



# Hailo TAPPAS User Guide

Release 3.11.0  
September 2021



## Table of Contents

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>Disclaimer and Proprietary Information Notice</b> | <b>3</b>  |
| 1.1       | Copyright  | 3         |
| 1.2       | General Notice                                       | 3         |
| <b>2</b>  | <b>Documentation Control</b>                         | <b>4</b>  |
| <b>3</b>  | <b>Getting Started</b>                               | <b>5</b>  |
| 3.1       | System Requirements                                  | 5         |
| 3.2       | Prerequisites  | 5         |
| 3.2.1     | Required Packages                                    | 5         |
| 3.3       | Installation Flow                                    | 7         |
| 3.4       | Running Apps   | 9         |
| 3.4.1     | CLI options  | 9         |
| <b>4</b>  | <b>Object Detection</b>                              | <b>11</b> |
| 4.1       | Detection App  | 11        |
| 4.1.1     | CenterNet-ResNet50                                   | 12        |
| 4.1.2     | CenterNet-ResNet18                                   | 13        |
| 4.1.3     | MobileNet v1 SSD                                     | 14        |
| 4.1.4     | YOLOv5   | 15        |
| 4.1.5     | YOLOv4   | 17        |
| 4.1.6     | YOLOv3   | 18        |
| 4.2       | Multi Stream   | 19        |
| 4.3       | Tiling   | 20        |
| 4.3.1     | MobileNetV1 SSD NMS - COCO                           | 21        |
| 4.3.2     | MobileNetV1 SSD NMS - Visdrone                       | 21        |
| 4.3.3     | Lightface slim                                       | 21        |
| 4.3.4     | Running the Tiling App                               | 21        |
| <b>5</b>  | <b>Semantic Segmentation Apps</b>                    | <b>24</b> |
| 5.1       | ResNet18-FCN8  | 24        |
| <b>6</b>  | <b>Pose Estimation</b>                               | <b>25</b> |
| 6.1       | CenterPose   | 25        |
| <b>7</b>  | <b>Face Recognition</b>                              | <b>26</b> |
| 7.1       | FaceNet  | 26        |
| 7.1.1     | Running the Demo                                     | 27        |
| <b>8</b>  | <b>Face Detection</b>                                | <b>28</b> |
| 8.1       | RetinaFace MobileNet HD                              | 28        |
| 8.2       | Lightface Slim Face Detection                        | 29        |
| <b>9</b>  | <b>Instance Segmentation</b>                         | <b>30</b> |
| 9.1       | YOLACT   | 30        |
| <b>10</b> | <b>Lane Detection</b>                                | <b>31</b> |
| 10.1      | PolyLaneNet  | 31        |
| <b>11</b> | <b>Depth Estimation</b>                              | <b>32</b> |
| 11.1      | Mono Depth2  | 32        |
| <b>12</b> | <b>Multiple Object Tracking</b>                      | <b>33</b> |
| 12.1      | FAIR MOT RegNet800                                   | 33        |

|  |           |
|--|-----------|
| <b>13 Classification</b>   | <b>34</b> |
| 13.1 ResNet50 . . . . .  | 34        |
| <b>14 3D Object Detection</b>                                    | <b>36</b> |
| 14.1 SMOKE_Regnet800 . . . . .                                   | 36        |
| <b>15 Multitask apps</b>   | <b>37</b> |
| 15.1 MonoDepthSSD . . . . .                                      | 37        |
| <b>16 Gstreamer Apps</b>   | <b>39</b> |
| 16.1 Gst Detection . . . . .                                     | 39        |
| 16.2 Gst Segmentation . . . . .                                  | 39        |
| 16.3 Gst Multi Stream . . . . .                                  | 40        |
| 16.4 Gst Pose Estimation . . . . .                               | 41        |
| <b>17 MIPI Apps</b>  | <b>42</b> |
| 17.1 Yolact_mipi App yolact_regnetx_600mf_d2s_9classes . . . . . | 42        |
| <b>18 Add Your Own App/Network</b>                               | <b>44</b> |
| 18.1 Known Q&A . . . . .   | 44        |
| <b>19 Tools</b>  | <b>45</b> |
| 19.1 Inference Acceleration and frame-rate control . . . . .     | 45        |
| 19.2 Inter-Process Communication: Pipes . . . . .                | 45        |
| 19.3 Extras . . . . .  | 46        |
| 19.3.1 Plailo . . . . .  | 46        |
| 19.3.2 ZeroMQ . . . . .  | 46        |
| <b>20 Debug and Troubleshooting</b>                              | <b>47</b> |
| 20.1 Breakpoints . . . . .                                       | 47        |
| 20.2 Profiling . . . . .   | 49        |
| 20.2.1 Inline Profiling . . . . .                                | 49        |
| 20.2.2 Profiler Reports . . . . .                                | 50        |
| 20.3 Queues and Blocks Status . . . . .                          | 52        |
| 20.4 Pipeline Timestamps . . . . .                               | 52        |
| 20.5 Common Pitfalls . . . . .                                   | 53        |
| <b>21 Version Changelog</b>                                      | <b>54</b> |
| 21.1 Version 3.11.0 . . . . .                                    | 54        |
| 21.2 Version 3.9.0 . . . . .                                     | 54        |
| 21.3 Version 3.8.0 . . . . .                                     | 54        |
| 21.4 Version 3.7.0 . . . . .                                     | 55        |
| 21.5 Version 3.6.0 . . . . .                                     | 56        |
| 21.6 Version 3.5.0 . . . . .                                     | 56        |
| 21.7 Version 3.4.0 . . . . .                                     | 56        |
| 21.8 Version 3.3.0 . . . . .                                     | 57        |
| 21.9 Version 3.2.0 . . . . .                                     | 57        |
| 21.10 Version 3.1.0 . . . . .                                    | 58        |
| <b>22 Movies Licensing</b>                                       | <b>60</b> |

# **1 Disclaimer and Proprietary Information Notice**

## **1.1 Copyright**

© 2021 Hailo Technologies Ltd ("Hailo"). All Rights Reserved. No part of this document may be reproduced or transmitted in any form without the expressed, written permission of Hailo. Nothing contained in this document should be construed as granting any license or right to use proprietary information for that matter, without the written permission of Hailo. This version of the document supersedes all previous versions.

## **1.2 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.



## 2 Documentation Control

| Version | Date           | Description  |
|---------|----------------|--|
| V3.11.0 | September 2021 | Added GStreamer apps, Post-Estimation, NVR based on-top of GStreamer. MIPI app   |
| V3.9.0  | June 2021      | Gstreamer full pipeline, multi stream YOLOv4/3   |
| V3.8.0  | May 2021       | Added dynamic acceleration, New yolov4, New smoke 3d object detection  |
| V3.7.0  | April 2021     | Added Lightface network. Removed JLFs.   |
| V3.6.0  | March 2021     | Added yolov3 network   |
| V3.5.0  | February 2021  | Added new example project and user guide. New Classification app. New app MonoDepthSSD with two tasks on single chip. New faster Resnet18-fcn16 network. Added on-chip pre-processing to Yolo networks. Merged NMS and Normalization on-chip for Fair-MOT. Added new pipes paradigm for improved pipeline performance. |
| V3.4.0  | January 2021   | Moved to HEF format. New Profiling tool. Added yolov5m network. Gstreamer based App. Refactored App closing sequence. Openpose network was removed.  |
| V3.3.0  | December 2020  | New Apps: yolov5s, MonoDepth2, Mobiledet, Yolact replaced FE to Regnet<br>Enabled Power and Temperature readings from chip.  |
| V3.2.0  | November 2020  | New Apps: Centernet Resnet18, Multiscale tiling.<br>Added on-chip post-processing to CenterPose  |
| V3.1.0  | October 2020   | First release for Hailo8 production chip   |

## 3 Getting Started

This document is meant to help installing and using Hailo's APPs package. It includes a user guide and debug tips.

### 3.1 System Requirements

- This release is built to work with Hailo8 production chip.
- This release should be installed on an **Ubuntu 18.04** machine with **python 3.6**.
- This release supports only the PCIe interface. UDP interface is not supported.

### 3.2 Prerequisites

The apps zip package **includes Hailo's HailoRT release which is used for this HailoApps version**. No further action is required if you want to use this version of HailoRT. If you have a different version of HailoRT that you want to use however, then when running the install script (see [3.3 Installation Flow](#) below), point the `--hailort-path` to the path where your version is. It is assumed that you have to pre-requisites required to install that HailoRT release. For HailoRT prerequisites see the relevant installation guide.

Check that your board is recognized by the host by running

```
lspci | grep Co-processor
```

You should get in response:

```
bb:dd:ff Co-processor: Hailo Technologies Ltd. Hailo-8 AI Processor (rev 01)
```

#### 3.2.1 Required Packages

- Cairo:

```
sudo apt-get install -y libcairo2-dev
```

- GObject-introspection:

```
sudo apt-get install -y libgirepository1.0-dev
```

- Gstreamer:

```
sudo apt-get install libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad
gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl
gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio python3-gst-1.0
```

- Tkinter:

```
sudo apt-get install -y python3-tk
```

- Python3 dev:

```
sudo apt-get install -y python3.6-dev
```

- Pip:

```
sudo apt-get install -y python3-pip
```

- Virtualenv:

```
sudo apt-get install python-virtualenv
```

- pybind11:

```
sudo apt-get install python-pybind11
```

- Psutil:

```
pip3 install psutil
```

### 3.3 Installation Flow

Download and unzip the HailoApps release package using the password provided to you. This should inflate the apps\_install.sh script and hailo\_apps.tar.gz, along with three files for the HailoRT package. Enter the hailo\_apps directory to see these files, this is your HailoApps root directory:

```
itaio@hai-186-lap:~/repos/releases/3.1$ unzip hailo_apps_v3.1.0.zip
Archive:  hailo_apps_v3.1.0.zip
  creating:  hailo_apps/
[hailo_apps_v3.1.0.zip] hailo_apps/platform.tar.gz password:
  inflating: hailo_apps/platform.tar.gz
  extracting: hailo_apps/md5s.txt
  inflating: hailo_apps/install.sh
  inflating: hailo_apps/apps_install.sh
  inflating: hailo_apps/hailo_apps.tar.gz
itaio@hai-186-lap:~/repos/releases/3.1$ ls
hailo_apps  hailo_apps_v3.1.0.zip
itaio@hai-186-lap:~/repos/releases/3.1$ cd hailo_apps/
itaio@hai-186-lap:~/repos/releases/3.1/hailo_apps$ ls
apps_install.sh  hailo_apps.tar.gz  install.sh  md5s.txt  platform.tar.gz
```

Change directory to the extracted directory, and run the installation script:

```
cd hailo_apps
./apps_install.sh
```

You should see the install script unpacking and installing HailoApps. You might be prompted if older configuration files exist, select Rewrite.

```
config
Config file /home/giladn/.hailo/config already exists. Rewrite? [Y/N]: y
```

In addition, you might be prompted to enter your password when elevated permissions are required. If the script finished successfully, then you are good to go! Source the virtual environment to see that the installation was successful

```
~/repos/releases/3.1/hailo_apps ~/repos/releases/3.1/hailo_apps
~/repos/releases/3.1/hailo_apps
<HailoApps Install> Done.
itaio@hai-186-lap:~/repos/releases/3.1/hailo_apps$ . hailo_apps_venv/bin/activate
(hailo_apps_venv) itaio@hai-186-lap:~/repos/releases/3.1/hailo_apps$ deactivate
```

To check that the firmware was loaded to the board successfully, use the command:

```
dmesg | grep hailo
```

You should see at the end of the log either "Firmware loaded successfully" or "Firmware was already loaded"

```
[274795.536133] hailo: Probing: Allocate memory for irq shadow buffer, 4096
[274795.536135] hailo: Firmware was already loaded for 1e60:2864
[274795.536210] hailo: Probing: Added board 1e60-2864, /dev/hailo0
```

Finally check that you can interface with a Hailo device. In the virtual environment, broadcast to see if a device is connected to your host:

```
source hailo_apps_venv/bin/activate
hailo broadcast
```

If everything is working properly, then you will get a response back from your Hailo device

```
(hailo_apps_venv) hailo@hlo-lab123:/local/us
Using pcie device
Board location: 03:00.0
[INFO/MainProcess] process shutting down
(hailo_apps_venv) hailo@hlo-lab123:/local/us
```

This tells us that we have a Hailo pcie device connected to slot 03:00.0 on the host computer. Congratulations! You are ready to run your first application!

