

aicastアプリ開発ハンズオン

2023.07.12 SA 新田・田中

目標と目的

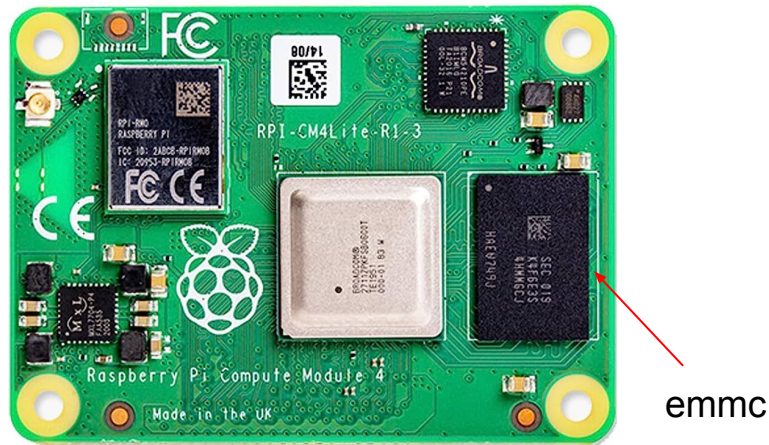
- aicastアプリを誰でも作れるようにする
 - 対象者: raspi向けactcastアプリを開発したことはあるがaicastアプリを開発したことのない人
- 事前準備をすませておいてください
 - [事前準備資料](#)
- 資料・ソースコード
 - <https://github.com/ldein/sa-aicast-tutorial-yolov5s>

目次

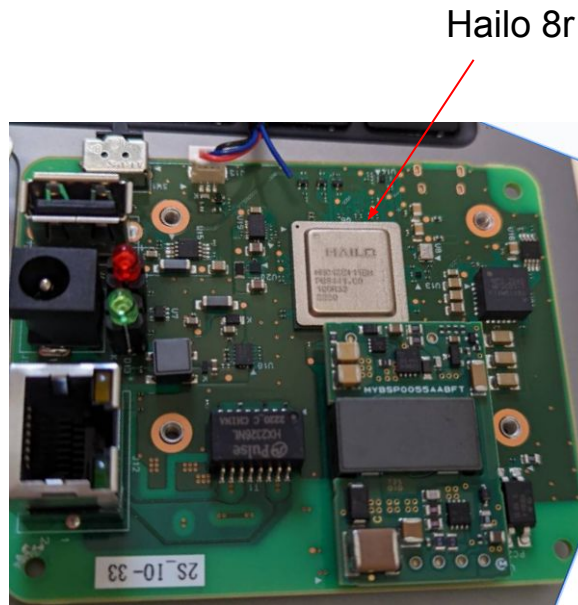
- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動作させる仕組み
 - 実践2: 物体検出アプリケーションを作ろう

raspiとの違い

- aicast = Raspi CM4 + Hailo8r
 - 基板2つがドッキングされている
- インタフェースは専用基板に繋がる



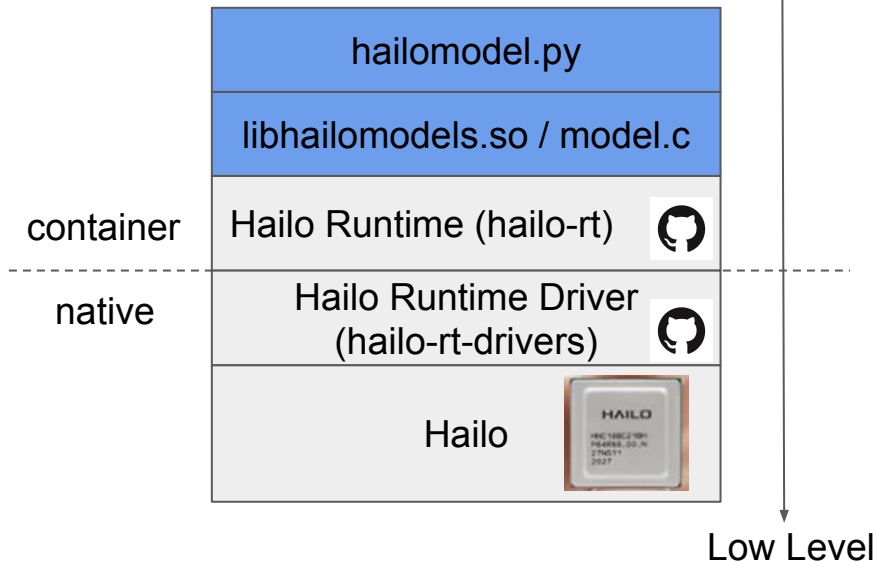
Raspi Compute Module 4



アイシンが設計した専用基板

Hailoについて

- Hailo社製 DNN推論専用アクセラレータ
 - 「量子化」された専用モデルが動作する
 - aicastに乗っているのは”hailo8r”
- AIモデルごとに専用の実行ファイルHEF (Hailo Executable Format)を作る
 - ONNX / tensorflowから変換する
 - すべてのレイヤに対応しているわけではない

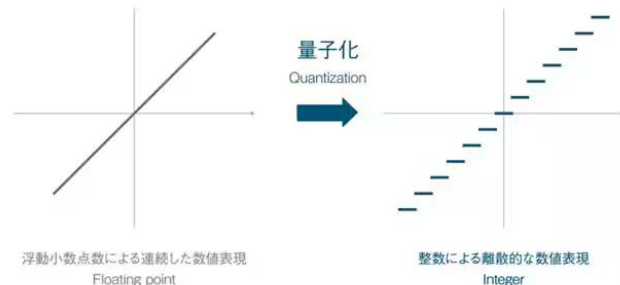


目次

- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動かす方法
 - 実践2: 物体検出アプリケーションを作ろう
- その他開発Tips

量子化とは

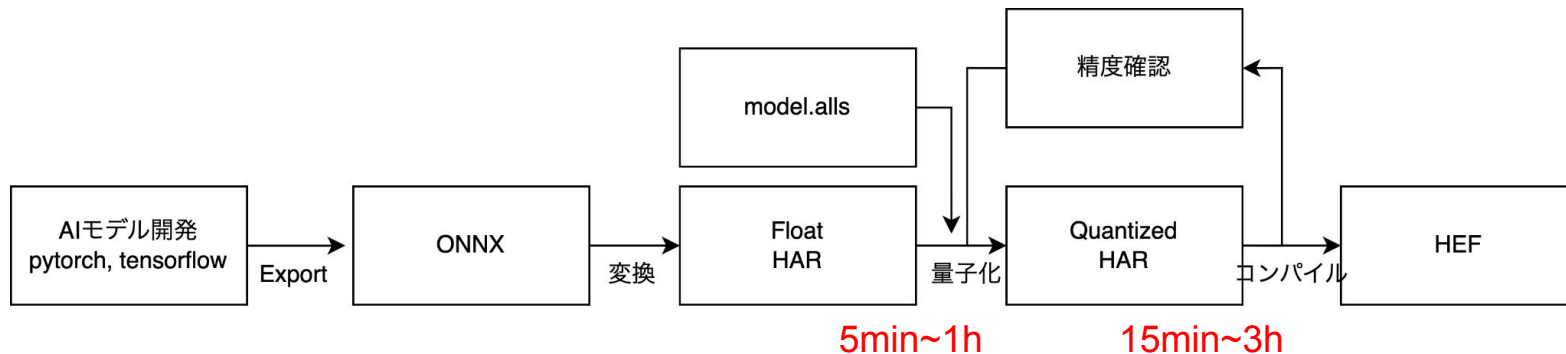
- モデル圧縮技術の1つ
 - 参考: [深層学習の量子化に入門してみた 理論編](#)
- 学習時はfloat 32bit (または半精度16bit)で計算
- 低bit整数に丸めることで**精度を犠牲に高速推論**
 - nnoirは32bitで推論するため精度劣化なし
- Hailoは16bit, 8bit, 4bitでの量子化に対応
 - 8bitがデフォルト
- 量子化に伴い増える作業
 - 量子化・キャリブレーション
 - 量子化パラメータの決定
 - 精度劣化の確認
 - 期待する精度が得られない場合は量子化精度を変更するなど工夫が必要



<https://www.tel.co.jp/museum/magazine/interview/202104/?section=2>

HEFができるまで

- ONNX→Float HAR→Quantized HAR→HEF の順で変換
- Hailoが提供するDataflow Compilerを使用する
 - 下のすべての作業を提供する
- HAR: Hailo ARchive File
 - 中間生成物
 - Hailo Runtimeを使ってホストPC上でtensorflow graphとして実行できる
 - ちなみにtar形式のアーカイブなので解凍すれば中身を見れる



HEFができるまで

- model.alls
 - 量子化時の最適化を指示
 - 量子化精度の指定
 - キャリブレーション画像枚数
 - そのほか、finetuneなど

[hailo_model_zoo / hailo_model_zoo / cfg / alls / hailo8 / base / yolov5m.alls](#) 



HailoModelZoo update-to-version-2.8.0

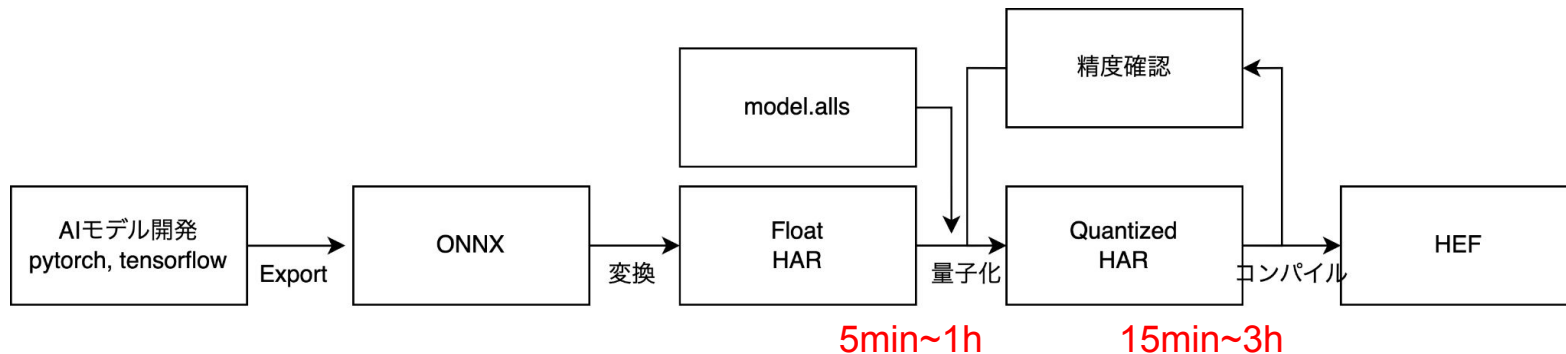
Code

Blame

12 lines (12 loc) · 889 Bytes

```
1 normalization1 = normalization([0.0, 0.0, 0.0], [255.0, 255.0, 255.0])
2 change_output_activation(sigmoid)
3 nms_postprocess("$HMZ_DATA/models_files/ObjectDetection/Detection-COCO/yolo/yolov5m_spp/pretrained/2023-04-25/yolov5m_nms,
4 model_optimization_config(calibration, batch_size=4, calibset_size=64)
5 quantization_param(conv45, precision_mode=a8_w4)
6 quantization_param(conv46, precision_mode=a8_w4)
7 quantization_param(conv51, precision_mode=a8_w4)
8 quantization_param(conv53, precision_mode=a8_w4)
9 quantization_param(conv84, precision_mode=a8_w4)
10 quantization_param(conv89, precision_mode=a8_w4)
11 quantization_param(conv91, precision_mode=a8_w4)
12 post_quantization_optimization(finetune, policy=enabled, learning_rate=0.0001, epochs=4, dataset_size=4000, loss_factors=
```

