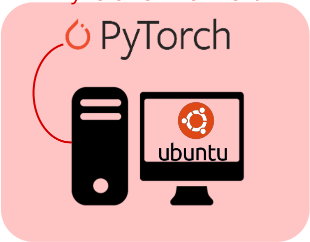
BIG IDEAS FOR EVERY SPACE RENESAS

2.4: Convert AI Model to ONNX Format

PyTorch[*1] → PyTorch-ONNX → ONNX

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

<PyTorch environment>



2. Move to the ONNX conversion working directory.

```
$ cd $WORK/darknet/yolo
```

3. Run the script to convert from the *pth* model created in the previous instruction to the *onnx* file.

   Following command is for YOLOv3 model conversion. BLUE is the conversion target model name.

```
$ python3 convert_to_onnx.py yolov3
```

If displayed as follows without any errors, the script is succeeded.
Note: Warning may be shown depending on the model. It does not affect the operation.

```
...

return (%output1, %output2, %output3)
```

4. Check the ONNX conversion working directory.

```
$ ls $WORK/darknet/yolo
```

Please confirm that *onnx* file is generated as follows.
Here, YOLOv3 model has been converted, therefore *d-yolov3.onnx* is generated.

```
convert_to_pytorch.py convert_to_onnx.py d-yolov3.onnx yolov3.pth
```

STEP-2 Summary

In this chapter, Darknet YOLO 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".

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

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.

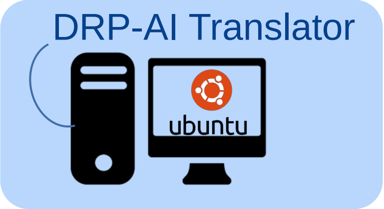
```
<path to the working directory>/rzv_ai_work
```

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
```



3.2: Check the File Configuration

Please confirm that each directory configuration is as follows.

<ONNX conversion working directory ($WORK)>

- rzv_ai_work
  - drpai_samples
    - onnx
      - d-yolov3.onnx
      - ⋮
    - yolov3_bmp
      - ⋮
    - addrmap_in_linux.yaml
  - darknet
    - yolo
  - drp-ai_translator_release

<DRP-AI Translator working directory ($DRPAI)>

- *drp-ai_translator_release*
  - *DRP-AI_translator*
  - *onnx*
  - *output*
  - *UserConfig*
  - *run_DRP-AI_translator_V2M.sh*
  - *run_DRP-AI_translator_V2L.sh*

To see the details of DRP-AI Translator directory, please refer to 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

3.3: Prepare the ONNX File

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

We use YOLOv3 model as an example.

- *drp-ai_translator_release*
  - …
  - *onnx*
    - *d-yolov3.onnx*
  - *UserConfig*
    - *addrmap_in_d-yolov3.yaml*
    - *prepost_d-yolov3.yaml*

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

```
$ cp -v $WORK/darknet/yolo/d-yolov3.onnx $DRPAI/onnx/
```

2. Check the *onnx* directory.

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

Check that there is *d-yolov3.onnx* file.

```
d-yolov3.onnx  tiny_yolov2.onnx  yolov2.onnx
```

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

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.

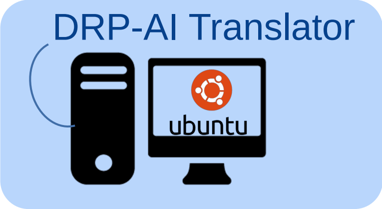
We use YOLOv3 model as an example.

- *drp-ai_translator_release*
  - *…*
  - *onnx*
    - *d-yolov3.onnx*
  - *UserConfig*
    - *addrmap_in_d-yolov3.yaml*
    - *prepost_d-yolov3.yaml*

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.



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.

d-yolov3.onnx

→

addrmap_in_d-yolov3.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_d-yolov3.yaml.*

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