### 3.4 Running Apps

To run the apps, you need to be inside the apps virtual environment.

```
source hailo_apps_venv/bin/activate
```

To run a demo run:

```
hailo run-app <app name>
```

For help and additional controls add '-h or --help' to command line.  
Most apps capture keystrokes while you are in the display window context.

```
(demo_virtualenv) hailo@hilo-demo153:~/repos/demos$ hailo run-app -h
usage: hailo run-app [-h]
                    {resnet50,mobilenet_ssd_multi_stream,facenet,yolact,basic_app,tiling,detection,retinaface_mobilenet_hd,
retinaface_facial_landmark,resnet18_fcn16,polyanenet_hailo,multiple_object_tracking,centerpose,gstreamer_app,mono_depth_2}
```

Figure 1: Running with --help lists the different apps available to run

**To quit an app type 'q' when the app window is selected.**

To change the input source for the app use the '--input / -i' flag. You can use a video file, image folder, or camera device. For example, if your camera is listed by the os as **/dev/video2**, you can use it explicitly with:

```
hailo run-app <app name> -i /dev/video2
```

#### 3.4.1 CLI options

Every app may have more specific command line options for its needs, you can see more details when running the app by using the '-h or --help' command after choosing an app:

```
hailo run-app <app name> -h
-h, --help            show this help message and exit
-i INPUT, --input INPUT Input source for example /dev/video0, video.avi
--no-display          Do not show display
--no-power            delete power printouts
--show-temperature    enable temperature printouts
--no-fps              delete fps printouts
--debug              Enable debug features
--profile             Enable profile tracing
--enable-acceleration Enable inference acceleration
--timeout TIMEOUT     Enable a timeout to kill the app
--source-fps SOURCE_FPS FPS to read from source, default is 30
```

Additional tools are accessible in sub commands menus.

For **logger** tools add the logger sub command.

```
hailo run-app detection logger -h
```

```
usage: hailo run-app detection logger [-h]
                                     [--log-level {NOTSET,SPAM,DEBUG,VERBOSE,INFO,NOTICE,WARNING,SUCCESS,ERROR,CRITICAL}]
                                     [--log-file] [--log-path LOG_PATH]
                                     [--no-colors]
```

optional arguments:

```
-h, --help            show this help message and exit
--log-level {NOTSET,SPAM,DEBUG,VERBOSE,INFO,NOTICE,WARNING,SUCCESS,ERROR,CRITICAL}
                        Set logger verbosity
--log-file            Enable log file
--log-path LOG_PATH   Path to write log file to
--no-colors           Disable log colors
```

For **record** tools add the record sub command.

```
hailo run-app detection record -h
```

```
usage: hailo run-app detection record [-h] [--record-video [RECORD_VIDEO]]
                                     [--record-blocks [RECORD_BLOCKS [RECORD_BLOCKS ...]]]
                                     [--records-path RECORDS_PATH]
                                     [--record-limit RECORD_LIMIT]
```

optional arguments:

```
-h, --help            show this help message and exit
--record-video [RECORD_VIDEO]
                       Enable output recording
--record-blocks [RECORD_BLOCKS [RECORD_BLOCKS ...]]
                       Enable output recording for all blocks, can name
                       specific blocks to record
--records-path RECORDS_PATH
                       Path to log block records to.
--record-limit RECORD_LIMIT
                       Limit recording to certain number of frames
```

A good use case for this feature is to record a demo video directly from the app.

```
hailo run-app detection record --record-video video.mp4
```

**Important: Do not stop apps by pressing ctrl-c. This can result in some processes not shutting down properly.**

## 4 Object Detection

### 4.1 Detection App

The detection app is used to run all our detection networks with a pipeline aimed for this task. It has a history tracker which allows to track detections and to smooth bounding boxes.

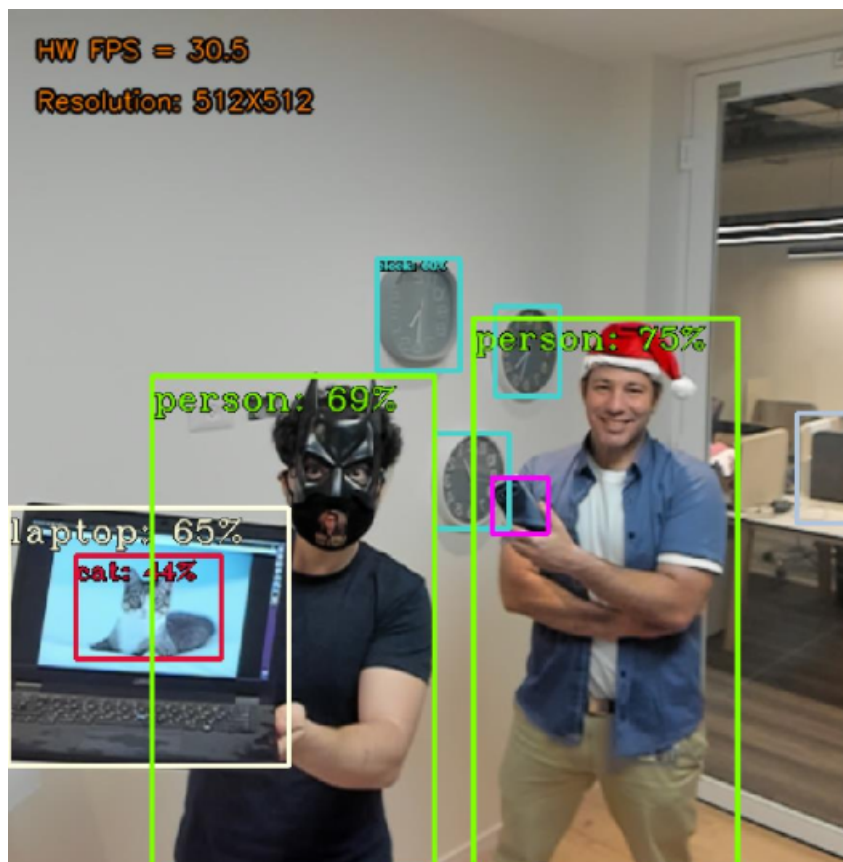
You can control the tracker thresholds online using sliders.

- **Distance threshold** is used to decide whether a certain detection should be matched to a different one from the previous frame (Or from older frames...). This threshold is matched to a metric measuring the bounding boxes locations over frames.
- **Aging threshold** is used to "delete" old entries which have not been "seen lately". Every time an entry is matched to a new detection its aging counter is reset. If the aging counter reaches the aging threshold it is evacuated from the history tracker.
- **BBOX smoothing** is an "alpha value" used to smooth BBOX between frames. Lower value is less smoothing





See below subsections [4.1.1](#) - [4.1.4](#) for the networks which are available for this app.



#### 4.1.1 CenterNet-ResNet50

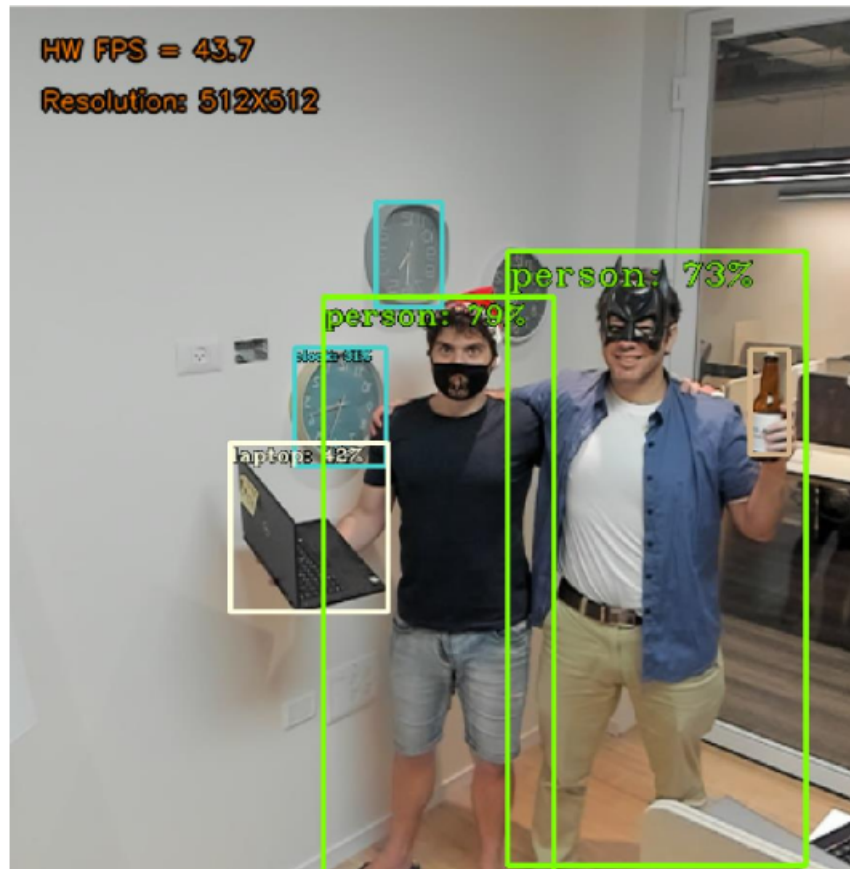






```
$> hailo run-app detection --network-selection centernet
```

|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 512x512x3   |
|  | <b>Native accuracy</b> | 31.7mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Public model. Taken from GluonCV detection <a href="#">model zoo</a> . Model name is center_net_resnet50_v1b_coco |

By default, the app assumes a webcam, so make sure you have one connected. To stream data from a different source, camera of file, use the `--input` flag.

## 4.1.2 CenterNet-ResNet18

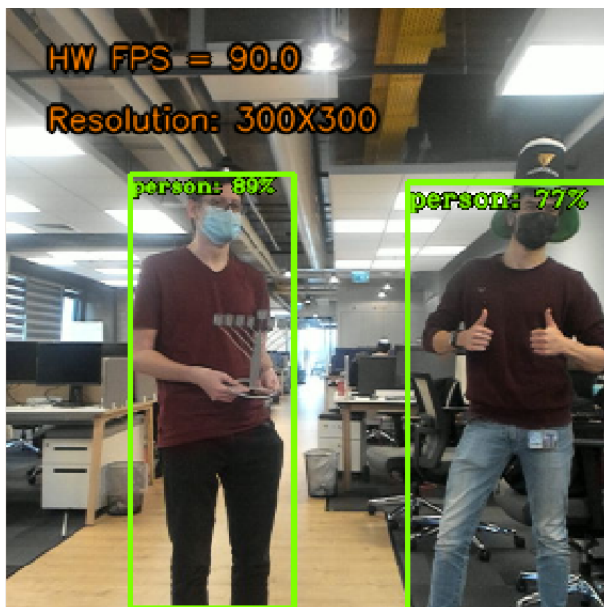






| \$> hailo run-app detection --network-selection centernet18                         |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 512x512x3   |
|  | <b>Native accuracy</b> | 26.3mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Public model. Taken from GluonCV detection <a href="#">model zoo</a> . Model name is center_net_resnet18_v1b_coco |





By default, the app assumes a webcam, so make sure you have one connected. To stream data from a different source, camera of file, use the --input flag.

### 4.1.3 MobileNet v1 SSD

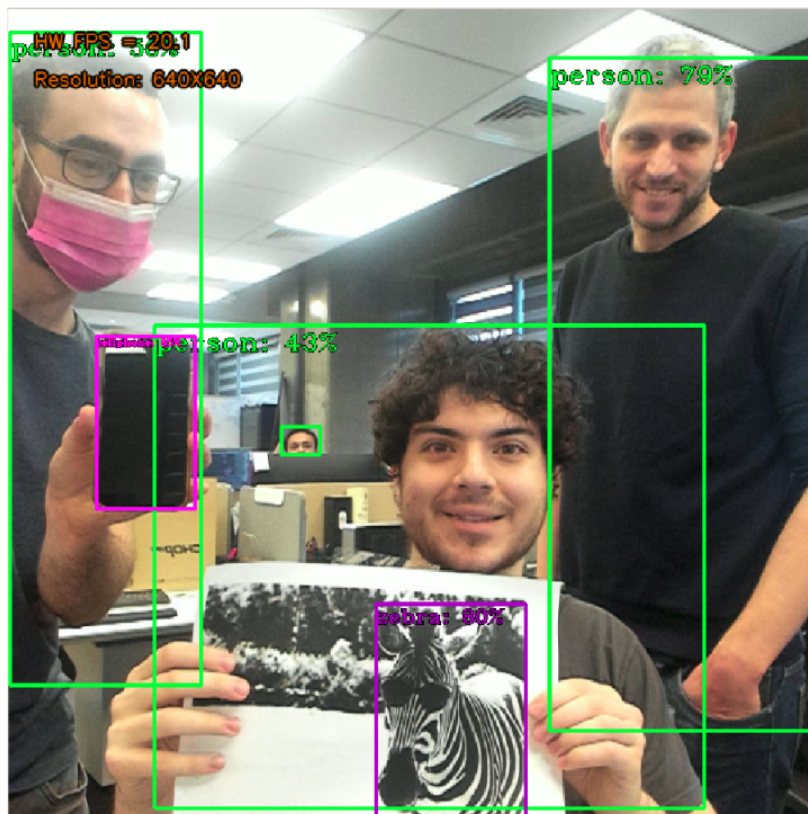
The NMS post-processing is done on chip. This allows to leverage the high FPS supported by Hailo w/o requiring a heavy postprocessing on the host. The network is trained for two flavors. One is based on the original network trained on COCO Dataset and its app is mobilenetSSD300. The other flavor is trained on VisDrone Dataset and is referred to as visdrone.



| \$> hailo run-app detection --network-selection mobilenetSSD300                     |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 300x300x3   |
|  | <b>Native accuracy</b> | 23.1mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Public model. Taken from TensorFlow detection <a href="#">model Zoo</a> . Link to <a href="#">model</a> |

| \$> hailo run-app detection --network-selection visdrone                            |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 300x300x3  |
|  | <b>Native accuracy</b> | 2.3mAP   |
|  | <b>Dataset</b>         | <a href="#">Visdrone object detection</a>  |
|  | <b>Model</b>           | Hailo trained. Trained with Tensorflow <a href="#">object detection API</a> on the vis-drone dataset |





#### 4.1.4 YOLOv5







Modern object detection architecture that is based on the YOLO-v3 meta-architecture with CSPNet backbone. The YOLO-v5 was released on 05/2020 with a very efficient design and SoTA accuracy results on the COCO benchmark.

