Empowering Intelligence

Model Zoo User Guide

Release v2.8.0

29 June 2023

Table of Contents

Disclaimer and Proprietary Information Notice

Copyright

This version of the document supersedes all previous versions.

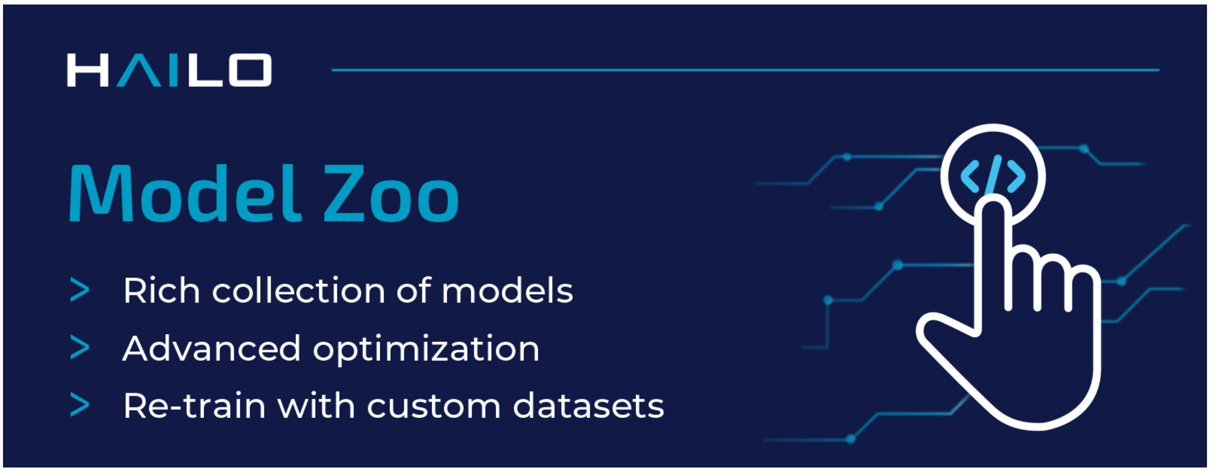
General Notice

Hailo, to the fullest extent permitted by law, provides this document "as-is" and disclaims all warranties, either express or implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties' rights, and fitness for particular purpose.

Although Hailo used reasonable efforts to ensure the accuracy of the content of this document, it is possible that this document may contain technical inaccuracies or other errors. Hailo assumes no liability for any error in this document, and for damages, whether direct, indirect, incidental, consequential or otherwise, that may result from such error, including, but not limited to loss of data or profits.

The content in this document is subject to change without prior notice and Hailo reserves the right to make changes to content of this document without providing a notification to its users.

1. Hailo Model Zoo



python 3.8 | 3.9 | 3.10 Tensorflow 2.9.2 CUDA 11.2 Hailo Dataflow Compiler 3.24.0 HailoRT (optional) 4.14.0 License MIT

The Hailo Model Zoo provides pre-trained models for high-performance deep learning applications. Using the Hailo Model Zoo you can measure the full precision accuracy of each model, the quantized accuracy using the Hailo Emulator and measure the accuracy on the Hailo-8 device. Finally, you will be able to generate the Hailo Executable Format (HEF) binary file to speed-up development and generate high quality applications accelerated with Hailo-8. The Hailo Model Zoo also provides re-training instructions to train the models on custom datasets and models that were trained for specific use-cases on internal datasets.

1.1. Models

Hailo provides different pre-trained models in ONNX / TF formats and pre-compiled HEF (Hailo Executable Format) binary file to execute on the Hailo-8 device.
The models are divided to:

- *PUBLIC MODELS* which were trained on publicly available datasets.
- *HAILO MODELS* which were trained in-house for specific use-cases on internal datasets.
  Each Hailo Model is accompanied with retraining instructions.

1.2. Retraining

Hailo also provides *RETRAINING INSTRUCTIONS* to train a network from the Hailo Model Zoo with custom dataset.

1.3. Benchmarks

List of Hailo's benchmarks can be found in hailo.ai.
In order to reproduce the measurements please refer to the following *page*.

1.4. Quick Start Guide

- Install Hailo Dataflow Compiler and enter the virtualenv. In case you are not Hailo customer please contact hailo.ai
- Install HailoRT (optional). Required only if you want to run on Hailo-8. In case you are not Hailo customer please contact hailo.ai
- Clone the Hailo Model Zoo

```
git clone https://github.com/hailo-ai/hailo_model_zoo.git
```

- Run the setup script

```
cd hailo_model_zoo; pip install -e .
```

- Run the Hailo Model Zoo. For example, print the information of the MobileNet-v1 model:

```
hailomz info mobilenet_v1
```

1.4.1. Getting Started

For full functionality please see the *INSTALLATION GUIDE* page (full install instructions and usage examples). The Hailo Model Zoo is using the Hailo Dataflow Compiler for parsing, model optimization, emulation and compilation of the deep learning models. Full functionality includes:

- Parse: model translation of the input model into Hailo's internal representation.
- Profiler: generate profiler report of the model. The report contains information about your model and expected performance on the Hailo hardware.
- Optimize: optimize the deep learning model for inference and generate a numeric translation of the input model into a compressed integer representation.
  For further information please see our *OPTIMIZATION* page.
- Compile: run the Hailo compiler to generate the Hailo Executable Format file (HEF) which can be executed on the Hailo hardware.
- Evaluate: infer the model using the Hailo Emulator or the Hailo hardware and produce the model accuracy.

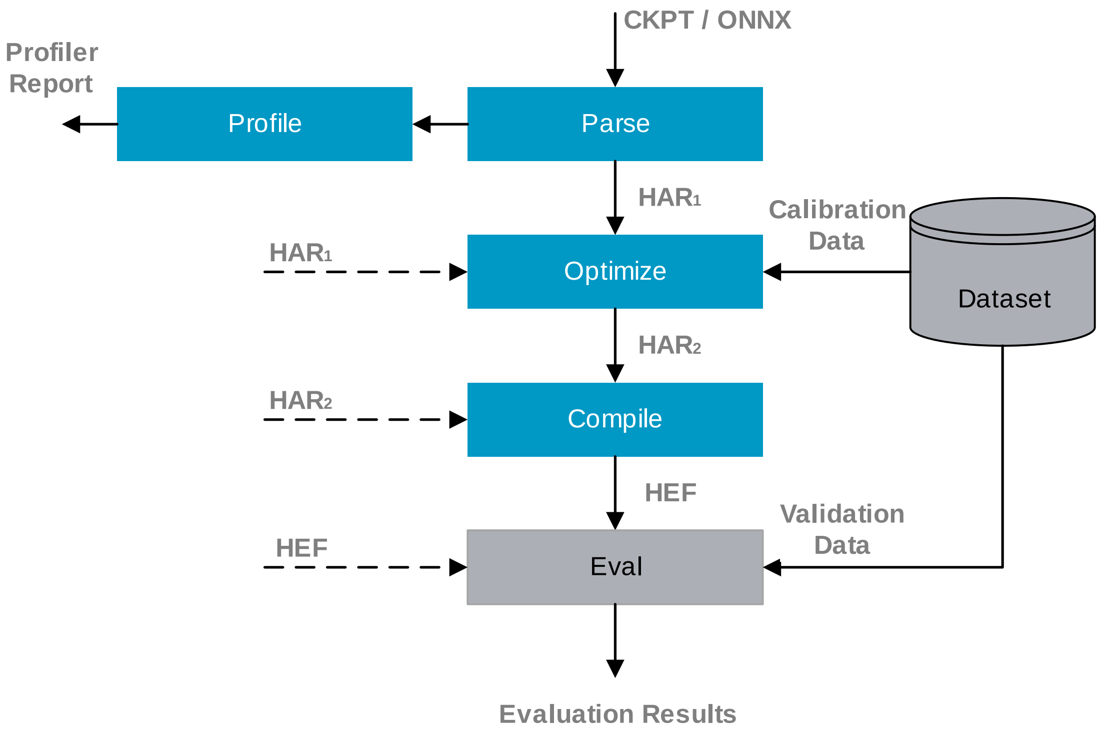
For further information about the Hailo Dataflow Compiler please contact hailo.ai.

1.5. License

The Hailo Model Zoo is released under the MIT license. Please see the LICENSE file for more information.

1.6. Contact

Please visit hailo.ai for support / requests / issues.



1.7. Changelog

For further information please see our *CHANGELOG* page.

2. Changelog

v2.8

- Update to use Dataflow Compiler v3.24.0 (developer-zone)
- Update to use HailoRT 4.14.0 (developer-zone)
- The Hailo Model Zoo now supports the following vision transformers models:
  - vit_tiny / vit_small / vit_base - encoder based transformer with batchnorm for classification
  - detr_resnet_v1_18_bn - encoder/decoder transformer for object detection
  - clip_resnet_50 - Contrastive Language-Image Pre-Training for zero-shot classification
  - yolov5s_c3tr - object detection model with a MHSA block
- Using HailoRT-pp for postprocessing of the following variants:
  - yolov5
  - yolox
  - ssd
  - efficientdet
  - yolov7
- New Models:
  - repvgg_a1 / repvgg_a2 - classification
  - yolov8_seg: yolov8n_seg / yolov8s_seg / yolov8m_seg - instance segmentation
  - yolov6n_0.2.1 - object detection
  - zero_dce - low-light enhancement
- New retraining dockers for:
  - yolov8
  - yolov8_seg
- Enable compilation for hailo15h device
- Enable evaluation of models with RGBX / NV12 input format
- Bug fixes

v2.7

- Update to use Dataflow Compiler v3.23.0 (developer-zone)
- Updated to use HailoRT 4.13.0 (developer-zone)
- Inference flow was moved to new high-level APIs
- New object detection variants:
  - yolov8: yolov8n / yolov8s / yolov8m / yolov8l / yolov8x
  - damoyolo: damoyolo_tinynasL20_T / damoyolo_tinynasL25_S / damoyolo_tinynasL35_M
- New transformers based models:
  - vit_base - classification model
  - yolov5s_c3tr - object detection model with a self-attention block
- Examples for using HailoRT-pp - support for seamless integration of models and their corresponding postprocessing
  - yolov5m_hpp

- Configuration YAMLs and model-scripts for networks with YUY2 input format
- DAMO-YOLO retraining docker
- Bug fixes

v2.6.1

- Bug fixes

v2.6

- Update to use Dataflow Compiler v3.22.0 (developer-zone)
- Updated to use HailoRT 4.12.0 (developer-zone)
- ViT (Vision Transformer) - new classification network with transformers-encoder based architecture
- New instance segmentation variants:
  - yolov5n_seg
  - yolov5s_seg
  - yolov5m_seg
  - yolov5l_seg
- New object detecion variants for high resolution images:
  - yolov7e6
  - yolov5n6_6.1
  - yolov5s6_6.1
  - yolov5m6_6.1
- New flag `--performance` to reproduce highest performance for a subset of networks
- Hailo model-zoo log is now written into `sdk_virtualenv/etc/hailo/modelzoo/hailo_examples.log`
- Bug fixes

v2.5

- Update to use Dataflow Compiler v3.20.1 (developer-zone)
- Model scripts use new bgr to rgb conversion
- New Yolact variants - with all COCO classes:
  - yolact_regnetx_800mf
  - yolact_regnetx_1.6gf
- Bug fixes

v2.4

- Updated to use Dataflow Compiler v3.20 (developer-zone)
- Required FPS was moved from models YAML into the models scripts
- Model scripts use new change activation syntax
- New models:
  - Face Detection - scrfd_500m / scrfd_2.5g / scrfd_10g
- New tasks:
  1. Super-Resolution
  - Added support for BSD100 dataset

- The following models were added: espcn_x2 / espcn_x3 / espcn_x4

2. Face Recognition

- Support for LFW dataset
- The following models were added:
  1. arcface_r50
  2. arcface_mobilefacenet
- Retraining docker for arcface architecture

• Added support for new hw-arch - hailo8l

v2.3

• Updated to use Dataflow Compiler v3.19 (developer-zone)
• New models:
  - yolov6n
  - yolov7 / yolov7-tiny
  - nanodet_repvgg_a1_640
  - efficientdet_lite0 / efficientdet_lite1 / efficientdet_lite2
• New tasks:
  - mspn_regnetx_800mf - single person pose estimation
  - face_attr_resnet_v1_18 - face attribute recognition
• Single person pose estimation training docker (mspn_regnetx_800mf)
• Bug fixes

v2.2

• Updated to use Dataflow Compiler v3.18 (developer-zone)
• CLI change:
  - Hailo model zoo CLI is now working with an entry point - hailomz
  - quantize sub command was changed to optimize
  - Hailo model zoo data directory by default will be `~/.hailomz`
• New models:
  - yolov5xs_wo_spp_nms - a model which contains bbox decoding and confidence thresholding on Hailo-8
  - osnet_x1_0 - person ReID network
  - yolov5m_6.1 - yolov5m network from the latest tag of the repo (6.1) including silu activation
• New tasks:
  - person_attr_resnet_v1_18 - person attribute recognition
• ReID training docker for the Hailo model repvgg_a0_person_reid_512/2048

NOTE: Ubuntu 18.04 will be deprecated in Hailo Model Zoo future version

NOTE: Python 3.6 will be deprecated in Hailo Model Zoo future version

v2.1

• Updated to use Dataflow Compiler v3.17 (developer-zone)
• Parser commands were moved into model scripts
• Support Market-1501 Dataset

- Support a new model zoo task - ReID
- New models:
  - yolov5s_personface - person and face detector
  - repvgg_a0_person_reid_512 / repvgg_a0_person_reid_2048 - ReID networks which outputs a person embedding
    These models were trained in-house as part of our upcoming new application
  - stdc1 - Segmentation architecture for Cityscapes

v2.0

- Updated to use Dataflow Compiler v3.16 (developer-zone) with TF version 2.5 which require CUDA11.2
- Updated to use HailoRT 4.6 (developer-zone)
- Retraining Dockers - each retraining docker has a corresponding README file near it. New retraining dockers:
  - SSD
  - YOLOX
  - FCN
- New models:
  - yolov5l
- Introducing Hailo Models, in-house pretrained networks with compatible Dockerfile for retraining
  - yolov5m_vehicles (vehicle detection)
  - tiny_yolov4_license_plates (license plate detection)
  - lprnet (license plate recognition)
- Added new documentation to the *YAML structure*

v1.5

- Remove HailoRT installation dependency.
- Retraining Dockers
  - YOLOv3
  - NanoDet
  - CenterPose
  - Yolact
- New models:
  - unet_mobilenet_v2
- Support Oxford-IIIT Pet Dataset
- New mutli-network example: detection_pose_estimation which combines the following networks:
  - yolov5m_wo_spp_60p
  - centerpose_repvgg_a0
- Improvements:
  - nanodet_repvgg mAP increased by 2%
- New Tasks:
  - hand_landmark_lite from MediaPipe
  - palm_detection_lite from MediaPipe

Both tasks are without evaluation module.

v1.4

- Update to use Dataflow Compiler v3.14.0 (developer-zone)
- Update to use HailoRT 4.3.0 (developer-zone)
- Introducing *Hailo Models* - in house pretrained networks with compatible Dockerfile for easy retraining:
  - yolov5m_vehicles - vehicle detector based on yolov5m architecture
  - tiny_yolov4_license_plates - license plate detector based on tiny_yolov4 architecture
- New Task: face landmarks detection
  - tddfa_mobilenet_v1
  - Support 300W-LP and AFLW2k3d datasets
- New features:
  - Support compilation of several networks together - a.k.a *multinets*
  - CLI for printing *network information*
- Retraining Guide:
  - New training guide for yolov4 with compatible Dockerfile
  - Modifications for yolov5 retraining

v1.3

- Update to use Dataflow Compiler v3.12.0 (developer-zone)
- New task: indoor depth estimation
  - fast_depth
  - Support NYU Depth V2 Dataset
- New models:
  - resmlp12 - new architecture support paper
  - yolox_l_leaky
- Improvements:
  - ssd_mobilenet_v1 - in-chip NMS optimization (de-fusing)
- Model Optimization API Changes
  - Model Optimization parameters can be updated using the networks' model script files (*.alls)
  - Deprecated: quantization params in YAMLs
- Training Guide: new training guide for yolov5 with compatible Dockerfile

v1.2

- New features:
  - YUV to RGB on core can be added through YAML configuration.
  - Resize on core can be added through YAML configuration.
- Support D2S Dataset
- New task: instance segmentation
  - yolact_mobilenet_v1 (coco)
  - yolact_regnetx_800mf_20classes (coco)

    - yolact_regnetx_600mf_31classes (d2s)
- New models:
    - nanodet_repvgg
    - centernet_resnet_v1_50_postprocess
    - yolov3 - darkent based
    - yolox_s_wide_leaky
    - deeplab_v3_mobilenet_v2_dilation
    - centerpose_repvgg_a0
    - yolov5s, yolov5m - original models from link
    - yolov5m_yuv - contains resize and color conversion on HW
- Improvements:
    - tiny_yolov4
    - yolov4
- IBC and Equalization API change
- Bug fixes

v1.1

- Support VisDrone Dataset
- New task: pose estimation
    - centerpose_regnetx_200mf_fpn
    - centerpose_regnetx_800mf
    - centerpose_regnetx_1.6gf_fpn
- New task: face detection
    - lightfaceslim
    - retinaface_mobilenet_v1
- New models:
    - hardnet39ds
    - hardnet68
    - yolox_tiny_leaky
    - yolox_s_leaky
    - deeplab_v3_mobilenet_v2
- Use your own network manual for YOLOv3, YOLOv4_leaky and YOLOv5.

v1.0

- Initial release
- Support for object detection, semantic segmentation and classification networks

3. Getting Started

This document provides install instructions and basic usage examples of the Hailo Model Zoo.