Hailo Model Zooのallsはゴリゴリにチューニング指示している



目次

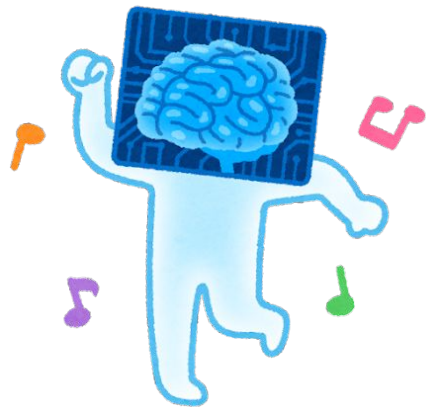
- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動作させる仕組み
 - 実践2: 物体検出アプリケーションを作ろう
- その他開発Tips

実践① HEFを作ろう

作業の流れ

事前準備: インスタンスを立ててコンテナを実行する ができることが前提です

1. yolov5を動かしてみる
2. onnxにエクスポート
3. onnxからHARをつくる
4. HARをホストPCで実行する
5. HARを量子化する
6. 量子化されたHARをホストPCで実行する
7. HARをHEFにコンパイルする



5, 7は時間がかかる為、今回は 3分クッキング形式で提供します

必要環境

ドキュメントより抜粋

3.1. System requirements

The Hailo Dataflow Compiler requires the following minimum hardware and software configuration:

1. Ubuntu 20.04/22.04, 64 bit (supported also on Windows, under WSL2)
2. 16+ GB RAM (32+ GB recommended)
3. Python 3.8/3.9/3.10, including pip and virtualenv
4. python3.X-dev and python3.X-distutils (according to the Python version), python3-tk, graphviz, and libgraphviz-dev packages. Use the command `sudo apt-get install PACKAGE` for installation.

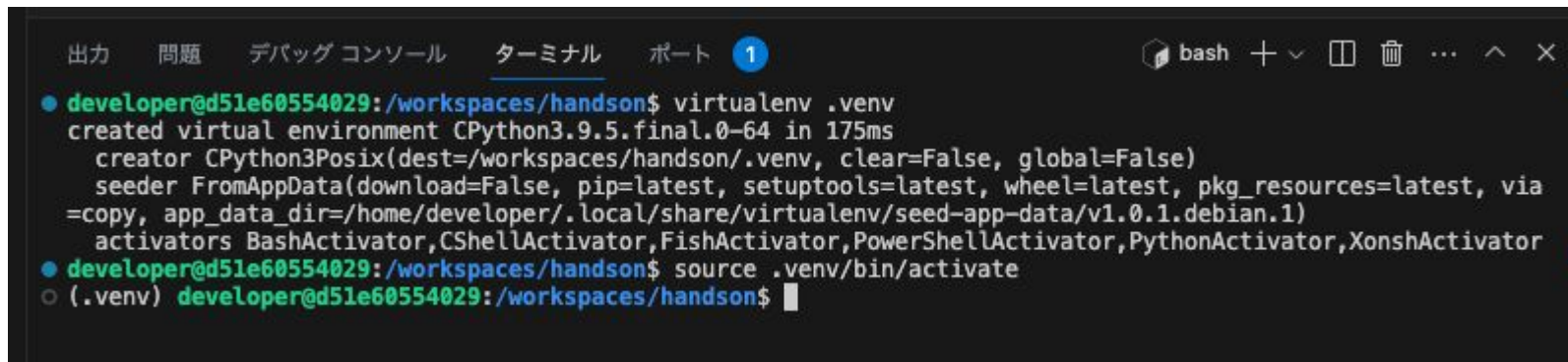
The following additional requirements are needed for GPU based hardware emulation:

1. Nvidia's Pascal/Turing/Ampere GPU architecture (such as Titan X Pascal, GTX 1080 Ti, RTX 2080 Ti, or RTX A4000)
2. GPU driver version 470
3. CUDA 11.2
4. CUDNN 8.1

Note: The Dataflow Compiler installs and runs Tensorflow, and when Tensorflow is installed from PyPi and runs on the CPU it requires AVX instructions support. Therefore, it is recommended to use a CPU that supports AVX instructions. Another option is to compile Tensorflow from sources without AVX.

環境構築

- VSCodeで開発コンテナに繋いで下さい
- dockeコンテナのnativeのpythonに既にDataflow Compilerはインストールされていますが、今回は仮想環境を使います。



The screenshot shows a terminal window within a VS Code editor. The terminal has tabs for '出力' (Output), '問題' (Problems), 'デバッグ コンソール' (Debug Console), 'ターミナル' (Terminal), and 'ポート' (Ports). The 'ターミナル' tab is active, showing a shell prompt 'bash'. The terminal content shows the execution of 'virtualenv .venv' and 'source .venv/bin/activate' commands, creating and activating a virtual environment.

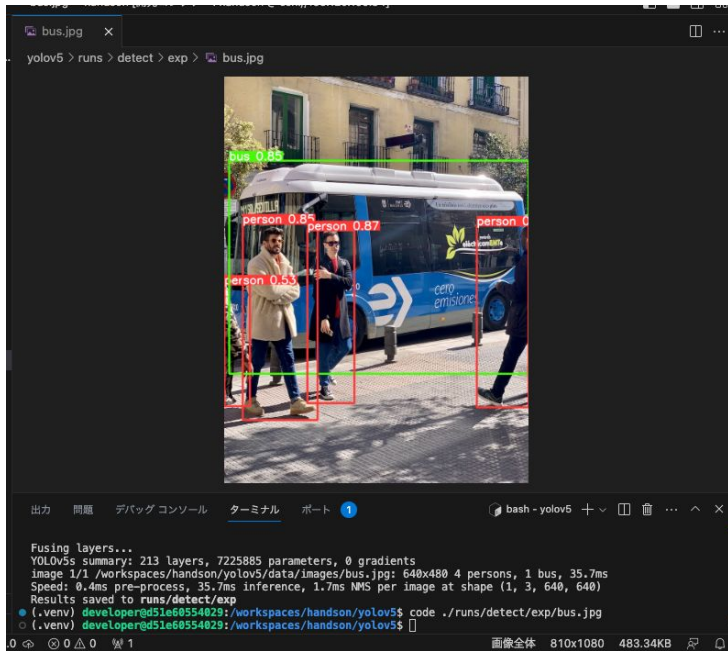
```
● developer@d51e60554029:/workspaces/handson$ virtualenv .venv
created virtual environment CPython3.9.5.final.0-64 in 175ms
creator CPython3Posix(dest=/workspaces/handson/.venv, clear=False, global=False)
seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, pkg_resources=latest, via
=copy, app_data_dir=/home/developer/.local/share/virtualenv/seed-app-data/v1.0.1.debian.1)
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
● developer@d51e60554029:/workspaces/handson$ source .venv/bin/activate
○ (.venv) developer@d51e60554029:/workspaces/handson$
```

yolov5のclone

```
git clone https://github.com/ultralytics/yolov5.git && cd yolov5  
git checkout v7.0  
pip install -r requirements.txt  
sudo apt-get install wget  
wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt
```

yolov5を動かしてみる

```
python3 detect.py --weight yolov5s.pt --source ./data/images/bus.jpg  
code ./runs/detect/exp/bus.jpg
```



onnxにエクスポート・眺める

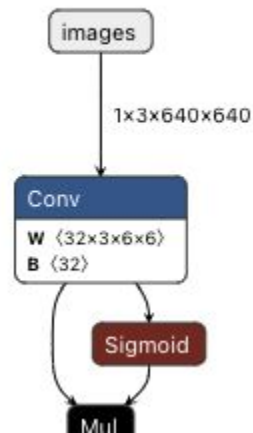
```
python3 export.py --weights yolov5s.pt --include onnx
```

onnxが自動的にインストールされます。onnxsimは自動で適用されます。

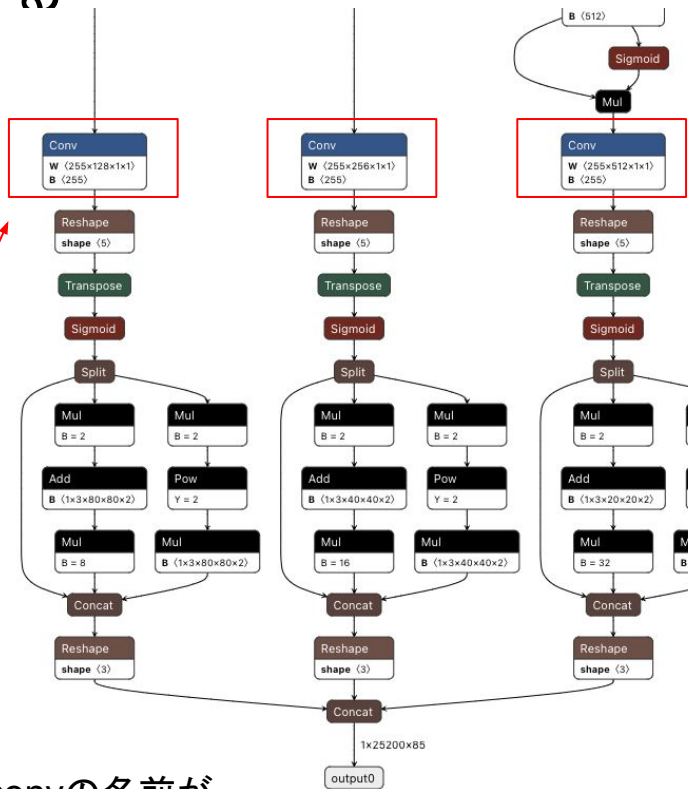
```
pip install netron  
netron yolov5s.onnx
```