We have two variants of the YOLO-v5 architecture **yolov5s** and **yolov5m** the 's' and 'm' stands for small and medium sized networks. See the [yolov5 git](#) for details.

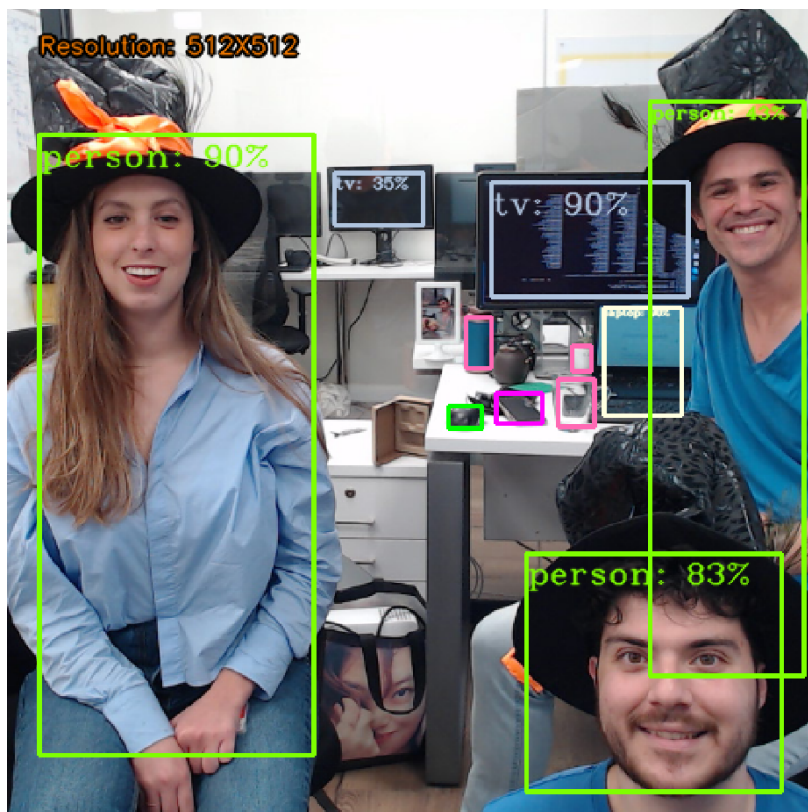
##### YOLOv5m

| \$> hailo run-app detection --network-selection yolov5m                             |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 640x640x3   |
|  | <b>Native accuracy</b> | 41.7mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Hailo Trained. Same as yolov5m in the original <a href="#">yolov5 git</a> with the SPP module removed |





## YOLOv5s

| \$> hailo run-app detection --network-selection yolov5s                           |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 640x640x3   |
|  | <b>Native accuracy</b> | 34.35mAP  |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Hailo Trained. Same as yolov5s in the original <a href="#">yolov5 git</a> with the SPP module removed |

#### 4.1.5 YOLOv4

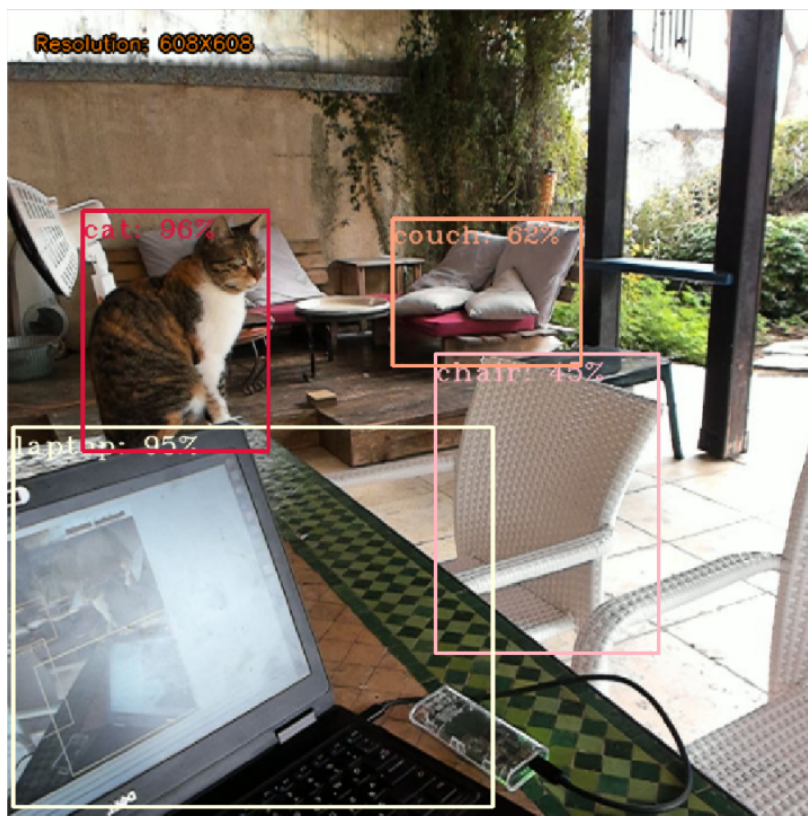


This state-of-the-art, Yolo-V4 object detection network is taken from Darknet. The network presents great accuracy on the COCO benchmark comparable with YOLOv5m; however it has large memory footprint and therefore less compatible with edge devices.





| \$> hailo run-app detection --network-selection yolov4                              |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 512x512x3  |
|  | <b>Native accuracy</b> | 42.4mAP  |
|  | <b>Dataset</b>         | <a href="#">coco2017</a>                           |
|  | <b>Model</b>           | YOLOv4-Leaky from <a href="#">YOLOv4-model-zoo</a> |







#### 4.1.6 YOLOv3

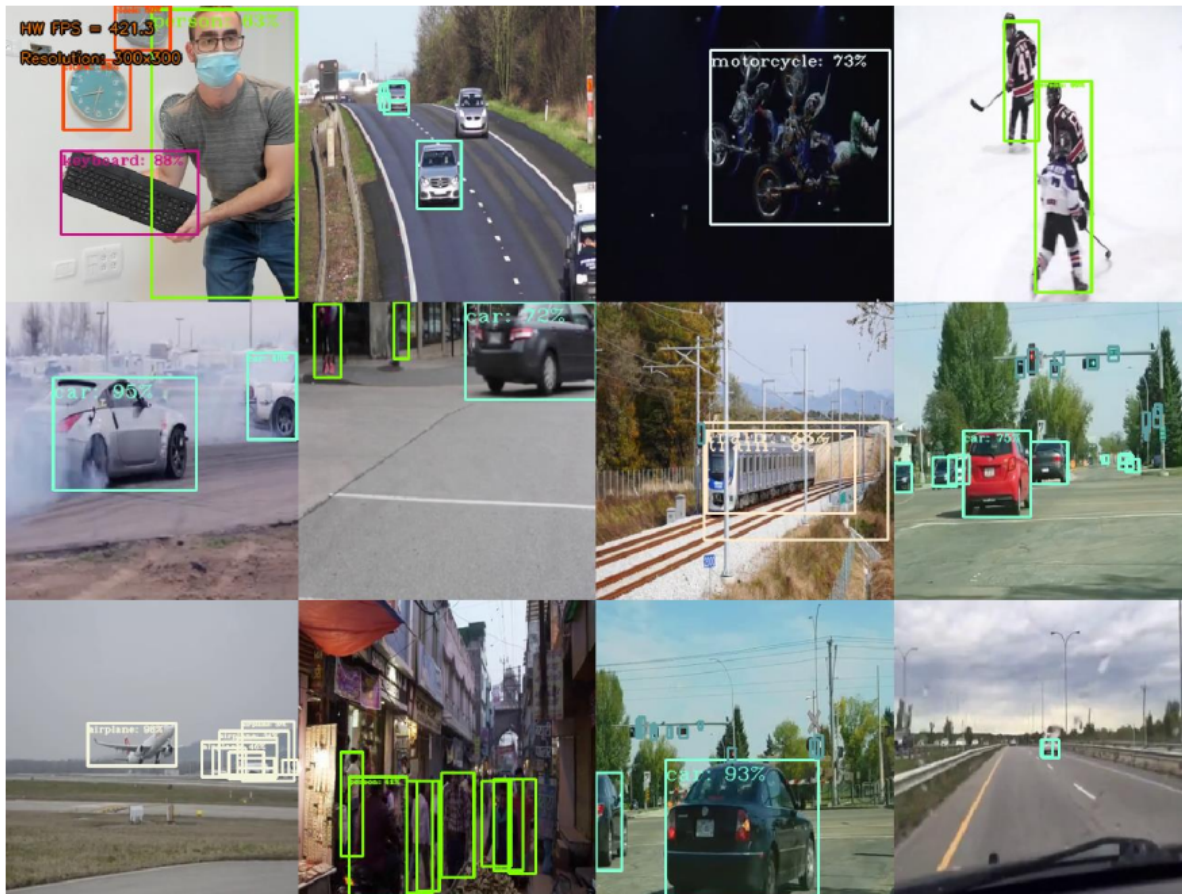


While no longer state-of-the-art, Yolo-V3 remains a popular architecture for object detection. Compared to yolov5, yolov3 has lower accuracy and is much less compatible with edge application due to its large memory footprint. Available in two resolutions 608x608 and 416x416.

| \$> hailo run-app detection --network-selection yolov3                              |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 608x608x3  |
|  | <b>Native accuracy</b> | 36.62mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>   |
|  | <b>Model</b>           | Public model. Taken from GluonCV detection <a href="#">model zoo</a> .<br>Model name is yolo3_darknet53_coco (608x608) |

| \$> hailo run-app detection --network-selection yolov3_416                          |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 416x416x3  |
|  | <b>Native accuracy</b> | 36.06mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>   |
|  | <b>Model</b>           | Public model. Taken from GluonCV detection <a href="#">model zoo</a> .<br>Model name is yolo3_darknet53_coco (416x416) |

## 4.2 Multi Stream



This app shows Hailo's ability to run on multiple streams, leveraging the device's high FPS. It supports multiple networks and varied amount of stream and input sources. You can select number of streams using the `--streams` flag. Inputs are selected as usual using the `--input` flag. However, in this app the `--input` flag can accept multiple inputs. You can select specific devices and files. If the number of streams is larger than the inputs you specify the app will add its default stream.

Supported networks mobilenetSSD300, yolov4, yolov3, yolov3\_416.