3.1. System Requirements

- Ubuntu 20.04/22.04, 64 bit (supported also on Windows, under WSL2)
- Python 3.8/3.9/3.10, including `pip` and `virtualenv`
- Hailo Dataflow Compiler v3.24.0 (Obtain from hailo.ai)
- HailoRT 4.14.0 (Obtain from hailo.ai) - required only for inference on Hailo-8.
- The Hailo Model Zoo supports Hailo-8 connected via PCIe only.
- Nvidia's Pascal/Turing/Ampere GPU architecture (such as Titan X Pascal, GTX 1080 Ti, RTX 2080 Ti, or RTX A4000)
- GPU driver version 470
- CUDA 11.2
- CUDNN 8.1

3.2. Install Instructions

3.2.1. Hailo Software Suite

The Hailo Software Suite includes all of Hailo's SW components and insures compatibility across products versions. The Hailo Model Zoo is already installed and ready to be used within the virtualenv of it.

3.2.2. Manual Installation

1. Install the Hailo Dataflow compiler and enter the virtualenv (visit hailo.ai for further instructions).
2. Install the HailoRT - required only for inference on Hailo-8 (visit hailo.ai for further instructions).
3. Clone the Hailo Model Zoo repo:

```
git clone https://github.com/hailo-ai/hailo_model_zoo.git
```

4. Run the setup script:

```
cd hailo_model_zoo; pip install -e .
```

5. For setting up datasets please see *DATA*.
6. Verify Hailo-8 is connected through PCIe (required only to run on Hailo-8. Full-precision / emulation run on GPU.)

```
hailo fw-control identify
```

Expected output:

```
(hailo) Running command 'fw-control' with 'hailortcli'
Identifying board
Control Protocol Version: 2
Firmware Version: 4.6.0 (release,app)
Logger Version: 0
```

(continues on next page)

(continued from previous page)

```
Board Name: Hailo-8
Device Architecture: HAILO8_B0
Serial Number: HLUTM20204900071
Part Number: HM218B1C2FA
Product Name: HAILO-8 AI ACCELERATOR M.2 MODULE
```

3.2.3. Upgrade Instructions

If you want to upgrade to a specific Hailo Model Zoo version within a suite or on top of a previous installation not in the suite.

1. Pull the specific repo branch:

```
git clone -b v2.6 https://github.com/hailo-ai/hailo_model_zoo.git
```

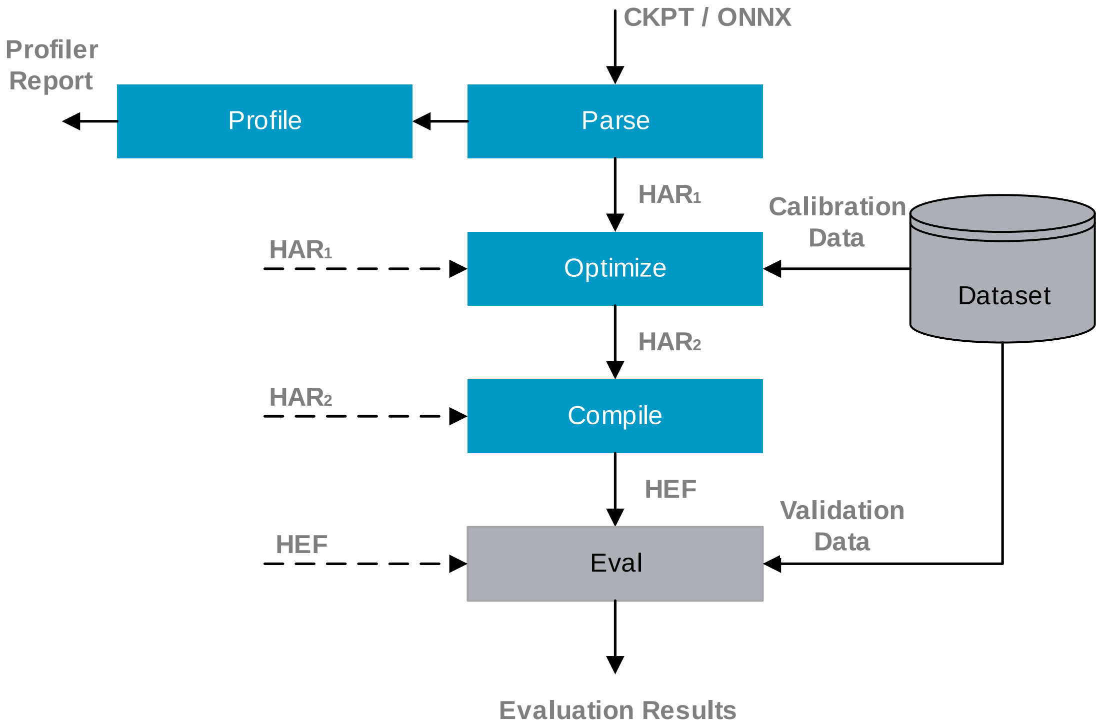
2. Run the setup script:

```
cd hailo_model_zoo; pip install -e .
```


4. Usage

4.1. Flow Diagram

The following scheme shows high-level view of the model-zoo evaluation process, and the different stages in between.



By default, each stage executes all of its previously necessary stages according to the above diagram. The post-parsing stages also have an option to start from the product of previous stages (i.e., the Hailo Archive (HAR) file), as explained below. The operations are configured through a YAML file that exist for each model in the cfg folder. For a description of the YAML structure please see *YAML*.

4.2. Parsing

The pre-trained models are stored on AWS S3 and will be downloaded automatically when running the model zoo into your data directory. To parse models into Hailo's internal representation and generate the Hailo Archive (HAR) file:

```
hailomz parse <model_name>
```

- The default compilation target is Hailo-8. To compile for different architecture (Hailo-15H for example), use `--hw_arch hailo15h` as CLI argument:

```
hailomz parse <model_name> --hw-arch hailo15h
```

4.3. Profiling

To generate the Hailo profiler report:

```
hailomz profile <model_name>
```

To generate the Hailo profiler report using a previously generated HAR file:

```
hailomz profile <model_name> --har /path/to/model.har
```

- The report contains information about your model and expected performance on the Hailo hardware.

4.4. Optimize

To optimize models, convert them from full precision into integer representation and generate a quantized Hailo Archive (HAR) file:

```
hailomz optimize <model_name>
```

To optimize the model starting from a previously generated HAR file:

```
hailomz optimize <model_name> --har /path/to/model.har
```

You can use your own images by giving a directory path to the optimization process, with the following supported formats (.jpg,.jpeg,.png):

```
hailomz optimize <model_name> --calib-path /path/to/calibration/imgs/dir/
```

- This step requires data for calibration. For additional information please see *OPTIMIZATION*.

In order to achieve highest performance, one could use the performance flag:

```
hailomz optimize <model_name> --performance
```

The flag will be ignored on models that do not support this feature. The default and performance model scripts are located on *hailo_model_zoo/cfg/alls/*

4.5. Compile

To run the Hailo compiler and generate the Hailo Executable Format (HEF) file:

```
hailomz compile <model_name>
```

By default the compilation target is Hailo-8. To compile for a different architecture use `--hw-arch` command line argument:

```
hailomz compile <model_name> --hw-arch hailo15h
```

To generate the HEF starting from a previously generated HAR file:

```
hailomz compile <model_name> --har /path/to/model.har --hw-arch <hailo8|hailo15h>
```

In order to achieve highest performance, one could use the performance flag:

```
hailomz optimize <model_name> --performance --hw-arch <hailo8|hailo15h>
```

The flag will be ignored on models that do not support this feature. The default and performance model scripts are located on *hailo_model_zoo/cfg/alls/*

4.6. Evaluation

To evaluate models in full precision:

```
hailomz eval <model_name>
```

To evaluate models starting from a previously generated Hailo Archive (HAR) file:

```
hailomz eval <model_name> --har /path/to/model.har
```

To evaluate models with the Hailo emulator (after quantization to integer representation - fast_numeric):

```
hailomz eval <model_name> --target emulator
```

To evaluate models on Hailo-8:

```
hailomz eval <model_name> --target hailo8
```

If multiple Hailo-8 devices are available, it's possible to select a specific one

```
# Device id looks something like 0000:41:00.0
hailomz eval <model_name> --target <device_id>
# This command can be used to list available devices
hailomz eval --help
```

To limit the number of images for evaluation use the following flag:

```
hailomz eval <model_name> --data-count <num-images>
```

To explore other options (for example: changing the default batch-size) use:

```
hailomz eval --help
```

- Currently MZ evaluation can be done only on hailo8

4.7. Visualization

To run visualization (without evaluation) and generate the output images:

```
hailomz eval <model_name> --visualize
```

To create a video file from the network predictions:

```
hailomz eval <model_name> --visualize --video-outpath /path/to/video_output.mp4
```

4.8. Info

You can easily print information of any network exists in the model zoo, to get a sense of its input/output shape, parameters, operations, framework etc.

To print a model-zoo network information:

```
hailomz info <model_name>
```

Here is an example for printing information about mobilenet_v1:


```
hailomz info mobilenet_v1
```

Expected output:

```
<Hailo Model Zoo Info> Printing mobilenet_v1 Information
<Hailo Model Zoo Info>
    task:             classification
    input_shape:          224x224x3
    output_shape:          1x1x1001
    operations:          0.57G
    parameters:          4.22M
    framework:          tensorflow
    training_data:          imagenet train
    validation_data:          imagenet val
    eval_metric:          Accuracy (top1)
    full_precision_result:   71.02
    source:          https://github.com/tensorflow/models/tree/v1.13.0/research/
↪slim
    license_url:          https://github.com/tensorflow/models/blob/v1.13.0/LICENSE
```

4.9. Compile multiple networks together

We can use multiple disjoint models in the same binary. This is useful for running several small models on the device.

```
python hailo_model_zoo/multi_main.py <config_name>
```

4.10. TFRecord to NPY conversion

In some situations you might want to convert the tfrecord file to npy file (for example, when explicitly using the Dataflow Compiler for quantization). In order to do so, run the command:

```
python hailo_model_zoo/tools/conversion_tool.py /path/to/tfrecord_file resnet_v1_
↪50 --npy
```

5. Model Optimization

5.1. Introduction

Model optimization is the stage of converting a full precision model to an optimized model which will be compiled to the Hailo device.

This stage includes numeric translation of the input model into a compressed integer representation as well as optimizing the model architecture to best fit the Hailo hardware.

Compressing deep learning model induce degradation for the model's accuracy.

For example, in the following table we compare the accuracy of full precision ResNet V1 18 with the compressed 8-bits weights and activation representation on ImageNet-1K:

Precision	Top-1
Full precision	68.85
8-bit (emulator)	68.54

The main goal of the model optimization step is to prepare the model for compilation with minimum degradation as possible.

5.2. Optimization Workflow

The model optimization has two main steps: full precision optimization and quantization optimization.

- Full precision optimization includes any changes to the model in the floating point precision domain, for example, Equalization (*Meller2019*), TSE (*Vosco2021*) and pruning.
- Quantization includes compressing the model from floating point to integer representation of the weights and activations (4/8/16 bits) and algorithms to improve the model's accuracy, such as IBC (*Finkelstein2019*) and QFT.

Both steps may degrade the model accuracy, therefore, evaluation is needed to verify the model accuracy. This workflow is depicted in the following diagram:

1. First step includes full precision validation. This step is important to make sure parsing was successful and we built the pre/post processing and evaluation of the model correctly. In the Hailo Model Zoo, we can execute the following which will infer a specific model in full precision to verify that the accuracy is correct (this will be our baseline for measuring degradation):

   ```
   hailomz eval <model_name>
   ```

2. Next, we call the model optimization API to generate an optimized model. Note, it is recommended to run this step on a GPU machine with dataset size of at least 1024 images.

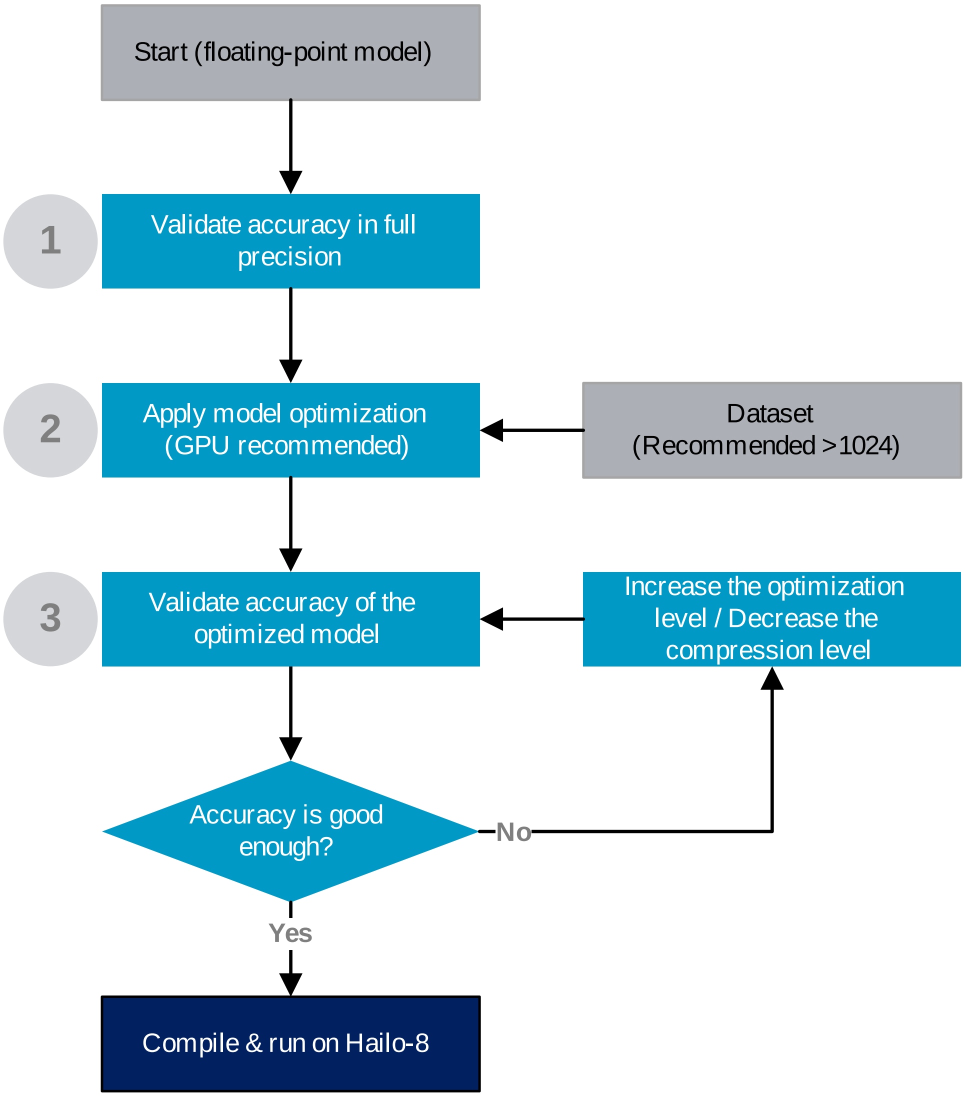
   ```
   hailomz optimize <model_name>
   ```

3. Lastly, we verify the accuracy of the optimized model. In case the results are not good enough we should repeat the process with different configurations of the optimization/compression levels:

   ```
   hailomz eval <model_name> --target emulator --har <model_name>.har
   ```

Once optimization is finished and met our accuracy requirements, we can compile the optimized model. For example:

```
hailomz compile <model_name> --har <model_name>.har
```



Start (floating-point model)
1
Validate accuracy in full precision
2
Apply model optimization (GPU recommended)
Dataset (Recommended >1024)
3
Validate accuracy of the optimized model
Increase the optimization level / Decrease the compression level
Accuracy is good enough?
No
Yes
Compile & run on Hailo-8

5.3. Citations

5.3.1. Vosco2021

```
@InProceedings{Vosco2021,
 title = {Tiled Squeeze-and-Excite: Channel Attention With Local Spatial Context},
 author = {Niv Vosco and Alon Shenkler and Mark Grobman},
 booktitle = {ICCV},
 year = {2021}
}
```

5.3.2. Finkelstein2019

```
@InProceedings{Finkelstein2019,
 title = {Fighting Quantization Bias With Bias},
 author = {Alexander Finkelstein and Uri Almog and Mark Grobman},
 booktitle = {CVPR},
 year = {2019}
}
```

5.3.3. Meller2019

```
@InProceedings{Meller2019,
 title = {Same, Same But Different - Recovering Neural Network Quantization Error
↪Through Weight Factorization},
 author = {Eldad Meller and Alexander Finkelstein and Uri Almog and Mark Grobman},
 booktitle = {ICML},
 year = {2019}
}
```


6. Public Pre-Trained Models

Here, we give the full list of publicly pre-trained models supported by the Hailo Model Zoo.