VSCodeを使用していると自動的にポートフォワードされてホスト PCのブラウザから <http://localhost:8080/> でモデルを見れます。

onnxにエクスポート・眺める



入力ノード名はimages



この赤枠の3つのconvの名前が
/model.24/m.0/Conv, /model.24/m.1/Conv, /model.24/m.2/Convであることを確認しておく(理由は後述)

HARを作る準備

チュートリアル用プロジェクトの clone

```
cd /workspaces/handson
git clone https://github.com/ldein/sa-aicast-tutorial-yolov5s.git
cd sa-aicast-tutorial-yolov5s
cp ../yolov5/yolov5s.onnx ./make_hef/
```

compilerインストール(仮想環境は yolov5と分ける)

```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s
deactivate
virtualenv .venv && source .venv/bin/activate
pip install /workspaces/handson/docker/hailo_dataflow_compiler-3.23.0-py3-none-linux_x86_64.whl
pip install pydantic==1.10
```

hailo_compilerの依存指定が悪くpydantic2.0がインストールされていてエラーを起こすので1.10を指定してインストール

HARを作ろう

/workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef内で作業
下記コマンドでHARを生成する

```
cd make_hef
hailo parser onnx yolov5s.onnx --start-node-names images --end-node-names
/model.24/m.0/Conv /model.24/m.1/Conv /model.24/m.2/Conv --hw-arch hailo8r
```

下記のようなメッセージが出ても無視で y で良い
(Dockerfileに問題があり環境情報を正常に取得できていないのが原因)

```
[info] First time Hailo Dataflow Compiler is being used. Checking system requirements... (this might take a
few seconds)
[Error] Unsupported OS - . Only Ubuntu is supported.
[Error] See full log: /workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef/.install_logs/hailo_installat
ion_log_2023.07.04T11:21:00.log
[Error] Unsupported release - . Only 20.04 22.04 are supported.
[Error] See full log: /workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef/.install_logs/hailo_installat
ion_log_2023.07.04T11:21:00.log
/workspaces/handson/sa-aicast-tutorial-yolov5s/.venv/lib/python3.9/site-packages/hailo_sdk_client/scripts/c
heck_system_requirements.sh: line 297: column: command not found
System requirements check failed (see table above for details). Continue? [Y/n]
* 0 18 2 行 213、列 21 スペース: 4 UTF-8 LF Python 3.9.5 ('.venv': venv)
```

HARを作ろう

HARができた！

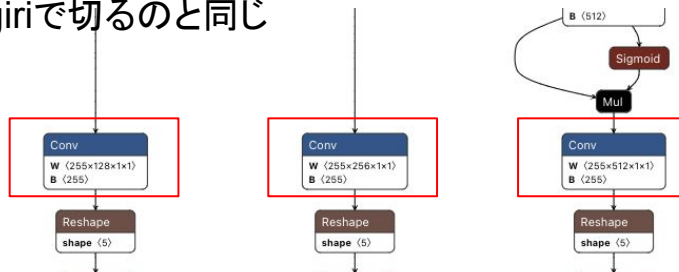
```
[warning] cannot use graphviz, so no visualizations will be created
[info] Translation completed on ONNX model yolov5s
[info] Initialized runner for yolov5s
[info] Saved Hailo Archive file at yolov5s.har
(.venv) developer@d51e60554029:/workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef$ ls
acceleras.log  hailo_sdk.client.log  yolov5s.har  yolov5s.onnx
(.venv) developer@d51e60554029:/workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef$
```

har作成にはONNXのグラフのうち入出力に指定したいノードを指定した。

--start-node-names images

--end-node-names /model.24/m.0/Conv /model.24/m.1/Conv /model.24/m.2/Conv

- 出力ノードを、ONNXの最後の出力にしなかったのは何故？
 - Hailoは4次元のReshapeに対応していない
 - これより後段の処理はPython側で処理する
 - nnoirの非対応ノードをonnigiriで切るのと同じ



NODE PROPERTIES

type Conv
name /model.24/m.2/Conv

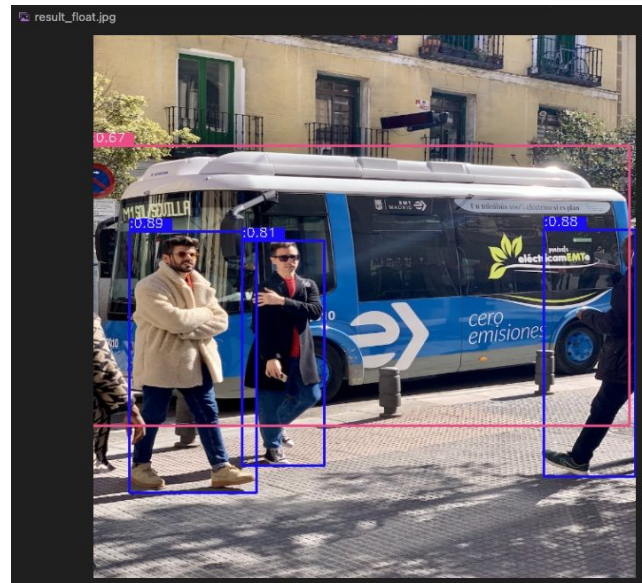
ATTRIBUTES

dilations 1, 1
group 1
kernel_shape 1, 1

HARをホストPCで実行する(float)

- Hailoが提供するEmulatorを用いてHARをホストPCで実行できる
 - Simulatorのほうが表現としては近い気がするが、公式の表現を使用
- 内部的にはHARをtensorflow graphとして実行している
- floatでの演算のため、ONNXと全く同じ結果に

```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s/host  
pip install opencv-python  
python3 test_model.py --target float
```



result_float.jpg

HARをホストPCで実行する 解説

- `net_eval()`はドキュメントからそのまま引用
- `target = SdkNative()`がfloatとして演算することを指定している

```
import cv2
from hailo_sdk_client import ClientRunner
from hailo_sdk_common.targets.inference_targets import SdkNative, SdkFPOptimized, SdkPartialNumeric, SdkNumeric
import tensorflow as tf

def net_eval(runner, target, images):
    with tf.Graph().as_default():
        network_input = tf.compat.v1.placeholder(dtype=tf.float32)
        sdk_export = runner.get_tf_graph(target, network_input)
        with sdk_export.session.as_default():
            sdk_export.session.run(tf.compat.v1.local_variables_initializer())
            probs_batch = sdk_export.session.run(sdk_export.output_tensors, feed_dict={network_input: images})
    return probs_batch

def test_float():
    model = "../make_hef/yolov5s.har"
    runner = ClientRunner(hw_arch='hailo8r', har_path=model)
    resized_img, input = get_input()
    input = input / 255.
    out0, out1, out2 = net_eval(runner, SdkNative(), input)
    result_img = postprocess(resized_img, out0, out1, out2)
    cv2.imwrite("result_float.jpg", result_img)
```

HARを量子化する

- 量子化の作業は時間がかかるので、今回のハンズオンでは行いません・・・
 - [sa-aicast-tutorial-yolov5s/precompiled/](#)にモデルをおいています
- 量子化を行う際はGPUがあることが望ましいです
- 作業工程を説明します

HARを量子化する

- 量子化プロセスでは「量子化パラメータ」を決定する
- 「キャリブレーション画像」が必要
- floatのモデルと出力を比較しながら、量子化パラメータを決定する

ここではimagenetの一部画像集 ([imagenette](https://s3.amazonaws.com/fast-ai-imageclas/imagenette2.tgz))を使うことにする

```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef
wget https://s3.amazonaws.com/fast-ai-imageclas/imagenette2.tgz
tar xvf imagenette2.tgz
python3 make_calib.py
```

```
sa-aicast-tutorial-yolov5s > make_hef > make_calib.py > ...
1  import numpy as np
2  import glob
3  from PIL import Image
4
5  images = []
6  for f in glob.glob('imagenette2/**/*.JPEG', recursive=True)[:1024]:
7      image = Image.open(f).convert('RGB').resize((640, 640))
8      images.append(np.asarray(image).astype('float32'))
9  np.save('calib_set.npy', np.stack(images, axis=0))
10
```

make_calib.pyは入力画像例集を直列化して numpyデータとして保存するだけ

HARを量子化する

キャリブレーションデータ・HAR・モデル最適化ファイル (alls)を指定して量子化実行

```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s/make_hef
hailo optimize yolov5s.har --calib-set-path calib_set.npy --model-script yolov5s.alls
--hw-arch hailo8r
```

yolov5s.allsの中身

Hailo側での正規化処理を指示

キャリブレーションサイズの指示

```
normalization1 = normalization([0.0, 0.0, 0.0], [255.0, 255.0, 255.0])
model_optimization_config(calibration, batch_size=4, calibset_size=1024)
quantization_param(output_layer1, precision_mode=a16_w16)
quantization_param(output_layer2, precision_mode=a16_w16)
quantization_param(output_layer3, precision_mode=a16_w16)
```

最終conv層のみ16bit量子化指定

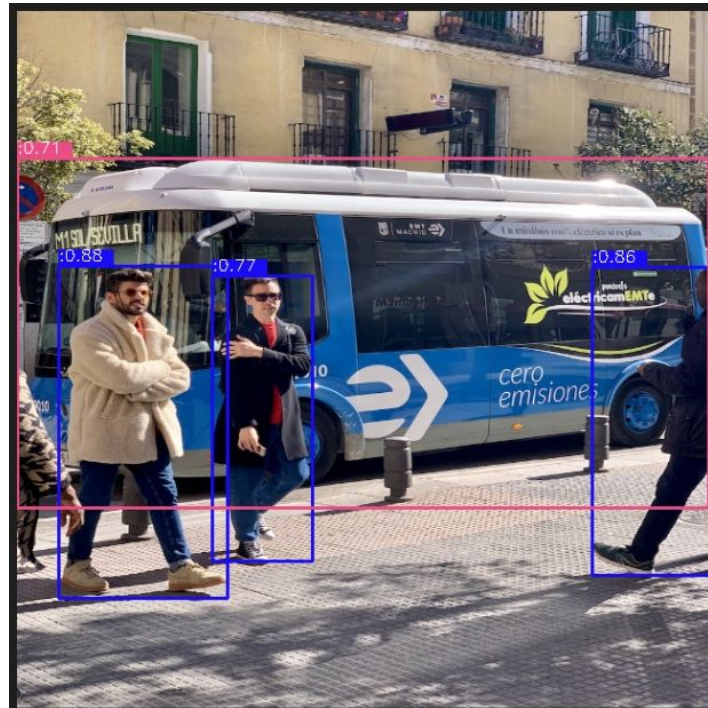
Dataflow Compilerのユーザガイドによると、最終層を 16bit量子化指定することで精度劣化が抑えられるらしい