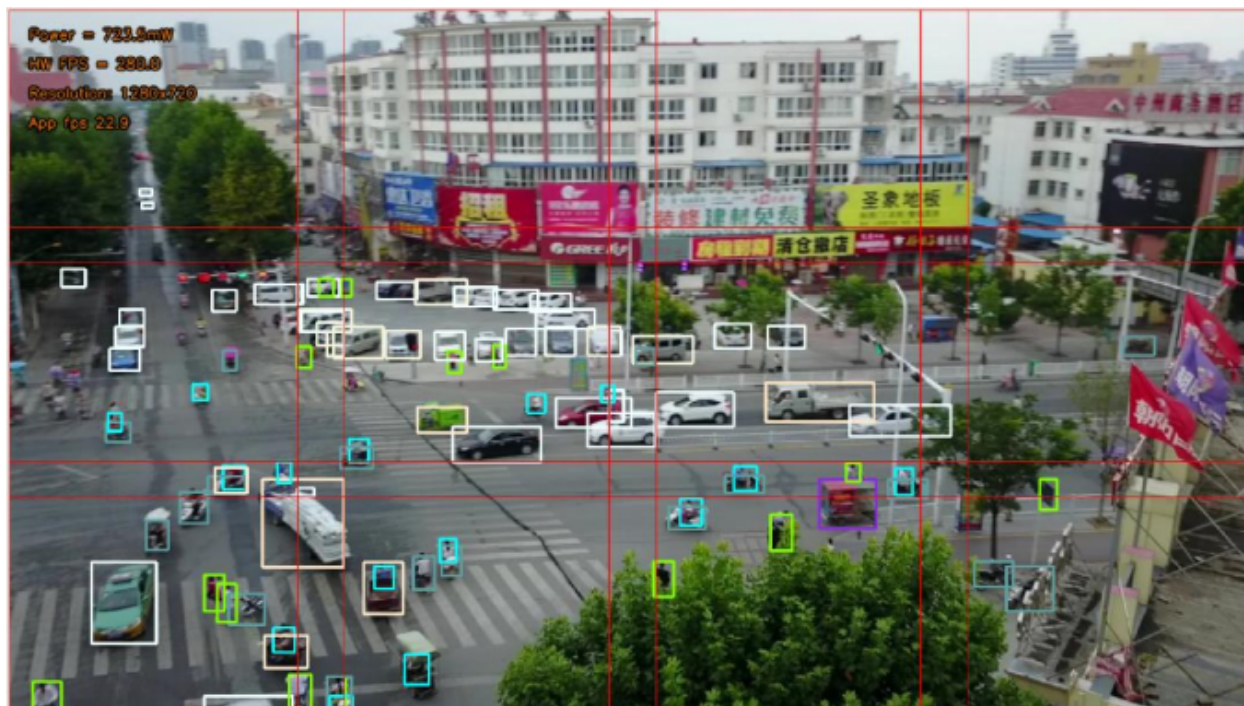
Example commands:





```
hailo run-app multi_stream --network-selection mobilenetSSD300 -i /dev/video0 /dev/video2 some_video.mp4
hailo run-app multi_stream --network-selection yolov3_416 -i /dev/video0 --streams 8
```

**Note:** Using more than 2 USB cameras with 'normal' configuration can lead to failure due to USB memory allocation and might require special configurations / HW. When adding your own videos resize them in advance for better performance.







### 4.3 Tiling







| \$> hailo run-app tiling  |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | Variable. A large frame is processed with overlapping 300x300x3 tiles. The default resolution is 1280x720x3          |
|  | <b>Native accuracy</b> | N/A  |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a> or <a href="#">VisDrone</a>   |
|  | <b>Model</b>           | The application supports three different networks: mobilnet-ssd COCO, mobilenet-ssd Visdrone and mobiledet-dsp COCO. |

In this app, we use a detection network trained on low input resolution (e.g 300x300) to process a high resolution input stream. This is done by breaking up each frame into several tiles which are processed independently. Tiling leverages the device's high compute by trading FPS for higher resolution. This method is especially effective for detecting small objects in high-resolution frames. This app supports 3 different detection networks.





### 4.3.1 MobileNetV1 SSD NMS - COCO

|   |                        |   |
|---|------------------------|---|
| \$> hailo run-app tiling --network-selection mobilenetSSD300                      |                        |   |
|  | <b>Resolution</b>      | 300x300x3   |
|  | <b>Native accuracy</b> | 23.1mAP <sup>1</sup>  |
|  | <b>Dataset</b>         | COCO val2017  |
|  | <b>Model</b>           | Public model. Taken from TensorFlow detection <a href="#">model Zoo</a> . Link to <a href="#">model</a> |

### 4.3.2 MobileNetV1 SSD NMS - Visdrone

|   |                        |  |
|---|------------------------|--|
| \$> hailo run-app tiling --network-selection visdrone                             |                        |  |
|  | <b>Resolution</b>      | 300x300x3  |
|  | <b>Native accuracy</b> | 2.3mAP <sup>1</sup>  |
|  | <b>Dataset</b>         | Visdrone object detection  |
|  | <b>Model</b>           | Hailo trained. Trained with Tensorflow <a href="#">object detection API</a> on the vis-drone dataset |

### 4.3.3 Lightface slim

|   |                        |   |
|---|------------------------|---|
| \$> hailo run-app tiling --network-selection lightface_slim                         |                        |   |
|  | <b>Resolution</b>      | 320x240x3   |
|  | <b>Native accuracy</b> | 38.99%  |
|  | <b>Dataset</b>         | WIDER-Face  |
|  | <b>Model</b>           | Based on <a href="#">Ultra-Light-Fast-Generic-Face-Detector-1MB</a> |

### 4.3.4 Running the Tiling App

By default the app will run with a sample video using the MobileNetV1 SSD VisDrone flavor. You can choose to use a different video file or the webcam from command line. To show the ability to trade-off fps and accuracy there are some 'pre-defined' tiling options you can choose from. To switch between tiling options, use the digit keys 0-7 in the video window context. This will switch between tiling modes; the tiles are marked with thin red frames. The actual resolution in pixels which runs on Hailo can be calculated based on the model input resolution. For example, for MobileNetV1 SSD this calculation is:  $300 \times 300 \times N_{tiles}$ . For comparison, using 12 tiles takes a little bit more than processing 720p resolution directly (Note that we have overlap between tiles). The input resolution, number of tiles and overlapping between tiles can be selected from command line. The tiling strategy that gives optimal performance is dataset dependent. The VisDrone dataset consists of only small objects which we can assume are always confined within an single tile. As such it is better suited for running single-scale tiling with little overlap and without additional filtering. On the other hand, the COCO dataset includes multi-scale objects. In this case dividing the frame to small tiles might miss large objects or "crop" them to tiles. For optimal

<sup>1</sup>Native accuracy for network without tiling

performance on COCO, one should use a multi-scale tiling strategy which allows us to filter the correct detection over several scales. In this mode we use 3 sets of tiles at 3 different scales:

- Large scale, one tile to cover the entire frame (1x1)
- Medium scale dividing the frame to 2x2 tiles.
- Small scale dividing the frame to 3x3 tiles.

In this mode we use  $1 + 4 + 9 = 14$  tiles for each frame. We have added smart merging filters which are controlled by the flags below. The best heuristic we found for filtering detections over scales is to "ignore" detections which are cropped by the tile boundaries. This heuristic is enabled as default when running with COCO dataset. Running the following CLI command:

```
hailo run-app tiling --network-selection mobilenetSSD300
```

The mode can also be controlled using these CLI flags:

|   |   |
|---|---|
| <code>--disable-remove-exceeded</code>  | Remove detections which reaches the tile boundaries         |
| <code>--border-threshold</code>         | Number of pixels from the boundary to filter. default is 15 |
| <code>--disable-filter-classes</code>   | Disables showing only the catIds selected                   |
| <code>--catIds</code>                   | which categories to keep on screen                          |
| <code>--disable-remove-landscape</code> | Heuristic to remove COCO's "is crowded" <sup>2</sup>        |
| <code>--network-selection</code>        | mobilenetSSD300, visdrone, or mobiledet                     |

In this mode the digit keys 0-4 are used to select 1x1, 2x2, 3x3, or multiscale tiling. See below comparison between results achieved by running with multiscale, compared to running with only 1 tile (COCO dataset)



<sup>2</sup>this might remove real detections of persons if they are large enough and have a landscape aspect ratio.

## 5 Semantic Segmentation Apps

### 5.1 ResNet18-FCN8



| \$> hailo run-app semantic_segmentation   |                         |   |
|---|-------------------------|---|
|  | <b>Resolution</b>       | 1920x1024x3   |
|  | <b>Numeric accuracy</b> | 65.18mIOU   |
|  | <b>Dataset</b>          | <a href="#">CityScapes</a>  |
|  | <b>Model</b>            | Hailo Trained. Trained on cityscapes using <a href="#">GlounCV</a> and a resnet-18-FCN8 architecture. |

You can run the app on a different video, with the `-input` flag followed by the path to the video.







## 6 Pose Estimation

### 6.1 CenterPose



```
$> hailo run-app centerpose
```

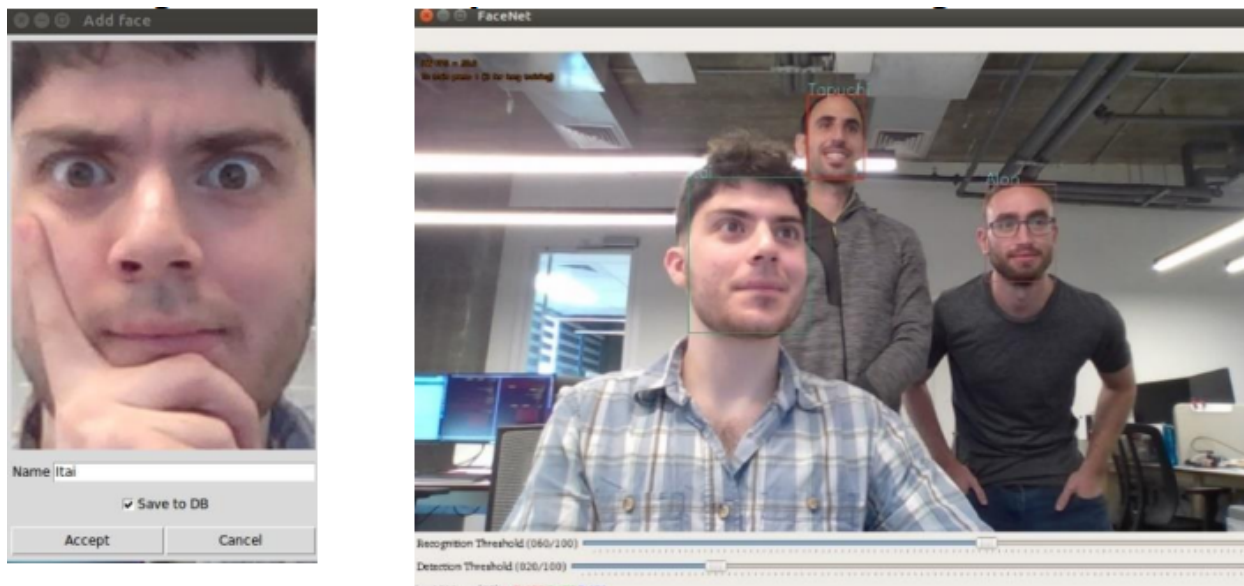
|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 640x640x3   |
|  | <b>Native accuracy</b> | 53.9 OKS  |
|  | <b>Dataset</b>         | COCO-Pose   |
|  | <b>Model</b>           | Hailo Trained. Based on <a href="#">centerpose</a> architecture with <a href="#">RegnetX_1.6FG</a> backbone |

Post processing is not as dependent on number of skeletons like openpose. The Centernet BBOX decoding post-process runs on Hailo. The backbone used is based on regnetx\_1.6<sup>3</sup>  
To enable tracking use the **--enable-tracking** flag.

<sup>3</sup>[https://github.com/facebookresearch/pycls/blob/master/MODEL\\_ZOO.md](https://github.com/facebookresearch/pycls/blob/master/MODEL_ZOO.md)

## 7 Face Recognition

### 7.1 FaceNet



| \$> hailo run-app facenet |                        |   |
|---------------------------|------------------------|---|
|                           | <b>Resolution</b>      | 112x112x3   |
|                           | <b>Native accuracy</b> | 98.5% (ArcFace-MNv1) 99.6% (ArcFace-RN50)   |
|                           | <b>Dataset</b>         | LFW   |
|                           | <b>Model</b>           | Hailo trained. Using <a href="#">insightFace</a> public git and changing the backbone to Resnet-50 and MobileNet-V1-1.0 |

In this app we have 2 networks running, one for *face detection* and the second for *face recognition*. This app has the capability to run on 2 Hailo-8 devices, in that case the detection network is RetinaFace Mobilenet HD, see [8.1](#).

In the case that there is only one Hailo device, the detection network is RetinaFace Mobilenet0.25<sup>4</sup> VGA. It is ran on the host machine.

The face recognition is accelerated on the Hailo-8 chip.

The FaceNet app supports 2 app flavors:

#### 1. Face recognition + Age-Gender (Default)

- A recognition network based on ArcFace with a modified Mobilenet-V2 backbone, working at a resolution of 112x112.  
The network outputs a 128-dimensional face embedding vector
- An age/gender estimation network based on a shortened Mobilenet-V2 backbone, working at a resolution of 224x224.  
The network outputs probability distribution per age from 1-100 and a probability of the subject being male from 0 to 1

<sup>4</sup> Mobilenet0.25, since only a 1/4 of the channels is used

## 2. Better recognition network

A better-recognition option activated using the `-better-recognition` flag. A recognition network based on ArcFace with a modified ResNet50 backbone, working at a resolution of 112x112. The network outputs 512-dimensional face embedding vector.