- Network available in Hailo Benchmark are marked with 🚀
- Networks available in TAPPAS are marked with ⭐
- Benchmark, TAPPAS and Recommended networks run in performance mode
- All models were compiled using Hailo Dataflow Compiler v3.24.0
- Supported tasks:
  - *Classification*
  - *Object Detection*
  - *Semantic Segmentation*
  - *Pose Estimation*
  - *Single Person Pose Estimation*
  - *Face Detection*
  - *Instance Segmentation*
  - *Depth Estimation*
  - *Facial Landmark Detection*
  - *Person Re-ID*
  - *Super Resolution*
  - *Face Recognition*
  - *Person Attribute*
  - *Face Attribute*
  - *Zero-shot Classification*
  - *Low Light Enhancement*
  - *Hand Landmark detection*

6.1. Classification

6.1.1. ImageNet

Network Name	Accuracy (top1)	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
efficientnet_l	80.46	79.36	300x300x3	10.55	19.4	download	link	download
efficientnet_lite0	74.99	73.81	224x224x3	4.63	0.78	download	link	download
efficientnet_lite1	76.68	76.21	240x240x3	5.39	1.22	download	link	download

Continued on next page

Table 1 – continued from previous page

Network Name	Accuracy (top1)	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
efficientnet_lite2	77.45	76.74	260x260x3	6.06	1.74	download	link	download
efficientnet_lite3	79.29	78.33	280x280x3	8.16	2.8	download	link	download
efficientnet_lite4	80.79	80.47	300x300x3	12.95	5.10	download	link	download
efficientnet_m 🚀	78.91	78.63	240x240x3	6.87	7.32	download	link	download
efficientnet_s	77.64	77.32	224x224x3	5.41	4.72	download	link	download
hardnet39ds	73.43	72.92	224x224x3	3.48	0.86	download	link	download
hardnet68	75.47	75.04	224x224x3	17.56	8.5	download	link	download
inception_v1	69.74	69.54	224x224x3	6.62	3	download	link	download
mobilenet_v1	70.97	70.15	224x224x3	4.22	1.14	download	link	download
mobilenet_v2_1.0 🚀	71.78	71.08	224x224x3	3.49	0.62	download	link	download
mobilenet_v2_1.4	74.18	73.07	224x224x3	6.09	1.18	download	link	download
mobilenet_v3	72.21	71.73	224x224x3	4.07	2	download	link	download
mobilenet_v3_large_minimalistic	72.11	70.92	224x224x3	3.91	0.42	download	link	download
regnetx_1.6gf	77.05	76.75	224x224x3	9.17	3.22	download	link	download
regnetx_800mf	75.16	74.84	224x224x3	7.24	1.6	download	link	download
regnety_200mf	70.38	70.02	224x224x3	3.15	0.4	download	link	download
repvgg_a1	74.4	73.61	224x224x3	12.79	4.7	download	link	download
repvgg_a2	76.52	75.08	224x224x3	25.5	10.2	download	link	download
resmlp12_relu	75.26	74.32	224x224x3	15.77	6.04	download	link	download
resnet_v1_18	71.26	71.06	224x224x3	11.68	3.64	download	link	download
resnet_v1_34	72.7	72.14	224x224x3	21.79	7.34	download	link	download

Continued on next page

Table 1 – continued from previous page

Network Name	Accuracy (top1)	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
resnet_v1_50 🚀 ⭐	75.12	74.47	224x224x3	25.53	6.98	download	link	download
resnet_v2_18	69.57	69.1	224x224x3	11.68	3.64	download	link	download
resnet_v2_34	73.07	72.72	224x224x3	21.79	7.34	download	link	download
resnext26_32x4d	76.18	75.78	224x224x3	15.37	4.96	download	link	download
resnext50_32x4d	79.31	78.39	224x224x3	24.99	8.48	download	link	download
shufflenet_g8_w1	66.3	65.5	224x224x3	2.46	0.36	download	link	download
squeezenet_v1.1	59.85	59.4	224x224x3	1.24	0.78	download	link	download
vit_base	79.98	78.88	224x224x3	86.5	34.2	download	link	download
vit_small	78.12	77.02	224x224x3	21.12	8.5	download	link	download
vit_tiny	68.02	66.08	224x224x3	5.41	4.72	download	link	download

6.2. Object Detection

6.2.1. COCO

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
centernet_resnet_v1_18_postprocess	26.29	23.39	512x512x3	14.22	31.26	download	link	download
centernet_resnet_v1_50_postprocess	31.78	29.64	512x512x3	30.07	56.92	download	link	download
damoyolo_tinynasL20_T	42.8	42.0	640x640x3	11.35	18.06	download	link	download
damoyolo_tinynasL25_S	46.53	46.04	640x640x3	16.25	37.7	download	link	download
damoyolo_tinynasL35_M	49.7	47.23	640x640x3	33.98	61.74	download	link	download
detr_resnet_v1_18_bn	33.9	30.6	800x800x3	32.42	59.16	download	link	download

Continued on next page

Table 2 – continued from previous page

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
efficientdet_lite0	27.32	26.48	320x320x3	3.56	1.98	download	link	download
efficientdet_lite1	32.27	31.72	384x384x3	4.73	4	download	link	download
efficientdet_lite2	35.95	34.67	448x448x3	5.93	6.84	download	link	download
nanodet_repvgg ⭐	29.3	28.53	416x416x3	6.74	11.28	download	link	download
nanodet_repvgg_a12	33.7	32.0	640x640x3	5.13	28.23	download	link	download
nanodet_repvgg_a1_640	33.28	32.88	640x640x3	10.79	42.8	download	link	download
ssd_mobilenet_v1 🚀 ⭐	23.17	22.17	300x300x3	6.79	2.5	download	link	download
ssd_mobilenet_v1_hd	17.66	15.55	720x1280x3	6.81	24.52	download	link	download
ssd_mobilenet_v2	24.15	22.94	300x300x3	4.46	1.52	download	link	download
tiny_yolov3	14.36	13.61	416x416x3	8.85	5.58	download	link	download
tiny_yolov4	19.18	17.73	416x416x3	6.05	6.92	download	link	download
yolov3 ⭐	38.42	37.32	608x608x3	68.79	158.34	download	link	download
yolov3_416	37.73	36.08	416x416x3	61.92	65.94	download	link	download
yolov3_gluon 🚀 ⭐	37.28	35.64	608x608x3	68.79	140.69	download	link	download
yolov3_gluon_416 ⭐	36.27	34.92	416x416x3	61.92	65.94	download	link	download
yolov4_leaky ⭐	42.37	41.08	512x512x3	64.33	91.04	download	link	download
yolov5l	46.01	44.01	640x640x3	48.54	121.56	download	link	download
yolov5m	42.59	41.19	640x640x3	21.78	52.28	download	link	download
yolov5m6_6.1	50.68	48.74	1280x1280x3	35.70	200.04	download	link	download
yolov5m_6.1	44.81	43.38	640x640x3	21.17	48.96	download	link	download
yolov5m_wo_spp 🚀	42.46	40.76	640x640x3	22.67	41.67	download	link	download

Continued on next page

Table 2 – continued from previous page

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
yolov5n6_6.1	35.63	33.68	1280x1280x3	3.24	18.34	download	link	download
yolov5s ⭐	35.33	33.98	640x640x3	7.46	17.44	download	link	download
yolov5s6_6.1	44.17	41.74	1280x1280x3	12.61	67.4	download	link	download
yolov5s_c3tr	37.13	35.33	640x640x3	10.29	17.02	download	link	download
yolov5xs_wo_spp	33.18	32.2	512x512x3	7.85	11.36	download	link	download
yolov5xs_wo_spp_nms_core	32.57	31.06	512x512x3	7.85	11.36	download	link	download
yolov6n	34.29	31.99	640x640x3	4.32	4.65	download	link	download
yolov6n_0.2.1	35.2	33.77	640x640x3	4.33	11.06	download	link	download
yolov7	50.6	47.8	640x640x3	36.91	104.68	download	link	download
yolov7_tiny	37.07	35.97	640x640x3	6.22	13.74	download	link	download
yolov7e6	55.37	53.17	1280x1280x3	97.20	515.12	download	link	download
yolov8l	52.61	51.95	640x640x3	43.7	165.3	download	link	download
yolov8m	50.08	48.73	640x640x3	25.9	78.93	download	link	download
yolov8n	37.23	36.23	640x640x3	3.2	8.8	download	link	download
yolov8s	44.75	44.15	640x640x3	11.2	28.6	download	link	download
yolov8x	53.61	52.21	640x640x3	68.2	258	download	link	download
yolox_l_leaky ⭐	48.68	46.77	640x640x3	54.17	155.3	download	link	download
yolox_s_leaky	38.13	37.15	640x640x3	8.96	26.74	download	link	download
yolox_s_wide_leaky	42.4	40.9	640x640x3	20.12	59.46	download	link	download
yolox_tiny	32.98	31.26	416x416x3	5.05	6.44	download	link	download

6.2.2. VisDrone

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
ssd_mobilenet_v1_visdrone ⭐	2.18	2.16	300x300x3	5.64	2.3	download	link	download

6.3. Semantic Segmentation

6.3.1. Cityscapes

Network Name	mIoU	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
fcn16_resnet_v1_18 ⭐	66.83	66.39	1024x1920x3	11.19	142.52	download	link	download
fcn8_resnet_v1_18 ⭐	68.75	67.85	1024x1920x3	11.20	143.02	download	link	download
fcn8_resnet_v1_22	67.55	67.39	1920x1024x3	15.12	300.08	download	link	download
stdc1 🚀	74.57	73.57	1024x1920x3	8.27	126.47	download	link	download

6.3.2. Oxford-IIIT Pet

Network Name	mIoU	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
unet_mobilenet_v2	77.32	77.02	256x256x3	10.08	28.88	download	link	download

6.3.3. Pascal VOC

Network Name	mIoU	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
deeplab_v3_mobilenet_v2	76.05	74.8	513x513x3	2.10	17.82	download	link	download
deeplab_v3_mobilenet_v2_wo_dilation	71.46	71.08	513x513x3	2.10	3.28	download	link	download

6.4. Pose Estimation

6.4.1. COCO

Network Name	AP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
centerpose_regnetx_1.6gf_fpn ⭐	53.54	48.29	640x640x3	14.28	64.76	download	link	download
centerpose_regnetx_800mf	44.07	42.02	512x512x3	12.31	86.12	download	link	download
centerpose_repvgg_a0 ⭐	39.17	37.37	416x416x3	11.71	24.76	download	link	download

6.5. Single Person Pose Estimation

6.5.1. COCO

Network Name	AP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
mspn_regnetx_800mf ⭐	70.8	70.3	256x192x3	7.17	2.94	download	link	download

6.6. Face Detection

6.6.1. WiderFace

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
lightface_slim ⭐	39.7	39.22	240x320x3	0.26	0.08	download	link	download
retinaface_mobilenet_v1 ⭐	81.27	81.17	736x1280x3	3.49	25.14	download	link	download
scrfd_10g	82.13	82.03	640x640x3	4.23	26.74	download	link	download
scrfd_2.5g	76.59	76.32	640x640x3	0.82	6.88	download	link	download
scrfd_500m	68.98	68.88	640x640x3	0.63	1.5	download	link	download

6.7. Instance Segmentation

6.7.1. COCO

Network Name	mAP	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
yolact_mobilenet_v1	14.98	14.86	512x512x3	19.11	103.84	download	link	download
yolact_regnetx_1.6gf	27.57	27.27	512x512x3	30.09	125.34	download	link	download
yolact_regnetx_800mf	25.61	25.5	512x512x3	28.3	116.75	download	link	download
yolact_regnetx_800mf_20classes ⭐	20.23	20.22	512x512x3	21.97	102.94	download	link	download
yolov5l_seg	39.78	39.09	640x640x3	47.89	147.88	download	link	download
yolov5m_seg	37.05	36.32	640x640x3	32.60	70.94	download	link	download
yolov5n_seg ⭐	23.35	22.24	640x640x3	1.99	7.1	download	link	download
yolov5s_seg	31.57	30.49	640x640x3	7.61	26.42	download	link	download
yolov8m_seg	40.6	39.88	640x640x3	27.3	104.6	download	link	download
yolov8n_seg	30.32	29.68	640x640x3	3.4	12.04	download	link	download
yolov8s_seg	36.63	35.8	640x640x3	11.8	40.2	download	link	download

6.7.2. D2S

Network Name	mAP-segme	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
yolact_regnetx_600mf_d2s_31classes	62.48	63.36	512x512x3	22.14	103.24	download	link	download

6.8. Depth Estimation

6.8.1. NYU

Network Name	RMSE	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
fast_depth ⭐	0.6	0.61	224x224x3	1.35	0.74	download	link	download

6.9. Facial Landmark Detection

6.9.1. AFLW2k3d

Network Name	NME	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
tddfa_mobilenet_v1 ⭐	3.68	4.05	120x120x3	3.26	0.36	download	link	download

6.10. Person Re-ID

6.10.1. Market1501

Network Name	rank1	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
osnet_x1_0	94.43	93.7	256x128x3	2.19	1.98	download	link	download
repvgg_a0_person_reid_2048	90.02	89.12	256x128x3	9.65	1.78	download	link	download
repvgg_a0_person_reid_512 ⭐	89.9	89.4	256x128x3	7.68	1.78	download	link	download

6.11. Super Resolution

6.11.1. BSD100

Network Name	PSNR	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
espcn_x2	31.4	30.3	156x240x1	0.02	1.6	download	link	download
espcn_x3	28.41	28.06	104x160x1	0.02	0.76	download	link	download
espcn_x4	26.83	26.58	78x120x1	0.02	0.46	download	link	download

6.12. Face Recognition

6.12.1. LFW

Network Name	lfw verification accuracy	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
arcface_mobilefacenet	99.43	99.41	112x112x3	2.04	0.88	download	link	download
arcface_r50	99.72	99.71	112x112x3	31.0	12.6	download	link	download

6.13. Person Attribute

6.13.1. PETA

Network Name	Mean Accuracy	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
person_attr_resnet_v1_18	82.5	82.61	224x224x3	11.19	3.64	download	link	download

6.14. Face Attribute

6.14.1. CELEBA

Network Name	Mean Accuracy	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
face_attr_resnet_v1_18	81.19	81.09	218x178x3	11.74	3	download	link	download

6.15. Zero-shot Classification

6.15.1. CIFAR100

Network Name	Accuracy (top1)	Quantized	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
clip_resnet_50	42.07	39.57	224x224x3	38.72	11.62	download	link	download

6.16. Low Light Enhancement

6.16.1. LOL

Network Name	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
zero_dce	600x400x3	0.21	38.2	download	link	download

6.17. Hand Landmark detection

6.17.1. Hand Landmark

Network Name	Input Resolution (HxWxC)	Params (M)	OPS (G)	Pretrained	Source	Compiled
hand_landmark_lite	224x224x3	1.01	0.3	download	link	download

7. Hailo Models

Here, we give the full list of models trained in-house for specific use-cases. Each model is accompanied with its own README, retraining docker and retraining guide.

- FLOPs in the table are counted as MAC operations.
- Supported tasks:
  - *Object Detection*
  - *Person & Face Detection*
  - *License Plate Recognition*
  - *Person Re-Identification*

Important: Retraining is not available inside the docker version of Hailo Software Suite. In case you use it, clone the hailo_model_zoo outside of the docker, and perform the retraining there: `git clone https://github.com/hailo-ai/hailo_model_zoo.git`

1. **Object Detection**

Network Name	mAP*	Input Resolution (HxWxC)	Params (M)	FLOPs (G)
yolov5m_vehicles	46.5	640x640x3	21.47	25.63
tiny_yolov4_license_plates	73.45	416x416x3	5.87	3.4
yolov5s_personface	47.5	640x640x3	7.25	8.38

2. **License Plate Recognition**

Network Name	Accuracy*	Input Resolution (HxWxC)	Params (M)	FLOPs (G)
lprnet	99.96	75x300x3	7.14	18.29

* Evaluated on internal dataset

3. **Person Re-ID**

Network Name	Accuracy*	Input Resolution (HxWxC)	Params (M)	FLOPs (G)
repvgg_a0_person_reid_512	89.9	256x128x3	7.68	0.89
repvgg_a0_person_reid_2048	90.02	256x128x3	9.65	0.89

* Evaluated on Market-1501

7.1. License Plate Detection

Hailo's license detection network (*tiny_yolov4_license_plates*) is based on Tiny-YOLOv4 and was trained in-house using Darknet with a single class. It expects a single vehicle and can work under various weather and lighting conditions, on different vehicle types and numerous camera angles.