```
$ mv -v ./addrmap_in_linux.yaml ./addrmap_in_d-yolov3.yaml
```

3.5: Prepare the Pre/Postprocessing Definition File

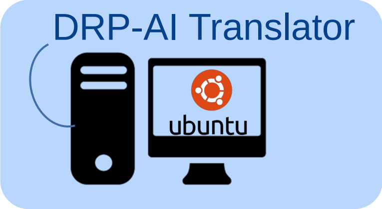
In this section, we will prepare the pre/postprocessing definition file which is necessary for DRP-AI Translator.

We use YOLOv3 model as an example.

- *drp-ai_translator_release*
  - *…*
  - *onnx*
    - *d-yolov3.onnx*
  - *UserConfig*
    - *addrmap_in_d-yolov3.yaml*
    - *prepost_d-yolov3.yaml*

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.



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.

- *drp-ai_translator_release*
  - *DRP-AI_translator*
    - …
    - *UserConfig*
      - *sample*
        - *prepost_resnet50v1.yaml*
        - *prepost_tiny_yolov2.yaml*
        - *prepost_vgg16.yaml*
        - *prepost_yolov2.yaml*

Pre/postprocessing for Darknet YOLO is similar to the definition stated in the *prepost_tiny_yolov2.yaml.* Modify this file to prepare the pre/postprocessing definition file for Darknet YOLO.

NOTE:

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

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

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

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

d-yolov3.onnx

prepost_d-yolov3.yaml

2. Rename *prepost_tiny_yolov2.yaml* to *prepost_d-yolov3.yaml.*

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