### 7.1.1 Running the Demo

When running the demo, you can train new faces with 2 options:

- **Capture One Frame** This option should be used to get the first sample and should be done when front facing the camera. This capture is initiated by pressing '1' in the video window context. This can be done when more than one person is in the frame. A pop up with the picture of the selected person will appear prompting to input the name of the person detected. The app will suggest faces which are not recognized. Faces are also tracked to enhance consistency, so it is possible that a face which was already registered will show up again. Save it under the same name to update the user with another sample. After pressing accept the face will be added to a temporary DB and will be recognized immediately. After the app is closed the temporary DB is not saved. If you want to save the trained DB to a file to be reloaded again later you should run with the `-use-db` flag. When this flag is used the app will load your DB file at startup and you will be able to update it and to save your updates. If you do not supply a DB file name using `-db-file` a default one will be selected. If the DB file you supplied does not exist the app will create it. In this mode a save to db checkbox will be added to the pop up window (this will save all current temporary DB, not only this face). The database file is an NPZ file.

**Important: The DB used for one recognition version is not compatible with the other one. i.e. DB trained on "default" (age-gender) mode will not work with the "better recognition" mode.**

- **Multi-Frame Capture** In order to get better performance, you can use more data and more angles to help recognition. This mode should be done after an initial capture is done. In this mode **only one person is allowed in the frame**. If more than one person is present other face embeddings will be assigned to the same person. To initiate this mode press '2' in the display window context. Now the app is looking for unrecognized samples. Move your head in different angles or try wearing glasses or growing a beard. When enough samples are collected or reaching timeout the 'save' box will appear. You should input the same name used before to add these samples to its DB.

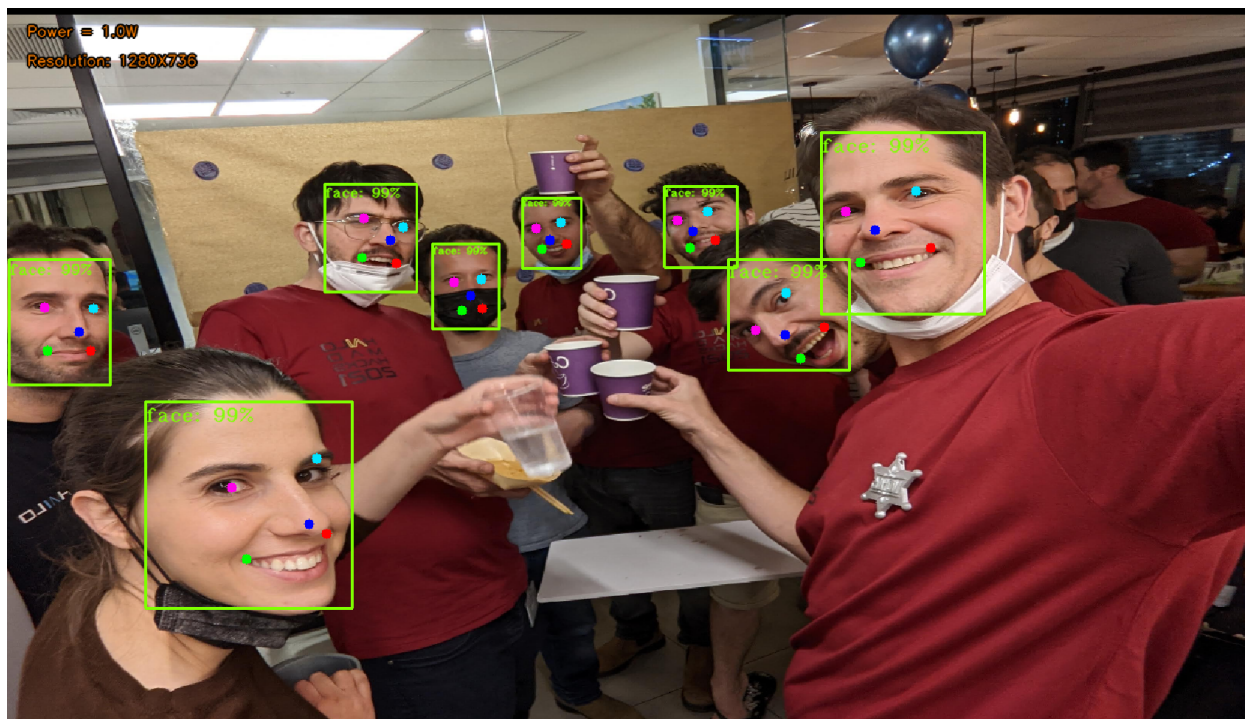
**Note:** training with 'bad' examples will lead to 'bad' recognition.







## 8 Face Detection

This application is used for face detection networks.

### 8.1 RetinaFace MobileNet HD



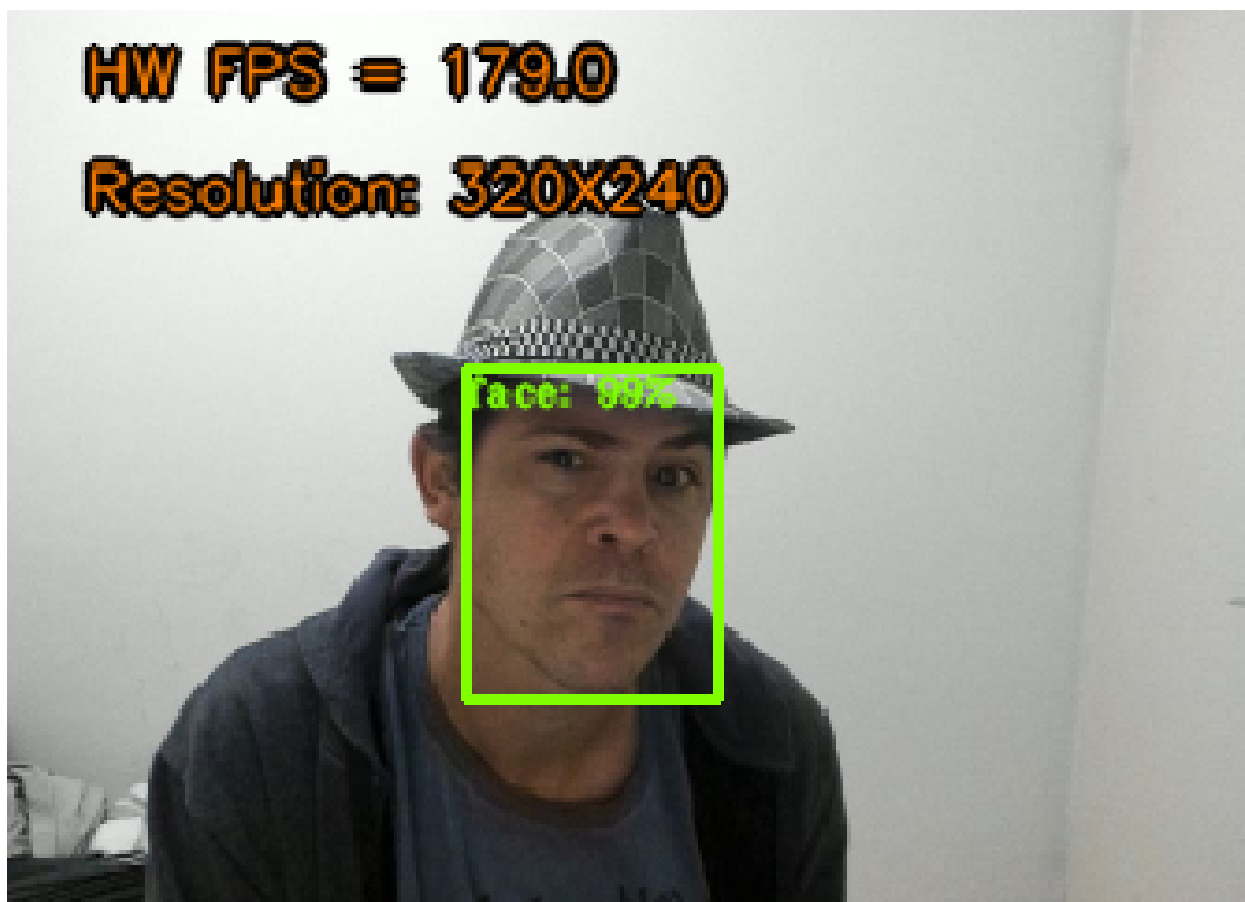
```
$> hailo run-app face_detection --network-selection retinaface
```

|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 1200x736x3  |
|  | <b>Native accuracy</b> | 81.2%   |
|  | <b>Dataset</b>         | WIDER-Face  |
|  | <b>Model</b>           | Hailo trained. Based on <a href="#">RetinaFace</a> architecture with mobilenet backbone and HD input resolution |





This demo runs the retinaface network for face and 5-landmark detection. For each face within the frame, the network outputs a bounding-box and 5 facial landmarks - the eyes, bottom top of the nose the corners of the mouth.

To change the source stream (camera or video) use the `-input` flag.

## 8.2 Lightface Slim Face Detection



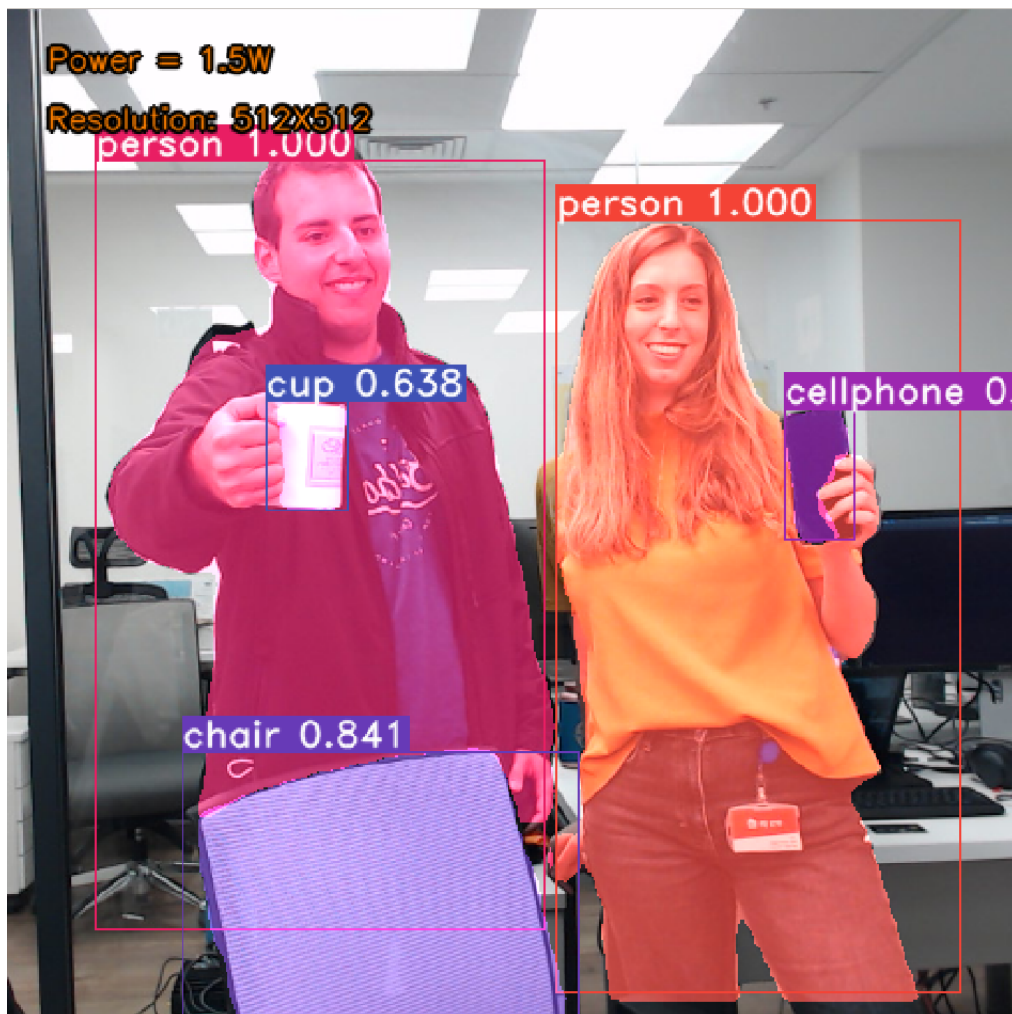
```
$> hailo run-app face_detection --network-selection lightface_slim
```



|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 320x240x3   |
|  | <b>Native accuracy</b> | 38.99%  |
|  | <b>Dataset</b>         | <a href="#">WIDER-Face</a>  |
|  | <b>Model</b>           | Based on <a href="#">Ultra-Light-Fast-Generic-Face-Detector-1MB</a> |

This network is a lightweight face detection which can be ran in high frame-rate with a small footprint on Hailo8. It makes it perfect to be added as another stage allocated on the same chip along side other networks.