7.1.1. Model Details

Architecture

- Tiny-YOLOv4
- Number of parameters: 5.87M
- GMACS: 3.4
- Accuracy* : 73.45 mAP
  * Evaluated on internal dataset containing 5000 images

Inputs

- RGB image with size of 416x416x3
- Image normalization occurs on-chip

Outputs

- Two output tensors with sizes of 13x13x18 and 26x26x18.
- Each output contains 3 anchors that hold the following information:
  - Bounding box coordinates ((x,y) centers, height, width)
  - Box objectness confidence score
  - Class probability confidence score
- The above 6 values per anchor are concatenated into the 18 output channels

Download

The pre-compiled network can be downloaded from here.
Use the following command to measure model performance on hailo's HW:

```
hailortcli benchmark tiny_yolov4_license_plates.hef
```

7.1.2. Train License Plate Detection on a Custom Dataset

Here we describe how to finetune Hailo's license plate detection network on your own custom dataset.

Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

Environment Preparations

1. **Build the docker image**

```
cd hailo_model_zoo/hailo_models/license_plate_detection/
docker build --build-arg timezone=`cat /etc/timezone` -t license_plate_
↪detection:v0 .
```

   - This command will build the docker image with the necessary requirements using the Dockerfile exists in this directory.

2. **Start your docker:**

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪data/dir:/path/to/docker/data/dir license_plate_detection:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `license_plate_detection:v0` the name of the docker image.

Finetuning and exporting to ONNX

1. **Train the network on your dataset**

   Once the docker is started, you can train the license plate detector on your custom dataset. We recommend following the instructions for YOLOv4 training that can be found in here. The important steps are specified below:

   - Update `data/obj.data` with paths to your training and validation `.txt` files, which contain the list of the image paths*.

     ```
     classes = 1
     train  = data/train.txt
     valid  = data/val.txt
     names = data/obj.names
     backup = backup/
     ```

   * Tip: specify the paths to the training and validation images in the training and validation `.txt` files relative to `/workspace/darknet/`

   - Place your training and validation images and labels in your data folder.
   - Start training on your dataset starting from our pre-trained weights in `tiny_yolov4_license_plates.weights` (or download it from here)

   ```
   ./darknet detector train data/obj.data ./cfg/tiny_yolov4_license_plates.cfg
   ↪tiny_yolov4_license_plates.weights -map -clear
   ```

   NOTE: If during training you get an error similar to

   ```
   cuDNN status Error in: file: ./src/convolutional_kernels.cu : () : line: 543 :
   ↪build time: Jun 21 2022 - 20:09:28

   cuDNN Error: CUDNN_STATUS_BAD_PARAM
   Darknet error location: ./src/dark_cuda.c, cudnn_check_error, line #204
   cuDNN Error: CUDNN_STATUS_BAD_PARAM: Success
   ```

   - then please try changing *subdivisions* in the *.cfg* file (e.g., from 16 to 32).
   - For further information, please see discussion here.

2. **Export to ONNX**

   Export the model to ONNX using the following command:

   ```
   python ../pytorch-YOLOv4/demo_darknet2onnx.py cfg/tiny_yolov4_license_plates.
   ↪cfg /path/to/trained.weights /path/to/some/image.jpg 1
   ```

Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the model YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/tiny_yolov4_license_plates.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt tiny_yolov4_license_plates_1_416_416.onnx --calib-path /
↪path/to/calibration/imgs/dir/ --yaml path/to/tiny_yolov4_license_plates.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.

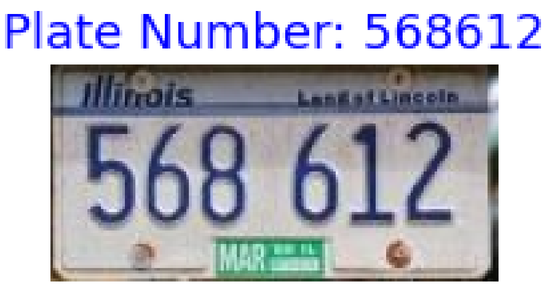
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Since it's an Hailo model, calibration set must be manually supplied.
- On tiny_yolov4_license_plates.yaml, change `postprocessing` section if anchors changed, `evaluation.classes` if classes amount is changed, and `evaluation.labels_offset` if it was changed on retraining.
- On yolo.yaml, change `preprocessing.input_shape` if the network is trained on other resolution.

More details about YAML files are presented *here*.

7.2. Licesen Plate Recognition



Hailo's license plate recognition network (*lprnet*) was trained in-house on a synthetic auto-generated dataset to predict registration numbers of license plates under various weather and lighting conditions.

7.2.1. Model Details

Architecture

- A convolutional network based on LPRNet, with several modifications:
  - A ResNet like backbone with 4 stages, each containing 2 residual blocks
  - Several kernel shape changes
  - Maximal license plate length of 19 digits
  - More details can be found here
- Number of parameters: 7.14M
- GMACS: 18.29
- Accuracy* : 99.96%
  * Evaluated on internal dataset containing 1178 images

Inputs

- RGB liscense plate image with size of 75x300x3
- Image normalization occurs on-chip

Outputs

- A tensor with size 5x19x11
  - Post-processing outputs a tensor with size of 1x19x11
  - The 11 channels contain logits scores for 11 classes (10 digits + *blank* class)
- A Connectionist temporal classification (CTC) greedy decoding outputs the final license plate number prediction

---

Download

The pre-compiled network can be downloaded from here.

Use the following command to measure model performance on hailo's HW:

```
hailortcli benchmark lprnet.hef
```

Training on Custom Dataset

- Hailo's LPRNet was trained on a synthetic auto-generated dataset containing 4 million license plate images. Auto-generation of synthetic data for training is cheap, allows one to obtain a large annotated dataset easily and can be adapted quickly for other domains
- A notebook for auto-generation of synthetic training data for LPRNet can be found here
- For more details on the training data autogeneration, please see the training guide

7.2.2. Train License Plate Recognition on a Custom Dataset

Here we describe how to finetune Hailo's optical character reader (OCR) model for license plate recognition with your own custom dataset.

Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

Environment Preparations

1. **Build the docker image**

```
cd hailo_model_zoo/hailo_models/license_plate_recognition/
docker build --build-arg timezone=`cat /etc/timezone` -t license_plate_
↪recognition:v0 .
```

   - This command will build the docker image with the necessary requirements using the Dockerfile that exists in this directory.

2. **Start your docker:**

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪data/dir:/path/to/docker/data/dir license_plate_recognition:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `license_plate_recognition:v0` the name of the docker image.

Finetuning and exporting to ONNX

1. **Train the network on your dataset**
   Once the docker is started, you can train the OCR on your custom dataset.

   - Create a folder with license plates images for training and testing. The folder should contain images whose file names correspond to the plate number, e.g. `12345678.png`.

   NOTE: Please make sure the file names **do not** contain characters which are not numbers or letters.

   - Alternatively, you can use the provided jupyter notebook in `dataset/lp_autogenerate.ipynb` as an example of how to auto-generate a synthetic dataset of license plates for training. The auto-generation uses several parameters to control the randomness in the dataset, such as allowed characters, font size, font color, character separation etc. (Please refer to the notebook for more details). In order to auto-generate the synthetic dataset, please provide the following:
     - Clean license plate images with no characters in the `dataset/plates/` folder
     - `.ttf` font files in the `dataset/fonts/` folder

   NOTE: We recommend the autogenerated training set to contain at least 4 million images

   - Start training on your dataset:
     - Start from our pre-trained weights in `pre_trained/lprnet.pth` (you can also download it from here)

```
python train_LPRNet.py --train_img_dirs path/to/train/images --test_
↪img_dirs path/to/test/images --max_epoch 30 --train_batch_size 64 --
↪test_batch_size 32 --resume_epoch 15 --pretrained_model pre_trained/
↪lprnet.pth --save_folder runs/exp0/ --test_interval 2000
```

     - Or train from scratch

```
python train_LPRNet.py --train_img_dirs path/to/train/images --test_
↪img_dirs path/to/test/images --max_epoch 15 --save_folder runs/exp0/
```

2. **Export to ONNX**

   Export the model to ONNX using the following command:

```
python export.py --onnx lprnet.onnx --weights /path/to/trained/model.pth
```

Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the model YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/lprnet.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt lprnet.onnx --calib-path /path/to/calibration/imgs/dir/ --
↪yaml path/to/lprnet.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Since it's an Hailo model, calibration set must be manually supplied.

More details about YAML files are presented *here*.

7.3. Person-Face Detection

Hailo's person-face detection network (*yolov5s_personface*) is based on YOLOv5s and was trained in-house with two classes [person, face]. It can work under various lighting conditions, number of people, and numerous camera angles.

7.3.1. Model Details

Architecture

- YOLOv5s
- Number of parameters: 7.25M
- GMACS: 8.38G
- Accuracy* : 47.5 mAP
  * Evaluated on internal dataset containing 6000 images

Inputs

- RGB image with various input sizes
  - Image resize to 640x640x3 occurs on-chip
- Image normalization occurs on-chip

Outputs

- Three output tensors with sizes of 20x20x21, 40x40x21 and 80x80x21
- Each output contains 3 anchors that hold the following information:
  - Bounding box coordinates ((x,y) centers, height, width)
  - Box objectness confidence score
  - Class probablity confidence score per class
- The above 7 values per anchor are concatenated into the 21 output channels

---

Comparison with Different Models

The table below shows the performance of our trained network on an internal validation set containing 6000 images, compared to other benchmark models from the model zoo*.

network	Person mAP (@IoU=0.5:0.95)
yolov5s_personface	**34.2**
yolov5s*	23.0
yolov5m*	25.6

* Benchmark models were trained on all COCO classes, and evaluated on our internal validation set, on 'Person' class only.

Download

The pre-compiled network can be downloaded from here

Use the following command to measure model performance on hailo's HW:

```
hailortcli benchmark yolov5s_personface.hef
```

7.3.2. Train Person-Face Detection on a Custom Dataset

Here we describe how to finetune Hailo's person-face detection network with your own custom dataset.

Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

Environment Preparations

1. **Build the docker image:**

```
cd hailo_model_zoo/hailo_models/personface_detection/
docker build --build-arg timezone=`cat /etc/timezone` -t personface_
↪detection:v0 .
```

   - This command will build the docker image with the necessary requirements using the Dockerfile that exists in this directory.

2. **Start your docker:**

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/
↪local/data/dir:/path/to/docker/data/dir personface_detection:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `personface_detection:v0` the name of the docker image.

Finetuning and exporting to ONNX

1. **Train the network on your dataset**

   Once the docker is started, you can train the person-face detector on your custom dataset. We recommend following the instructions for YOLOV5 training that can be found in here. The important steps are specified below:

   - Update the dataset config file `data/personface_data.yaml` with the paths to your training and validation images files.

```
#update your data paths
train: /path/to/personface/images/train/
val: /path/to/personface/images/val
# number of classes
nc: 2
# class names
names: ['person', 'face']
```

- Start training on your dataset starting from our pre-trained weights in weights/yolov5s_personface.pt (you can also download it from here

```
python train.py --data ./data/personface_data.yaml --cfg ./models/yolov5s_
↪personface.yaml --weights ./weights/yolov5s_personface.pt --epochs 300 --
↪batch 128 --device 1,2,3,4
```

2. **Export to ONNX** Export the model to ONNX using the following command:

```
python models/export.py --weights ./runs/exp<#>/weights/best.pt --img-size 640
↪--batch-size 1
```

- The best model's weights will be saved under the following path:
  `./runs/exp<#>/weights/best.pt`
  , where <#> is the experiment number.
- Export at 640x640 with batch size 1

Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment.
Choose the model YAML from our networks configuration directory, i.e. hailo_model_zoo/cfg/networks/yolov5s_personface.yaml, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov5s_personface.onnx --calib-path /path/to/calibration/
↪imgs/dir/ --yaml path/to/yolov5s_personface.yaml
```

- --ckpt - path to your ONNX file.
- --calib-path - path to a directory with your calibration images in JPEG/png format
- --yaml - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Since it's an Hailo model, calibration set must be manually supplied.
- On yolo.yaml, change preprocessing.input_shape if changed on retraining

More details about YAML files are presented *here*.

7.3.3. Anchors Extraction

The training flow will automatically try to find more fitting anchors values then the default anchors. In our TAPPAS environment we use the default anchors, but you should be aware that the resulted anchors might be different.
The model anchors can be retrieved from the trained model using the following snnipet:

```
m = torch.load("last.pt")["model"]
detect = list(m.children())[0][-1]
print(detect.anchor_grid)
```

7.4. Person-ReID

Hailo's person Re-Identification network is based on RepVGG_A0 and was trained in-house using several Person ReID datasets. It can work under various lighting conditions and numerous camera angles. 2 Models were trained independently - using 512 & 2048 embedding dimensions.

7.4.1. Model Details

Architecture (2048 / 512)

- RepVGG_A0
- Number of parameters: 9.65M / 7.68M
- GMACS: 0.89 / 0.89
- Rank1* : 89.8% / 89.3%
  * Evaluated on Market1501 dataset

Inputs

- RGB image with various input sizes
  - Image resize to 256x128x3 occurs on-chip
- Image normalization occurs on-chip

Outputs

- Single embedding vector (2048 / 512 dim) per query

7.4.2. Comparison with Different Models

The table below shows the performance of our trained network on Market1501 dataset.

network	Person Rank1
repvgg_a0_person_reid_512	**89.3**
repvgg_a0_person_reid_2048	**89.8**

Download

The pre-compiled network can be download from:

- 512-dim
- 2048-dim

  Use the following command to measure model performance on hailo's HW:

  ```
  hailortcli benchmark repvgg_a0_person_reid_512.hef
  hailortcli benchmark repvgg_a0_person_reid_2048.hef
  ```

7.4.3. Train Person-ReID on a Custom Dataset

Here we describe how to finetune Hailo's person-reid network with your own custom dataset.

Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

Environment Preparations

1. **Build the docker image:**

   ```
   cd hailo_model_zoo/hailo_models/reid/
   docker build --build-arg timezone=`cat /etc/timezone` -t person_reid:v0 .
   ```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"

- `user` - username for a local non-root user. Defaults to 'hailo'.
- `group` - default group for a local non-root user. Defaults to 'hailo'.
- `uid` - user id for a local non-root user.
- `gid` - group id for a local non-root user.

This command will build the docker image with the necessary requirements using the Dockerfile that exists in this directory.

2. **Start your docker:**

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪drive:<span
val="docker_vol_path">/path/to/docker/dir person_reid:v0
```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `personface_detection:v0` the name of the docker image.

Finetuning and exporting to ONNX

1. **Train the network on your dataset**
   Once the docker is started, you can train the person-reid model on your custom dataset. We recommend following the instructions from the torchreid repo that can be found in here.

   - Insert your dataset as described in use-your-own-dataset.
   - Start training on your dataset starting from our pre-trained weights in `models/repvgg_a0_person_reid_512.pth` or `models/repvgg_a0_person_reid_2048.pth` (you can also download it from 512-dim & 2048-dim) - to do so, you can edit the added yaml `configs/repvgg_a0_hailo_pre_train.yaml` and take a look at the examples in torchreid.

   ```
   python scripts/main.py --config-file configs/repvgg_a0_hailo_pre_train.
   ↪yaml
   ```

2. **Export to ONNX**
   Export the model to ONNX using the following command:

```
python scripts/export.py --model_name <model_name> --weights /path/to/model/pth
```

Compile the Model using Hailo Model Zoo

In case you exported to onnx based on one of our provided RepVGG models, you can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the model YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/repvgg_a0_person_reid_512.yaml` (or 2048), and run compilation using the model zoo:

```
hailomz compile --ckpt repvgg_a0_person_reid_512.onnx --calib-path /path/to/
↪calibration/imgs/dir/ --yaml path/to/repvgg_a0_person_reid_512.yaml
```

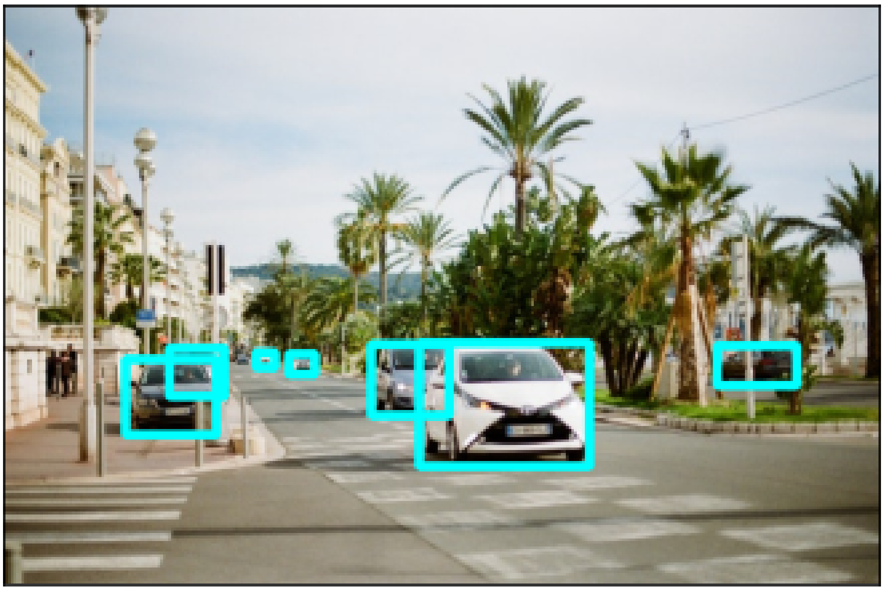
- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Since it's an Hailo model, calibration set must be manually supplied.
- On market1501.yaml, change `preprocessing.input_shape` if changed on retraining

More details about YAML files are presented *here*.