```
$ mv -v ./prepost_tiny_yolov2.yaml ./prepost_d-yolov3.yaml
```

3.5: Prepare the Pre/Postprocessing Definition File

Each Darknet YOLO model has its own output format.

Please change the `output_from_body` and relative items in pre/postprocessing definition file according to the table below.

For other items, follow the instructions explained in this chapter.

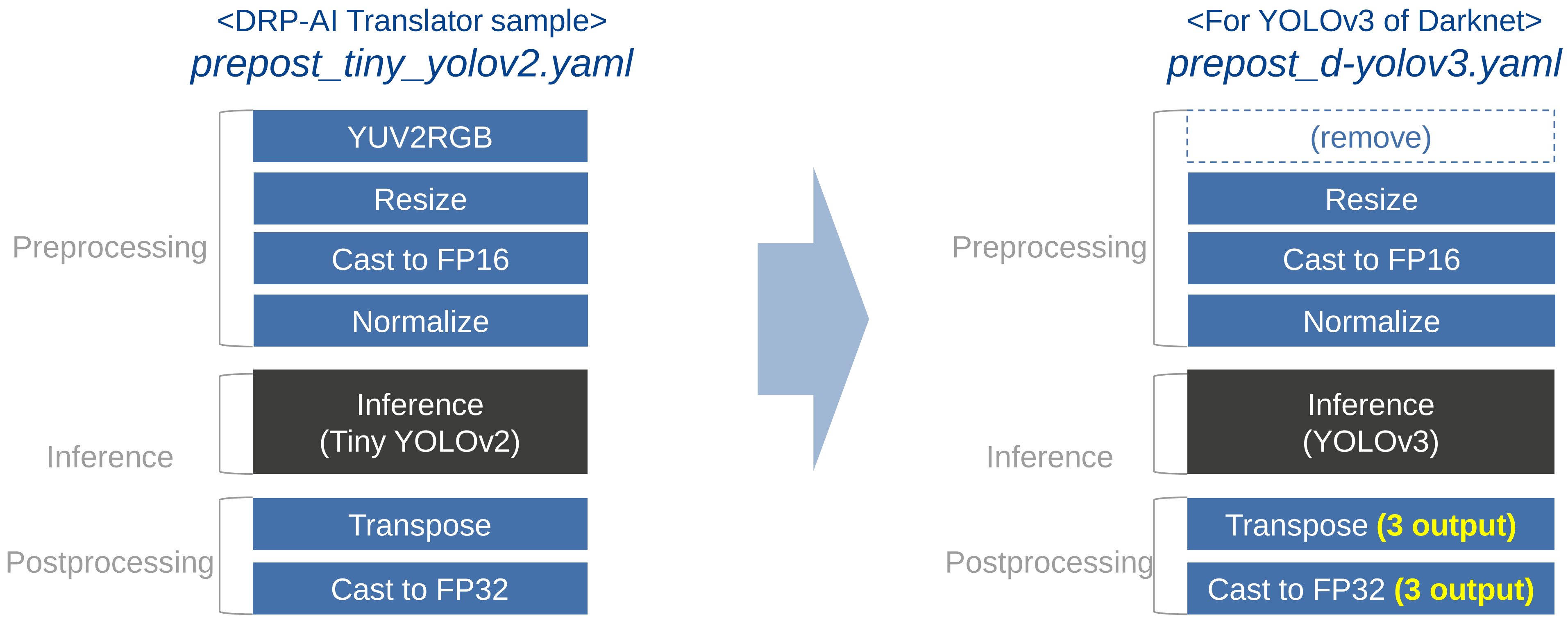
Model	name	shape (HWC)
YOLOv3	output1 output2 output3	[13, 13, 255] [26, 26, 255] [52, 52, 255]
YOLOv2	output1	[13, 13, 125]
Tiny YOLOv3	output1 output2	[13, 13, 255] [26, 26, 255]
Tiny YOLOv2	output1	[13, 13, 125]

Note: "name" is defined when it is converted to the ONNX format in STEP-2.

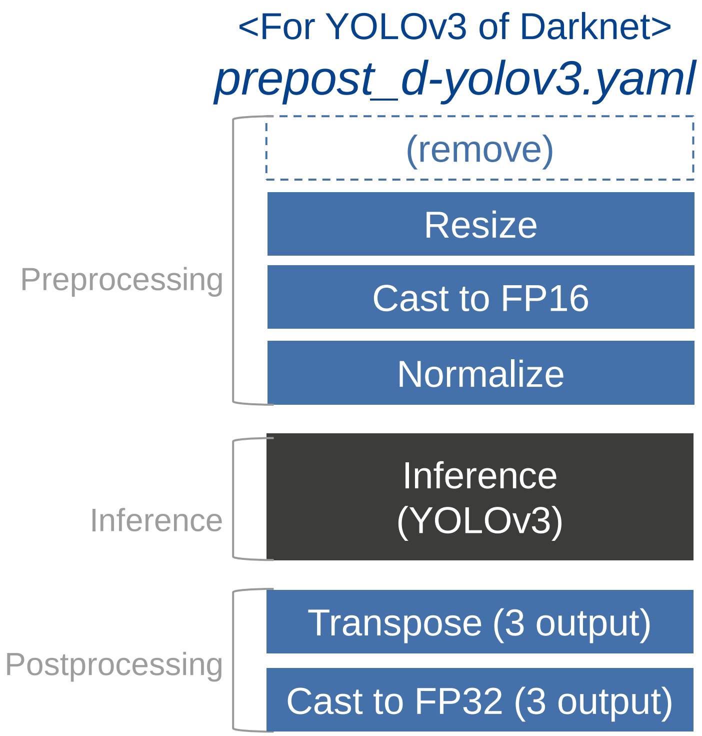
3.5: Prepare the Pre/Postprocessing Definition File

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 YOLOv3 of Darknet shown in the right figure, which includes the modification of input data format from YUV to BGR.



3.5: Prepare the Pre/Postprocessing Definition File



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

1. Change the number of model output data to 3.
2. 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.
3. Change the input data format for the preprocessing from YUY2 to RGB.
4. Set the parameters of normalize to the training value.

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

Definition of *prepost_d-yolov3.yaml*

Definition	Items
Input data definition	(2), (3)
Output data definition	(1), (2)
Preprocessing definition	(2), (3), (4)
Postprocessing definition	(1), (2)

3.5: Prepare the Pre/Postprocessing Definition File

First, change the number of model output data to 3.

In the YOLOv3 model, there are 3 output layers, and each output layer size can be calculated as follows.

$[\ H \times W \times (3*(4 + 1 + N_{class}))]$

N_{class} = Number of detection class

H, W = Height and width of each output layer.

Note: Please refer to the following website for more information of model output.

YOLOv3: An Incremental Improvement https://arxiv.org/pdf/1804.02767.pdf

YOLOv3: Real-Time Object Detection (Joseph Redmon) https://pjreddie.com/darknet/yolo/

The Darknet YOLOv3 model used in this document outputs the result in the following format.

Output 1: [13 x 13 x (3*(4 + 1 + 80))] → [13 x 13 x 255]

Output 2: [26 x 26 x (3*(4 + 1 + 80))] → [26 x 26 x 255]

Output 3: [52 x 52 x (3*(4 + 1 + 80))] → [52 x 52 x 255]

This chapter explains how to rewrite the pre/postprocessing definition file for above output information.

Note: For Tiny YOLOv3, please use appropriate number of output layers, their names and shapes.
For YOLOv2 and Tiny YOLOv2, since the number of output layer is 1, this procedure is unnecessary.

3.5: Prepare the Pre/Postprocessing Definition File

1. Open the *prepost_d-yolov3.yaml* with the editor.