## 9 Instance Segmentation

### 9.1 YOLACT



| \$> hailo run-app yolact  |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 512x512x3   |
|  | <b>Native accuracy</b> | 27.7mAP   |
|  | <b>Dataset</b>         | COCO val2017  |
|  | <b>Model</b>           | Hailo trained. Based on the Yolact architecture with a RegnetX-800MF backbone |

This app performs instance segmentation using the single-stage Yolact architecture. Each detected object is treated as a separate instance and shaded with a different color even if belonging to the same class. The sliders at the bottom of the window adjust the network's sensitivity.





Note that the application only works on a subset of classes from COCO which are relevant for indoor setting. The supported classes are: ('person', 'bicycle', 'bench', 'backpack', 'handbag', 'tie', 'suitcase', 'bottle', 'cup', 'bowl', 'chair', 'couch', 'potted plant', 'dining table', 'tv', 'laptop', 'keyboard', 'cellphone', 'book')

## 10 Lane Detection

### 10.1 PolyLaneNet



```
$> hailo run-app polylanenet_hailo
```

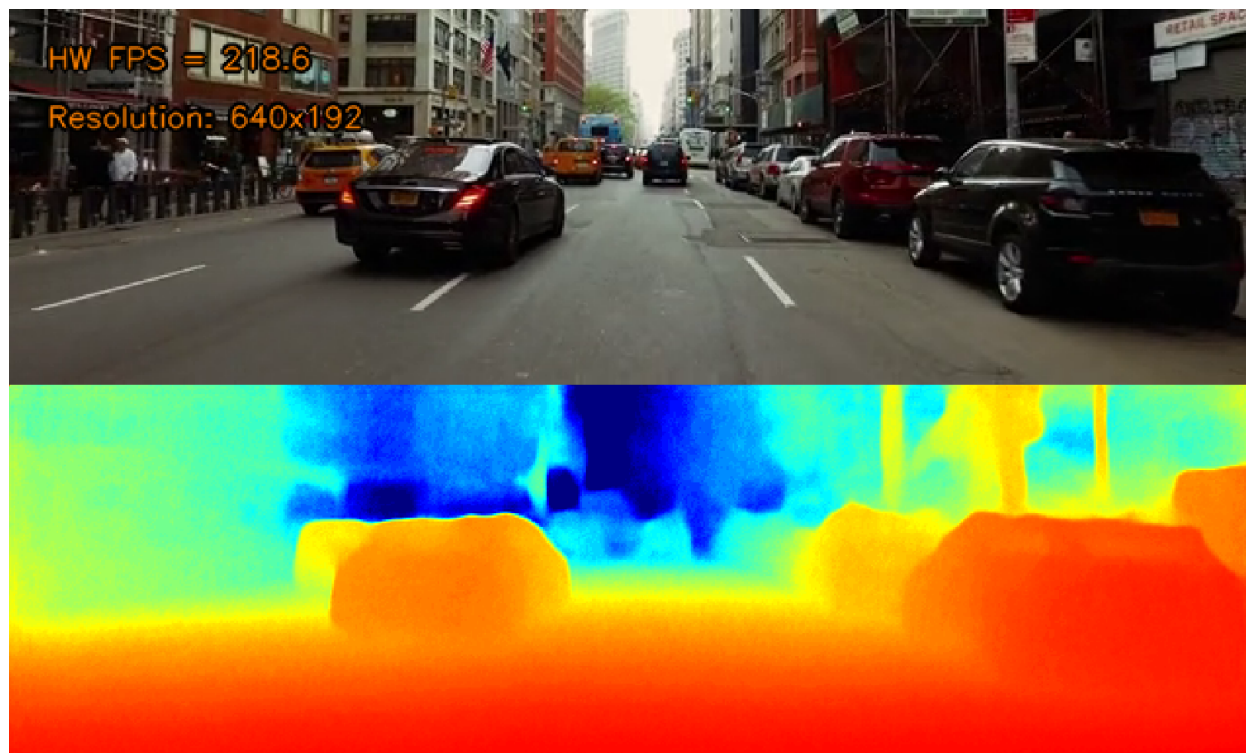
|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 640x360x3   |
|  | <b>Native accuracy</b> | 91.31%  |
|  | <b>Dataset</b>         | <a href="#">Tusimple</a>  |
|  | <b>Model</b>           | Public Model. Taken from <a href="#">PolyLaneNet</a> with ResNet34 backbone |




This demo runs the polylanenet network for lane detection with Resnet34 backbone with small classification head. The left and right sides of the lane are highlighted along with a dotted line for the center of the lane.



## 11 Depth Estimation

### 11.1 Mono Depth2

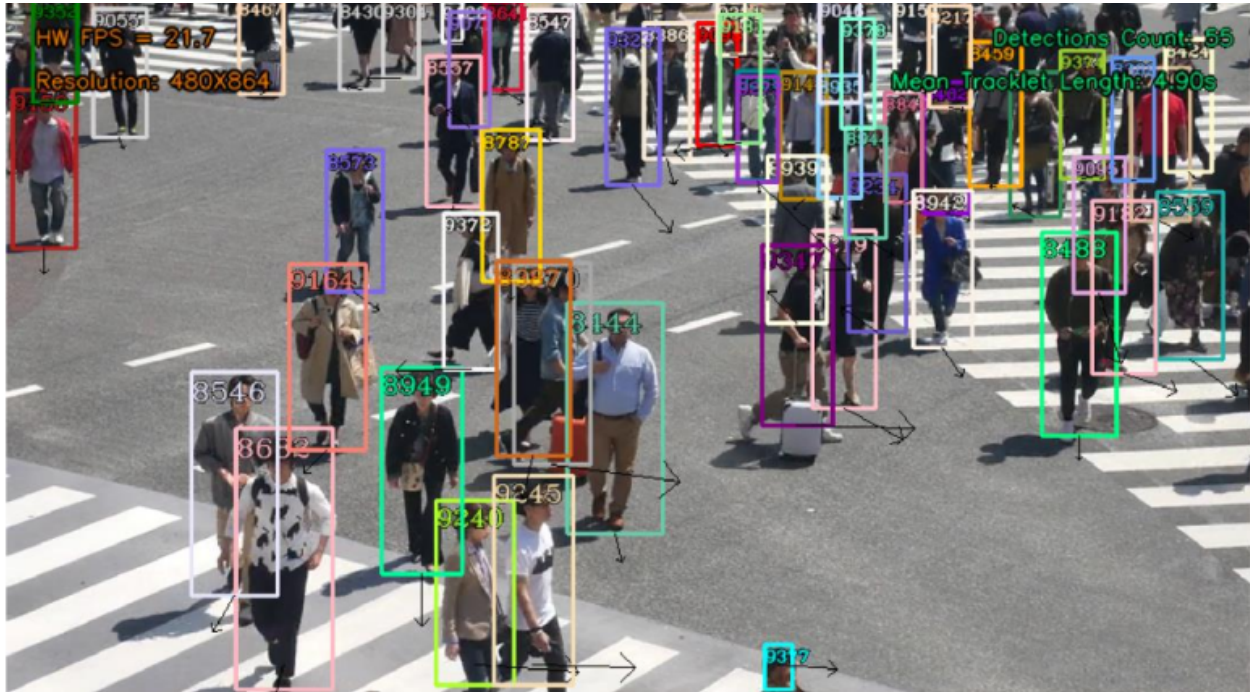


| \$> hailo run-app mono_depth_2  |                        |                             |
|---|------------------------|-----------------------------|
|  | <b>Resolution</b>      | 640x192x3                   |
|  | <b>Native accuracy</b> | 0.106 Abs. Rel.             |
|  | <b>Dataset</b>         | <a href="#">KITTI depth</a> |





Self supervised monocular depth estimation architecture that is based on ResNet18-UNET architecture and shows great accuracy on the KITTI benchmark for predicting per pixels depth estimation.

## 12 Multiple Object Tracking

### 12.1 FAIR MOT RegNet800



```
$> hailo run-app multiple_object_tracking
```

|   |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 480x864x3  |
|  | <b>Native accuracy</b> | 70.4%  |
|  | <b>Dataset</b>         | MOT-16   |
|  | <b>Model</b>           | Hailo trained. Based on the FairMOT architecture with a RegnetX-800MF backbone |

This app performs tracking of multiple pedestrians using the fairMOT meta-architecture with a RegnetX-800MF backbone. The application's output per pedestrian is:

- Bounding box
- An ID label based on the tracking of that pedestrian along his entire appearance in the video
- A arrow representing its velocity vector, based on the last second of movement by that pedestrian



## 13 Classification

### 13.1 ResNet50



```
$> hailo run-app classification
```

|  |                 |  |
|--|-----------------|--|
|  | Resolution      | 224x224x3  |
|  | Native accuracy | 75.21% Top-1 accuracy  |
|  | Dataset         | ImageNet-1K  |
|  | Model           | Public model. Taken from Tensorflow slim <a href="#">model zoo</a> . Link to <a href="#">model</a> |

The app performs object classification using the Resnet-50 model trained on Imagenet. The softmax is done on chip and the FPS is improved. The top right stream is taken live from camera (or video use the **--input** flag).

The Imagenet dataset does not include 'person' and it must output 'some' classification. So if no "known" objects are presented low score "random" classifications will be shown.

Not all classifications can be displayed on screen, you can change the **"Display refresh delay"** using the in app slider.

The updated allocation we did allows to reach very high FPS and hence draw a lot of power. If your system's heat dissipation is not good enough you might need to limit the FPS using the **--fps-limit** flag in order to work for long periods. (This issue was found only while running with cheap m.2 to thunderbolt converters)

**Note:** The APPs release includes only a small number of images. This is done to reduce size and comply with copyrights. Not all frames classified are presented to screen. But all 128 images are classified every round. The user can change the directory from which the images are taken to have the application work with different images. However, the correct resize should be done in advance. To allow to reach Hailo's max throughput.

You can use your own pictures by giving the app folder of images as an input. To use all "streams" with your input add the **-only-live** flag. Example command:

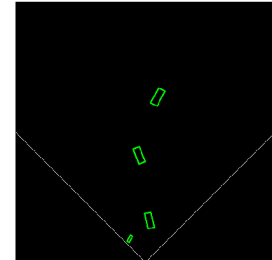
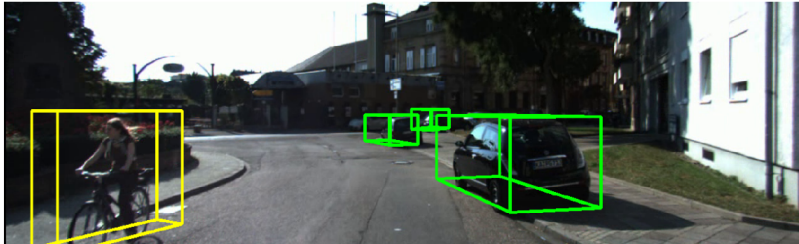
```
1 hailo run-app classification -i IMAGES-FOLDER --only-live
```







## 14 3D Object Detection

### 14.1 SMOKE\_Regnet800

Resolution: 304x1280



```
$> hailo run-app smoke_regnet800
```

|   |                        |  |
|---|------------------------|--|
|  | <b>Resolution</b>      | 1280x384x3   |
|  | <b>Native accuracy</b> | car BEV moderate mAP: 14.14%   |
|  | <b>Dataset</b>         | <a href="#">kitti 3d object detection</a>  |
|  | <b>Model</b>           | Based on <a href="#">SMOKE: Single-Stage Monocular 3D Object Detection via Key-point Estimation</a> . Downloaded from <a href="#">SMOKE github</a> |

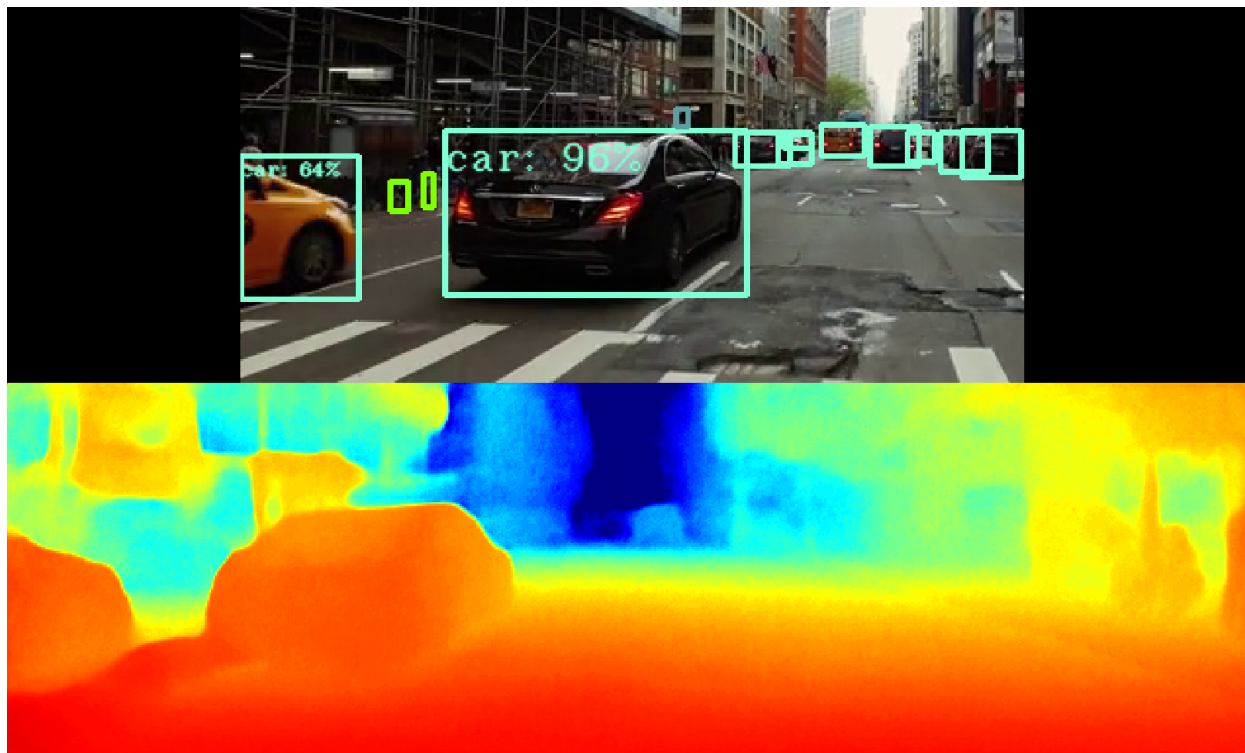
SMOKE is a 3D monocular object detection network based on anchor-less architecture. The network was trained on KITTY to detect 3D boxes and project them to BEV (Birds Eye View).