7.5. Vehicle Detection



Hailo's vehicle detection network (*yolov5m_vehicles*) is based on YOLOv5m and was trained in-house with a single class. It can work under various weather and lighting conditions, and numerous camera angles.

7.5.1. Model Details

Architecture

- YOLOv5m
- Number of parameters: 21.47M
- GMACS: 25.63
- Accuracy* : 46.0 mAP
  * Evaluated on internal dataset containing 5000 images

Inputs

- RGB image with size of 1080x1920x3
  - Image resize to 640x640x3 occurs on-chip
- Image normalization occurs on-chip

Outputs

- Three output tensors with sizes of 20x20x18, 40x40x18 and 80x80x18
- Each output contains 3 anchors that hold the following information:
  - Bounding box coordinates ((x,y) centers, height, width)
  - Box objectness confidence score
  - Class probability confidence score
- The above 6 values per anchor are concatenated into the 18 output channels

---

Comparison with Different Models

The table below shows the performance of our trained network on an internal validation set containing 5000 images, compared with the performance of other benchmark models from the model zoo*.

network	**Vehicle mAP (@IoU=0.5:0.95)**
yolov5m_vehicles	**46.0**
yolov5m	33.95
yolov4_leaky	33.13
yolov3_gluon	29.89

* Benchmark models were trained on all COCO classes

Download

The pre-compiled network can be downloaded from here.

Use the following command to measure model performance on hailo's HW:

```
hailortcli benchmark yolov5m_vehicles.hef
```

7.5.2. Train Vehicle Detection on a Custom Dataset

Here we describe how to finetune Hailo's vehicle detection network with your own custom dataset.

Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

Environment Preparations

1. **Build the docker image**

```
cd hailo_model_zoo/hailo_models/vehicle_detection/
docker build --build-arg timezone=`cat /etc/timezone` -t vehicle_detection:v0 .
```

   - This command will build the docker image with the necessary requirements using the Dockerfile that exists in this directory.

2. **Start your docker:**

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪drive:/path/to/docker/dir vehicle_detection:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `vehicle_detection:v0` the name of the docker image.

Finetuning and exporting to ONNX

1. **Train the network on your dataset** Once the docker is started, you can train the vehicle detector on your custom dataset. We recommend following the instructions for YOLOV5 training that can be found in here. The important steps are specified below:

   - Update the dataset config file `data/vehicles.yaml` with the paths to your training and validation images files.

   ```
   #update your data paths
   train: /path/to/vehicles/training/images/
   val: /path/to/vehicles/validation/images/

   # number of classes
   nc: 1

   # class names
   names: ['vehicle']
   ```

   - Start training on your dataset starting from our pre-trained weights in `weights/yolov5m_vehicles.pt` (you can also download it from here)

   ```
   python train.py --data ./data/vehicles.yaml --cfg ./models/yolov5m.yaml --
   ↪weights ./weights/yolov5m_vehicles.pt --epochs 300 --batch 128 --device 1,2,3,
   ↪4
   ```

2. **Export to ONNX** Export the model to ONNX using the following command:

   ```
   python models/export.py --weights ./runs/exp<#>/weights/best.pt --img 640 --
   ↪batch 1
   ```

   - The best model's weights will be saved under the following path: `./runs/exp<#>/weights/best.pt`, where <#> is the experiment number.

Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment.
Choose the model YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov5m_vehicles.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov5m_vehicles.onnx --calib-path /path/to/calibration/
↪imgs/dir/ --yaml path/to/yolov5m_vehicles.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Since it's an Hailo model, calibration set must be manually supplied.

- This model has an on-chip resize from the video input [1080x1920] to the model's input ([640x640], the resolution the model is trained with). Model Zoo automatically adds the resize for this model using a model script command on yolov5m_vehicles.alls. Therefore, the `input_resize` command should be updated if the video input resolution is different (or even removed if it is equal to the resolution the model is trained with).
- On yolov5m_vehicles.yaml, change `input_resize` field to match the input_resize command on the model script.
- On yolo.yaml, change `preprocessing.input_shape` if the network is trained on other resolution.

More details about YAML files are presented *here*.

7.5.3. Anchors Extraction

The training flow will automatically try to find more fitting anchors values then the default anchors. In our TAPPAS environment we use the default anchors, but you should be aware that the resulted anchors might be different.

The model anchors can be retrieved from the trained model using the following snnipet:

```
m = torch.load(”last.pt”)[”model”]
detect = list(m.children())[0][-1]
print(detect.anchor_grid)
```

8. Datasets

The Hailo Model Zoo works with TFRecord files which store the images and labels of the dataset for evaluation and calibration.

The instructions on how to create the TFRecord files are given below. By default, datasets are stored in the following path:

```
~/.hailomz
```

We recommend to define the data directory path yourself, by setting the `HMZ_DATA` environment variable.

```
export HMZ_DATA=/new/path/for/storage/
```


8.1. ImageNet

To evaluate/optimize/compile the classification models of the Hailo Model Zoo you should generate the ImageNet TFRecord files (manual download is required).

1. Download the ImageNet dataset from here. The expected dataset structure:

```
imagenet
|_ train
|  |_ n01440764
|  |_ ...
|  |_ n15075141
|_ val
|  |_ n01440764
|  |_ ...
```

(continues on next page)

(continued from previous page)

```
| |_n15075141
|_...
```

* To avoid downloading the ImageNet training data, you may consider using the validation dataset for calibration (does not apply for finetune).

2. Run the create TFRecord scripts:

```
python hailo_model_zoo/datasets/create_imagenet_tfrecord.py val --img /path/
↪to/imagenet/val/
python hailo_model_zoo/datasets/create_imagenet_tfrecord.py calib --img /path/
↪to/imagenet/train/
```

8.2. COCO2017

To evaluate/optimize/compile the object detection / pose estimation models of the Hailo Model Zoo you should generate the COCO (link) TFRecord files.
Run the create TFRecord scripts to download the dataset and generate the TFRecord files:

```
python hailo_model_zoo/datasets/create_coco_tfrecord.py val2017
python hailo_model_zoo/datasets/create_coco_tfrecord.py calib2017
```

To evaluate/optimize/compile the single person pose estimation models of the Hailo Model Zoo you should generate the single-person COCO TFRecord files.
Run the create TFRecord scripts to download the dataset and generate the TFRecord files:

```
python hailo_model_zoo/datasets/create_coco_single_person_tfrecord.py val2017
python hailo_model_zoo/datasets/create_coco_single_person_tfrecord.py calib2017
```

8.2.1. Manual Download (Optional)

1. Download COCO (here). The expected dataset structure:

   Annotations:

```
annotations
|_ captions_train2017.json
|_ captions_val2017.json
|_ instances_train2017.json
|_ instances_val2017.json
|_ person_keypoints_train2017.json
|_ person_keypoints_val2017.json
```

   Validation set:

```
val2017
|_ 000000000139.jpg
|_ 000000000285.jpg
```

(continues on next page)

(continued from previous page)

```
|_ 000000000632.jpg
|_ 000000000724.jpg
|_ 000000000776.jpg
|_ 000000000785.jpg
|_ 000000000802.jpg
|_ 000000000872.jpg
|_ 000000000885.jpg
|_ ...
```

Training set:

```
train2017
|_ 000000000009.jpg
|_ 000000000025.jpg
|_ 000000000030.jpg
|_ 000000000034.jpg
|_ 000000000036.jpg
|_ 000000000042.jpg
|_ 000000000049.jpg
|_ 000000000061.jpg
|_ 000000000064.jpg
|_ ...
```

2. Run the creation scripts:

```
python hailo_model_zoo/datasets/create_coco_tfrecord.py val2017 --img /path/
↪to/val2017 --det /path/to/annotations
python hailo_model_zoo/datasets/create_coco_tfrecord.py calib2017 --img /path/
↪to/train2017 --det /path/to/annotations
```

8.3. Cityscapes

To evaluate/optimize/compile the semantic segmentation models of the Hailo Model Zoo you should generate the Cityscapes TFRecord files (manual download is required).

1. Download the Cityscapes dataset from here. The expected dataset structure:

```
Cityscapes
|_ gtFine
|  |_ train
|  |_ test
|  |_ val
|_ leftImg8bit
|  |_ train
|  |_ test
|  |_ val
|  |_ train_extra
|_ ...
```

2. Run the create TFRecord scripts:

```
python hailo_model_zoo/datasets/create_cityscapes_tfrecord.py val --data /
↪path/to/Cityscapes/
python hailo_model_zoo/datasets/create_cityscapes_tfrecord.py calib --data /
↪path/to/Cityscapes/
```

8.4. WIDERFACE

To evaluate/optimize/compile the face detection models of the Hailo Model Zoo you should generate the WIDERFACE (link) TFRecord files.
Run the create TFRecord scripts to download the dataset and generate the TFRecord files:

```
python hailo_model_zoo/datasets/create_widerface_tfrecord.py calib
python hailo_model_zoo/datasets/create_widerface_tfrecord.py val
```

8.4.1. Manual Download (Optional)

1. Download the following from here:
   - WIDER Face Training Images
   - WIDER Face Validation Images
   - Face annotations
2. Download the following from here
   - wider_hard_val.mat

   Expected directory structure:

```
widerface/
|_ wider_face_split
|  |_ readme.txt
|  |_ wider_face_test_filelist.txt
|  |_ wider_face_test.mat
|  |_ wider_face_train_bbx_gt.txt
|  |_ wider_face_train.mat
|  |_ wider_face_val_bbx_gt.txt
|  |_ wider_face_val.mat
|  |_ wider_hard_val.mat
|_ WIDER_train
|  |_ images
|     |_ 0--Parade
|     |_ 10--People_Marching
|     |_ 11--Meeting
|     |_ ...
|_ WIDER_val
   |_ images
      |_ 0--Parade
      |_ 10--People_Marching
      |_ 11--Meeting
      |_ ...
```

3. Run the creation scripts

```
python hailo_model_zoo/datasets/create_widerface_tfrecord.py calib --img /
↪path/to/widerface --gt_mat_path /path/to/wider_face_split --hard_mat_path /
↪path/to/wider_face_split
python hailo_model_zoo/datasets/create_widerface_tfrecord.py val --img /path/
↪to/widerface --gt_mat_path /path/to/wider_face_split --hard_mat_path /path/
↪to/wider_face_split
```


8.5. VisDrone

To evaluate/optimize/compile the visdrone object detection models of the Hailo Model Zoo you should generate the VisDrone (link) TFRecord files.
Run the create TFRecord scripts to download the dataset and generate the TFRecord files:

```
python hailo_model_zoo/datasets/create_visdrone_tfrecord.py train
python hailo_model_zoo/datasets/create_visdrone_tfrecord.py val
```

8.5.1. Manual Download (Optional)

1. Download VisDrone (here). The expected dataset structure:

   Training set:

   ```
   VisDrone2019-DET-train/
   |_ annotations
   |  |_ 0000002_00005_d_0000014.txt
   |  |_ 0000002_00448_d_0000015.txt
   |  |_ ...
   |_ images
     |_ 0000002_00005_d_0000014.jpg
     |_ 0000002_00448_d_0000015.jpg
     |_ ...
   ```

   Validation set:

   ```
   VisDrone2019-DET-val/
   |_ annotations
   |  |_ 0000001_02999_d_0000005.txt
   |  |_ 0000001_03499_d_0000006.txt
   |  |_ ...
   |_ images
     |_ 0000001_02999_d_0000005.jpg
     |_ 0000001_03499_d_0000006.jpg
     |_ ...
   ```

2. Run the creation scripts:

   ```
   python hailo_model_zoo/datasets/create_visdrone_tfrecord.py train -d /path/to/
   ↪VisDrone2019-DET-train
   python hailo_model_zoo/datasets/create_visdrone_tfrecord.py val -d /path/to/
   ↪VisDrone2019-DET-val
   ```

8.6. Pascal VOC augmented dataset

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_pascal_tfrecord.py calib
python hailo_model_zoo/datasets/create_pascal_tfrecord.py val
```

8.6.1. Manual Download (Optional)

1. Download the dataset from here. Expected dataset structure:

```
benchmark_RELEASE
|_ dataset
 |_ cls
 |_ img
 |_ inst
 |_ train.txt
 |_ val.txt
```

2. run the creation scripts:

```
python hailo_model_zoo/datasets/create_pascal_tfrecord.py calib --root
↪benchmark_RELEASE/dataset
python hailo_model_zoo/datasets/create_pascal_tfrecord.py val --root
↪benchmark_RELEASE/dataset
```

8.7. D2S augmented dataset

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_d2s_tfrecord.py calib
python hailo_model_zoo/datasets/create_d2s_tfrecord.py val
```

8.7.1. Manual Download (Optional)

1. Download the dataset from here. Extract using 'tar -xf d2s_images_v1.1.tar.xz'. Expected dataset structure:

```
|_ images
 |_ D2S_000200.jpg
 |_ D2S_000201.jpg
 |_ ...
```

2. Download the annotations from here. Extract using 'tar -xf d2s_annotations_v1.1.tar.xz'. Expected annotations structure:

```
|_ annotations
 |_ D2S_augmented.json
 |_ D2S_validation.json
 |_ ...
```

3. run the creation scripts:

```
python hailo_model_zoo/datasets/create_d2s_tfrecord.py calib --img /path/to/
↪dataset --det /path/to/annotations/D2S_augmented.json
python hailo_model_zoo/datasets/create_d2s_tfrecord.py val --img /path/to/
↪dataset --det /path/to/annotations/D2S_validation.json
```


8.8. NYU Depth V2

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_nyu_depth_v2_tfrecord.py calib
python hailo_model_zoo/datasets/create_nyu_depth_v2_tfrecord.py val
```

8.8.1. Manual Download (Optional)

1. Download the dataset from here. Extract using 'tar -xf nyudepthv2.tar.gz'. Expected dataset structure:

```
|_ train
 |_ study_0300
   |_ 00626.h5
   |_ 00631.h5
   |_ ...
 |_ ...
|_ val
 |_ official
   |_ 00001.h5
   |_ 00002.h5
   |_ 00009.h5
   |_ 00014.h5
   |_ ...
```

2. run the creation scripts:

```
python hailo_model_zoo/datasets/create_nyu_depth_v2_tfrecord.py calib --data .
↪/nyu_depth_v2/
python hailo_model_zoo/datasets/create_nyu_depth_v2_tfrecord.py val --data ./
↪nyu_depth_v2/
```

8.9. AFLW2k3d and 300W-LP

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_300w-lp_tddfa_tfrecord.py
python hailo_model_zoo/datasets/create_aflw2k3d_tddfa_tfrecord.py
```

8.9.1. Manual Download (Optional)

1. Download the augmented_cropped 300W-LP dataset from here and extract. Expected structure:

```
train_aug_120x120
|_ AFW_AFW_1051618982_1_0_10.jpg
|_ AFW_AFW_1051618982_1_0_11.jpg
|_ AFW_AFW_1051618982_1_0_12.jpg
|_ AFW_AFW_1051618982_1_0_13.jpg
|_ AFW_AFW_1051618982_1_0_1.jpg
|_ AFW_AFW_1051618982_1_0_2.jpg
|_ AFW_AFW_1051618982_1_0_3.jpg
|_ AFW_AFW_1051618982_1_0_4.jpg
|_ ...
```

2. Run


```
python hailo_model_zoo/datasets/create_300w-lp_tddfa_tfrecord.py --dir /path/
↪to/train_aug_120x120
```

3. Download the following files:

   - the official dataset from here
   - the cropped dataset from here
   - The following files from here
     - AFLW2000-3D.pose.npy
     - AFLW2000-3D.pts68.npy
     - AFLW2000-3D-Reannotated.pts68.npy
     - AFLW2000-3D_crop.roi_box.npy

   The expected structure:

```
aflw2k3d_tddfa
|_ AFLW2000-3D_crop.roi_box.npy
|_ AFLW2000-3D.pose.npy
|_ AFLW2000-3D.pts68.npy
|_ AFLW2000-3D-Reannotated.pts68.npy
|_ test.data
  |_ AFLW2000
  |  |_ Code
  |  |  |_ Mex
  |  |  |_ ModelGeneration
  |  |_ image00002.jpg
  |  |_ image00002.mat
  |  |_ image00004.jpg
  |  |_ image00004.mat
  |  |_ ...
  |_ AFLW2000-3D_crop
  |  |_ image00002.jpg
  |  |_ image00004.jpg
  |  |_ image00006.jpg
  |  |_ image00008.jpg
  |  |_ ...
  |_ AFLW2000-3D_crop.list
  |_ AFLW_GT_crop
  |  |_ ...
  |_ AFLW_GT_crop.list
```