```
$ vi $DRPAI/UserConfig/prepost_d-yolov3.yaml
```

2. Change the output data from the model in the output data definition.
   Rewrite the output format from the model according to the YOLOv3 output information described in previous page.

```
output_from_body:
  -
    name: "grid"
    shape: [13, 13, 125]
    order: "HWC"
    type: "fp16"
```

Copy the output data

```
output_from_body:
  -
    name: "grid"
    shape: [13, 13, 125]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [13, 13, 125]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [13, 13, 125]
    order: "HWC"
    type: "fp16"
```

Output format from model (output 1)
Output format from model (output 2)
Output format from model (output 3)

```
output_from_body:
  -
    name: "grid"
    shape: [13, 13, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [26, 26, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [52, 52, 255]
    order: "HWC"
    type: "fp16"
```

Note: How to change the output name from the model will be explained in "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."

3.5: Prepare the Pre/Postprocessing Definition File

3. Change the output data from the postprocessing in the output data definition.

   Rewrite the output shape from the postprocessing according to the YOLOv3 output information described before.

```
output_from_post:
  -
    name: "post_out"
    shape: [125, 13, 13]
    order: "CHW"
    type: "fp32"
```

Copy output data

```
output_from_post:
  -
    name: "post_out"
    shape: [125, 13, 13]
    order: "CHW"
    type: "fp32"
  -
    name: "post_out"
    shape: [125, 13, 13]
    order: "CHW"
    type: "fp32"
  -
    name: "post_out"
    shape: [125, 13, 13]
    order: "CHW"
    type: "fp32"
```

Output name and shape from postprocessing (output 1)

Output name and shape from postprocessing (output 2)

Output name and shape from postprocessing (output 3)

```
output_from_post:
  -
    name: "post_out1"
    shape: [255, 13, 13]
    order: "CHW"
    type: "fp32"
  -
    name: "post_out2"
    shape: [255, 26, 26]
    order: "CHW"
    type: "fp32"
  -
    name: "post_out3"
    shape: [255, 52, 52]
    order: "CHW"
    type: "fp32"
```

Note 1: Output name from the postprocessing specified here will also be used in later instruction.

Note 2: Users can change the output name from the postprocessing appropriately. However, please use the different name for each output.

3.5: Prepare the Pre/Postprocessing Definition File

4. Change the output data from the model in the postprocessing definition to 3 outputs.

Copy the postprocessing definition to be 3 outputs.

```
postprocess:
  -
    src: ["grid"]

    dest: ["post_out"]

    operations:
      -
        op : transpose
        param:
          WORD_SIZE: 1     # 2Byte
          IS_CHW2HWC: 0    # HWC to CHW

      -
        op : cast_fp16_fp32
        param:
          CAST_MODE: 0 # FP16 to FP32
```

output 1

```
postprocess:
  -
    src: ["grid"]
      ⋮
      -
        op : cast_fp16_fp32
        param:
          CAST_MODE: 0 # FP16 to FP32
```

output 2

```
  -
    src: ["grid"]
      ⋮
      -
        op : cast_fp16_fp32
        param:
          CAST_MODE: 0 # FP16 to FP32
```

output 3

```
  -
    src: ["grid"]
      ⋮
      -
        op : cast_fp16_fp32
        param:
          CAST_MODE: 0 # FP16 to FP32
```

3.5: Prepare the Pre/Postprocessing Definition File

5. Rewrite the output data from the model in the postprocessing definition, which was added in instruction 4.

Before:

```
postprocess:
  -
    src: ["grid"]

    dest: ["post_out"]
    :
  -
    src: ["grid"]

    dest: ["post_out"]
    :
  -
    src: ["grid"]

    dest: ["post_out"]
```

Output name from postprocessing (output 1) *1 → dest: ["post_out1"]

Output name from postprocessing (output 2) *1 → dest: ["post_out2"]

Output name from postprocessing (output 3) *1 → dest: ["post_out3"]

After:

```
postprocess:
  -
    src: ["grid"]

    dest: ["post_out1"]
    :
  -
    src: ["grid"]

    dest: ["post_out2"]
    :
  -
    src: ["grid"]

    dest: ["post_out3"]
```

Note 1: For output name from the postprocessing, use the name specified in instruction 3.

Note 2: How to change the output name from the model will be explained in "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."

3.5: Prepare the Pre/Postprocessing Definition File

Next, 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 *convert_to_onnx.py* in "STEP-2.4: Convert AI Model to ONNX Format".

```
<convert_to_onnx.py>
      ⋮
    # Define the input tensor and output tensor name of the converted onnx neural network
    input_names = model_params.get('input')
    output_names = model_params.get('output')

    # Starts the pythorch to onnx conversion
    torch.onnx.export(model, dummy_input, model_params.get('onnx'), verbose=True, opset_version=12, input_names=input_names, output_names=output_names)
```

Input name of first layer of the model (Will be used in STEP-3)

Output name of last layer of the model (Will be used in STEP-3)

The actual name is listed in the *yolo.ini* file.

We will set this input name (*input1*) and the output name (*output1, output2, output3*) in each definition.

```
<yolo.ini>
[yolov3]
cfg     =yolov3.cfg
weights =yolov3.weights
pth     =yolov3.pth
input   =["input1"]
output  =["output1", "output2", "output3"]
onnx    =d-yolov3.onnx
 ⋮
```

Defined here.
Input name : *input1*
Output name : *output1, output2, output3*

3.5: Prepare the Pre/Postprocessing Definition File

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

```
input_to_body:
  -
    name: "image2"
    format: "RGB"
    order: "HWC"
    shape: [416, 416, 3]
    type: "fp16"
```

Input name to model →

```
input_to_body:
  -
    name: "input1"
    format: "RGB"
    order: "HWC"
    shape: [416, 416, 3]
    type: "fp16"
```

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

```
preprocess:
  -
    src      : ["camera_data"]

    dest     : ["image2"]
```

Input name to model →

```
preprocess:
  -
    src      : ["camera_data"]

    dest     : ["input1"]
```

3.5: Prepare the Pre/Postprocessing Definition File

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

```
output_from_body:
  -
    name: "grid"
    shape: [13, 13, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [26, 26, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "grid"
    shape: [52, 52, 255]
    order: "HWC"
    type: "fp16"
```

Output name from model (output 1)

Output name from model (output 2)

Output name from model (output 3)

```
output_from_body:
  -
    name: "output1"
    shape: [13, 13, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "output2"
    shape: [26, 26, 255]
    order: "HWC"
    type: "fp16"
  -
    name: "output3"
    shape: [52, 52, 255]
    order: "HWC"
    type: "fp16"
```

3.5: Prepare the Pre/Postprocessing Definition File

9. Rewrite output name from model in postprocessing definition.

```
postprocess:
  -
    src: ["grid"]
    dest: ["post_out1"]
    :
  -
    src: ["grid"]
    dest: ["post_out2"]
    :
  -
    src: ["grid"]
    dest: ["post_out3"]
```

Output name from model (output 1) → `src: ["output1"]`

Output name from model (output 2) → `src: ["output2"]`

Output name from model (output 3) → `src: ["output3"]`

```
postprocess:
  -
    src: ["output1"]
    dest: ["post_out1"]
    :
  -
    src: ["output2"]
    dest: ["post_out2"]
    :
  -
    src: ["output3"]
    dest: ["post_out3"]
```

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.

