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

2022年9月

ルネサス エレクトロニクス株式会社

R11AN0621JJ0720

はじめに

本書は、AI Implementation Guide Get Startedドキュメントの内容をMMPoseフレームワークが提供する
HRNet事前学習済みモデル（以降、MMPose HRNet）を使用して解説します。

事前にGet Startedドキュメントを必ずお読みください。

本書では以下のドキュメント及びファイルを使用します。

名称	ファイル名	説明
Get Startedドキュメント	r11an0616jj0720-rzv-ai-imp-getstarted.pdf	AI Implementation Guide環境構築方法及び AI開発フローが記載されたドキュメント。
Get Startedソースコード	rzv_ai-implementation-guide_ver7.20.tar.gz	AI Implementation Guide全体で使用するソースコード。
MMPose HRNet向けドキュメント	r11an0621jj0720-rzv-ai-imp-hrnet.pdf	本書。MMPose HRNetモデル向け手順が記載されたドキュメント。
MMPose HRNet向けソースコード	mmpose_hrnet_ver7.20.tar.gz	MMPose HRNet向けドキュメント内で使用する ソースコード及び実行例。

MMPose

https://github.com/open-mmlab

ガイドの流れ

Step内で作成する物
前stepの成果物

目次

Step内で作成する物

前stepの成果物

STEP-1

Introduction

STEP-2

ONNXに
変換しよう

ONNX変換用スクリプト

学習済みモデル

STEP-3

DRP-AI
Translatorを
使ってみよう

変換用設定ファイル

ONNXモデル

STEP-4

推論実行
してみよう

AI実行環境

DRP-AI Object files

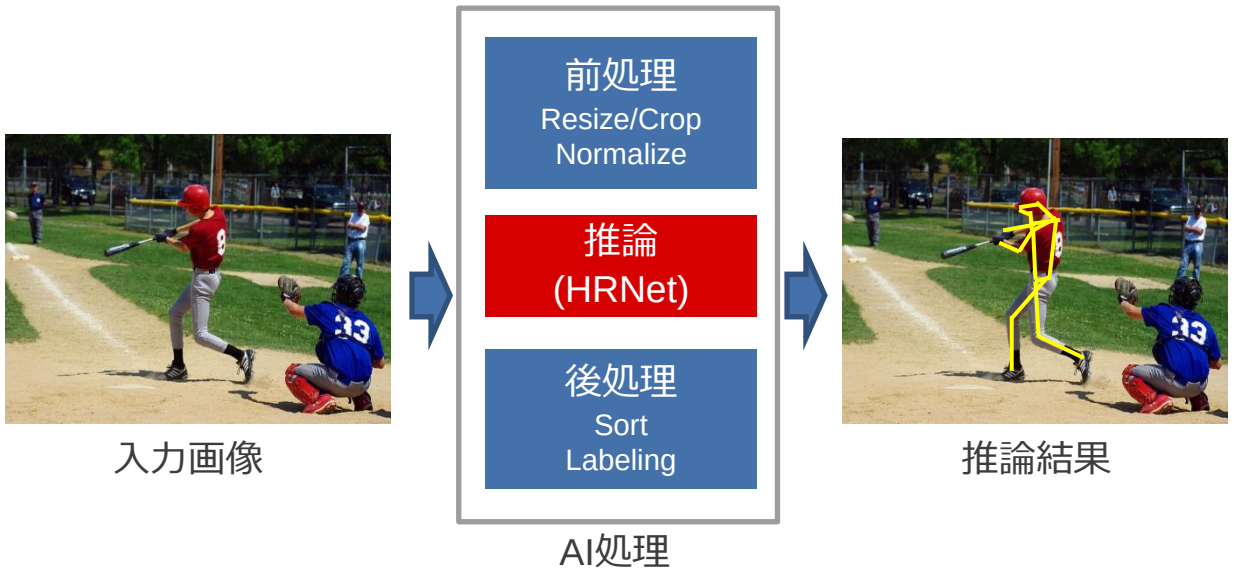
HRNet

本書ではMMPoseフレームワークが提供するHRNetモデルを使用します。

HRNet(**H**igh-**R**esolution **Net**work)は、人間向け姿勢推定（Human Pose Estimation）用の画像認識Neural Networkです。

HRNetでは高解像度サブネットワークからネットワークが深くなるにつれて低解像度サブネットワークを追加します。

これらのサブネットワークを並列に学習することで精度の向上を図っています。



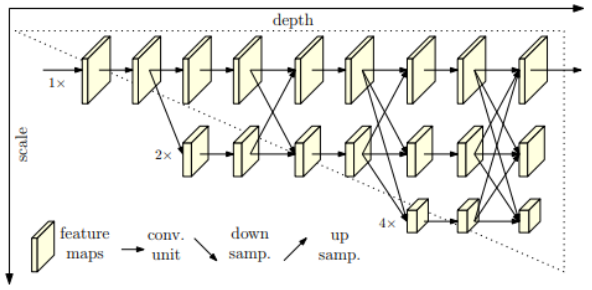


Figure 1. Illustrating the architecture of the proposed HRNet. It consists of parallel high-to-low resolution subnetworks with repeated information exchange across multi-resolution subnetworks (multi-scale fusion). The horizontal and vertical directions correspond to the depth of the network and the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

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

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

Top-down vs Bottom-up

2次元における複数人姿勢推定では主にTop-downアプローチ、Bottom-upアプローチという2種類に分けられます。

Top-downアプローチでは、画像内の人物を検出した後、各人物の姿勢を推定します。

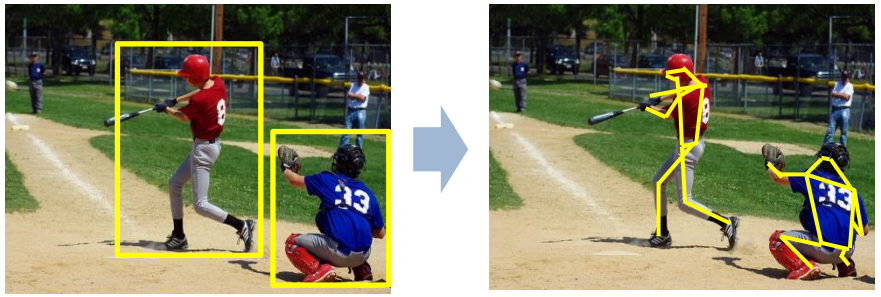
Bottom-upアプローチでは画像内の関節点を検出し、人物毎に関節点を分けることで複数人の姿勢を推定します。