4. Run the following:

```
python hailo_model_zoo/datasets/create_aflw2k3d_tddfa_tfrecord.py --dir /path/
↪to/aflw2k3d_tddfa
```

8.10. Hand Landmark

Run the creation script:

```
python hailo_model_zoo/datasets/create_hand_landmark_tfrecord.py
```

8.10.1. Manual Download (Optional)

1. Download the dataset from here and extract. Expected structure:

```
Hands         00 000
|_ Hand_0011695.jpg
|_ Hand_0011696.jpg
|_ Hand_0011697.jpg
|_ ...
```

2. Run

```
python hailo_model_zoo/datasets/create_hand_landmark_tfrecord.py --img /path/
↪to/Hands
```

8.11. Market1501

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_market_tfrecord.py val
python hailo_model_zoo/datasets/create_market_tfrecord.py calib
```

8.11.1. Manual Download (Optional)

1. Download the dataset from here and extract.

   Expected structure:

```
Market-1501-v15.09.15
|_ bounding_box_test
 |_ 0000_c1s1_000151_01.jpg
 |_ 0000_c1s1_000376_03.jpg
 |_ ...
|_ bounding_box_train
 |_ 0002_c1s1_000451_03.jpg
 |_ 0002_c1s1_000551_01.jpg
 |_ ...
|_ gt_bbox
 |_ 0001_c1s1_001051_00.jpg
 |_ 0001_c1s1_002301_00.jpg
 |_ ...
|_ gt_query
 |_ 0001_c1s1_001051_00_good.mat
 |_ 0001_c1s1_001051_00_junk.mat
 |_ ...
|_ query
 |_ 0001_c1s1_001051_00.jpg
 |_ 0001_c2s1_000301_00.jpg
 |_ ...
```

2. Run

```
python hailo_model_zoo/datasets/create_market_tfrecord.py val --img path/to/
↪Market-1501-v15.09.15/
python hailo_model_zoo/datasets/create_market_tfrecord.py calib --img path/to/
↪Market-1501-v15.09.15/bounding_box_train/
```


8.12. PETA

To evaluate/optimize/compile the person attribute models of the Hailo Model Zoo you should generate the PETA TFRecord files (manual download is required).

1. Download the PETA dataset from here. The expected dataset structure:

```
PETA
|_ images
|  |_ 00001.png
|  |_ ...
|  |_ 19000.png
|_ PETA.mat
```

2. Run the create TFRecord scripts:

```
python hailo_model_zoo/datasets/create_peta_tfrecord.py test --data /path/to/
↪PETA/
python hailo_model_zoo/datasets/create_peta_tfrecord.py train --data /path/to/
↪PETA/
```

8.13. CelebA

To evaluate/optimize/compile the face attribute models of the Hailo Model Zoo you should generate the CelebA TFRecord files (manual download is required).

1. Download the CelebA dataset from here. The expected dataset structure:

```
Celeba
|_ img_align_celeba_png
|  |_ 000001.jpg
|  |_ ...
|  |_ 202599.jpg
|_ list_attr_celeba.txt
|_ list_eval_partition.txt
```

2. Run the create TFRecord scripts:

```
python hailo_model_zoo/datasets/create_celeba_tfrecord.py val --data /path/to/
↪CelebA/
python hailo_model_zoo/datasets/create_celeba_tfrecord.py train --data /path/
↪to/CelebA/
```

8.14. LFW

To evaluate/optimize/compile the face recognition models of the Hailo Model Zoo you should generate the arcface_lfw TFRecord files

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_arcface_lfw_tfrecord.py calib
python hailo_model_zoo/datasets/create_arcface_lfw_tfrecord.py val
```


8.14.1. Manual Download (Optional)

1. Download LFW dataset from here
2. Download LFW pairs file from *here <http://vis-www.cs.umass.edu/lfw/pairs.txt>*
3. Run the scripts:

```
python hailo_model_zoo/datasets/create_arcface_lfw_tfrecord.py calib -
↪-tgz /path/to/lfw.tgz --pairs /path/to/pairs.txt
python hailo_model_zoo/datasets/create_arcface_lfw_tfrecord.py val --
↪tgz /path/to/lfw.tgz --pairs /path/to/pairs.txt
```

8.15. BSD100

To evaluate/optimize/compile the super resolution models of the Hailo Model Zoo you should generate the BSD100 TFRecord files.

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_bsd100_tfrecord.py val
python hailo_model_zoo/datasets/create_bsd100_tfrecord.py calib
```

8.15.1. Manual Download (Optional)

1. Download the BSD100 dataset from here and extract. The expected dataset structure:

```
BSD100
|_ GTmod12
|  |_ 101085.png
|  |_ ...
|  |_ 97033.png
|_ GTmod16
|  |_ ...
|_ LRbicx8
|  |_ ...
|_ LRbicx4
|  |_ ...
|_ LRbicx3
|  |_ ...
|_ LRbicx2
|  |_ ...
|_ LRbicx16
|  |_ ...
|_ original
|  |_ ...
```

2. Run the scripts:

```
python hailo_model_zoo/datasets/create_bsd100_tfrecord.py val --lr /path/to/
↪LRbicx4 --hr /path/to/GTmod12
python hailo_model_zoo/datasets/create_bsd100_tfrecord.py calib --lr /path/to/
↪LRbicx4 --hr /path/to/GTmod12
```

8.16. CIFAR100

To evaluate/optimize/compile the CLIP models of the Hailo Model Zoo you should generate the CIFAR100 TFRecord files.

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_cifar100_tfrecord.py val
python hailo_model_zoo/datasets/create_cifar100_tfrecord.py calib
```

8.17. LOL

To evaluate/optimize/compile the low light enhancement models of the Hailo Model Zoo you should generate the LOL TFRecord files.

Run the creation scripts:

```
python hailo_model_zoo/datasets/create_lol_tfrecord.py val
python hailo_model_zoo/datasets/create_lol_tfrecord.py calib
```

8.17.1. Manual Download (Optional)

1. Download the LOL dataset from here and extract. The expected dataset structure:

```
lol_dataset
|_ eval15
  |_ high
  |   |_ 111.png
  |   |_ 146.png
  |   |_ ...
  |_ low
  |   |_ 111.png
  |   |_ 146.png
  |   |_ ...
|_ our485
  |_ high
  |   |_ 100.png
  |   |_ 101.png
  |   |_ ...
  |_ low
  |   |_ 100.png
  |   |_ 101.png
  |   |_ ...
```

2. Run the scripts:

```
python hailo_model_zoo/datasets/create_lol_tfrecord.py val --ll /path/to/val/
↪lowlight/images --lle /path/to/val/highlight/images
python hailo_model_zoo/datasets/create_lol_tfrecord.py calib --ll /path/to/
↪train/lowlight/images --lle /path/to/train/highlight/images
```


9. Benchmarks

In order to measure FPS, power and latency of the Hailo Model Zoo networks you can use the HailoRT command line interface.
For more information please refer to the HailoRT documentation in hailo.ai.

9.1. Example

The HailoRT command line interface works with the Hailo Executable File (HEF) of the model.
To generate the HEF file use the following command:

```
hailomz compile <model_name>
```

After building the HEF you will be able to measure the performance of the model by using the HailoRT command line interface.
Example for measuring performance of resnet_v1_50:

```
hailortcli benchmark resnet_v1_50.hef
```

Example output:

```
=======
Summary
=======
FPS   (hw_only)          = 1328.83
    (streaming)         = 1328.8
Latency (hw)            = 2.93646 ms
Power in streaming mode (average) = 3.19395 W
             (max)    = 3.20456 W
```

9.2. Using Datasets from the Hailo Model Zoo

To use datasets from the Hailo Model Zoo, you can use the command:

```
python hailo_model_zoo/tools/conversion_tool.py /path/to/tfrecord_file resnet_v1_
↪50
```

which will generate a bin file with serialized images. This bin file can be used inside the HailoRT:

```
hailortcli benchmark resnet_v1_50.hef --input-files tfrecord_file.bin
```


10. Hailo Model Zoo YAML Description

10.1. Properties

- **network**
  - **network_name** *(['string', 'null'])*: The network name. Default: `None`.
- **paths**
  - **network_path** *(array)*: Path to the network files, can be ONNX / TF Frozen graph (pb) / TF CKPT files / TF2 saved model.
  - **alls_script** *(['string', 'null'])*: Path to model script in the alls directory.
- **parser**
  - **nodes** *(['array', 'null'])*: List of [start node, [end nodes]] for parsing.
    For example: ["resnet_v1_50/conv1/Pad", "resnet_v1_50/predictions/Softmax"].
  - **normalization_params**: Add normalization on chip.
    For example: normalization_params: { "normalize_in_net": true, "mean_list": [123.68, 116.78, 103.94], "std_list": [58.395, 57.12, 57.375] }.
    * **normalize_in_net** *(boolean)*: Whether or not the network includes an on-chip normalization layer. If so, the normalization layer will appear on the .alls file that is used. Default: `False`.
      Example alls command: `normalization1 = normalization([123.675, 116.28, 103.53], [58.395, 57.12, 57.375])`
      If the alls doesn't include the required normalization, then the MZ (and the user application) will apply normalization before feeding inputs to the network
    * **mean_list** *(['array', 'null'])*: Used only in case normalize_in_net=false. The MZ automatically performs normalization to the calibration set, so the network receives already-normalized input (saves the user the need to normalize the dataset). Default: `None`.
    * **std_list** *(['array', 'null'])*: Used only in case normalize_in_net=false: The MZ automatically performs normalization to the calibration set, so the network receives already-normalized input (saves the user the need to normalize the dataset). Default: `None`.
  - **start_node_shapes** *(['array', 'null'])*: Dict for setting input shape of supported models that does not explicitly use it.
    For example, models with input shape of [?, ?, ?, 3] can be set with {"Preprocessor/sub:0": [1, 512, 512, 3]}. Default: `None`.
- **preprocessing**
  - **network_type** *(['string', 'null'])*: The type of the network. Default: `classification`.
  - **meta_arch** *(['string', 'null'])*: The network preprocessing meta-architecture.
    For example: mobilenet_ssd, efficientnet, etc. Default: `None`.
  - **padding_color** *(['integer', 'null'])*: On the training environments, the input images to the model have used this color to indicate "padding" around resized images. Default: `114` for YOLO architectures, `0` for others.
- **quantization**
  - **calib_set** *(['array', 'null'])*: List contains the calibration set path.
    For example: ['models_files/imagenet/2021-06-20/imagenet_calib.tfrecord']. Default: `None`.
  - **calib_set_name** *(['string', 'null'])*: Name of the dataset used for calibration. By default (null) uses the evaluation dataset name. Default: `None`.
- **postprocessing**
  - **meta_arch** *(['string', 'null'])*: Postprocessing meta architecture name.

For example: yolo_v3, yolo_v4, etc. Default: None.

- **postprocess_config_file** *(['string', 'null'])*: Path to a file with the postprocessing configuration (for example, for offloading NMS to the Hailo-8). Default: None.
- **device_pre_post_layers**: Whether to use postprocessing on chip or do it on the host.
  - **bilinear** *(boolean)*: Activate the bilinear PPU layer. Default: False.
  - **argmax** *(boolean)*: Activate the Argmax PPU layer. Default: False.
  - **softmax** *(boolean)*: Activate the Softmax PPU layer. Default: False.
  - **nms** *(boolean)*: Activate the NMS PPU layer and the relevant decoding layers. Default: False.

- **evaluation**
  - **dataset_name** *(['string', 'null'])*: Name of the dataset to be used in evaluation. Default: None.
  - **data_set** *(['string', 'null'])*: Path to TFrecord dataset for evaluation. Default: None.
  - **classes** *(['integer', 'null'])*: Number of classes in the model. Default: 1000.
  - **labels_offset** *(['integer', 'null'])*: Offset of labels. Default: 0.
  - **network_type** *(['string', 'null'])*: The type of the network used for evaluation.
    Use this field if evaluation type is different than preprocessing type. Default: None.
- **hn_editor**
  - **yuv2rgb** *(boolean)*: Add YUV to RGB layer. Default: False.
  - **flip** *(boolean)*: Rotate input by 90 degrees. Default: False.
  - **input_resize**: Add resize bilinear layer at the start of the network.
    - **enabled** *(boolean)*: Whether this is enabled or disabled. Default: False.
    - **input_shape** *(array)*: List of input shape to resize from [H, W].
  - **bgr2rgb** *(boolean)*: Add BGR to RGB layer.
    On some training frameworks, the models are trained on BGR inputs. When we want to feed RGB images to the network (whether on the MZ or on the user application), we need to transform the images from RGB to BGR. The MZ automatically inserts this layer to the on-chip model.
    We have already set the "bgr2rgb" flag on the yaml files that correspond to the relevant retraining dockers. Default: False.

10.2. YAML hierarchies

- **The MZ uses hierarchical .yaml infrastructure for configuration. For example, for yolov5m_vehicles:**
  - Network yaml is networks/yolov5m_vehicles.yaml
  - It includes at the top the lines:

```
base:
- base/yolov5.yaml
```

  - Meaning it inherits from base/yolov5.yaml
  - Which inherits from base/yolo.yaml
  - Which inherits from base/base.yaml
- Each property on the child hierarchies replaces the properties on the parent ones. For example, if *preprocessing.input_shape* is defined both in *base/yolov5.yaml* and *base/base.yaml*, the one from *base/yolov5.yaml* will be used
- Therefore, if we want to change some property, we can just update the last child file that is using that property

10.3. Notes for Retraining

- `evaluation` and `postprocessing` properties aren't needed for compilation as they are used by the Model-Zoo for model evaluation (which isn't supported yet for retrained models). Also `info` field is just used for description.
  - Only on YOLOv4 family, the `evaluation.classes` and `postprocessing.anchors.sizes` fields are used for compilation, that's why you should update those values even if just for compilation
- You might want to update those default values on some advanced scenarios:
  - preprocessing.padding_color
    * Change those values only if you have used a different value for training your model
  - parser.normalization_params.normalize_in_net
    * If you have manually changed the normalization values on the retraining docker, and *normalize_in_net=true*, remember to update the corresponding alls command
  - parser.normalization_params.mean_list
    * Update those values if *normalize_in_net=false* and you have manually changed the normalization values on the retraining docker
  - parser.normalization_params.std_list
    * Update those values if *normalize_in_net=false* and you have manually changed the normalization values on the retraining docker

11. Retrain on Custom Dataset

This page will help you ramping-up your training environment for various tasks and architectures. Each architecture has its own README which describes how to:

1. Setup the environment using a compatible Dockerfile.
2. Start the training process.
3. Export your model.

Important: Retraining is not available inside the docker version of Hailo Software Suite. In case you use it, clone the hailo_model_zoo outside of the docker, and perform the retraining there: `git clone https://github.com/hailo-ai/hailo_model_zoo.git`

Object Detection

- *YOLOv3*
- *YOLOv4*
- *YOLOv5*
- *YOLOv8*
- *YOLOX*
- *DAMO-YOLO*
- *NanoDet*
- *SSD*

Pose Estimation

- *CenterPose*

Single Person Pose Estimation

- *MSPN*

Semantic Segmentation

- *FCN*

Instance Segmentation

- *YOLACT*
- *YOLOv8_seg*

Face Recognition

- *ArcFace*

11.1. YOLOv3 Retraining

- To learn more about yolov3 look here

11.1.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you use Hailo Software Suite docker, make sure you are doing all the following instructions outside of this docker.

11.1.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolov3
docker build --build-arg timezone=`cat /etc/timezone` -t yolov3:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.
   - This command will build the docker image with the necessary requirements using the Dockerfile exists in this directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪data/dir:/path/to/docker/data/dir yolov3:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `yolov3:v0` the name of the docker image.

11.1.3. Training and exporting to ONNX

1. Train your model: Once the docker is started, you can start training your YOLOv3.
   - Prepare your custom dataset - Follow the full instructions described here:
     - Create `data/obj.data` with paths to your training and validation `.txt` files, which contain the list of the image paths*.

       ```
       classes = 80
       train = data/train.txt
       valid = data/val.txt
       names = data/coco.names
       backup = backup/
       ```

       * Tip: specify the paths to the training and validation images in the training and validation `.txt` files relative to `/workspace/darknet/`

       Place your training/validation images and labels in your data folder and make sure you update the number of classes.

     - Labels - each image should have labels in YOLO format with corresponding txt file for each image.
   - Start training - The following command is an example for training the yolov3.

   ```
   ./darknet detector train data/obj.data cfg/yolov3.cfg yolov3.weights -map -clear
   ```

   Final trained weights will be available in `backup/` directory.

2. Export to ONNX:
   In order to export your trained YOLOv3 model to ONNX run the following script:

   ```
   python ../pytorch-YOLOv4/demo_darknet2onnx.py cfg/yolov3.cfg /path/to/trained.
   ↪weights /path/to/some/image.jpg 1
   ```

   - The ONNX would be available in `/workspace/darknet/`

11.1.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov3_416.yaml` (for the default YOLOv3 model).

Align the corresponding alls, i.e. `hailo_model_zoo/cfg/alls/base/yolov3_416.alls` with the size of the calibration set using `dataset_size=<number_of_jpgs_in_folder>` parameter.