Due to licensing restrictions of the KITTY data-set the video sample is not released with TAPPAS. To download and prepare the video run the provided script.

```
cd apps/smoke/
./download_unzip_and_create_movie_and_calib.sh
```

## 15 Multitask apps

### 15.1 MonoDepthSSD







```
$> hailo run-app mono_depth_ssd
```

#### MonoDepth2 Network

|   |                        |                             |
|---|------------------------|-----------------------------|
|  | <b>Resolution</b>      | 640x192x3                   |
|  | <b>Native accuracy</b> | 0.106 Abs. Rel.             |
|  | <b>Dataset</b>         | <a href="#">KITTI depth</a> |

#### MobilenetSSD300 Network

|   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 300x300x3   |
|  | <b>Native accuracy</b> | 23.1mAP   |
|  | <b>Dataset</b>         | <a href="#">COCO val2017</a>  |
|  | <b>Model</b>           | Public model. Taken from TensorFlow detection <a href="#">model Zoo</a> . Link to <a href="#">model</a> |

This app showcases Hailo's ability to run several networks on one context concurrently. In this app we merged MobilenetSSD300 (4.1.3) with MonoDepth2 (11.1). Both networks runs in parallel

and are the same networks as in the single network examples. On this app the allocated FPS is somewhat lower due to the fact that less resources are used for each network inference. This demonstrates Hailo's ability to trade performance with additional tasks.

## 16 Gstreamer Apps

In Gstreamer applications the entire pipeline is running in the Gstreamer framework.

The Gstreamer project is the "go to" framework for media applications and is heavily supported by the open source community. This means you can use many ready-to-use plugins across a wide variety of supported platforms.

You can read more about this framework at [gstreamer documentation](#).

In order to get the most of the gstreamer framework, we created a set of elements that enable inference, post-processing, and drawing as part of the gstreamer media pipeline.

In this release, three Gstreamer elements are provided as part of the Hailo plugin:

- **hailosend** - sends buffers to the Hailo device for inference.
- **hailorecv** - receives inference output tensors from the Hailo device.
- **hailofilter** - applies a given post-process/drawing to a buffer.

With these elements, you can create apps with better CPU and memory utilization than ever before, while utilizing our Hailo-8 accelerator to it's maximum potential on both regular and "weak" hosts.

\*\*Gstreamer apps support many of the flags used in the python apps.

### 16.1 Gst Detection

This app runs detection networks using the Gstreamer framework. Currently supported networks are yolov5m, yolov5s, yolov4 and yolov3.

Run command:

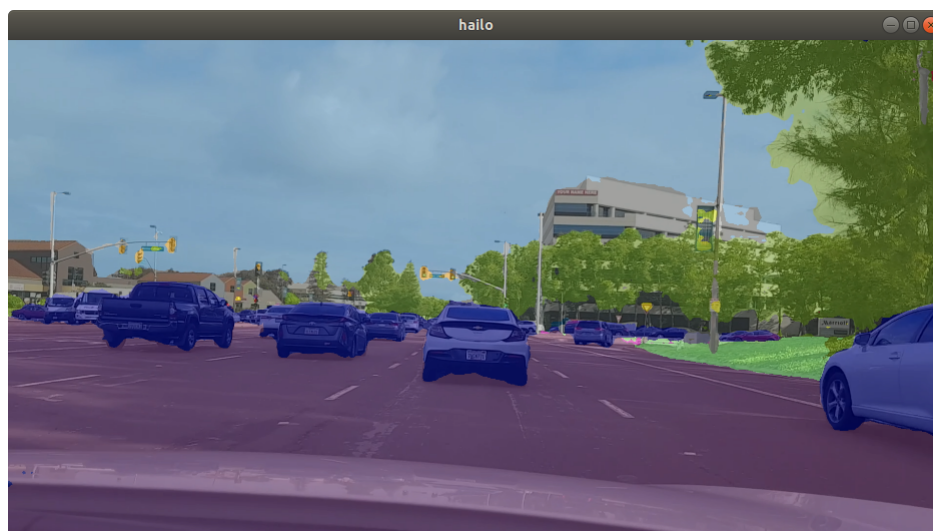
```
hailo run-app gst_detection --network-selection yolov5m
hailo run-app gst_detection --network-selection yolov5s
hailo run-app gst_detection --network-selection yolov4
hailo run-app gst_detection --network-selection yolov3
```

### 16.2 Gst Segmentation

This app runs the fcn8\_resnet\_v1\_18 semantic segmentation network using the Gstreamer framework.

Run command:

```
hailo run-app gst_segmentation
```



### 16.3 Gst Multi Stream

This app shows how to create a multi-streaming pipeline with a Hailo device using the Gstreamer framework. Multiple input sources can be selected including video files and live cameras. Currently supported networks are yolov5m, yolov5s, yolov4 and yolov3.

The number of streams shown can be controlled with the **-n** flag.

Run command:

```
hailo run-app gst_multi_stream # Default 4 streams
hailo run-app gst_multi_stream -n 6 # Run with 6 streams
hailo run-app gst_multi_stream -n 8 --no-input # Run with 8 streams, all file sources
hailo run-app gst_multi_stream --network-selection yolov5s # Run with yolov5m network
```

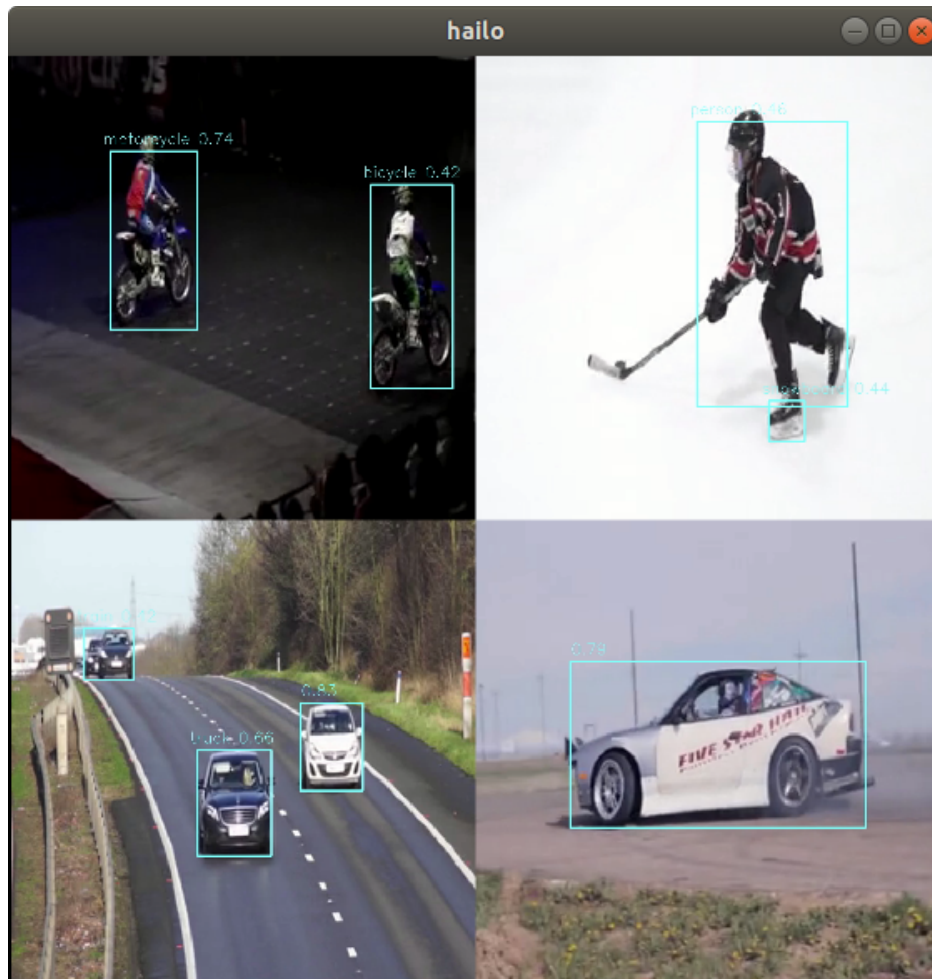
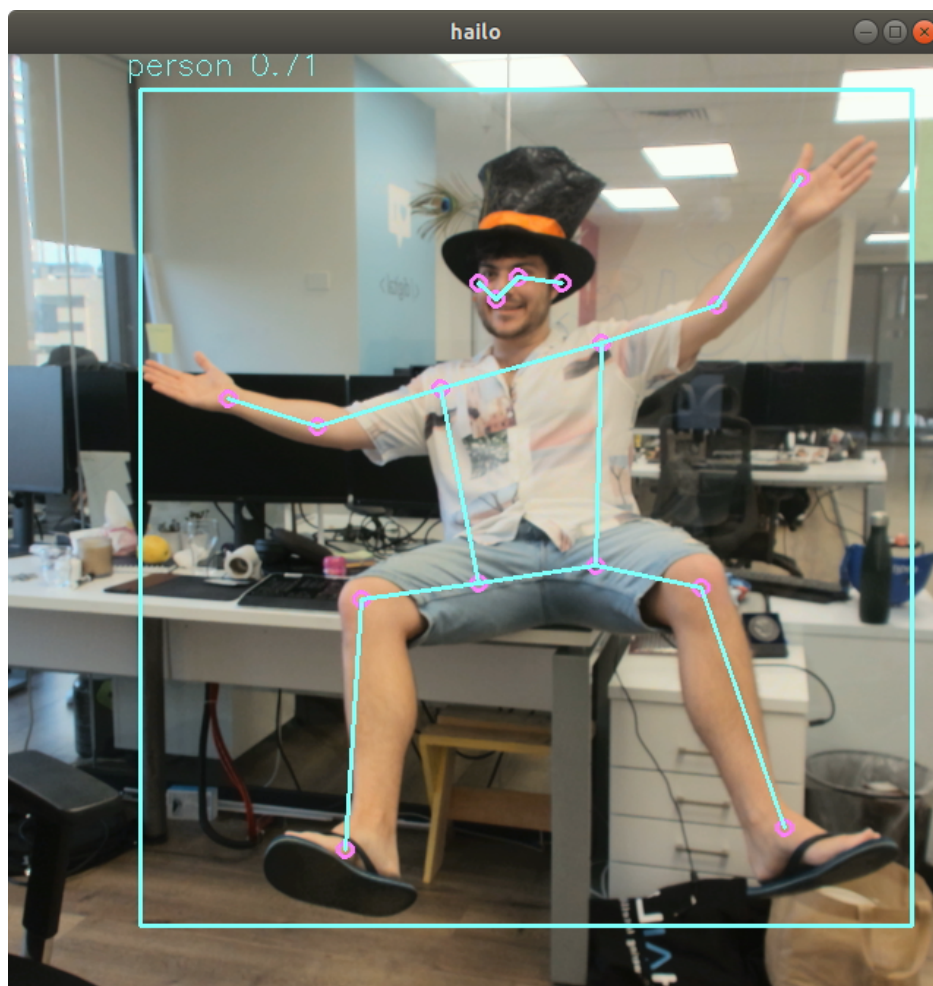


Figure 2: hailo run-app gst\_multi\_stream -no-input

## 16.4 Gst Pose Estimation

This app runs the CenterposeRegnetx\_16gf pose estimation network using the Gstreamer framework. Run command:

```
hailo run-app gst_pose_estimation
```





## 17 MIPI Apps

MIPI Apps require specific HW, including an Industrial camera board or an EVB, and a MIPI camera. MIPI Apps can run exclusively with UDP connection.