HRNetは、人物検出を実行済みという前提のTop-downアプローチ向けの姿勢推定モデルです。

そのため、本書では人物検出済みと想定し、単体人物に対する姿勢推定向けのモデルとして使用方法を解説します。

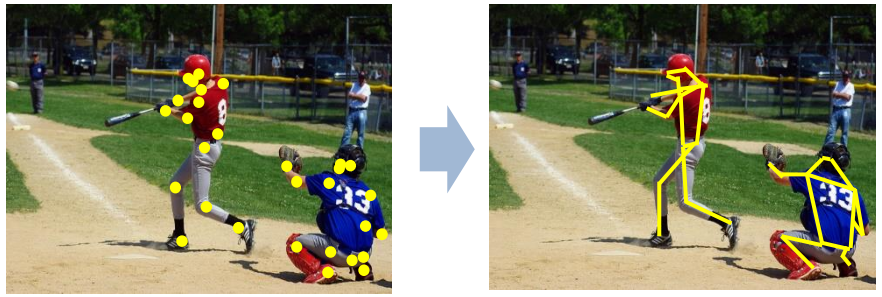
Top-down

人物を検出後、人物毎の関節点を検出



Bottom-up

関節点を検出後、人物毎にグループ分け



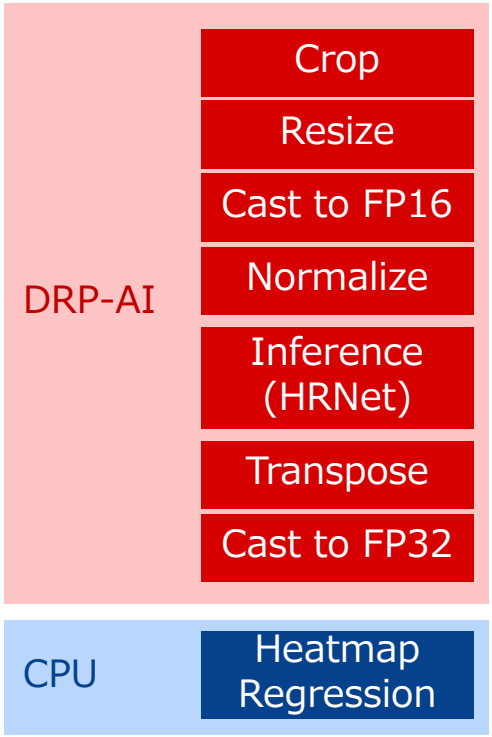
(論文) Deep High-Resolution Representation Learning for Human Pose Estimation https://arxiv.org/pdf/1902.09212.pdf

【参考】 HRNet on DRP-AI

以下はDRP-AIでHRNetモデルを実行する際の使用オペレータ例です。

入力データは640x480サイズのBGR画像を想定しています。

DRP-AIで対応していないオペレータはCPUで実行する必要があります。



<Pre-processing>
Crop : Input image size (640x480 BGR) to portrait image size (270x480 BGR).
Resize : Image size (270x480 BGR) to HWC tensor size (192x256x3).
Cast to fp16 : Cast to fp16 from int8 for DRP-AI processing.
Normalize: Normalized using 'mean' and 'std'.

<Post-processing>
Transpose : HWC (64x48x17) to CHW (17x64x48).
Cast to fp32

【参考】 MMPose & PyTorch

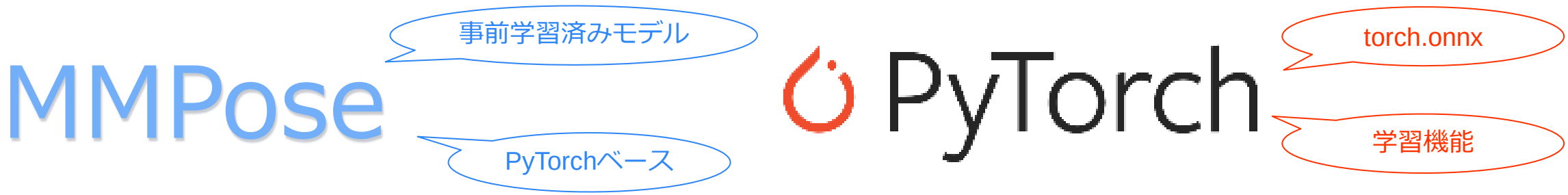
本書では、MMPoseが提供している事前学習済みモデルを使用します。

MMPoseはPyTorchフレームワークをベースにした姿勢推定向けのツールボックスです。

MMPoseでは、PyTorchの学習機能やONNX変換機能(torch.onnx)を使って、姿勢推定モデルの学習やONNX変換を可能にしています。

PyTorchの詳細については公式ドキュメント(https://pytorch.org/docs/1.12/)をご参照ください。

MMPoseの詳細については公式サイト(https://github.com/open-mmlab/mmpose) をご参照ください。



必要環境

本書の各STEPで必要な環境イメージは以下のとおりです。

環境構築方法はGet Startedドキュメントをご参照ください。

＜ONNX変換環境＞

MMPose PyTorch[*1]

以下のSTEPで使用します。
- STEP-2 ONNXに変換しよう

＜DRP-AI Translator環境＞

DRP-AI Translator

以下のSTEPで使用します。
- STEP-3 DRP-AI Translatorを使ってみよう

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

必要ファイル

本書で使用するソースコードはmmpose_hrnet_ver7.20.tar.gzに含まれています。

- mmpose_hrnet_ver7.20
  - drpai_samples
    - onnx
    - hrnet_bmp
  - mmpose
    - hrnet

drpai_samples、onnx、hrnet_bmp：DRP-AI Translatorの実行に使用するファイル例です。STEP-3で使用します。

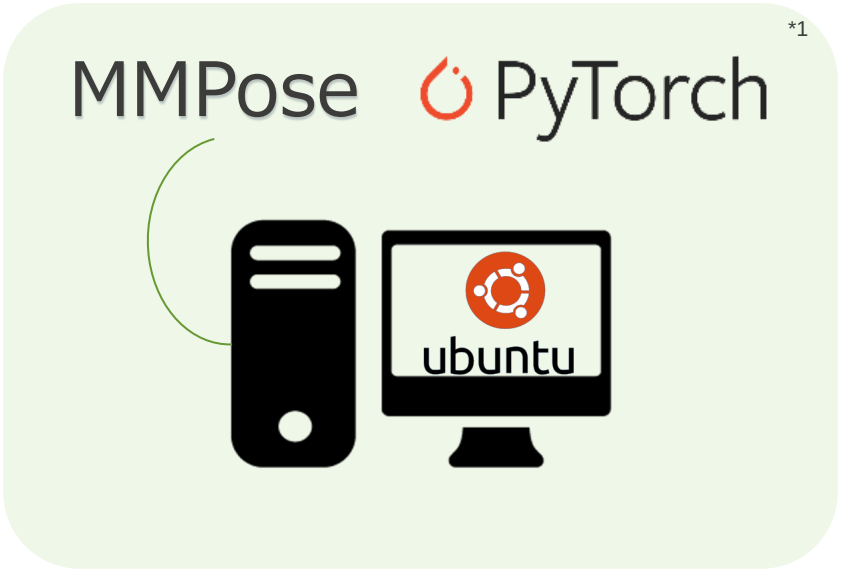
mmpose、hrnet：ONNX変換スクリプトや後処理スクリプトです。STEP-2、STEP-4で使用します。

【補足】HRNetの学習環境

本ガイドではDeep Learningの学習工程については解説しません。

HRNetモデルを自作データセットで学習する場合、またはモデル構造をカスタマイズする場合は、MMPose公式サイト (https://github.com/open-mmlab/mmpose)をご参照ください。