Run compilation using the model zoo:

```
hailomz compile yolov3_416 --ckpt yolov3_1_416_416.onnx --calib-path /path/to/
↪calibration/imgs/dir/
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- On your desired YOLOv3 YAML, make sure `preprocessing.input_shape` fits your model's resolution.
- For TAPPAS, retrain the model with a resolution of 608x608, and on compilation use `yolov3_gluon.yaml`.

More details about YAML files are presented *here*.

11.2. YOLOv4-leaky Retraining

- To learn more about yolov4 look here

11.2.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.2.2. Environment Preparations

1. Build the docker image:

   ```
   cd hailo_model_zoo/training/yolov4
   docker build --build-arg timezone=`cat /etc/timezone` -t yolov4:v0 .
   ```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.
   - This command will build the docker image with the necessary requirements using the Dockerfile exists in this directory.

2. Start your docker:

   ```
   docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
   ↪data/dir:/path/to/docker/data/dir yolov4:v0
   ```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `yolov4:v0` the name of the docker image.

11.2.3. Training and exporting to ONNX

1. Train your model:

   Once the docker is started, you can start training your YOLOv4-leaky.

   - Prepare your custom dataset - Follow the full instructions described here:
     - Create `data/obj.data` with paths to your training and validation `.txt` files, which contain the list of the image paths*.

       ```
       classes = 80
       train  = data/train.txt
       valid  = data/val.txt
       names = data/coco.names
       backup = backup/
       ```

       * Tip: specify the paths to the training and validation images in the training and validation `.txt` files relative to `/workspace/darknet/`
       Place your training/validation images and labels in your data folder and make sure you update the number of classes.

     - Labels - each image should have labels in YOLO format with corresponding txt file for each image.
   - Start training - The following command is an example for training the yolov4-leaky model.

   ```
   ./darknet detector train data/obj.data cfg/yolov4-leaky.cfg -map -clear
   ```

   Final trained weights will be available in `backup/` directory.

2. Export to ONNX:

   In order to export your trained YOLOv4 model to ONNX run the following script:

   ```
   python ../pytorch-YOLOv4/demo_darknet2onnx.py cfg/yolov4-leaky.cfg /path/to/
   ↪trained.weights /path/to/some/image.jpg 1
   ```

- The ONNX will be available in `/workspace/darknet/`

Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov4_leaky.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov4_1_3_512_512.onnx --calib-path /path/to/calibration/
↪imgs/ --yaml path/to/yolov4_leaky.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- On your desired YOLOv4 YAML, update `postprocessing.anchors.sizes` property if anchors changed, and `preprocessing.input_shape` if the network is trained on other resolution.
- On yolo.yaml, change `evaluation.classes` if classes amount is changed.

More details about YAML files are presented *here*.

11.3. YOLOv5 Retraining

- To learn more about yolov5 look here

11.3.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.3.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolov5
docker build --build-arg timezone=`cat /etc/timezone` -t yolov5:v0 .
```

the following optional arguments can be passed via –build-arg:

- `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
- `user` - username for a local non-root user. Defaults to 'hailo'.
- `group` - default group for a local non-root user. Defaults to 'hailo'.
- `uid` - user id for a local non-root user.
- `gid` - group id for a local non-root user.

* This command will build the docker image with the necessary requirements using the Dockerfile exists in yolov5 directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/
↪local/data/dir:/path/to/docker/data/dir yolov5:v0
```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `yolov5:v0` the name of the docker image.

11.3.3. Training and exporting to ONNX

1. Train your model:
   Once the docker is started, you can start training your model.
   - Prepare your custom dataset - Follow the steps described here in order to create:
     - `dataset.yaml` configuration file
     - Labels - each image should have labels in YOLO format with corresponding txt file for each image.
     - Make sure to include number of classes field in the yaml, for example: `nc: 80`
   - Start training - The following command is an example for training a *yolov5s* model.

```
python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --weights
↪yolov5s.pt --cfg models/yolov5s.yaml
```

   - `yolov5s.pt` - pretrained weights. You can find the pretrained weights for *yolov5s*, *yolov5m*, *yolov5l*, *yolov5x* and *yolov5m_wo_spp* in your working directory.
   - `models/yolov5s.yaml` - configuration file of the yolov5 variant you would like to train. In order to change the number of classes make sure you update this file.

NOTE: We recommend to use *yolov5m_wo_spp* for best performance on Hailo-8

2. Export to ONNX:

In order to export your trained YOLOv5 model to ONNX run the following script:

```
python models/export.py --weights /path/to/trained/model.pt --img 640 --batch 1
↪ # export at 640x640 with batch size 1
```

11.3.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov5s.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov5s.onnx --calib-path /path/to/calibration/imgs/dir/ --
↪yaml path/to/yolov5s.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Make sure to also update `preprocessing.input_shape` field on yolo.yaml, if it was changed on re-training.

More details about YAML files are presented *here*.

11.3.5. Anchors Extraction

The training flow will automatically try to find more fitting anchors values then the default anchors. In our TAPPAS environment we use the default anchors, but you should be aware that the resulted anchors might be different.
The model anchors can be retrieved from the trained model using the following snippet:

```
m = torch.load("last.pt")["model"]
detect = list(m.children())[0][-1]
print(detect.anchor_grid)
```

11.4. YOLOv8 Retraining

- To learn more about yolov8 look here

11.4.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.4.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolov8
docker build --build-arg timezone=`cat /etc/timezone` -t yolov8:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

   * This command will build the docker image with the necessary requirements using the Dockerfile exists in yolov8 directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/
↪local/data/dir:/path/to/docker/data/dir yolov8:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `yolov8:v0` the name of the docker image.

11.4.3. Training and exporting to ONNX

1. Train your model:
   Once the docker is started, you can start training your model.
   - Prepare your custom dataset - Follow the steps described here in order to create:
     - `dataset.yaml` configuration file
     - Labels - each image should have labels in YOLO format with corresponding txt file for each image.
     - Make sure to include number of classes field in the yaml, for example: `nc: 80`
   - Start training - The following command is an example for training a *yolov8s* model.

   ```
   yolo detect train data=coco128.yaml model=yolov8s.pt name=retrain_yolov8s
   ↪epochs=100 batch=16
   ```

     - `yolov8s.pt` - pretrained weights. The pretrained weights for *yolov8n*, *yolov8s*, *yolov8m*, *yolov8l* and *yolov8x* will be downloaded to your working directory when running this command.
     - `coco128.yaml` - example file for data.yaml file. Can be found at ultralytics/ultralytics/datasets.
     - `retrain_yolov8s` - the new weights will be saved at ultralytics/runs/detect/retrain_yolov8s.
     - `epochs` - number of epochs to run. default to 100.
     - `batch` - number of images per batch. default to 16.

   NOTE:more configurable parameters can be found at https://docs.ultralytics.com/modes/train/

2. Export to ONNX:

   In order to export your trained YOLOv8 model to ONNX run the following script:

   ```
   yolo export model=/path/to/trained/best.pt imgsz=640 format=onnx opset=11 #
   ↪export at 640x640
   ```

   NOTE:more configurable parameters can be found at https://docs.ultralytics.com/modes/export/

11.4.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov8s.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov8s.onnx --calib-path /path/to/calibration/imgs/dir/ --
↪yaml path/to/yolov8s.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Make sure to also update `preprocessing.input_shape` field on yolo.yaml, if it was changed on retraining.

More details about YAML files are presented *here*.

11.5. YOLOX Retraining

- To learn more about yolox look here

11.5.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

  NOTE: In case you use Hailo Software Suite docker, make sure you are doing all the following instructions outside of this docker.

11.5.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolox
docker build --build-arg timezone=`cat /etc/timezone` -t yolox:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /
↪path/to/local/data/dir:/path/to/docker/data/dir yolox:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `yolox:v0` the name of the docker image.

11.5.3. Training and exporting to ONNX

1. Prepare your data:

   You can use coco format, which is already supported for training on your own custom dataset. More information can be found here

2. Training:

   Start training with the following command:

```
python tools/train.py -f exps/default/yolox_s_leaky.py -d 8 -b 64 -c yolox_s.pth

                    exps/default/yolox_m_leaky.py
                    exps/default/yolox_l_leaky.py
                    exps/default/yolox_x_leaky.py
                    exps/default/yolox_s_wide_leaky.py
```

   - -f: experiment description file
   - -d: number of gpu devices
   - -b: total batch size, the recommended number for -b is num-gpu * 8
   - -c: path to pretrained weights which can be found in your working directory

```
|_ yolox_s.pth
|_ yolox_m.pth
|_ yolox_l.pth
|_ yolox_x.pth
```

3. Exporting to onnx:

   After finishing training run the following command:

```
python tools/export_onnx.py --output-name yolox_s_leaky.onnx -f ./exps/default/
↪yolox_s_leaky.py -c YOLOX_outputs/yolox_s_leaky/best_ckpt.pth
```

NOTE: Your trained model will be found under the following path: `/workspace/YOLOX/YOLOX_outputs/yolox_s_leaky/`, and the exported onnx will be written to `/workspace/YOLOX/yolox_s_leaky.onnx`

11.5.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolox_s_leaky.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolox_s_leaky.onnx --calib-path /path/to/calibration/imgs/
↪dir/ --yaml path/to/yolox_s_leaky.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note: More details about YAML files are presented *here*.

11.6. DAMO-YOLO Retraining

- To learn more about DAMO-YOLO visit the official repository

---

11.6.1. Prerequisites

- docker (docker installation instructions)
- nvidia-docker2 (nvidia docker installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.6.2. Environment Preparations

1. Build the docker image:

   ```
   cd hailo_model_zoo/training/damoyolo
   docker build --build-arg timezone=`cat /etc/timezone` -t damoyolo:v0 .
   ```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. “Asia/Jerusalem”
   - `user` - username for a local non-root user. Defaults to ‘hailo’.
   - `group` - default group for a local non-root user. Defaults to ‘hailo’.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

   * This command will build the docker image with the necessary requirements using the Dockerfile exists in damoyolo directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/
↪local/data/dir:/path/to/docker/data/dir damoyolo:v0
```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `damoyolo:v0` the name of the docker image.

11.6.3. Training and exporting to ONNX

1. Train your model:
   Once the docker is started, you can start training your model.
   - Prepare your custom dataset (must be coco format) - Follow the steps described here.
   - Modify `num_classes` and `class_names` in the configuration file, for example `damoyolo_tinynasL20_T.py`
   - Use `self.train.batch_size` / `self.train.total_epochs` in the configuration file to modify the batch_size and number of epochs
   - Update the symbolic link to your dataset: ln -sfn /your/coco/like/dataset/path datasets/coco
   - Start training - The following command is an example for training a *damoyolo_tinynasL20_T* model.

   ```
   python tools/train.py -f configs/damoyolo_tinynasL20_T.py

               configs/damoyolo_tinynasL25_S.py
               configs/damoyolo_tinynasL35_M.py
   ```

     - `configs/damoyolo_tinynasL20_T.py` - configuration file of the DAMO-YOLO variant you would like to train. In order to change the number of classes make sure you update `num_classes` and `class_names` in this file.

2. Export to ONNX:

   In order to export your trained DAMO-YOLO model to ONNX run the following script:

```
python tools/converter.py -f configs/damoyolo_tinynasL20_T.py -c /path/to/
↪trained/model.pth --batch_size 1 --img_size 640 # export at 640x640 with batch
↪size 1
```

11.6.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/damoyolo_tinynasL20_T.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt damoyolo_tinynasL20_T.onnx --calib-path /path/to/
↪calibration/imgs/dir/ --yaml path/to/damoyolo/variant.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

  More details about YAML files are presented *here*.

11.7. Nanodet Retraining

- To learn more about NanoDet look here

11.7.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.7.2. Environment Preparations

1. Build the docker image:

   ```
   cd hailo_model_zoo/training/nanodet
   docker build -t nanodet:v0 --build-arg timezone=`cat /etc/timezone` .
   ```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. “Asia/Jerusalem”
   - `user` - username for a local non-root user. Defaults to ‘hailo’.
   - `group` - default group for a local non-root user. Defaults to ‘hailo’.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /
↪path/to/local/data/dir:/path/to/docker/data/dir nanodet:v0
```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-u <username>` same username as used for building the image.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `nanodet:v0` the name of the docker image.

11.7.3. Training and exporting to ONNX

1. Prepare your data:

   Data is expected to be in coco format. More information can be found here

2. Training:

   Configure your model in a .yaml file. We'll use /workspace/nanodet/config/legacy_v0.x_configs/RepVGG/nanodet-RepVGG-A0_416.yml in this guide.
   Modify the path for the dataset in the .yaml configuration file:

```
data:
 train:
  name: CocoDataset
  img_path: <path-to-train-dir>
  ann_path: <path-to-annotations-file>
  ...
 val:
  name: CocoDataset
  img_path: <path-to-validation-dir>
  ann_path: <path-to-annotations-file>
  ...
```

   Start training with the following commands:

```
cd /workspace/nanodet
ln -s /workspace/data/coco/ /coco
python tools/train.py ./config/legacy_v0.x_configs/RepVGG/nanodet-RepVGG-A0_
↪416.yml
```

In case you want to use the pretrained nanodet-RepVGG-A0_416.ckpt, which was predownloaded into your docker modify your configurationf file:

```
schedule:
 load_model: ./pretrained/nanodet-RepVGG-A0_416.ckpt
```

Modifying the batch size and the number of GPUs used for training can be done also in the configuration file:

```
device:
 gpu_ids: [0]
 workers_per_gpu: 1
 batchsize_per_gpu: 128
```

3. Exporting to onnx

After training, install the ONNX and ONNXruntime packages, then export the ONNX model:

```
python tools/export_onnx.py --cfg_path ./config/legacy_v0.x_configs/RepVGG/
↪nanodet-RepVGG-A0_416.yml --model_path /workspace/nanodet/workspace/RepVGG-
↪A0-416/model_last.ckpt
```

NOTE: Your trained model will be found under the following path: /workspace/nanodet/workspace/<backbone-name> /model_last.ckpt, and exported onnx will be written to /workspace/nanodet/nanodet.onnx

11.7.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/nanodet_repvgg.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt nanodet.onnx --calib-path /path/to/calibration/imgs/dir/ --
↪yaml path/to/nanodet_repvgg.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- On your desired YAML file, change `preprocessing.input_shape` if changed on retraining.

More details about YAML files are presented *here*.

11.8. SSD-Mobilenet-V1 Retraining

Deprecation Warning: This retraining image will be deprecated in next release.

- To learn more about ssd look here

11.8.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you use Hailo Software Suite docker, make sure you are doing all the following instructions outside of this docker.

11.8.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/ssd
docker build -t tf1od:v0 --build-arg timezone=`cat /etc/timezone` .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "docker-name" -it --gpus all -u "username" --ipc=host -v /path/
↪to/local/data/dir:/path/to/docker/data/dir tf1od:v0
```

   - `docker run` create a new docker container.
   - `--name <docker-name>` name for your container.
   - `-u <username>` same username as used for building the image.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `tf1od:v0` the name of the docker image.

11.8.3. Training and exporting

1. Prepare your data:

   Data is required to be in a specific TFRecord format, accompanied by a label map file. Infomation about the dataset structure can be found here. In addition you can find useful datasets conversion utils in the following path `object_detection/dataset_tools/`

2. Training:

   Configure your model by modifying the config file `/home/tensorflow/models/research/pipeline.config` found in your working directory. You should first update your dataset information, update the input_path and label_map_path to correspond the data prepared in the previous stage.

```
train_input_reader {
 label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
 tf_record_input_reader {
  input_path: "PATH_TO_BE_CONFIGURED/<you-own-dataset>_train.record-?????-of-
↪00100"
 }
}
eval_input_reader {
 label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
 shuffle: false
 num_readers: 1
 tf_record_input_reader {
  input_path: "PATH_TO_BE_CONFIGURED/<your-own-dataset>_val.record-?????-of-
↪00010"
 }
```

   The pretrained weights were already downloaded for you and configured into your config file.

```
fine_tune_checkpoint: "/home/tensorflow/models/research/ssd_mobilenet_v1_
↪coco_2018_01_28/model.ckpt"
from_detection_checkpoint: true
```

   Start training with the following command:

```
python object_detection/model_main.py --pipeline_config_path=/home/
↪tensorflow/models/research/pipeline.config --model_dir=ssd_mobilenet_v1_
↪training --num_train_steps=200000 --sample_1_of_n_eval_examples=3 --
↪alsologtostderr
```

   - `--pipeline_config_path` - path to your training configuration file.
   - `--model_dir` - output training directory.
   - `--num_train_steps` - exists also in the configuration file but can be overwritten as cli argument.
   - `--sample_1_of_n_eval_examples` - sample of one every n eval input examples, where n is provided. Modifying training hyper parameters (batch size, learning rate, optimizer etc...) can be done in the `train_config` section:

```
train_config {
batch_size: 24
data_augmentation_options {
 random_horizontal_flip {
```

(continues on next page)

(continued from previous page)

```
 }
}
data_augmentation_options {
 ssd_random_crop {
 }
}
optimizer {
 rms_prop_optimizer {
  learning_rate {
   exponential_decay_learning_rate {
    initial_learning_rate: 0.00400000018999
    decay_steps: 800720
    decay_factor: 0.949999988079
   }
  }
  momentum_optimizer_value: 0.899999976158
  decay: 0.899999976158
  epsilon: 1.0
 }
}
```

3. Exporting the model After training, run the following command:

```
python object_detection/export_inference_graph.py --input_type image_tensor --
↪input_shape -1,300,300,3 --pipeline_config_path pipeline.config --trained_
↪checkpoint_prefix ./ssd_mobilenet_v1_training/model.ckpt-"iteration-number
↪" --output_directory ./ssd_mobilenet_v1
```

Exported `model.ckpt` files will be found in the given output directory.