より細かな最適化指示も可能: 参考 [hailo_model_zooのalls](#)

量子化されたHARをホストPCで実行する

```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s/host  
python3 test_model.py --target quantized
```

- 量子化されたモデルでも問題なく検出できている
- 精度検証の結果、問題がある場合は allsを修正して再度最適化をかける



result_quantized.jpg

量子化されたHARをホストPCで実行する

- test_quantized()とtest_floatの違い
 - target: 演算精度の違い
 - 正規化処理の有無
 - allsでHARファイル内に正規化処理を追加したので、quantizedではHARの外では正規化しない

```
def test_quantized():
    quantized_model = "../make_hef/yolov5s_quantized.har"
    runner = ClientRunner(hw_arch='hailo8r', har_path=quantized_model)
    resized_img, input = get_input()
    out0, out1, out2 = net_eval(runner, SdkNumeric(), input)
    result_img = postprocess(resized_img, out0, out1, out2)
    cv2.imwrite("result_quantized.jpg", result_img)

def test_float():
    model = "../make_hef/yolov5s.har"
    runner = ClientRunner(hw_arch='hailo8r', har_path=model)
    resized_img, input = get_input()
    input = input / 255.
    out0, out1, out2 = net_eval(runner, SdkNative(), input)
    result_img = postprocess(resized_img, out0, out1, out2)
    cv2.imwrite("result_float.jpg", result_img)
```

HARをHEFにコンパイルする

- いよいよHEFを作ります！
- コンパイルも時間がかかるので、今回のハンズオンでは行いません
 - `sa-aicast-tutorial-yolov5s/precompiled/`にモデルをおいています
- Hailoの計算資源にどのようにモデルの推論を実行するか、の資源割り当てを行う
 - 割り当て問題なので時間がかかるのは仕方ない

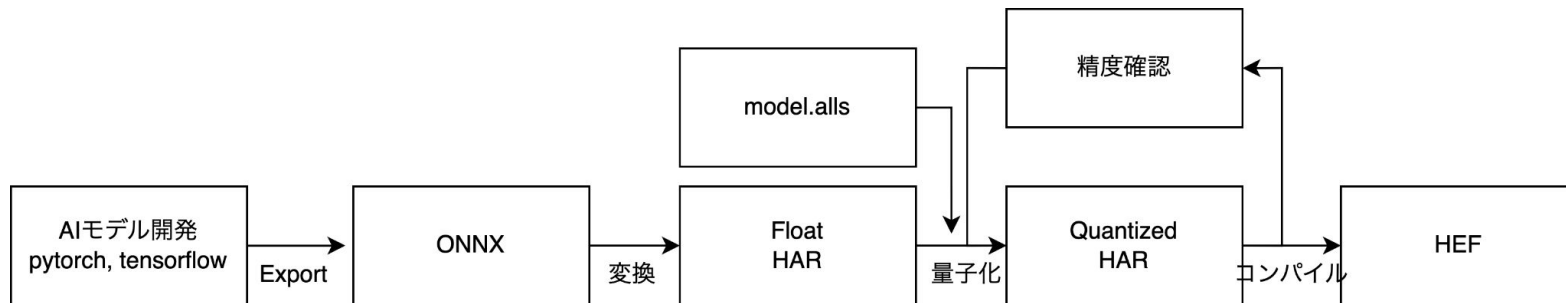
```
cd /workspaces/handson/sa-aicast-tutorial-yolov5s/make_hf
hailo compiler yolov5s_quantized.har --hw-arch hailo8r
```

[illegible]

コンパイルログ cherryで19minで完了

実践①まとめ

- Pytorch→ONNX→HAR→QuantizedHAR→HEFの順に変換/量子化/コンパイル
- HARはホストPC上で実行することができる
- 量子化にはキャリブレーション画像が必要
- 量子化・最適化オプションはallsファイルで設定可能

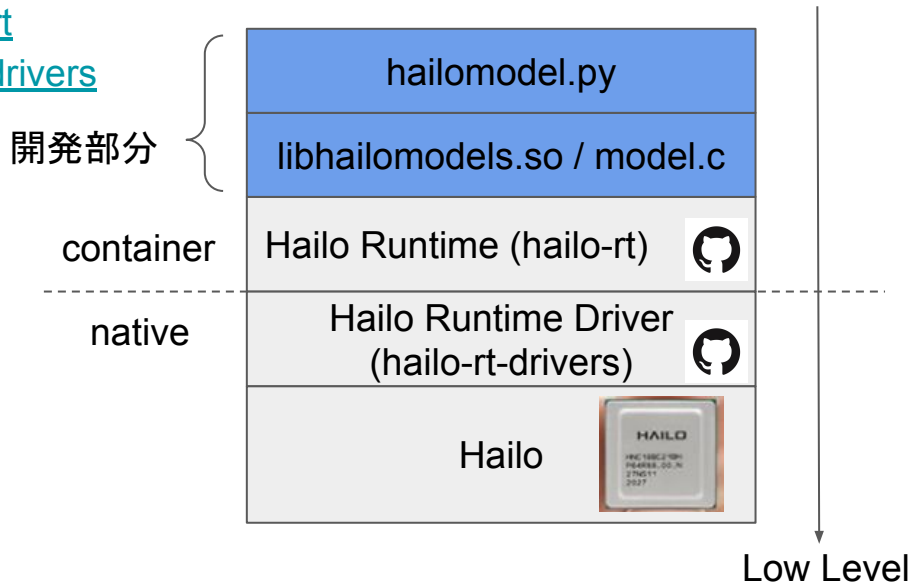


目次

- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動作させる仕組み
 - 実践2: 物体検出アプリケーションを作ろう
- その他開発Tips

aicastアプリでのSWスタック

- aicastアプリはPythonで開発する
- Pythonから各レイヤを通してHailoチップを駆動させている
- Runtime, DriverはHailo提供のものを使用する
 - Runtime: <https://github.com/hailo-ai/hailort>
 - Driver: <https://github.com/hailo-ai/hailort-drivers>



aicastアプリでのSWスタック

- Pythonからcdll経由での.so呼び出し

```
from ctypes import cdll
import numpy as np
```

```
class YoloX_Tiny:
```

```
    def __init__(self):
```

```
        self.lib = cdll.LoadLibrary('./yolox_tiny.so')
```

```
        self.lib.init()
```

```
    def __del__(self):
```

```
        self.lib.destroy()
```

```
    def infer(self, image):
```

```
        out0 = np.zeros((52, 52, 85), dtype=np.float32)
```

```
        out1 = np.zeros((26, 26, 85), dtype=np.float32)
```

```
        out2 = np.zeros((13, 13, 85), dtype=np.float32)
```

```
        self.lib.infer(
```

```
            image.ctypes.data,
```

```
            out0.ctypes.data,
```

```
            out1.ctypes.data,
```

```
            out2.ctypes.data
```

```
        )
```

```
        return out0, out1, out2
```

yolox_tiny.so



master aicast_model_zoo / yolox_tiny / edge / src / model.c

Blame 97 lines (78 loc) · 3.71 KB

```
void infer(unsigned char *input0, float *out0, float *out1, float *out2)
{
    hailo_status status = HAILO_UNINITIALIZED;
    status = hailo_vstream_write_raw_buffer(input_vstreams[0], input0, 416 * 416 * 3);
    assert(status == HAILO_SUCCESS);
    status = hailo_flush_input_vstream(input_vstreams[0]);
    assert(status == HAILO_SUCCESS);

    read_and_dequantize(0, out0, 52 * 52 * 85);
    read_and_dequantize(1, out1, 26 * 26 * 85);
    read_and_dequantize(2, out2, 13 * 13 * 85);
}
```


aicastアプリでのSWスタック

- hailo_vstream_write_raw_bufferなどはHailo Runtimeの呼び出し

```
from ctypes import cdll
import numpy as np
```

```
class YoloX_Tiny:
    def __init__(self):
        self.lib = cdll.LoadLibrary('./yolox_tiny.so')
        self.lib.init()
```

hailort_4.13.0_user_guide.pdf

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`status` `hailo_vstream_write_raw_buffer`(`hailo_input_vstream` `input_vstream`, `const void`

Writes buffer to hailo device via input virtual stream `input_vstream`.

Parameters

- `input_vstream` - [in] A `hailo_input_vstream` object.
- `buffer` - [in] A pointer to a buffer to be sent. The buffer format comes from the vstream's *format* (Can be obtained using `hailo_get_input_vstream_user_format`) and the shape comes from *shape* inside `hailo_vstream_info_t` (Can be obtained using `hailo_get_input_vstream_info`).

master aicast_model_zoo / yolox_tiny / edge / src / model.c

Blame 97 lines (78 loc) · 3.71 KB

```
void infer(unsigned char *input0, float *out0, float *out1, float *out2)
{
    hailo_status status = HAILO_UNINITIALIZED;
    status = hailo_vstream_write_raw_buffer(input_vstreams[0], input0, 416 * 416 * 3);
    assert(status == HAILO_SUCCESS);
    status = hailo_flush_input_vstream(input_vstreams[0]);
    assert(status == HAILO_SUCCESS);

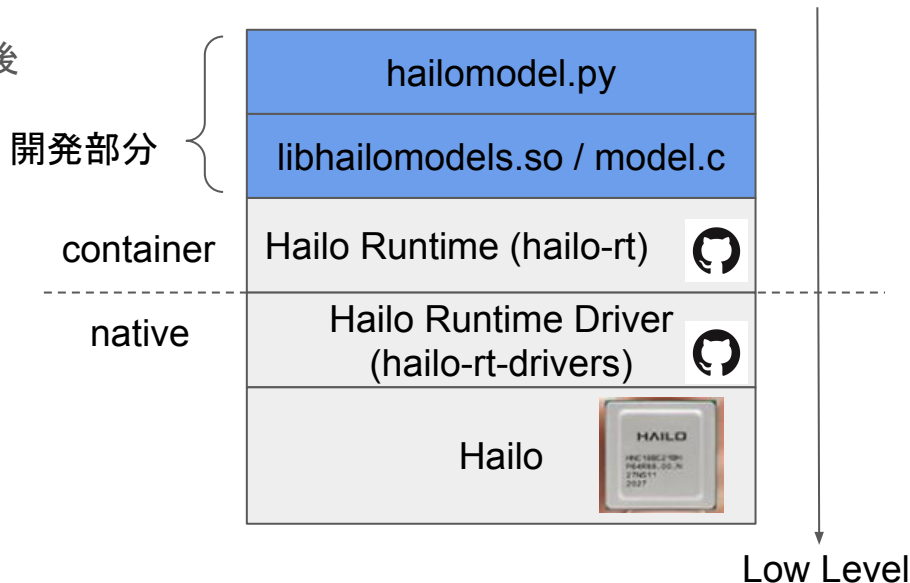
    read_and_dequantize(0, out0, 52 * 52 * 85);
    read_and_dequantize(1, out1, 26 * 26 * 85);
    read_and_dequantize(2, out2, 13 * 13 * 85);
}
```

aicastアプリでのSWスタック

- Runtimeはcontainerレベルで提供 = アプリに同梱する
 - root.tar をアプリに追加している理由
- Driverはnativeレベルでの提供 = Writerレベルで提供する
 - aicast専用writerが用意されている理由
 - raspiと共通のactsimイメージを書き込んだ後ドライバをインストールした理由