```
input_to_pre:
  -
    name: "camera_data"
    format: "YUY2"
    order: "HWC"
    shape: [480, 640, 2]
    type: "uint8"
```

Input name → Format name → Number of channels →

```
input_to_pre:
  -
    name: "bgr_data"
    format: "BGR"
    order: "HWC"
    shape: [480, 640, 3]
    type: "uint8"
```

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

```
preprocess:
  -
    src      : ["camera_data"]
```

Input name →

```
preprocess:
  -
    src      : ["bgr_data"]
```


BIG IDEAS FOR EVERY SPACE RENESAS

3.5: Prepare the Pre/Postprocessing Definition File

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

```
  -
        op: conv_yuv2rgb
        param:
          DOUT_RGB_FORMAT: 0 # "RGB"
```

```
# -
#       op: conv_yuv2rgb
#       param:
#         DOUT_RGB_FORMAT: 0 # "RGB"
```

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

Output format of normalize operation

```
  -
        op: normalize
        param:
          DOUT_RGB_ORDER: 0 # Output RGB order = Input RGB order
          cof_add: [0.0, 0.0, 0.0]
          cof_mul: [0.00392157, 0.00392157, 0.00392157]
```

```
  -
        op: normalize
        param:
          DOUT_RGB_ORDER: 1
          cof_add: [0.0, 0.0, 0.0]
          cof_mul: [0.00392157, 0.00392157, 0.00392157]
```

3.5: Prepare the Pre/Postprocessing Definition File

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

Although normalization with mean and std (standard deviation) is not necessary for the Darknet pre-trained model, input data must be converted to floating point number between 0 and 1.

Therefore, we use ***mean=[0, 0, 0]*** and ***std=[1, 1, 1]*** to set the normalize parameter.

To compute, use the formulae explained in "STEP-3.5: Prepare the Pre/Postprocessing Definition File" in Get Started Document.

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

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

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

12. Rewrite *cof_add* and *cof_mul* in the *normalize* operation. (In this case, they are the same values as sample)

```
-
    op: normalize
    param:
      DOUT_RGB_ORDER: 1
      cof_add: [0.0, 0.0, 0.0]
      cof_mul: [0.00392157, 0.00392157, 0.00392157]
```

```
-
    op: normalize
    param:
      DOUT_RGB_ORDER: 1
      cof_add: [0.0, 0.0, 0.0]
      cof_mul: [0.00392157, 0.00392157, 0.00392157]
```

3.6: Translate the Model Using DRP-AI Translator

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

We use the YOLOv3 model as an example.

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

- *drp-ai_translator_release*
  - …
  - *onnx*
    - *d-yolov3.onnx*
  - *UserConfig*
    - *addrmap_in_d-yolov3.yaml*
    - *prepost_d-yolov3.yaml*

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 yolov3 -onnx ./onnx/d-yolov3.onnx
```

For RZ/V2L:

```
$ ./run_DRP-AI_translator_V2L.sh yolov3 -onnx ./onnx/d-yolov3.onnx
```

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

```
[Run DRP-AI Translator] Ver. 1.80
[Input file information]
   PREFIX               : yolov3
   ONNX Model           : ./onnx/d-yolov3.onnx
   Prepost file         : ./UserConfig/prepost_d-yolov3.yaml
   Address mapping file : ./UserConfig/addrmap_in_d-yolov3.yaml
      ...

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

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 -1 $DRPAI/output/yolov3/
```

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
yolov3.json
yolov3_addrmap_intm.txt
yolov3_addrmap_intm.yaml
yolov3_data_in_list.txt
yolov3_data_out_list.txt
yolov3_drpcfg.mem
yolov3_prepost_opt.yaml
yolov3_summary.xlsx
yolov3_tbl_addr_data.txt
yolov3_tbl_addr_data_in.txt
yolov3_tbl_addr_data_out.txt
yolov3_tbl_addr_drp_config.txt
yolov3_tbl_addr_merge.txt
yolov3_tbl_addr_weight.txt
yolov3_tbl_addr_work.txt
yolov3_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-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".