11.8.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained checkpoint. In order to do so you need a working model-zoo environment. Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/ssd_mobilenet_v1.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt ssd_mobilenet_v1.ckpt --calib-path /path/to/calibration/
↪imgs/dir/ --yaml path/to/ssd_mobilenet_v1.yaml
```

- `--ckpt` - path to your ckpt file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- This model uses on-chip NMS capabilities. Therefore, the on-chip NMS parameters should be updated if they have changed:
  - Set ModelZoo data folder location: `export HMZ_DATA=/local/workspace/data`
  - Use `hailomz parse ssd_mobilenet_v1` so the model is downloaded

- `cd $HMZ_DATA/models_files/ObjectDetection/Detection-COCO/ssd/ssd_mobilenet_v1/pretrained/`
- Update the `mobilenet_ssd_nms_postprocess_config.json` file to match the updated NMS post-process configuration.
- Run the command from above.

More details about YAML files are presented *here*.

11.9. FCN Retraining

- To learn more about FCN look here

11.9.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.9.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/fcn
docker build -t fcn:v0 --build-arg timezone=`cat /etc/timezone` .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /path/to/local/data/dir:/path/to/docker/data/dir fcn:v0
```

   - `docker run` create a new docker container.
   - `--name <docker-name>` name for your container.
   - `-u <username>` same username as used for building the image.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.

- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `fcn:v0` the name of the docker image.

11.9.3. Training and exporting to ONNX

1. Prepare your data:

   Data is expected to be in coco format, and by default should be in /workspace/data/<dataset_name>.
   The expected structure is as follows:

```
/workspace
|-- mmsegmentation
`-- |-- data
    `-- cityscapes
      |-- gtFine
      |  | -- train
      |  |   | -- aachem
      |  |   | -- | -- *.png
      |  |   ` -- ...
      |  ` -- test
      |     | -- berlin
      |     | -- | -- *.png
      |     ` -- ...
      `-- leftImg8bit
        | -- train
        | -- | -- aachem
        | -- | -- | -- *.png
        | -- ` -- ...
        ` -- test
          | -- berlin
          | -- | -- *.png
          ` -- ...
```

   more information can be found here

2. Training:

   Configure your model in a .py file. We'll use /workspace/mmsegmentation/configs/fcn/fcn_r18_hailo.py in this guide.
   start training with the following command:

```
cd /workspace/mmsegmentation
./tools/dist_train.sh configs/fcn/fcn8_r18_hailo.py 2
```

   Where 2 is the number of GPUs used for training.

3. Exporting to onnx

After training, run the following command:

```
cd /workspace/mmsegmentation
python ./tools/pytorch2onnx.py configs/fcn/fcn_r18_hailo.py --opset-version
↪11 --checkpoint ./work_dirs/fcn8_r18_hailo/latest.pth --shape 1024 1920 --
↪output-file fcn.onnx
```

11.9.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e.
`hailo_model_zoo/cfg/networks/fcn16_resnet_v1_18.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt fcn.onnx --calib-path /path/to/calibration/imgs/dir/ --yaml
↪path/to/fcn16_resnet_v1_18.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note: More details about YAML files are presented *here*.

11.10. YOLACT Retraining

- To learn more about Yolact look here

11.10.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you use Hailo Software Suite docker, make sure you are doing all the following instructions outside of this docker.

11.10.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolact
docker build --build-arg timezone=`cat /etc/timezone` -t yolact:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.
   - This command will build the docker image with the necessary requirements using the Dockerfile exists in the yolact directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/local/
↪data/dir:/path/to/docker/data/dir yolact:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `yolact:v0` the name of the docker image.

11.10.3. Training and exporting to ONNX

1. Prepare your data:
   Data is expected to be in coco format, and by default should be in /workspace/data/<dataset_name>.
   The expected structure is as follows:

```
/workspace
|-- data
`-- |-- coco
  `-- |-- annotations
    |  |-- instances_train2017.json
    |  |-- instances_val2017.json
    |  |-- person_keypoints_train2017.json
    |  |-- person_keypoints_val2017.json
    |  |-- image_info_test-dev2017.json
    |-- images
      |-- train2017
        |-- *.jpg
      |-- val2017
```

(continues on next page)

(continued from previous page)

```
    |-- *.jpg
  |-- test2017
  `  |-- *.jpg
```

For training on custom datasets see here

2. Train your model:

   Once your dataset is prepared, create a soft link to it under the yolact/data work directory, then you can start training your model:

```
cd /workspace/yolact
ln -s /workspace/data/coco data/coco
python train.py --config=yolact_regnetx_800MF_config
```

   - `yolact_regnetx_800MF_config` - configuration using the regnetx_800MF backbone.

3. Export to ONNX: In orded to export your trained YOLACT model to ONNX run the following script:

```
python export.py --config=yolact_regnetx_800MF_config --trained_model=path/to/
↪trained/model --export_path=path/to/export/model.onnx
```

   - `--config` - same configuration used for training.
   - `--trained_model` - path to the weights produced by the training process.
   - `--export_path` - path to export the ONNX file to. Include the `.onnx` extension.

11.10.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolact.yaml`, and run compilation using the model zoo:

```
hailomz compile yolact --ckpt yolact.onnx --calib-path /path/to/calibration/imgs/
↪dir/ --yaml path/to/yolact_regnetx_800mf_20classes.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- The **'yolact_regnetx_800mf_20classes.yaml<https://github.com/hailo-ai/hailo_model_zoo/blob/master/hailo_model_zoo/cfg/** is an example yaml where some of the classes (out of 80) were removed. If you wish to change the number of classes, the easiest way is to retrain with the exact number of classes, erase the `channels_remove` section (lines 18 to 437).

More details about YAML files are presented *here*.

11.11. YOLOv8-seg Retraining

- To learn more about yolov8 look here

11.11.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.11.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/yolov8_seg
docker build --build-arg timezone=`cat /etc/timezone` -t yolov8_seg:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

   * This command will build the docker image with the necessary requirements using the Dockerfile exists in yolov8-seg directory.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all --ipc=host -v /path/to/
↪local/data/dir:/path/to/docker/data/dir yolov8_seg:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `yolov8:v0` the name of the docker image.

11.11.3. Training and exporting to ONNX

1. Train your model:
   Once the docker is started, you can start training your model.

   - Prepare your custom dataset - Follow the steps described here in order to create:
     - `dataset.yaml` configuration file
     - Labels - each image should have labels in YOLO format with corresponding txt file for each image.
     - Make sure to include number of classes field in the yaml, for example: `nc: 80`
   - Start training - The following command is an example for training a *yolov8s-seg* model.

   ```
   yolo segment train data=coco128-seg.yaml model=yolov8s-seg.pt name=retrain_
   ↪yolov8s_seg epochs=100 batch=16
   ```

     - `yolov8s-seg.pt` - pretrained weights. The pretrained weights for *yolov8n-seg*, *yolov8s-seg*, *yolov8m-seg*, *yolov8l-seg* and *yolov8x-seg* will be downloaded to your working directory when running this command.
     - `coco128-seg.yaml` - example file for data.yaml file. Can be found at ultralytics/ultralytics/datasets.
     - `retrain_yolov8s_seg` - the new weights will be saved at ultralytics/runs/segment/retrain_yolov8s_seg.
     - `epochs` - number of epochs to run. default to 100.
     - `batch` - number of images per batch. default to 16.

   NOTE:more configurable parameters can be found at https://docs.ultralytics.com/modes/train/

2. Export to ONNX:

   In order to export your trained YOLOv8-seg model to ONNX run the following script:

   ```
   yolo export model=/path/to/trained/best.pt imgsz=640 format=onnx opset=11 #
   ↪export at 640x640
   ```

   NOTE:more configurable parameters can be found at https://docs.ultralytics.com/modes/export/

11.11.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/yolov8s-seg.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt yolov8s-seg.onnx --calib-path /path/to/calibration/imgs/dir/
↪ --yaml path/to/yolov8s-seg.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.

- The model zoo will take care of adding the input normalization to be part of the model.

Note:

- Make sure to also update `preprocessing.input_shape` field on yolo.yaml, if it was changed on re-training.

More details about YAML files are presented *here*.

11.12. Centerpose Retraining

- To learn more about CenterPose look here

11.12.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.12.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/centerpose
docker build -t centerpose:v0 --build-arg timezone=`cat /etc/timezone` .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /
↪path/to/local/data/dir:/path/to/docker/data/dir centerpose:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.

- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `centerpose:v0` the name of the docker image.

11.12.3. Training and exporting to ONNX

1. Prepare your data:

   Data is expected to be in coco format, and by default should be in /workspace/data/<dataset_name>.
   The expected structure is as follows:

```
/workspace
|-- data
`-- |-- coco
  `-- |-- annotations
    |  |-- instances_train2017.json
    |  |-- instances_val2017.json
    |  |-- person_keypoints_train2017.json
    |  |-- person_keypoints_val2017.json
    |  |-- image_info_test-dev2017.json
  `-- |-- images
    |---|-- train2017
    |---|---|-- *.jpg
    |---|-- val2017
    |---|---|-- *.jpg
    |---|-- test2017
    `---|---|-- *.jpg
```

   The path for the dataset can be configured in the .yaml file, e.g. centerpose/experiments/regnet_fpn.yaml

2. Training:

   Configure your model in a .yaml file. We'll use /workspace/centerpose/experiments/regnet_fpn.yaml in this guide.
   start training with the following command:

```
cd /workspace/centerpose/tools
python -m torch.distributed.launch --nproc_per_node 4 train.py --cfg ../
↪experiments/regnet_fpn.yaml
```

   Where 4 is the number of GPUs used for training.
   If using a different number, update both this and the used gpus in the .yaml configuration.

3. Exporting to onnx After training, run the following command:

```
cd /workspace/centerpose/tools
python export.py --cfg ../experiments/regnet_fpn.yaml --TESTMODEL /workspace/
↪out/regnet1_6/model_best.pth
```

(continues on next page)

(continued from previous page)

11.12.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/centerpose_regnetx_1.6gf_fpn.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt coco_pose_regnet1.6_fpn.onnx --calib-path /path/to/
↪calibration/imgs/dir/ --yaml path/to/centerpose_regnetx_1.6gf_fpn.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note: More details about YAML files are presented *here*.

11.13. MSPN Retraining

11.13.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

NOTE: In case you are using the Hailo Software Suite docker, make sure to run all of the following instructions outside of that docker.

11.13.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/mspn
docker build -t mspn:v0 --build-arg timezone=`cat /etc/timezone` .
```

the following optional arguments can be passed via –build-arg:

- `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
- `user` - username for a local non-root user. Defaults to 'hailo'.
- `group` - default group for a local non-root user. Defaults to 'hailo'.
- `uid` - user id for a local non-root user.
- `gid` - group id for a local non-root user.

2. Start your docker:


```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /
→path/to/local/data/dir:/path/to/docker/data/dir mspn:v0
```

- `docker run` create a new docker container.
- `--name <your_docker_name>` name for your container.
- `-it` runs the command interactively.
- `--gpus all` allows access to all GPUs.
- `--ipc=host` sets the IPC mode for the container.
- `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
- `centerpose:v0` the name of the docker image.

11.13.3. Training and exporting to ONNX

1. Prepare your data:

   Data is expected to be in coco format, and by default should be in /workspace/data/<dataset_name>.

   The expected structure is as follows:

```
/workspace
|-- data
`-- |-- coco
  `-- |-- annotations
    |  |-- instances_train2017.json
    |  |-- instances_val2017.json
    |  |-- person_keypoints_train2017.json
    |  |-- person_keypoints_val2017.json
    |  |-- image_info_test-dev2017.json

  `---|-- train2017
  |---|---|-- *.jpg
  `---|-- val2017
  |---|---|-- *.jpg
  `---|-- test2017
  `---|---|-- *.jpg
```

   The path for the dataset can be configured in the .py config file, e.g. `configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/regnetx_800mf_256x192.py`

2. Training:

   Configure your model in a .py config file. We will use `/workspace/mmpose/configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/coco/regnetx_800mf_256x192.py` in this guide. Start training with the following command:

```
cd /workspace/mmpose
./tools/dist_train.sh ./configs/body/2d_kpt_sview_rgb_img/topdown_heatmap/
→coco/regnetx_800mf_256x192.py 4 --work-dir exp0
```

   Where 4 is the number of GPUs used for training. In this example, the trained model will be saved under `exp0` directory.

3. Export to onnx

   In orded to export your trained model to ONNX run the following script:

```
cd /workspace/mmpose
python tools/deployment/pytorch2onnx.py ./configs/body/2d_kpt_sview_rgb_img/
↪topdown_heatmap/coco/regnetx_800mf_256x192.py exp0/best_AP_epoch_310.pth --
↪output-file mspn_regnetx_800mf.onnx
```

   where `exp0/best_AP_epoch_310.pth` should be replaced by the trained model file path.

11.13.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model.
In order to do so you need a working model-zoo environment.
Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/mspn_regnetx_800mf.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt mspn_regnetx_800mf.onnx --calib-path /path/to/calibration/
↪imgs/dir/ --yaml path/to/mspn_regnetx_800mf.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note: More details about YAML files are presented *here*.

11.14. Arface Retraining

- To learn more about arcface look here

11.14.1. Prerequisites

- docker (installation instructions)
- nvidia-docker2 (installation instructions)

  NOTE: In case you use Hailo Software Suite docker, make sure you are doing all the following instructions outside of this docker.

11.14.2. Environment Preparations

1. Build the docker image:

```
cd hailo_model_zoo/training/arcface
docker build --build-arg timezone=`cat /etc/timezone` -t arcface:v0 .
```

   the following optional arguments can be passed via –build-arg:

   - `timezone` - a string for setting up timezone. E.g. "Asia/Jerusalem"
   - `user` - username for a local non-root user. Defaults to 'hailo'.
   - `group` - default group for a local non-root user. Defaults to 'hailo'.
   - `uid` - user id for a local non-root user.
   - `gid` - group id for a local non-root user.

2. Start your docker:

```
docker run --name "your_docker_name" -it --gpus all -u "username" --ipc=host -v /
→path/to/local/data/dir:/path/to/docker/data/dir arcface:v0
```

   - `docker run` create a new docker container.
   - `--name <your_docker_name>` name for your container.
   - `-it` runs the command interactively.
   - `--gpus all` allows access to all GPUs.
   - `--ipc=host` sets the IPC mode for the container.
   - `-v /path/to/local/data/dir:/path/to/docker/data/dir` maps `/path/to/local/data/dir` from the host to the container. You can use this command multiple times to mount multiple directories.
   - `arcface:v0` the name of the docker image.

11.14.3. Training and exporting to ONNX

1. Prepare your data:

   For more information on obtraining datasets here
   The repository supports the following formats:

   1. ImageFolder dataset - each class (person) has its own directory
      Validation data is packed .bin files

```
data_dir/
    agedb_30.bin
    cfp_fp.bin
    lfw.bin
    person0/
    person1/
    ...
    personlast/
```

   2. MxNetRecord - train.rec and train.idx files. This is the format of insightface datasets.

Validation data is packed .bin files

```
data_dir/
    agedb_30.bin
    cfp_fp.bin
    lfw.bin
    train.idx
    train.rec
```

2. Training:

   Start training with the following command:

```
python -m torch.distributed.launch --nproc_per_node=2 --nnodes=1 --node_rank=0
↪--master_addr="127.0.0.1" --master_port=12581 train_v2.py /path/to/config
```

   - nproc_per_node: number of gpu devices

3. Exporting to onnx:

   After finishing training run the following command:

```
python torch2onnx.py /path/to/model.pt --network mbf --output /path/to/model.
↪onnx --simplify true
```

11.14.4. Compile the Model using Hailo Model Zoo

You can generate an HEF file for inference on Hailo-8 from your trained ONNX model. In order to do so you need a working model-zoo environment. Choose the corresponding YAML from our networks configuration directory, i.e. `hailo_model_zoo/cfg/networks/arcface_mobilefacenet.yaml`, and run compilation using the model zoo:

```
hailomz compile --ckpt arcface_s_leaky.onnx --calib-path /path/to/calibration/imgs/
↪dir/ --yaml /path/to/arcface_mobilefacenet.yaml
```

- `--ckpt` - path to your ONNX file.
- `--calib-path` - path to a directory with your calibration images in JPEG/png format
- `--yaml` - path to your configuration YAML file.
- The model zoo will take care of adding the input normalization to be part of the model.

Note: More details about YAML files are presented *here*.