<モデル学習環境>



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

STEP-1まとめ

以上でHRNetモデルを実装するための基礎知識を学習することができました。

それでは実際にONNXモデルを作成してみましょう。

「STEP-2 ONNXに変換しよう」へ進みましょう。

Step内で作成する物
前stepの成果物

STEP-1

Introduction

STEP-2

ONNXに変換しよう

ONNX変換用スクリプト

学習済みモデル

STEP-3

DRP-AI Translatorを使ってみよう

変換用設定ファイル

ONNXモデル

STEP-4

推論実行してみよう

AI実行環境

DRP-AI Object files

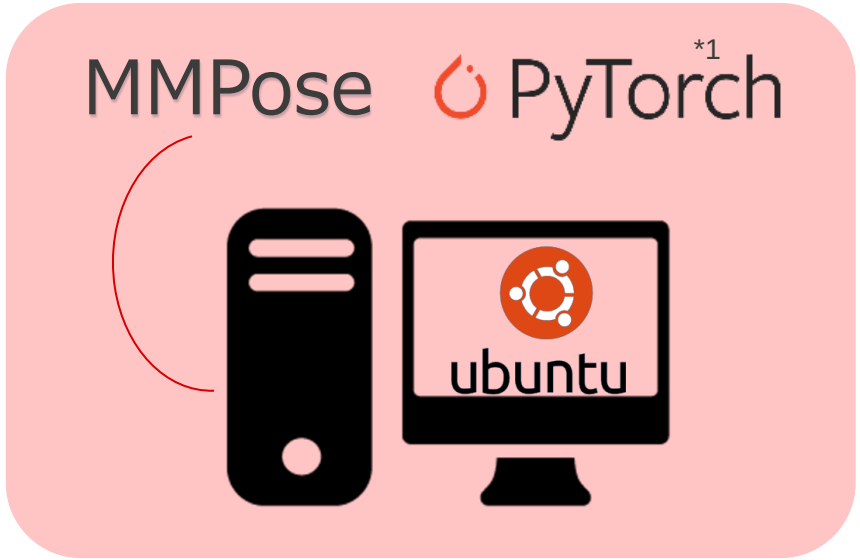
2.1: ONNXに変換するには？

MMPoseでは、事前学習済みモデルの構造とその重みパラメータを提供しています。

本章ではGet Startedドキュメントの「STEP-2 ONNXに変換しよう」を、
このMMPoseが提供するHRNetモデルを使用して解説します。

使用する環境は下図の通りです。

＜ONNX変換環境＞



+

MMPose

事前学習済みのHRNetモデル（COCOデータセット）

HRNet COCO Datasetモデル:
https://mmpose.readthedocs.io/en/latest/papers/algorithms.html#topdown-heatmap-hrnet-on-coco

ONNX

MMPoseからONNXへの変換

ONNX tutorials:
https://github.com/onnx/tutorials
MMPose公式チュートリアル:
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.

2.2: ONNX変換環境を構築しよう

本章は、以下を前提としております。

- Get Startedドキュメントの「STEP-2.2: ONNX変換環境を構築しよう」を実施し、ONNX変換環境が構築済み。

以下を実行してください。

1. 環境変数が登録されているか確認してください。緑色は環境変数です。

```
$ printenv WORK
```

以下のように作業ディレクトリのパスが表示されたら、環境変数は登録できています。

```
<作業ディレクトリのパス>/rzv_ai_work
```

2. シンボリックリンクを登録してください。（本手順はターミナルを起動するごとに実行する必要があります。）

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

2.3: 必要ファイルを準備しよう

本章は、以下を前提としております。

- Get Startedドキュメントの「STEP-2.3: 必要ファイルを準備しよう」を実施し、必要ファイルを展開済み。

本書で必要なファイルを準備します。

1. 作業ディレクトリへ移動します。緑色は環境変数です。

```
$ cd $WORK
```

2. tar.gzファイルを作業ディレクトリ下に解凍します。

```
$ tar xvzf <ファイルパス>/mmpose_hrnet_ver7.20.tar.gz -C $WORK
```

3. 作業ディレクトリ下に解凍できたか確認します。

```
$ ls $WORK
```

コマンド実行後、以下が表示されていることを確認してください。

```
drpai_samples     mmpose
```

drpai_samplesディレクトリは、DRP-AI Translator用の各種サンプルコード及び実行例が格納されています。

mmposeディレクトリは、MMPose用の各種サンプルコードが格納されています。

2.3: 必要ファイルを準備しよう

以上の手順を実行後、各ディレクトリのファイル構成が以下のようになっていることを確認してください。

<ONNX変換の作業ディレクトリ($WORK)>

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

2.4: AIモデルをONNX変換しよう

MMPoseが提供しているONNX変換スクリプトはPyTorchのONNX変換機能を使用しているため、

Get Startedドキュメントの「STEP-2.4: AIモデルをONNX変換しよう」で解説している通り、

以下のようなNNモデル構造とその重みパラメータを使って、ONNXモデルに変換します。

PyTorch [*1]

NNモデル構造
*.pyファイル

NNモデル重みパラメータ
*.pthファイル

ONNX
*.onnxファイル

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

MMPoseのONNX変換スクリプトでは、これらのファイルをPyTorchの以下の関数でONNXフォーマットに変換します。

```
torch.onnx.export(model, …)
```

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

PyTorch公式チュートリアル: https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

MMPose公式チュートリアル: https://mmpose.readthedocs.io/en/latest/tutorials/5_export_model.html

2.4: AIモデルをONNX変換しよう

MMPoseが提供するHRNetモデルのONNX変換には以下のスクリプトを使用します。

これらのファイルはMMPoseをインストール時にダウンロードされるため、既にMMPose環境に含まれています。

本書ではDRP-AI Translatorで実行しやすくするために、ONNX変換スクリプトを一部修正して使用します。

名称	ファイル名	用途	入手方法
ONNX変換スクリプト	pytorch2onnx.py	MMPose-ONNX変換に使用	MMPose提供
HRNetモデル構造	hrnet_w32_coco_256x192.py	MMPose-ONNX変換に使用	MMPose提供
HRNet重みパラメータ	hrnet_w32_coco_256x192-c78dce93_20200708.pth	MMPose-ONNX変換に使用	MMPose提供

2.4: AIモデルをONNX変換しよう

1. ONNX変換用スクリプト（格納場所は以下）を修正します。

ファイルパス：$WORK/mmpose/tools/deployment/pytorch2onnx.py

<変更前>

```
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)
```

<変更後>

```
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)
```

ここで指定した名称はSTEP-3で使用します。
input_names: モデルの先頭レイヤへの入力名
output_names: モデルの最終レイヤの出力名

2.4: AIモデルをONNX変換しよう

2. モデル構造ファイルのパスを環境変数に登録します。

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

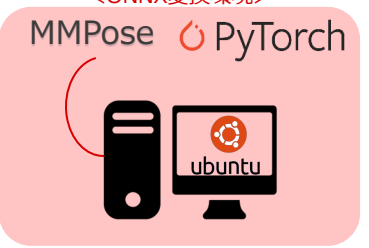
3. 重みファイルのURLを環境変数に登録します。

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

4. 出力ファイル名を環境変数に登録します。

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

<ONNX変換環境>



2.4: AIモデルをONNX変換しよう

5. ONNX変換用の作業ディレクトリへ移動します。緑色は環境変数です。

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

6. ONNX変換スクリプトを実行します。緑色は環境変数です。

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

注意事項

DRP-AI Translatorのサポート対象ONNX opset versionは12ですが、MMPoseではopset version 12に対応していないため、本書ではopset versionとして11を指定します。