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

Execute Inference with DRP-AI

This section will explain the instruction with the following assumption

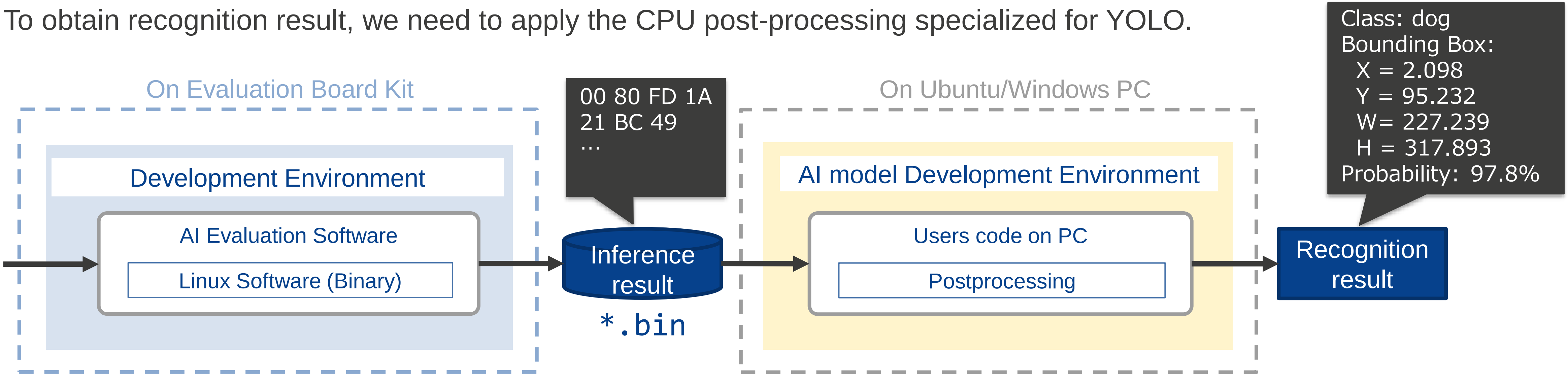
- Read "STEP-4 Execute Inference with DRP-AI" in the Get Started Document.

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

The DRP-AI Object files for YOLO 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 YOLO.



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 Darknet YOLO, which runs on Linux PC.

Path: *$WORK/darknet/yolo/postprocess_yolo.py*

Details of the script are as follow.

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

Notes

- ✓ Assumed to have run following postprocessing on DRP-AI
  - transpose
  - castFP16toFP32
- ✓ Install the necessary packages listed in Confirmed Operational Environment below appropriately.
- ✓ For more details of the algorithm, refer to the paper. (YOLOv3 https://arxiv.org/pdf/1804.02767.pdf, YOLOv2 https://arxiv.org/pdf/1612.08242.pdf.)
- ✓ The script specifies the result of DRP-AI Sample Application sample input image as the inference result binary file.
- ✓ The script draws the detected bounding boxes on the input image. Please specify the name of input image used when executing the AI Evaluation Software.
- ✓ Obtain the label list of detection class, *coco-labels-2014_2017.txt*, from the following URL.
  https://github.com/amikelive/coco-labels/blob/master/coco-labels-2014_2017.txt

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
                        Pillow        == 9.2.0
```

DRP-AI Inference Result Binary Data

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

- Number of data : 904995

(Depends on model output size. For YOLOv3, (1x255x13x13)+(1x255x26x26)+(1x255x52x52))

- Data width : 4byte (Because of castFP16toFP32, width is FP32=4byte)
- Byte order : Little endian

Execute Inference with DRP-AI

When running the postprocessing script, please specify the name of model as shown below with BLUE.

Note: If not specified, YOLOv3 model will be selected by default.

```
$ python3 postprocess_yolo.py yolov3
```

Parameter	Model
`yolov3`	YOLOv3 (Default)
`yolov2`	YOLOv2
`tinyyolov3`	Tiny YOLOv3
`tinyyolov2`	Tiny YOLOv2

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

Execute Inference with DRP-AI

postprocess_yolo.py

```python
if __name__ == '__main__':
    if ( len(sys.argv) > 1 and sys.argv[1] in ["yolov3", "yolov2", "tinyyolov3", "tinyyolov2"]):
        ⋮
   if model_name=="yolov3":
        anchors = anchors_yolov3
        out_layer_num = 3
        num_class = len(labels)
        output_shape.append([1, 3*(5+num_class), 13, 13]) # [N, C, H, W]
        output_shape.append([1, 3*(5+num_class), 26, 26]) # [N, C, H, W]
        output_shape.append([1, 3*(5+num_class), 52, 52]) # [N, C, H, W]
    elif model_name=="yolov2":
        ⋮
    result_bin = open("sample.bmp.bin", 'rb')
        ⋮
    # Load DRP-AI output binary
    for n in range(out_layer_num): # Output number
        for c in range(output_shape[n][1]): # C
            for h in range(output_shape[n][2]): # H
                for w in range(output_shape[n][3]): # W
                    a = struct.unpack('<f', result_bin.read(4))
                    data[n][0, c, h, w] = a[0]