● Hailo ドライバをインストールするため以下を実行します。

```
$ sudo apt install git
$ sudo apt install raspberrypi-kernel-headers
$ git clone https://github.com/hailo-ai/hailort-drivers.git
$ cd hailort-drivers
$ git checkout v4.10.0
$ cd linux/pcie
$ sed -i "s/raspi/raspberrypi/g" Kbuild
$ make all
$ sudo make install
$ sudo modprobe hailo_pcie
$ cd ../../
$ ./download_firmware.sh
$ sudo mkdir /lib/firmware/hailo
$ sudo mv hailo8_fw.4.10.0.bin /lib/firmware/hailo/hailo8_fw.bin
$ sudo cp ./linux/pcie/51-hailo-udev.rules /etc/udev/rules.d/
```

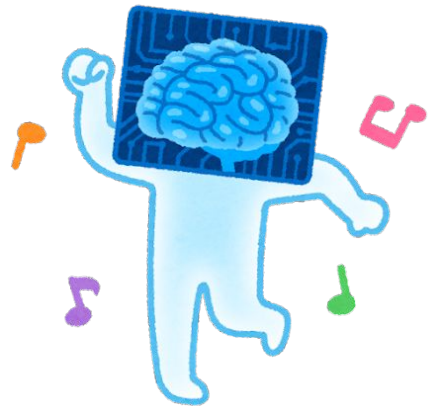


目次

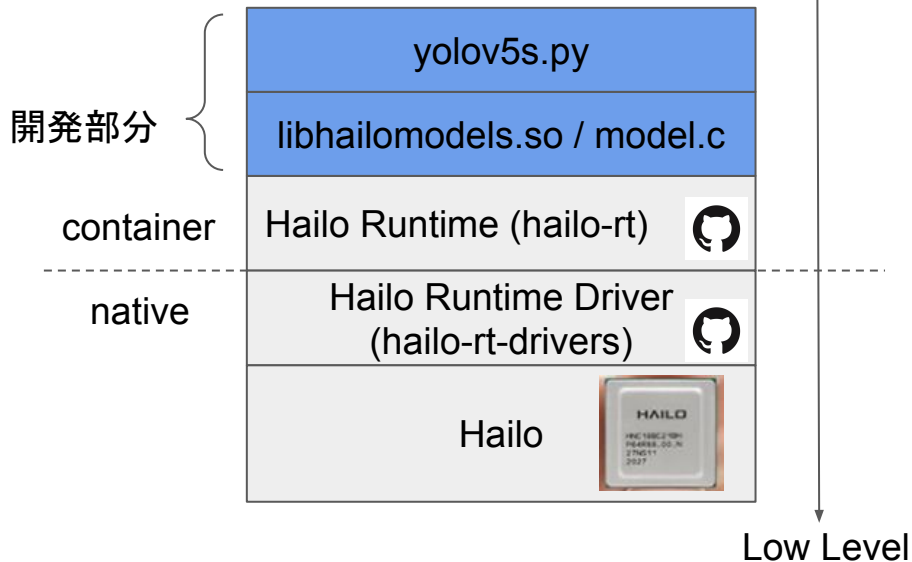
- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動作させる仕組み
 - 実践2: 物体検出アプリケーションを作ろう
- その他開発Tips

実践② 物体検出アプリを作ろう

事前準備: actsim, actdkのインストールが終わっていることが前提です



1. 物体検出アプリの動作確認
2. model.cの解説
3. yolov5s.pyの解説
4. 後処理の移植の解説
5. Dockerfile, Makefile, root.tarの解説
6. アプリ全体解説



物体検出アプリの動作確認

actdkのインストールされたホスト PC上で実行

```
git clone https://github.com/ldein/sa-aicast-tutorial-yolov5s.git
cd sa-aicast-tutorial-yolov5s
cp ../precompiled/yolov5s.hef ./app/
```

aicastのactdk remoteへの登録が終わっていない場合は登録

```
actdk remote add aicast@192.168.xx.xx
```

model.cのコンパイル

```
make
```

アプリのコンテナのビルド

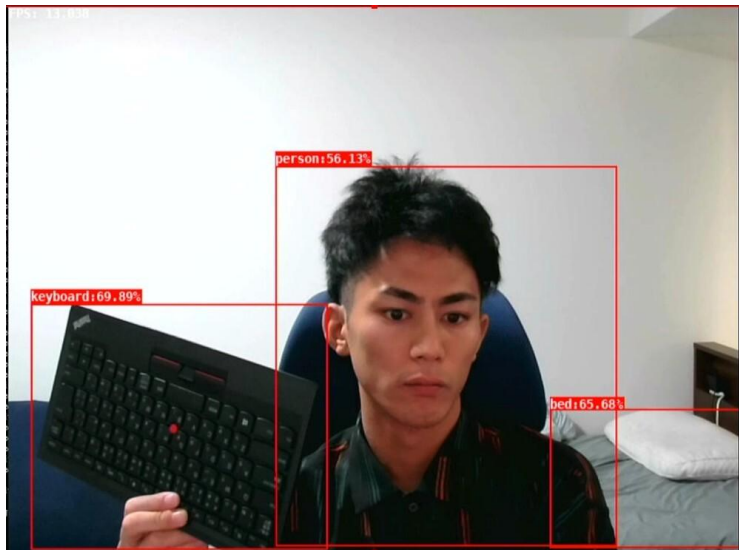
```
actdk build aicast
```

物体検出アプリの動作確認

アプリ実行

```
actdk run aicast
```

動いた！



model.cの解説

- Pythonから呼ばれる関数は3つ
- `init()`: アプリケーションの最初に1度だけ呼ばれる
- `infer()`: 推論1回につき1回呼ばれる
- `destroy()`: アプリケーションの終了時に1度だけ呼ばれる

```
class YOLOv5s:
    def __init__(self, n_class: int, conf_thresh: float, input_img_size=Tuple[int, int]):
        lib_path = os.path.join(os.path.dirname(
            os.path.abspath(__file__)), "../libhailomodels.so")
        self.lib = cdll.LoadLibrary(lib_path)
        self.lib.init()
        self.out0 = np.zeros((80, 80, 3, n_class + 5), dtype=np.float32)
        self.out1 = np.zeros((40, 40, 3, n_class + 5), dtype=np.float32)
        self.out2 = np.zeros((20, 20, 3, n_class + 5), dtype=np.float32)
        self.yolo_layer = YoloLayer(
            n_class=n_class, in_size=640, conf_thresh=conf_thresh, apply_sigmoid=True)
        self.lb_decoder = LetterBoxDecoder(input_img_size, (640, 640))

    def __del__(self):
        self.lib.destroy()

    def preproc(self, image: Image) -> np.ndarray:
        lb_img = self.lb_decoder.get_letterbox_image(image)
        return np.asarray(lb_img)

    def infer(self, input: np.ndarray):
        self.lib.infer(
            input.ctypes.data,
            self.out0.ctypes.data,
            self.out1.ctypes.data,
            self.out2.ctypes.data)

        return self.out0, self.out1, self.out2
```

model.cの解説

- init
- Hailoデバイスの初期化・入出力ストリームの初期化・ネットワーク有効化
 - 基本的に全アプリケーションでこのままコピペで使えます

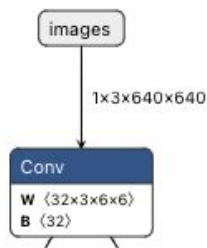
```
#define HEF_FILE ("yolov5s.hef")
int init()
{
    hailo_status status = HAILO_UNINITIALIZED;

    status = hailo_create_vdevice(NULL, &vdevice);
    status = hailo_create_hef_file(&hef, HEF_FILE);
    status = hailo_init_configure_params(hef, HAILO_STREAM_INTERFACE_PCIE, &config_params);
    status = hailo_configure_vdevice(vdevice, hef, &config_params, &network_group, &network_group_size);
    status = hailo_make_input_vstream_params(network_group, true, HAILO_FORMAT_TYPE_AUTO,
        input_vstream_params, &input_vstreams_size);
    status = hailo_make_output_vstream_params(network_group, true, HAILO_FORMAT_TYPE_AUTO,
        output_vstream_params, &output_vstreams_size);
    status = hailo_create_input_vstreams(network_group, input_vstream_params, input_vstreams_size, input_vstreams);
    status = hailo_create_output_vstreams(network_group, output_vstream_params, output_vstreams_size, output_vstreams);
    status = hailo_activate_network_group(network_group, NULL, &activated_network_group);

    for (size_t i = 0; i < output_vstreams_size; i++)
        hailo_get_output_vstream_info(output_vstreams[i], &output_vstreams_info[i]);
    return status;
}
```


model.cの解説

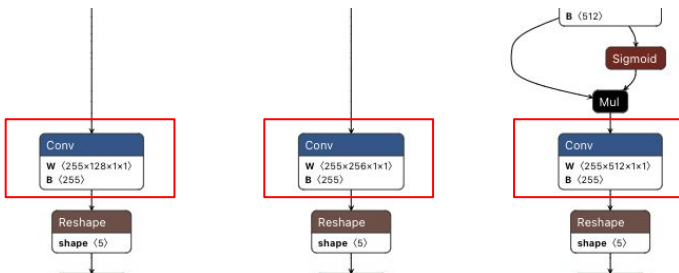
- infer: 推論の実行
- モデルに応じて書き換える



入力は3*640*640

```
#define READ_AND_DEQUANTIZE(idx, type, out, size) \
do { \
    type buf[size]; \
    hailo_status status = HAILO_UNINITIALIZED; \
    status = hailo_vstream_read_raw_buffer(output_vstreams[idx], buf, size * sizeof(type)); \
    assert(status == HAILO_SUCCESS); \
    float scale = output_vstreams_info[idx].quant_info.qp_scale; \
    float zp = output_vstreams_info[idx].quant_info.qp_zp; \
    for (int i = 0; i < size; i++) \
        *out++ = scale * (buf[i] - zp); \
} while (0)

void infer(
    unsigned char *input0,
    float *pred80,
    float *pred40,
    float *pred20)
{
    hailo_status status = HAILO_UNINITIALIZED;
    status = hailo_vstream_write_raw_buffer(input_vstreams[0], input0, 640*640*3);
    status = hailo_flush_input_vstream(input_vstreams[0]);
    READ_AND_DEQUANTIZE(0, uint16_t, pred80, 80*80*255);
    READ_AND_DEQUANTIZE(1, uint16_t, pred40, 40*40*255);
    READ_AND_DEQUANTIZE(2, uint16_t, pred20, 20*20*255);
}
```