コマンド実行後、エラーが無く、以下のように表示されていれば変換成功です。

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

以下のようにワーニングが出ますが問題ありません。

```
/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/mmdeploy
/mmcv/onnx/symbolic.py:481: UserWarning: DeprecationWarning: This function will be deprecated in future. Welcome to use the u
nified model deployment toolbox MMDeploy: https://github.com/open-mmlab/mmdeploy
```

エラーが発生した場合は、「STEP-2.2 ONNX変換環境を構築しよう」の手順2 シンボリックリンクの登録を実施後、再度お試しください。

2.4: AIモデルをONNX変換しよう

7. mmposeディレクトリにonnxファイルができているか確認します。

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

以下のようにhrnet.onnxがあることを確認してください。

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

STEP-2まとめ

以上でMMPoseが提供する事前学習済みHRNetモデルをONNXフォーマットに変換することができました。

次は作成したONNXモデルをDRP-AI Translatorを使って変換してみましょう。

「STEP-3 DRP-AI Translatorを使ってみよう」へ進みましょう。

Step内で作成する物

前stepの成果物

STEP-1

Introduction

STEP-2

ONNXに
変換しよう

ONNX変換用スクリプト

学習済みモデル

STEP-3

**DRP-AI
Translatorを
使ってみよう**

変換用設定ファイル

ONNXモデル

STEP-4

推論実行
してみよう

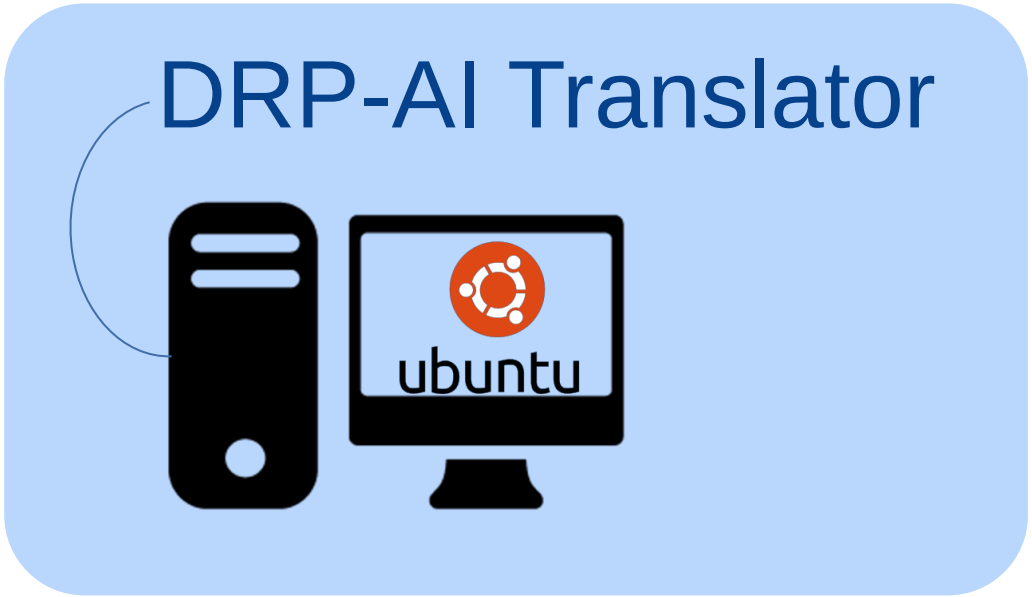
AI実行環境

DRP-AI Object files

DRP-AI Translatorを使ってみよう

STEP-2で作成したONNXフォーマットのモデルをDRP-AI Object filesへ変換してみましょう。

＜DRP-AI Translator環境＞

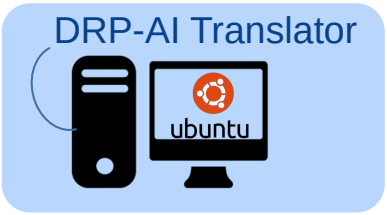


＋

ONNX

STEP-2で作成した
ONNXフォーマットのモデル

<DRP-AI Translator環境>



3.1: DRP-AI Translator環境を構築しよう

本章は、以下を前提としております。

- Get Startedドキュメントの「STEP-3.1: DRP-AI Translator環境を構築しよう」を実施し、DRP-AI Translator環境を構築済み。

DRP-AI Translator環境が構築されているかどうか、以下の手順でご確認ください

1. 作業ディレクトリの環境変数が登録されているか確認してください。

```
$ printenv WORK
```

以下のように作業ディレクトリのパスが表示されたら、環境変数は登録できています。

```
<作業ディレクトリのパス>/rzv_ai_work
```

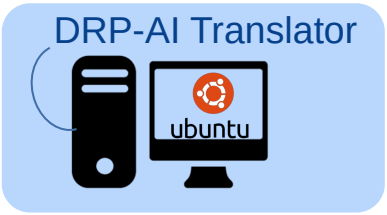
2. DRP-AI Translator作業ディレクトリの環境変数が登録されているか確認してください。

```
$ printenv DRPAI
```

以下のようにDRP-AI Translator作業ディレクトリのパスが表示されたら、環境変数として登録できています。

```
<$WORKに指定したパス>/drp-ai_translator_release
```

<DRP-AI Translator環境>



3.2: ファイル構成を確認しよう

各ディレクトリの構成が以下のようになっていることを確認してください。

<PyTorchの作業ディレクトリ($WORK)>

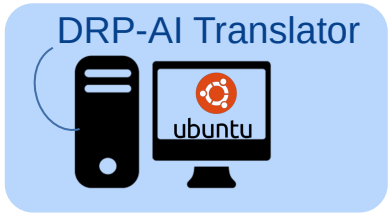
- rzv_ai_work
  - drpai_samples
    - onnx
      - hrnet.onnx
    - hrnet_bmp
    - addrmap_in_linux.yaml
  - mmpose
    - hrnet
    - …
  - drp-ai_translator_release

<DRP-AI Translatorの作業ディレクトリ($DRPAI)>

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

DRP-AI Translatorフォルダの詳細についてはGet Startedドキュメントをご参照ください。

<DRP-AI Translator環境>



3.3: ONNXファイルを用意しよう

本章では、 DRP-AI Translatorを実行するために必要なONNXファイルを用意していきます。

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

1. STEP-2で作成したonnxモデルをDRP-AI Translatorのonnxディレクトリへコピーします。

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

2. onnxディレクトリにコピーされたか確認します。

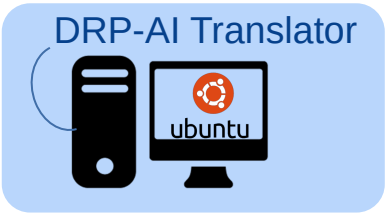
```
$ ls $DRPAI/onnx/
```

以下のようにhrnet.onnxがあることを確認してください。

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

└これら2ファイルはもともとDRP-AI Translatorに同梱されているモデルです。

<DRP-AI Translator環境>



3.4: アドレスマップ定義ファイルを用意しよう

本章では、 DRP-AI Translatorを実行するために必要なアドレスマップ定義ファイルを用意していきます。

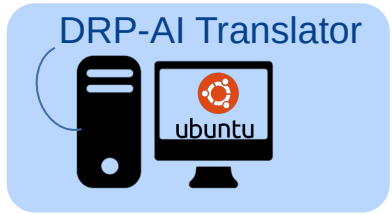
- drp-ai_translator_release
  - …
  - onnx
    - hrnet.onnx
  - UserConfig
    - addrmap_in_hrnet.yaml
    - prepost_hrnet.yaml