    # Read image to draw bounding boxes
    im = Image.open("sample.bmp")
    im = im.resize((model_in_size, model_in_size), resample=Image.BILINEAR)

    # Post-processing
    preds = {}
    for i in range(out_layer_num):
        preds[i] = torch.from_numpy(data[i]).clone()
    detections = all_post_process(model_name, preds, anchors, input_size=model_in_size,
        grid_size=output_shape, n_classes=num_class, obj_th=0.5, nms_th=0.5)

    # Draw bounding box
    img = np.asarray(im) # RGB
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)# BGR
    ret_im = draw_predictions(img, detections, labels) # BGR
    cv2.imwrite("result.jpg", ret_im)
```

Obtain the target model from command line argument.

Specify model dependent parameters.

1. Bounding Box anchor
2. Number of output layers
3. Number of detected class
4. Output shape

* sample.bmp.bin is the result of AI Evaluation Software with DRP-AI Sample Application input image

Load DRP-AI inference result file

Read the number from inference result

Specify little endian

* sample.bmp is the sample input image of DRP-AI Sample Application

Load and resize the input image for bounding boxes

Set the threshold for Probability in *obj_th*, and threshold for NMS in *nms_th*

* explained in next page

Draw the bounding boxes

STEP-1

STEP-2

STEP-3

STEP-4

Execute Inference with DRP-AI

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

Followings are the parameters used in the postprocessing script.

`obj_th` : Threshold for the probability of detected results.

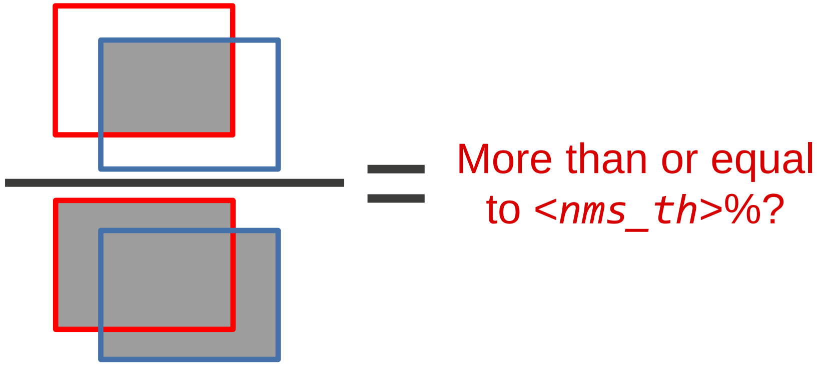
Detected results with probability that is below this value will be ignored. Default value is 0.5.

`nms_th` : Threshold for Non-Maximum Suppression (NMS) algorithm, which can eliminate over detected results.

Default value is 0.5.

Non-Maximum Suppression

Sometimes, multiple bounding boxes are detected for single object. NMS will compute the percentage of intersection of two boxes over their union and eliminate the bounding box with lower probability if the percentage is more than or equal to value *x*.
(Shown in figure below)
The parameter *nms_th* will be this value *x*.



Following example is the result of *postprocess_yolo.py* when running YOLOv3 model with the sample input image provided in DRP-AI Sample Application for Darknet YOLO.

Terminal Log

```
Class: bicycle | Probability 98.5% | [X1, Y1, X2, Y2] = [61,93,311,313]
Class: truck | Probability 95.4% | [X1, Y1, X2, Y2] = [254,62,375,121]
Class: dog | Probability 99.9% | [X1, Y1, X2, Y2] = [69,165,171,388]
```

Class : Detected object
Probability : Objectness * Predictions(Class) of detected object

Output Image (result.jpg)

(X1, Y1)

(X2, Y2)

BIG IDEAS FOR EVERY SPACE

STEP-4 Summary

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

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 Darknet YOLO model explained in this document.

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

Conclusion

Throughout STEP-1 to STEP-4, we have explained how to run the YOLO model provided by Darknet 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)