NODE PROPERTIES

type Conv
name /model.24/m.2/Conv

ATTRIBUTES

dilations 1, 1
group 1
kernel_shape 1, 1

出力は80*80*255, 40*40*255, 20*20*255

model.cの解説

- READ_AND_DEQUANTIZE()
 - 量子化された値をfloatに戻している
 - バッファのメモリ列を区切る型情報が必要

allsで指定した通りに出力レイヤの
型 uint16_tを指定

```
#define READ_AND_DEQUANTIZE(idx, type, out, size) \  
do { \  
    type buf[size]; \  
    hailo_status status = HAILO_UNINITIALIZED; \  
    status = hailo_vstream_read_raw_buffer(output_vstreams[idx], buf, size * sizeof(type)); \  
    assert(status == HAILO_SUCCESS); \  
    float scale = output_vstreams_info[idx].quant_info.qp_scale; \  
    float zp = output_vstreams_info[idx].quant_info.qp_zp; \  
    for (int i = 0; i < size; i++) \  
        *out++ = scale * (buf[i] - zp); \  
} while (0)  
  
void infer(  
    unsigned char *input0,  
    float *pred80,  
    float *pred40,  
    float *pred20)  
{  
    hailo_status status = HAILO_UNINITIALIZED;  
    status = hailo_vstream_write_raw_buffer(input_vstreams[0], input0, 640*640*3);  
    status = hailo_flush_input_vstream(input_vstreams[0]);  
    READ_AND_DEQUANTIZE(0, uint16_t, pred80, 80*80*255);  
    READ_AND_DEQUANTIZE(1, uint16_t, pred40, 40*40*255);  
    READ_AND_DEQUANTIZE(2, uint16_t, pred20, 20*20*255);  
}
```

normalization1 = normalization([0.0, 0.0, 0.0], [255.0, 255.0, 255.0])

model_optimization_config(calibration, batch_size=4, calibset_size=1024)

quantization_param(output_layer1, precision_mode=a16_w16) 最終conv層のみ16bit量子化指定

quantization_param(output_layer2, precision_mode=a16_w16)

quantization_param(output_layer3, precision_mode=a16_w16)

model.cの解説

- 入出力の型は基本的にそのまま
- 入力: unsigned char (uint8)
 - 8bit RGBの画像をそのまま渡す
- 出力: float
 - 逆量子化することでGPUと同じように後処理で値を扱える

```
#define READ_AND_DEQUANTIZE(idx, type, out, size) \  
do { \  
    type buf[size]; \  
    hailo_status status = HAILO_UNINITIALIZED; \  
    status = hailo_vstream_read_raw_buffer(output_vstreams[idx], buf, size * sizeof(type)); \  
    assert(status == HAILO_SUCCESS); \  
    float scale = output_vstreams_info[idx].quant_info.qp_scale; \  
    float zp = output_vstreams_info[idx].quant_info.qp_zp; \  
    for (int i = 0; i < size; i++) \  
        *out++ = scale * (buf[i] - zp); \  
} while (0)  
  
void infer(  
    unsigned char *input0,  
    float *pred80,  
    float *pred40,  
    float *pred20)  
{  
    hailo_status status = HAILO_UNINITIALIZED;  
    status = hailo_vstream_write_raw_buffer(input_vstreams[0], input0, 640*640*3);  
    status = hailo_flush_input_vstream(input_vstreams[0]);  
    READ_AND_DEQUANTIZE(0, uint16_t, pred80, 80*80*255);  
    READ_AND_DEQUANTIZE(1, uint16_t, pred40, 40*40*255);  
    READ_AND_DEQUANTIZE(2, uint16_t, pred20, 20*20*255);  
}
```

model.cの解説

- destroy
- Hailoデバイスの解放・入出カストリームの解放・ネットワーク無効化
 - 基本的に全アプリケーションでこのままコピペで使えます

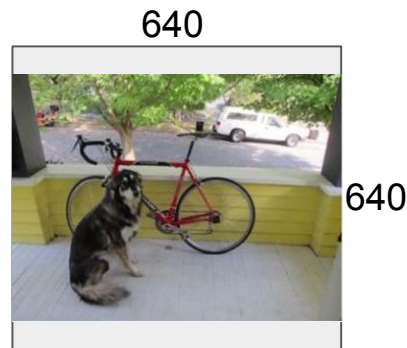
```
void destroy() {  
    (void) hailo_deactivate_network_group(activated_network_group);  
    (void) hailo_release_output_vstreams(output_vstreams, output_vstreams_size);  
    (void) hailo_release_input_vstreams(input_vstreams, input_vstreams_size);  
    (void) hailo_release_hef(hef);  
    (void) hailo_release_vdevice(vdevice);  
}
```

yolov5s.pyの解説

app/model/yolov5s.py

- 前処理

```
def preproc(self, image: Image) -> np.ndarray:  
    lb_img = self.lb_decoder.get_letterbox_image(image)  
    return np.asarray(lb_img)
```



yoloでよく使用される Letterbox画像化

- 推論

```
def infer(self, input: np.ndarray):  
    self.lib.infer(  
        input.ctypes.data,  
        self.out0.ctypes.data,  
        self.out1.ctypes.data,  
        self.out2.ctypes.data)  
  
    return self.out0, self.out1, self.out2
```

```
self.out0 = np.zeros((80, 80, 3, n_class + 5), dtype=np.float32)  
self.out1 = np.zeros((40, 40, 3, n_class + 5), dtype=np.float32)  
self.out2 = np.zeros((20, 20, 3, n_class + 5), dtype=np.float32)
```

numpy配列の .ctypes.data でポインタを .so に渡す

yolov5s.pyの解説

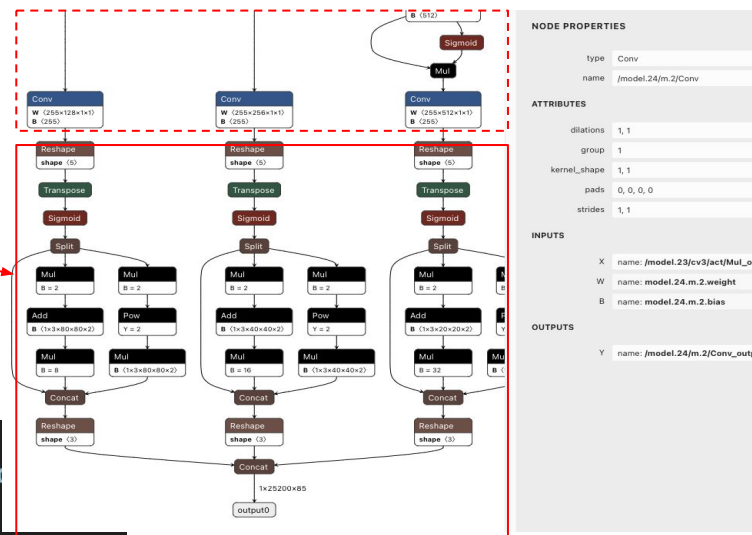
app/model/yolov5s.py

- 後処理

```
def postprocess(self, outs: List[np.ndarray]):  
    bboxes, scores, classes = self.yolo_layer.run(outs)  
    bboxes, scores, classes = nms(  
        bboxes, scores, classes, per_class=True, iou_threshold=0.45)  
    bboxes = self.lb_decoder.decode_box_letter_to_orig(bboxes)  
    return bboxes, scores, classes
```

```
self.yolo_layer = YoloLayer(  
    n_class=n_class, in_size=640, conf_thresh=conf_thresh, apply_sigmoid=True)  
self.lb_decoder = LetterBoxDecoder(input_img_size, (640, 640))
```

ONNXからHARを作る時に、削った部分の処理はPythonで実装



その他、NMS(Non-Max-Suppression), bboxの座標変換

YOLOLayer()が処理するレイヤ

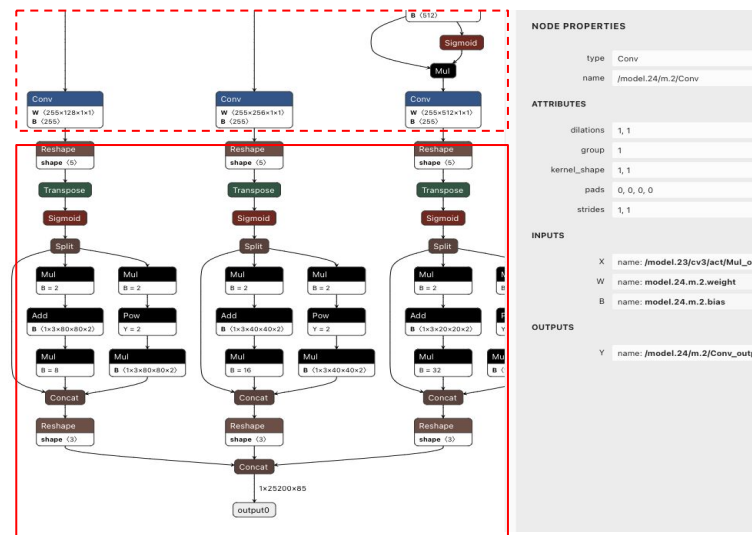
後処理の移植の解説

app/model/yolo_layer.py

- numpyで実装

```
You, 5 時間前 | 1 autor (you)
class YoloLayer():
    """
    Fast implementation of yolo layer.
    skip coordinate and confidence calculation for bboxes with confidence under threshold.

    apply_sigmoid: when model alls script contains 'change_output_activation(sigmoid)', set False
    """
    def __init__(
        self,
        n_class,
        conf_thresh,
        apply_sigmoid=True,
        in_size=640,
        output_sizes=[[80, 80], [40, 40], [20, 20]],
        anchors=[
            [10,13, 16,30, 33,23],
            [30,61, 62,45, 59,119],
            [116,90, 156,198, 373,326],
        ],
        n_max_output_bbox=30000
    ):
        self.n_class = n_class
        self.conf_thresh = conf_thresh
        self.activate_fn = sigmoid if apply_sigmoid else (lambda x: x)
        self.inv_conf_thres = np.log(conf_thresh/(1-conf_thresh))
        self.n_output = len(output_sizes)
        self.anchors = np.array(anchors, dtype=np.float32).reshape(self.n_output, self.n_output, 2)
        self.stride = [in_size // o[0] for o in output_sizes]
        self.grid = [
            self.make_grid(nx, ny)
            for nx, ny in output_sizes
        ]
```