本章は実際に実行する手順のみを記載しております。

DRP-AI Object filesを格納する領域の先頭アドレスを書き換える必要があります。

詳細については、 Get Startedドキュメントの「STEP-3.4: アドレスマップ定義ファイルを用意しよう」をご参照ください。

<DRP-AI Translator環境>



3.4: アドレスマップ定義ファイルを用意しよう

アドレスマップ定義ファイルは、rzv_ai-implementation-guide_ver7.20.tar.gzにてご提供している addrmap_in_linux.yamlを使用します。

このファイルをアドレスマップ定義ファイルの命名ルールに従ってファイル名を変更します。

hrnet.onnx

addrmap_in_hrnet.yaml

1. addrmap_in_linux.yamlをdrp-ai_translator_release/UserConfigディレクトリへコピーします。

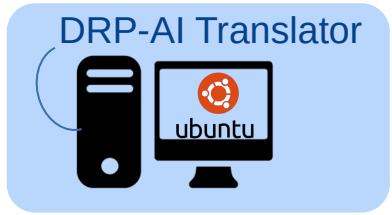
```
$ cp -v $WORK/drpai_samples/addrmap_in_linux.yaml $DRPAI/UserConfig
```

2. addrmap_in_linux.yamlをaddrmap_in_hrnet.yamlに名前を変更します。

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

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

<＜DRP-AI Translator環境＞>



3.5: 前後処理定義ファイルを用意しよう

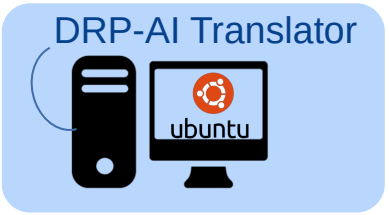
本章では、 DRP-AI Translatorを実行するために必要な前後処理定義ファイルを用意していきます。

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

本章はHRNetモデルに関する説明及び手順のみを記載しております。

前後処理定義ファイルの詳細については、 Get Startedドキュメントの
「STEP-3.5: 前後処理定義ファイルを用意しよう」をご参照ください。

<DRP-AI Translator環境>



3.5: 前後処理定義ファイルを用意しよう

DRP-AI Translatorに同梱されているサンプルを元に、前後処理定義ファイルを作成していきましょう。

前後処理定義ファイルのサンプルは、./UserConfig/sampleディレクトリの中にあります。（下図参照）

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

今回は、MMPoseのHRNetを変換するので、
後処理をそのまま使用できそうな
prepost_tiny_yolov2.yamlをカスタマイズしていきましょう。

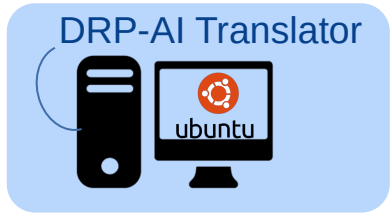
カスタマイズしたyamlファイルは、
全てUserConfigディレクトリ直下に置いてください。

1. prepost_tiny_yolov2.yamlをUserConfigディレクトリ直下へコピーします。

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

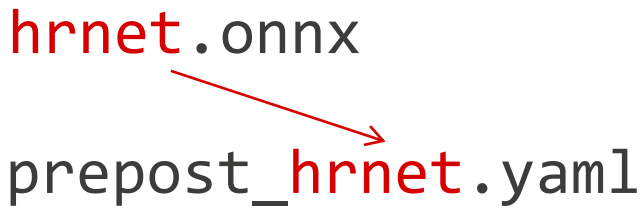
```
$ cp -v ./sample/prepost_tiny_yolov2.yaml ./
```

<DRP-AI Translator環境>



3.5: 前後処理定義ファイルを用意しよう

DRP-AI TranslatorはONNXモデルファイル名をもとに前後処理定義ファイルを指定するため、
前後処理定義ファイル名を変更します。



2. prepost_tiny_yolov2.yamlをprepost_hrnet.yamlに名前を変更します。

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

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

3.5: 前後処理定義ファイルを用意しよう

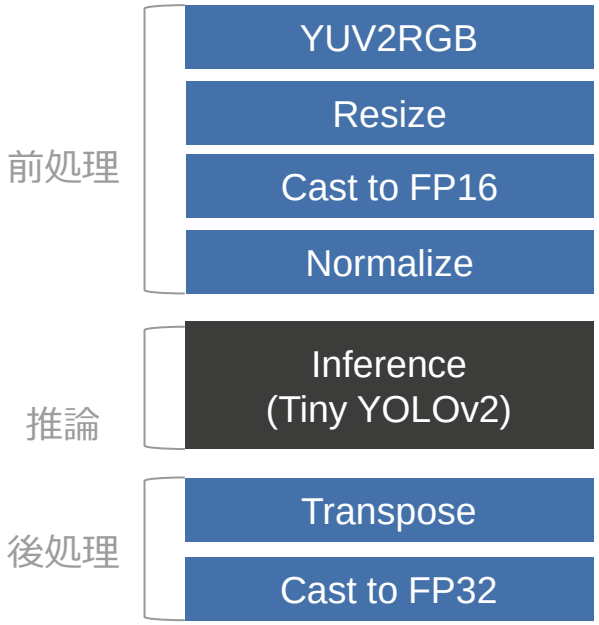
DRP-AI Translatorに同梱されているサンプルの前後処理定義ファイルは左下図のような前後処理になっています。

これを、MMPoseのHRNet向けに書き換えていきましょう。

また、サンプルでは入力データ形式がYUY2に設定されているので、BGR形式に変更してみましょう。

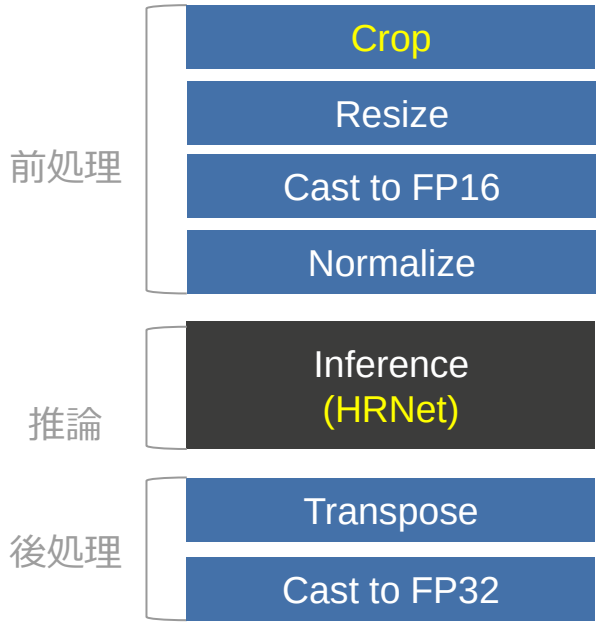
<DRP-AI Translator同梱>

prepost_tiny_yolov2.yaml



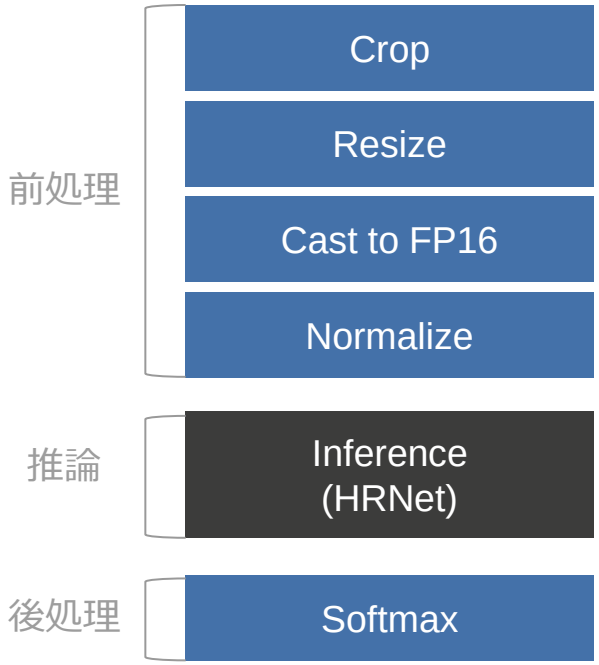
<MMPoseのHRNet向け>

prepost_hrnet.yaml



3.5: 前後処理定義ファイルを用意しよう

<MMPoseのHRNet向け>
prepost_hrnet.yaml



前後処理定義ファイルを左図の形にするために、5つの情報を書き換えます。

(1) モデルの入力データ名と出力データ名を、STEP-2のONNX変換時に命名したモデルの入出力レイヤ名に合わせる
(2) 前処理への入力データフォーマットをYUY2からBGRにする
(3) 前処理のNormalizeのパラメータ値を学習時の設定に合わせる
(4) モデルの入出力サイズをHRNetに合わせる
(5) Crop処理を追加する

書き換える情報と各定義の対応は下図のとおりです。

prepost_hrnet.yamlの構成

入力データの定義	(1), (2), (4)
出力データの定義	(1), (4)
前処理の定義	(1), (2), (3), (5)
後処理の定義	(1)

3.5: 前後処理定義ファイルを用意しよう

まずは、「モデルの入力データ名と出力データ名を、STEP-2のONNX変換時に命名したモデルの入出力レイヤ名に合わせる」をしましょう。

「STEP-2.4: AIモデルをONNX変換しよう」で、
モデルの先頭レイヤへの入力名と最終レイヤの出力名を定義してONNX変換しました。
その時の入力名（input1）と出力名（output1）になるように、各定義を書き換えていきます。

<変更後のpytorch2onnx.py>

```
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"]

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)
```

ここで各名称を定義しています。
入力名：input1
出力名：output1

3.5: 前後処理定義ファイルを用意しよう

3. エディタでprepost_hrnet.yamlを開きます。

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

4. 入力データの定義部分にあるモデルへの入力名を書き換えます。

```
12 input_to_body:
13   -
14     name: "image2"
15     format: "RGB"
16     order: "HWC"
17     shape: [416, 416, 3]
18     type: "fp16"
```

モデルへの入力名 →

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

5. 出力データの定義部分にあるモデルからの出力名を書き換えます。

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

モデルからの出力名 →

```
23 output_from_body:
24   -
25     name: "output1"
26     shape: [13, 13, 125]
27     order: "HWC"
28     type: "fp16"
```

3.5: 前後処理定義ファイルを用意しよう

6. 前処理の定義部分にあるモデルへの入力名を書き換えます。

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

モデルへの入力名

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

7. 後処理の定義部分にあるモデルからの出力名を書き換えます。

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

モデルからの出力名

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

3.5: 前後処理定義ファイルを用意しよう

次に、「前処理への入力データフォーマットをYUY2からBGRにする」をしましょう。

8. 入力データの定義部分にある前処理への入力データフォーマットを書き換えます。

```
4  input_to_pre:
5    -
6      name: "camera_data"
7      format: "YUY2"
8      order: "HWC"
9      shape: [480, 640, 2]
10     type: "uint8"
```

入力名
フォーマット名
Channel数

```
4  input_to_pre:
5    -
6      name: "bgr_data"
7      format: "BGR"
8      order: "HWC"
9      shape: [480, 640, 3]
10     type: "uint8"
```

9. 前処理の定義部分にある前処理への入力データフォーマットを書き換えます。

```
40 preprocess:
41   -
42     src       : ["camera_data"]
43
```

入力名

```
40 preprocess:
41   -
42     src       : ["bgr_data"]
43
```

3.5: 前後処理定義ファイルを用意しよう

10. 前処理の定義部分にあるconv_yuv2rgbオペレーション（YUV2RGB変換処理）をコメントアウトします。

```
47  -
48      op: conv_yuv2rgb
49      param:
50        DOUT_RGB_FORMAT: 0 # "RGB"
```

```
47  # -
48  #     op: conv_yuv2rgb
49  #     param:
50  #       DOUT_RGB_FORMAT: 0 # "RGB"
```

11. 前処理の定義部分にあるnormalizeオペレーション（正規化処理）のパラメータを書き換えます。

```
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]
```

normalize処理の出力形式

```
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]
```

3.5: 前後処理定義ファイルを用意しよう

次に、「前処理のNormalizeのパラメータ値を学習時の設定に合わせる」をしましょう。

今回は、MMPoseが提供している事前学習済みモデルを使用しているため、学習時のNormalizeのパラメータ値は、Neural Networkモデル構造ファイル（hrnet_w32_coco_256x192.py）で確認できます。

ファイルパス：$WORK/mmpose/configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/hrnet_w32_coco_256x192.py

hrnet_w32_coco_256x192.py一部抜粋

```
104   train_pipeline = [
105       dict(type='LoadImageFromFile'),
106       dict(type='TopDownRandomFlip', flip_prob=0.5),
107       dict(
      .
      .
      .
115       dict(
116           type='NormalizeTensor',
117           mean=[0.485, 0.456, 0.406],
118           std=[0.229, 0.224, 0.225]),
119       dict(type='TopDownGenerateTarget', sigma=2),
```

この引数 **mean=[0.485, 0.456, 0.406]** と **std=[0.229, 0.224, 0.225]** を、DRP-AI Translatorに設定します。

3.5: 前後処理定義ファイルを用意しよう

Normalizeパラメータ mean=[0.485, 0.456, 0.406] と std=[0.229, 0.224, 0.225] を addとmulに変換しましょう。

計算式はGet Startedドキュメントの「STEP-3.5 前後処理定義ファイルを用意しよう」 のものを使用します。

add = - (mean × range)

mul = 1 / (std × range)

モデルへの入力を0～1の浮動小数へ正規化するため、rangeは入力画像のダイナミックレンジの最大値255を使用します。

12. Normalizeオペレーションの引数であるcof_addとcof_mulを計算します。
    計算結果（以下）を前後処理定義ファイルに記入します。
    cof_add=[-123.675, -116.28, -103.53]
    cof_mul=[0.01712475, 0.017507, 0.01742919]

```
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]
```