全ての計算を愚直に行うのではなく、閾値を超えるbboxについてのみ計算
(onnigiri+onnion runtimeを使うより高速)

後処理の移植の解説

Hailo Model Zooのモデルを使用する時は注意！

- `change_output_activation()`によりHEFの最終層にsigmoidが追加される
- Python製後処理ではsigmoidをしない

```
17     apply_sigmoid=True,  
18     in_size=640,  
19     output_sizes=[[80, 80], [40, 40], [20, 20]],  
20     anchors=[  
21         [10,13, 16,30, 33,23],  
22         [30,61, 62,45, 59,119],  
23         [116,90, 156,198, 373,326],  
24     ],  
25     n_max_output_bbox=30000  
26 ):  
27     self.n_class = n_class  
28     self.conf_thresh = conf_thresh  
29     self.activate_fn = sigmoid if apply_sigmoid else (lambda x: x)
```



master

[hailo_model_zoo](#) / [hailo_model_zoo](#) / [cfg](#) / [alls](#) / [generic](#) / [yolov5s_personface.alls](#)



HailoModelZoo update-to-version-2.8.0

Code

Blame

4 lines (4 loc) · 474 Bytes

```
1     normalization1 = normalization([0.0, 0.0, 0.0], [255.0, 255.0, 255.0])  
2     change_output_activation(sigmoid)  
3     nms_postprocess("$HMZ_DATA/models_files/HailoNets/MCPReID/personface_detector/yolov5s_personface/.  
4     post_quantization_optimization(finetune, policy=enabled, learning_rate=0.0001, epochs=4, dataset_
```


Dockerfile, Makefileの解説

- model.cのクロスコンパイル
- actdk uploadしてもリモートビルドではmakeは実行されないなので、手元で実行してからupload
 - [actcast-app-idein-ci-orb](#)に準じてCIを設定すればCIでは実行される
 - 参考: [新規アプリケーション作成時のCI設定方法](#)

```
Dockerfile
You, 15 時間前 | 1 author (You)
1 FROM idein/cross-rpi:armv6-slim
2
3 ENV SYSROOT /home/idein/x-tools/armv6-rpi-linux-gnueabi
4
5 ADD root.tar $SYSROOT
6 RUN sudo rm $SYSROOT/usr/lib/libhailort.so
7 RUN sudo ln -s $SYSROOT/usr/lib/libhailort.so.4.10.0 $S

Makefile
You, 1 秒前 | 1 author (You)
1 SOURCE=model.c
2 TARGET=libhailomodels.so
3
4 all: app/${TARGET}
5
6 app/${TARGET}: src/${SOURCE}
7     docker build -t cross-rpi .
8     docker run -it --rm -d --name cross cross-rpi /bin/bash
9     docker cp src cross:/home/idein/src
10    docker exec -it cross armv6-rpi-linux-gnueabihf-gcc -W -Wall
11    docker cp cross:/home/idein/${TARGET} app/${TARGET}
12    docker stop cross
13
```

root.tarの解説

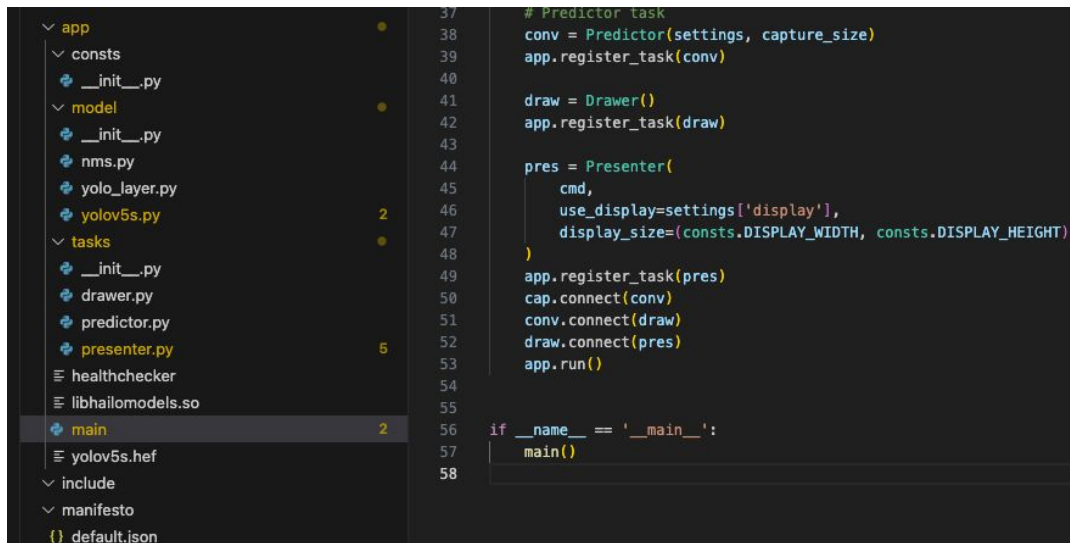
- アプリコンテナ内のrootに展開されるファイル
- actdkの隠し仕様・ファイルシステムに追加したい場合に使用
 - 基本的にそのままでもいい
 - Hailo Runtime v4.10が同梱

root.tarの中身

```
(.venv) lp6m-m1mac@lp6m ~/idein/sa-aicast-tutorial-yolov5s/aicast_app$ tar -tf root.tar
./share/opt/hailo/linux/pcie/build/include/hailo_pcie_version.h
./lib/firmware/hailo/hailo8_fw.bin
./lib/systemd/system/hailort.service
./usr/include/hailo/
./usr/include/hailo/hailort.hpp
./usr/include/hailo/vdevice.hpp
./usr/include/hailo/vstream.hpp
./usr/include/hailo/hailort.h
./usr/include/hailo/platform.h
./usr/include/hailo/event.hpp
./usr/include/hailo/hef.hpp
./usr/include/hailo/hailort_common.hpp
./usr/include/hailo/stream.hpp
./usr/include/hailo/quantization.hpp
./usr/include/hailo/network_group.hpp
./usr/include/hailo/expected.hpp
./usr/include/hailo/buffer.hpp
./usr/include/hailo/device.hpp
./usr/include/hailo/network_rate_calculator.hpp
./usr/include/hailo/inference_pipeline.hpp
./usr/include/hailo/runtime_statistics.hpp
./usr/include/hailo/transform.hpp
./usr/include/hailo/hailort.hpp
./usr/include/hailo/hailort.h
./usr/include/hailo/hailort_common.hpp
./usr/bin/hailortcli
./usr/lib/libhailort.so
./usr/lib/libhailort.so.4.10.0
./usr/local/bin/hailort_service
./etc/udev/rules.d/51-hailo-udev.rules
./lib/modules/5.4.83-v7l+/modules.symbols
./lib/modules/5.4.83-v7l+/modules.alias
./lib/modules/5.4.83-v7l+/modules.dep.bin
./lib/modules/5.4.83-v7l+/modules.builtin
./lib/modules/5.4.83-v7l+/modules.builtin.modinfo
./lib/modules/5.4.83-v7l+/modules.order
./lib/modules/5.4.83-v7l+/modules.softdep
./lib/modules/5.4.83-v7l+/modules.builtin.bin
./lib/modules/5.4.83-v7l+/modules.devname
./lib/modules/5.4.83-v7l+/kernel/drivers/misc/hailo_pcie.ko
```

アプリ全体の解説

- 普通のactcastアプリと同じ
 - cap: カメラキャプチャ
 - predictor: yolov5s推論
 - drawer: 結果描画
 - presenter: 結果画面表示



The screenshot shows a code editor with a file explorer on the left and Python code on the right. The file explorer lists the following files and folders:

- app
 - consts
 - __init__.py
 - model
 - __init__.py
 - nms.py
 - yolo_layer.py
 - yolov5s.py
 - tasks
 - __init__.py
 - drawer.py
 - predictor.py
 - presenter.py
 - healthchecker
 - libhailomodels.so
 - main
 - yolov5s.hef
 - include
 - manifesto
 - default.json

The Python code on the right is as follows:

```
37
38 # Predictor task
39 conv = Predictor(settings, capture_size)
40 app.register_task(conv)
41
42 draw = Drawer()
43 app.register_task(draw)
44
45 pres = Presenter(
46     cmd,
47     use_display=settings['display'],
48     display_size=(consts.DISPLAY_WIDTH, consts.DISPLAY_HEIGHT)
49 )
50 app.register_task(pres)
51 cap.connect(conv)
52 conv.connect(draw)
53 draw.connect(pres)
54 app.run()
55
56 if __name__ == '__main__':
57     main()
58
```

実践②まとめ

- 普通のactcastアプリ開発と異なるところ
 - src/model.cの実装
 - infer()の部分はモデルの入出力に応じて変更する
 - Dockerfile, Makefile, root.tar
 - 既存aicastアプリからコピーしてこればOK