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

3.5: 前後処理定義ファイルを用意しよう

次に、「モデルの入出力サイズをHRNetに合わせる」をしましょう。

今回は、MMPoseが提供している事前学習済みモデルを使用しているため、HRNetの入出力サイズは
Neural Networkモデル構造ファイル（hrnet_w32_coco_256x192.py）で確認できます。

ファイルパス：$WORK/mmpose/configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/hrnet_w32_coco_256x192.py

hrnet_w32_coco_256x192.py一部抜粋

```
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,
```

入力サイズ (HWC) = [256, 192, 3]
出力サイズ (HWC) = [64, 48, 17]

この引数 **入力サイズ=[256，192，3]**と **出力サイズ=[64，48，17]** を、DRP-AI Translatorに設定します。

3.5: 前後処理定義ファイルを用意しよう

13. 入力データの定義部分にあるモデルへの入力サイズを書き換えます。

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

モデルへの入力サイズ

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

14. 前処理の定義部分にあるresize処理のパラメータをモデルへの入力サイズに合わせて書き換えます。

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

モデルへの入力サイズ

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

3.5: 前後処理定義ファイルを用意しよう

15. 出力データの定義部分にあるモデルからの出力サイズを書き換えます。

```
23 output_from_body:
24   -              モデルからの出力サイズ
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. 出力データの定義部分にある後処理からの出力サイズを書き換えます。

```
30 output_from_post:
31   -              後処理からの出力サイズ
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"
```

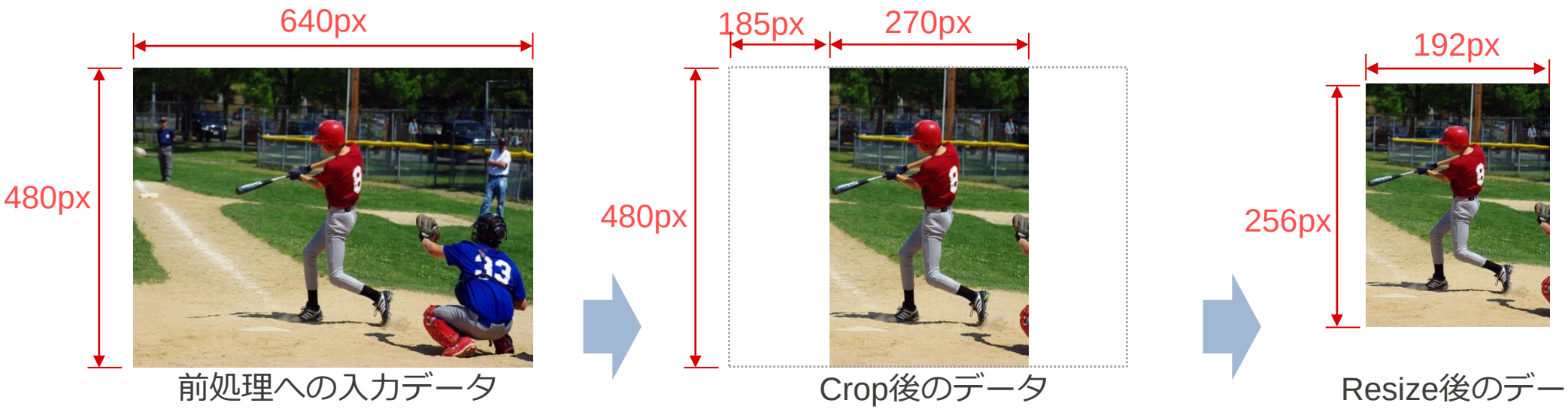
3.5: 前後処理定義ファイルを用意しよう

次に、「Crop処理を追加する」をしましょう。

本ガイドでは入力データとして、幅640px、高さ480pxの画像を使用しています。
しかし、HRNetモデルへの入力データサイズは幅192px、高さ256pxの縦長の画像です。

元の画像をresizeするだけでは、アスペクト比が大幅に異なってしまい、正当な認識結果を得ることができません。

本書では、crop処理を使って縦長の画像を切り出してから、モデルへの入力データサイズにresizeするように
前処理を変更します。



3.5: 前後処理定義ファイルを用意しよう

17. 前処理の定義部分にcrop処理を追加します。

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

crop処理を追加

```
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
```

主な設定項目は以下の通りです。

param > CROP_POS_X : 切り出し開始位置（左上）のX座標

param > CROP_POS_Y : 切り出し開始位置（左上）のY座標

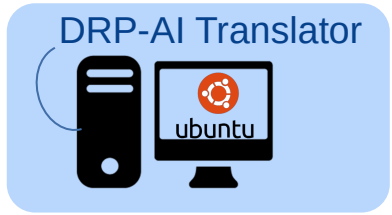
param > DATA_TYPE : データ型(uint8の場合は"0")

param > DATA_FORMAT : データ形式(HWCの場合は"0")

param > shape_out : cropオペレータからの出力データサイズ

cropオペレータの詳細はDRP-AI TranslatorのUMをご参照ください。

<＜DRP-AI Translator環境＞>



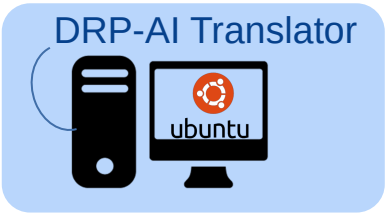
3.6: DRP-AI Translatorで変換してみよう

それでは実際にDRP-AI Translatorで変換してみましょう。

1. drp-ai_translator_releaseディレクトリに以下のように３つのファイルがあることを確認してください。

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

<DRP-AI Translator環境>



3.6: DRP-AI Translatorで変換してみよう

2. DRP-AI Translatorの作業ディレクトリへ移動します。

```
$ cd $DRPAI
```

3. 以下のコマンドでDRP-AI Translatorを実行します。（必ずdrp-ai_translator_releaseディレクトリ上で実施してください）
青字はPREFIXなので、任意の名前を指定してください。

RZ/V2M、 RZ/V2MAの場合:

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

RZ/V2Lの場合：

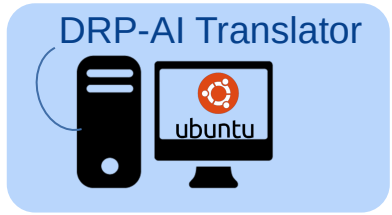
```
$ ./run_DRP-AI_translator_V2L.sh hrnet -onnx ./onnx/hrnet.onnx
```

エラーが無く、以下のように表示されていれば変換成功です。

```
[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
```

<DRP-AI Translator環境>



3.7: 変換結果を確認してみよう

1. 変換結果は、outputディレクトリの中の変換時に付けたPREFIX名が付いたディレクトリに格納されています。

```
$ ls -1 $DRPAI/output/hrnet/
```

正しく変換されていれば、以下のように表示されます。

```
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
```

黄色のファイルが
実機動作時に必要な
DRP-AI Object filesです。

緑色のファイルは
処理時間の目安を見積もるときに
使用するサマリファイルです。

STEP-3まとめ

以上でONNXフォーマットからDRP-AI Object filesへ変換することができました。

DRP-AI Object filesが作成できたら、「STEP-4 推論実行してみよう」へ進みましょう。

Step内で作成する物

前stepの成果物

STEP-1

Introduction

STEP-2

ONNXに
変換しよう

ONNX変換用スクリプト

学習済みモデル

STEP-3

DRP-AI
Translatorを
使ってみよう

変換用設定ファイル

ONNXモデル

STEP-4

**推論実行
してみよう**

AI実行環境

DRP-AI Object files

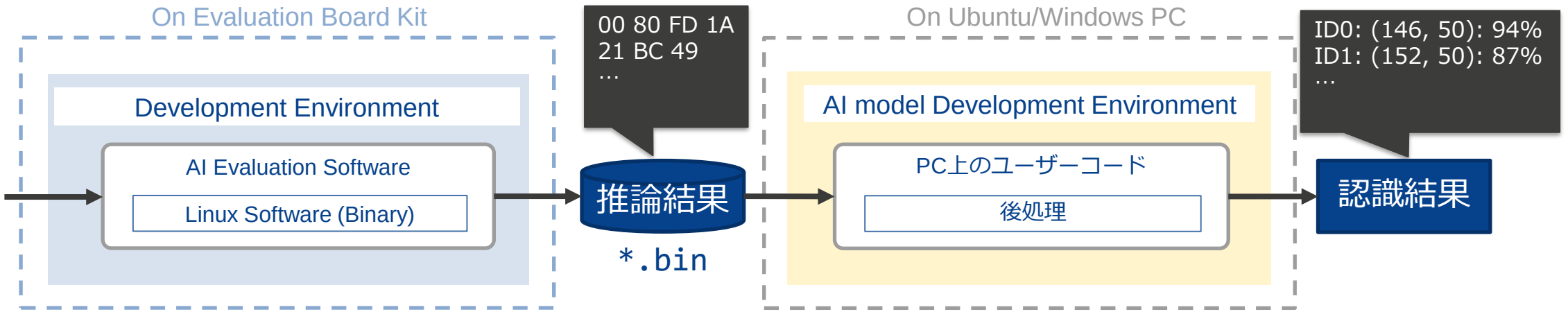
推論実行してみよう

本章は、以下を前提としております。

- Get Startedドキュメントの「STEP-4 推論実行してみよう」を確認済み。

STEP-3で作成したHRNet用DRP-AI Object filesはAI Evaluation Software でそのまま動かすことができます。

AI Evaluation Softwareを実行するとDRP-AIの演算結果が書かれたバイナリファイルが出力されますが、
このファイルでは認識結果を見ることはできません。認識結果を得るにはCPUによるHRNet用の後処理が必要です。



※AI Evaluation Software の使い方はAI Evaluation Software Guideをご参照ください。

推論実行してみよう

本書では、 MMPoseの提供するHRNetモデル用のLinux PC向け後処理スクリプトをご用意しております。

ファイルパス： $WORK/mmpose/hrnet/postprocess_hrnet.py

実行スクリプトの詳細は以下の通りです。

注意事項

- ✓ DRP-AIで以下の後処理を実施しているものとします。
  - transpose
  - castFP16toFP32
- ✓ 必要パッケージ（動作確認済み環境に記載のもの）は適宜インストールしてください。
- ✓ アルゴリズムの詳細はHRNetの論文（https://arxiv.org/pdf/1902.09212.pdf）をご参照ください。
- ✓ 実行スクリプトでは $WORK/mmpose/hrnet/sample.bmp画像の Evaluation Software実行結果を推論結果バイナリファイルとして指定しています。
- ✓ 実行スクリプトは入力画像に骨格を描画します。AI Evaluation Softwareの入力画像として使用した画像をご指定下さい。

動作確認済み環境

```
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推論結果バイナリデータ

DRP-AI推論結果バイナリファイル内のデータ詳細は以下です。

- データ数 : 52224 (モデルの出力サイズに依存。今回は[1x17x64x48])
- データ幅 : 4byte (castFP16toFP32を実行したため, FP32=4byte)
- バイトオーダー : リトルエンディアン

推論実行してみよう

postprocess_hrnet.pyを一部抜粋

```
...
if __name__ == '__main__':
    output_shape = [1, 17, 64, 48] # [N, C, H, W]
    input_size_x = 192 # Width of model input
    input_size_y = 256 # Height of model input

    # Drawing parameters
    kpt_score_thr=0.3
    thickness=1
    radius=4

    # dataset: 'TopDownCocoDataset'
    palette = np.array([[255, 128, 0], [255, 153, 51], [255, 178, 102],
    ...
    # Label from COCO dataset "https://arxiv.org/abs/1405.0312"
    label = {
    ...

    # Load DRP-AI output binary
    result_bin = open("sample.bmp.bin", 'rb') ※sample.bmp.binはサンプル入力画像($WORK/mmpose/hrnet/sample.bmp)のAI Evaluation Software実行結果
    data = np.zeros(output_shape, dtype=float)
    for c in range(output_shape[1]):
        for h in range(output_shape[2]):
            for w in range(output_shape[3]):
                a = struct.unpack('<f', result_bin.read(4))
                data[0, c, h, w] = a[0]

    # Postprocess
    ...
    # Image processing
    img = mmcv.imread("sample.bmp") ※sample.bmpはサンプル入力画像（$WORK/mmpose/hrnet/sample.bmp）
    ...
    # Print out result
    for i in range(len(all_preds[0])):
        print('ID {:2} {:14}: ({:3.0f}, {:3.0f}): {:5.1%}'.format(i, label[i], all_preds[0,i,0],all_preds[0,i,1],all_preds[0,i,2]))

    # Draw skeleton
    imshow_keypoints(img, all_preds, skeleton, kpt_score_thr,
    ...
    imwrite(img, "result.jpg")
```

モデルの入出力サイズ

描画用パラメータ

COCOデータセット向けの描画用パラメータ

キーポイント（関節点）のラベル

DRP-AI推論結果ファイルをロード

FP32の数値を[64x48x17]個読むリトルエンディアンを指定

描画用の入力画像をロード

後処理結果をコンソールに表示

骨格を描画

骨格描画画像を保存

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

推論実行してみよう

postprocess_hrnet.pyを実行してみましょう。

以下はサンプル入力画像($WORK/mmpose/hrnet/sample.bmp)をAI Evaluation Softwareで実行した場合の結果です。

ターミナルログ

```
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

入力画像(sample.bmp)

出力画像(result.jpg)

STEP-4まとめ

以上でDRP-AIを使ったHRNetモデルの推論実行についての説明は終了です。

DRP-AI Sample Applicationでは、本書でご説明したMMPoseの提供するHRNetモデルを推論からCPU後処理までをボード上で実行できるサンプルアプリケーションをご提供しております。

詳細はDRP-AI Sample Application Noteをご参照ください。

おわりに

STEP-1～STEP-4にかけて、MMPoseの提供するHRNetモデルをRZ/V2x上で推論実行する方法をご説明させていただきました。

もし、ご不明な点がございましたらルネサスまでお問い合わせください。

最後までお読みいただきありがとうございました。

Renesas.com

変更履歴

Date	Version	Chapter	変更内容
Sep. 29, 2022	7.20	-	初版（RZ/V2L, RZ/V2M, RZ/V2MAのAI Implementation Guide を統一）