目次

- aicast・Hailoの紹介
- Hailoで動作するAIモデルについて
 - 実践1: HEFを作ろう
- aicastアプリからHailoを動作させる仕組み
 - 実践2: 物体検出アプリケーションを作ろう
- その他開発Tips

aicast_model_zooの紹介

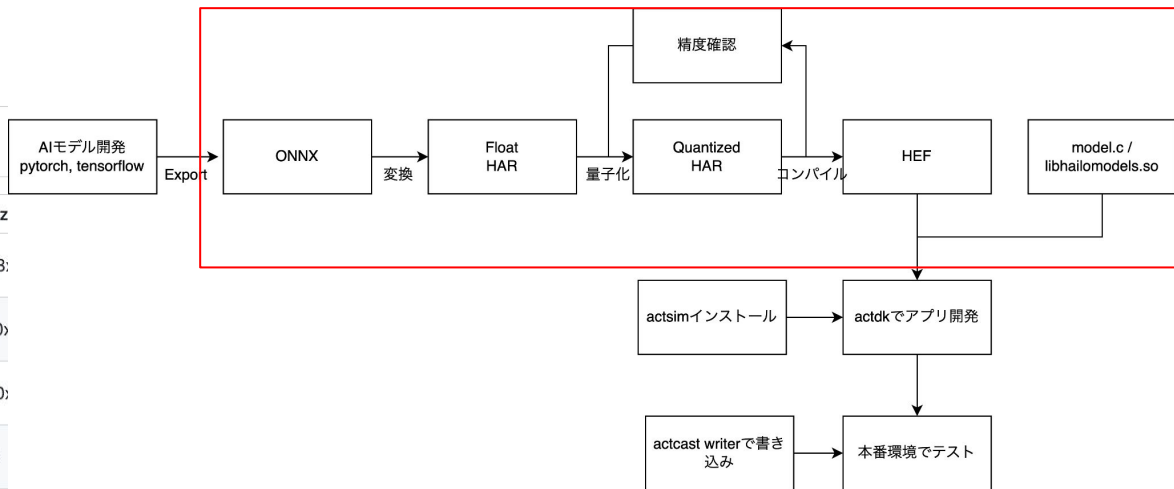
- cloneしてすぐに使えるモデル一覧

aicast_model_zooにより削減される作業

☰ README.md

aicast_model_zoo

name	description	latency[ms]	FPS	input siz	
body_yolov7	人物の身体検出	71.11	13.72	448x448:	
centerface	顔+5キーポイント検出	4.30	22.13	256x320>	
crowdhuman_yolov5m	人物+頭部検出	60.60	15.08	640x640:	
face_mask_judge	顔マスク判定	1.20	833.30	32x32x1	
face_yolov7	人物の顔検出	82.80	11.78	640x640x3	https://github.com/WongKin
head_yolov7	人物の頭部検出	75.17	13.00	448x448x3	https://github.com/WongKin
mm_mobilenetv2	COCO姿勢推定	3.90	161.00	256x196x3	https://github.com/open-mmlab/mmpose/
yolox_tiny	COCO一般物体検出	22.36	41.09	416x416x3	https://github.com/Megvii-BaseDetection/YOLOX/



PRお待ちしております

HEFの入出力情報を見る

- src/model.cでコメントアウトされていた部分
 - 入出力について
 - shape, 型, 次元orderを確認できる
 - うまく実行できない時は確認する

```
for (size_t i = 0; i < input_vstreams_size; i++) {
    hailo_vstream_info_t input_vstreams_info;
    hailo_get_input_vstream_info(input_vstreams[i], &input_vstreams_info);
    printf("==== input[%d] =====\n", i);
    printf("direction: %d\n", input_vstreams_info.direction);
    printf("format.type: %d\n", input_vstreams_info.format.type);
    printf("format.order: %d\n", input_vstreams_info.format.order);
    printf("format.flags: %d\n", input_vstreams_info.format.flags);
    printf("height: %d\n", input_vstreams_info.shape.height);
    printf("width: %d\n", input_vstreams_info.shape.width);
    printf("features: %d\n", input_vstreams_info.shape.features);
    printf("qp_zp: %f\n", input_vstreams_info.quant_info.qp_zp);
    printf("qp_scale: %f\n", input_vstreams_info.quant_info.qp_scale);
}
printf("=====\n");

for (size_t i = 0; i < output_vstreams_size; i++) {
    printf("==== output[%d] =====\n", i);
    printf("direction: %d\n", output_vstreams_info[i].direction);
```



```
==== input[0] ====
direction: 0
format.type: 1
format.order: 1
format.flags: 1
height: 640
width: 640
features: 3
qp_zp: 0.000000
qp_scale: 1.000000

==== output[0] ====
direction: 1
format.type: 2
format.order: 3
format.flags: 1
height: 80
width: 80
features: 255
qp_zp: 24074.000000
qp_scale: 0.000918

==== output[1] ====
direction: 1
format.type: 2
format.order: 3
format.flags: 1
height: 40
width: 40
features: 255
qp_zp: 23004.000000
qp_scale: 0.000833

==== output[2] ====
direction: 1
format.type: 2
format.order: 3
format.flags: 1
height: 20
width: 20
features: 255
qp_zp: 21529.000000
qp_scale: 0.000759
```

format.typeやformat.orderはドキュメントで各値が指す意味を確認

hailortのhailo_format_order_t enum

```
556 typedef enum {
557     /**
558      * Chosen automatically to match the format expected by the device.
559      */
560     HAILO_FORMAT_ORDER_AUTO = 0,
561
562     /**
563      * - Host side: [N, H, W, C]
564      * - Device side: [N, H, W, C], where width is padded to 8 elements
565      */
566     HAILO_FORMAT_ORDER_NHWC = 1,
567
568     /**
569      * - Not used for host side
570      * - Device side: [N, H, C, W], where width is padded to 8 elements
571      */
572     HAILO_FORMAT_ORDER_NHCW = 2,
573 }
```

ドキュメントでも確認できる

enum hailo_format_order_t

Data format orders, i.e. how the rows, columns and features are ordered:

- N: Number of images in the batch
- H: Height of the image
- W: Width of the image
- C: Number of channels of the image (e.g. 3 for RGB, 1 for grayscale...)

Values:

enumerator HAILO_FORMAT_ORDER_AUTO

Chosen automatically to match the format expected by the device.

enumerator HAILO_FORMAT_ORDER_NHWC

- Host side: [N, H, W, C]
- Device side: [N, H, W, C], where width is padded to 8 elements

enumerator HAILO_FORMAT_ORDER_NHCW

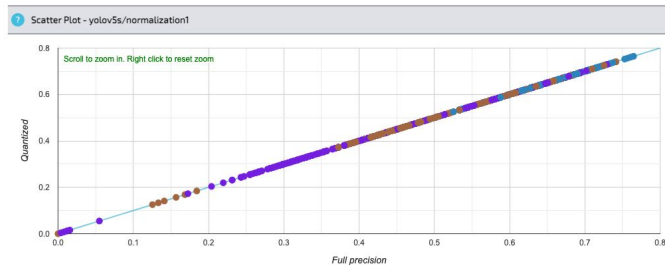
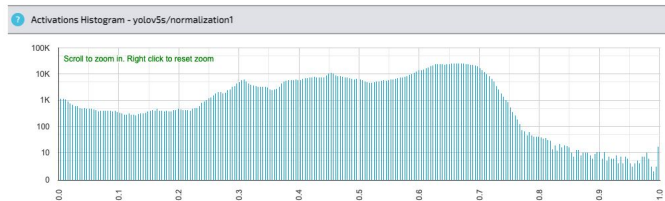
- Not used for host side
- Device side: [N, H, C, W], where width is padded to 8 elements

Profilerの使用

- モデルの入出力・計算量・メモリ使用量・最適化結果など詳細なレポートを見れる

hailo profiler yolov5s_quantized.har

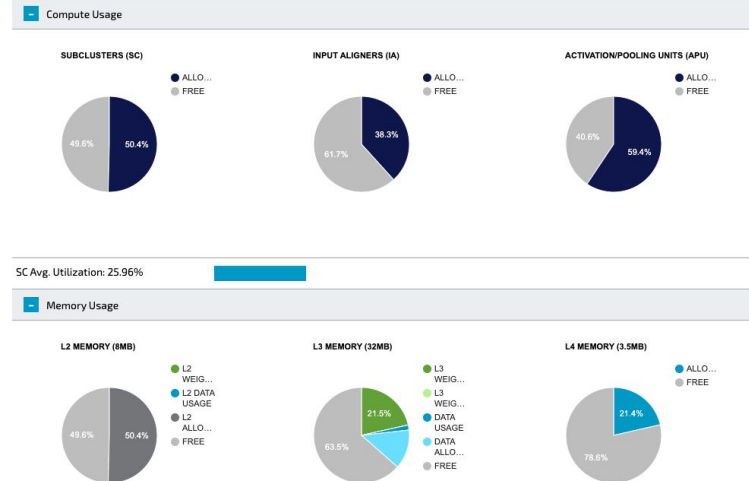
- ただしかなり時間がかかります
- 結果はHTML形式で出力されます



Detailed Device Utilization

Number of contexts used for allocation: 1

Total Utilization, context: context_0



その他開発関係Idein内部資料

- ImageNet Classification Tutorial
 - [actcast-app-hailo-imagenet-classification](#)
- [aicast PJ \(bizdev\)](#)
 - 毎週木曜定例をやっています
- [aicastで2つ以上モデルを実行する](#)
 - model.cを改変して複数実行に対応させます

Hailo公式資料

- 今回の内容はHailo公式ドキュメントに基づいています。
 - ※ Developer ZoneのFullアクセスでアカウント申請が必要
 - Dataflow Compiler, Runtimeのドキュメントに一度目を通すと良い

The screenshot shows the Hailo Developer Zone Documentation page. The header includes the Hailo logo and navigation links for Products, Industries, Resources, Company, and Partners. A search bar and links for Developer Zone, Sign In, and English are also present. The main navigation bar highlights Documentation, with links for SW Downloads, Tutorials, and Model Explorer. The page title is 'Hailo Documentation and Datasheets' with a search bar. Below the title, there's a breadcrumb trail: Home Page > Developer Zone > Documentation. A 'Latest Documents' button is visible, along with an 'Archive' button and an 'Open a Ticket' button. A 'PLEASE NOTE' section advises users to download and use the latest Hailo Software Suite for best consistency. At the bottom, there's a table of documents with columns for Title, File type, Version, and Date modified. The table lists three documents: 'Hailo-15™ Product Brief' (PDF, Rev01, March 8, 2023), 'Hailo Software Suite User Guide' (HTML, 2023-07, July 1, 2023), and 'Hailo Dataflow Compiler User Guide' (HTML, 2023-07, July 1, 2023).

Hailo

PRODUCTS INDUSTRIES RESOURCES COMPANY PARTNERS

DEVELOPER ZONE SIGN IN ENGLISH

SW Downloads Documentation Tutorials Model Explorer

Hailo Documentation and Datasheets

Search anything...

Home Page > Developer Zone > Documentation

Latest Documents Archive Open a Ticket

PLEASE NOTE
We highly recommend downloading and using only the latest Hailo Software Suite for best consistency across products and versions. Only Hailo Software Suite versions are supported for commercial products.

All Products

HW

SW

Select document type Select file type

Title	File type	Version	Date modified
Hailo-15™ Product Brief	PDF	Rev01	March 8, 2023
Hailo Software Suite User Guide	HTML	2023-07	July 1, 2023
Hailo Dataflow Compiler User Guide	HTML	2023-07	July 1, 2023