Network configuration:

The device default IP address is 10.0.0.93

in order to work with the industrial camera module, two configurations need to be set:

1. Assign a local IP address and a Netmask address to the relevant Network Interface.





example for linux: `ifconfig eno2 10.0.0.50 netmask 255.255.255.0`

2. enlarge host Ethernet buffers

### 17.1 Yolact\_mipi App yolact\_regnetx\_600mf\_d2s\_9classes





| \$> hailo run-app yolact_mipi   |                        |   |
|---|------------------------|---|
|  | <b>Resolution</b>      | 512x512x3   |
|  | <b>Native accuracy</b> | 61.75% mAP  |
|  | <b>Dataset</b>         | D2S   |
|  | <b>Model</b>           | Hailo trained. Based on the Yolact architecture with a RegnetX-600MF backbone |

The Yolact Mipi app performs instance segmentation using the single-stage Yolact architecture. Each detected object is treated as a separate instance and shaded with a different color even if it belongs to the same class.

Note that the application only works on a subset of classes from D2S. The supported classes are: ('apple', 'avocado', 'banana\_single', 'clementine\_single', 'kiwi', 'orange\_single', 'pear', 'cucumber', 'carrot').

When running the app for the first time on a new board, the config-sensor-and-isp flag is needed:

```
hailo run-app yolact_mipi --config-sensor-and-isp
```

The app can be controlled using these CLI flags:

|                         |   |
|-------------------------|---|
| --sensor-fps-config     | select sensor FPS configurations: 10, 20, 23 FPS. Default is 23 |
| --config-sensor-and-isp | configure sensor and isp for a new board                        |

## 18 Add Your Own App/Network

For your convenience, we added an example to our TAPPAS. the example is based on centerpose\_regnetx\_1.6gf\_fpn. It is located at **example/centerpose/** and contains 5 important files:

- Compiled HEF of this network, located under 'files' sub-directory.
- centerpose\_preprocess.py - Contains the pre-processing function for this network.
- centerpose\_postprocess.py - Contains the post-processing and drawing function for this network.
- network.py - Holds a class which encapsulate all the functions and data required to run a specific network on Hailo chip. This class will include for example the HEF file, network input resolution, preprocessing / post processing functions etc.
- app.py - An example of pose-estimating app.

With some minor changes to these files, you will be able to create a new app that runs your own network. The python files in the example are highly documented with all the required functions for this app to work. All files are heavily documented and got inline instructions for adding new network\creating your own app.

**Prerequisites** - Compiled HEF for your network.

Steps to create a new network:

1. Make a new directory under example/ with your <NEW\_NETWORK> name
2. Copy the content of example/centerpose/ to <NEW\_NETWORK> directory.
3. put files required by your network at example/NEW\_NETWORK/files - including the HEF file (current files can be removed)
4. Update the pre-processing function if you network require one, at centerpose\_preprocessing.py (You should change the name).
5. Update the post-processing and drawing function if you network require one, at centerpose\_postprocessing.py (You should change the name).
6. Open example/<NEW\_NETWORK>/network.py in a text editor (e.g. vim) and follow the inline instructions.

Once done, we would need to open the app.py file and make some changes.

Open example/NEW\_NETWORK/app.py read the function's documetation and adjust the code for your purposes. You are all done!

Run python example/run-app example/NEW\_NETWORK/app.py<sup>5</sup>

### 18.1 Known Q&A

1. Q - Where Can I put my new HEF file and what lines do I need to change in order to make it work?  
A - You can put your hef anywhere you want as long as the dir name is called "files". In order to point to this HEF, the "DATA\_DIR" constant in the network class in network.py should refer to the parent directory of "files" directory. For Example, if your hef is under "/home/user/apps/files/", set the data dir to "/home/user/apps/":

```
# file is network.py
class MyNetwork(Network):
    DATA_DIR="/home/user/apps"
    ...
```

Got ideas / remarks to make it better? Did something go wrong? Please let us know!

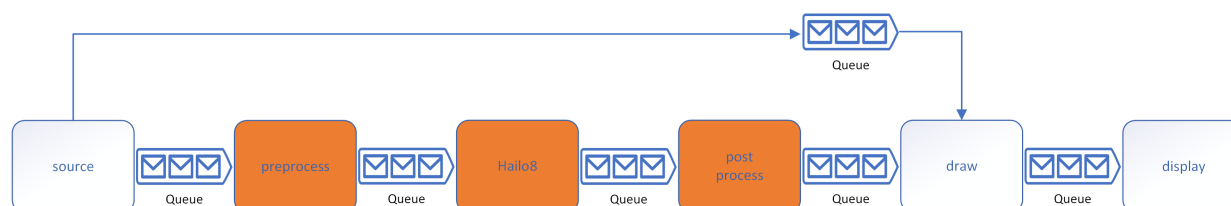
<sup>5</sup>You can run with -debug to get more data on the pipeline to finetune the performance.

## 19 Tools

### 19.1 Inference Acceleration and frame-rate control

As default single stream application will run at 30 frame per second. This will provide best user experience in terms of latency and showcasing "realtime application" power, memory, and host CPU usage. This work point can be overridden by the user using the `--source-fps` flag. In order to test the network high performance we are using the inference acceleration feature described below. We added an option to switch between "realtime mode" and "throughput mode" on the fly while the application is running. When this option is available a text on screen will prompt you can change mode by pressing 'a'. Not all applications and networks support this option of dynamic mode changing. Apps like multistream and tiling do not need this option because the chip is already working in high capacity. Some networks which their FPS is lower or close to 30fps will not gain from enabling this feature and will only suffer from worse latency, these network does not have this feature enabled. You can always force this feature using `--enable-acceleration` flag. The main goal of the inference acceleration tool is to test a network performance w/o taking into account "external limitations" like camera FPS or display FPS.

If for example your network can run @100 FPS but you have a camera which supports only 30 FPS you would still want to have the option to run the inference faster, for testing or demonstration purposes. The Inference Acceleration tool allows you to do just that. By duplicating frames we are able to demonstrate higher FPS. We do want to make sure that the FPS reached in this mode is feasible on your host this is why we include pre-process, inference, and post-processing in our "accelerated pipeline" see highlighted block in the figure below.



Note that video manipulation as video decoding, and resizing are assumed to be done in the source block. However, network related tasks as normalization are included in the accelerated pipeline.

The inference acceleration implementation is divided into two modules. `queue_start_acceleration` and `queue_end_acceleration`. These modules will handle the queue interface replacing the regular queue put and get functions. The start acceleration module will send the original frame as before and will try to send more duplicates of this frame according to the queue status. It will keep doing it until it is time to read a new frame from source. The duplicated frames' metadata will be marked in order to distinguish them from the originals. The end acceleration function will read all frames from queue and will dump the duplicated frames.

The pitfall in this method is that we might duplicate "too much" and add unwanted latency. To overcome this issue we added some knobs you can fine tune to 'manage' latency to performance balance. The inference acceleration functions are defined in `backbone/common.py` see inline comments for details.

### 19.2 Inter-Process Communication: Pipes

TAPPAS is a multiprocessing application, where separate processes (ie: preprocess, postprocess, etc...) communicate data (frames) with each other using queues. Because Python queues add overhead to serialize the frame, latency can increase as frame size increases. For especially large (and therefore slow!) frames, we have implemented a communication paradigm that passes the data between processes using `os.pipe` with our own encoding method that reduces such overhead. This means the frame can be communicated at a much faster rate.

You can enable pipes when creating queue instances through the `enable_pipes` parameter:

```
example_queue = HailoAppsQueue(maxsize=3, name='example_queue', enable_pipe=True)
```

The key difference between using this paradigm and the normal queues is that queues can buffer multiple frames, whereas pipes are limited by the kernel buffer size limit. This means that if a frame is larger than the buffer size (typically 1MB), the number of frames that can be buffered is effectively 1. It is therefore recommended to use pipes only between blocks that need to transmit large frames.

## 19.3 Extras

Extras are tools and functions which are not part of the TAPPAS core delivery. They are not tested and may require some additional dependencies. They are supplied as is for you to enjoy. Extras are available using the 'extras' sub group in hailo CLI.

### 19.3.1 Plailo

Plailo was developed in an internal Hackathon in Hailo and allows to use CenterPose outputs to emulate keystrokes. It requires to pip install pyautogui for keyboard emulation. It uses the player location to control the arrows and rising your right hand up will send an 'Enter'. This code is very simple and using only one player and not all available data. It is kept simple in order to allow simple hacking and tinkering. To try it open a game like [Temple-Run](#). To run the app use:

```
hailo run-app centerpose extras --enable-plailo
```

### 19.3.2 ZeroMQ

[ZeroMQ](#), or ZMQ in short, is a cross platform open-source universal messaging library. It allows to send data between different programs running on the same machine or a remote machine. In the example we showcase here we are using zmq to send centerpose outputs. It was used in an internal Hackathon in Hailo to send data between hosts and which were running different OS and code written in different languages. This allows for example to simply connect Hailo pipeline to external programs like Unity for example.

The class handling zmq can be found under apps/apps\_tools/zmq\_wrapper.py this file when running as `__main__` will start a zmq subscriber to test the connection.

To run this example use:

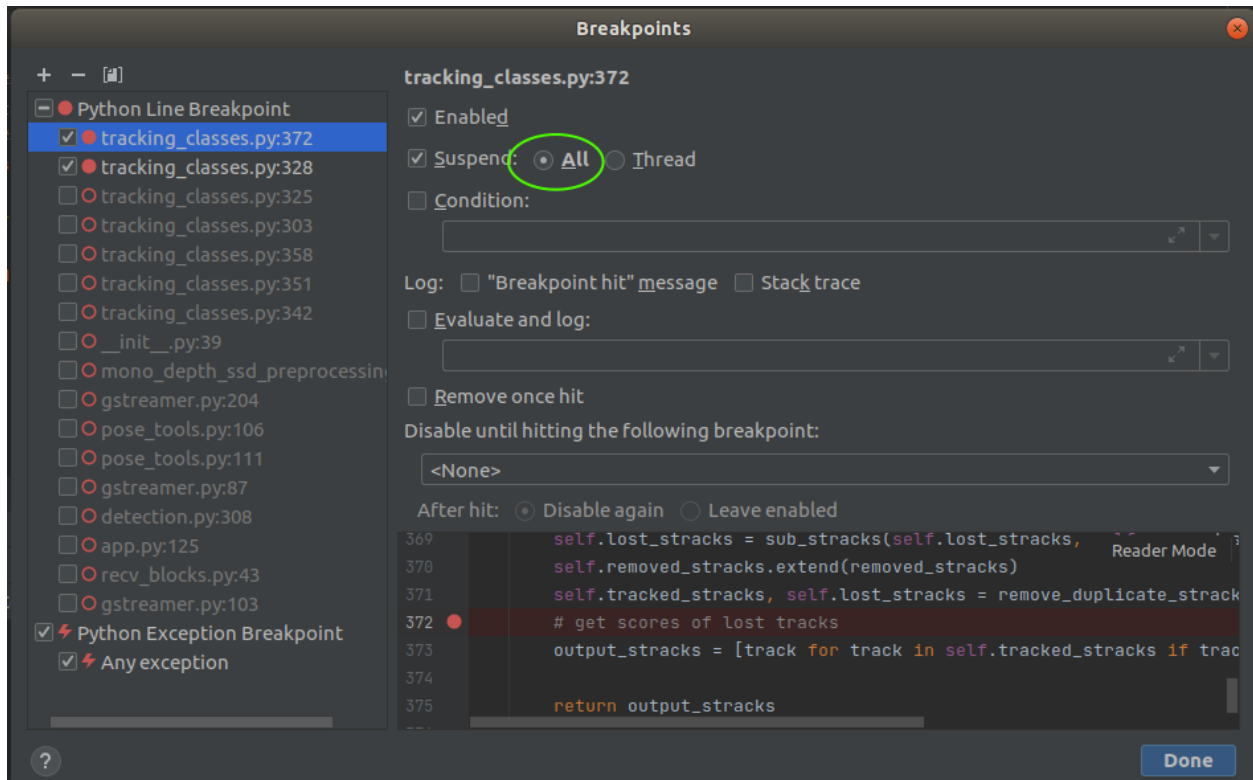
```
# to start a subscriber
python apps/app_tools/zmq_wrapper.py
# on another terminal run the demo
hailo run-app centerpose extras --enable-zmq
# to also get ID per skeleton enable the tracking option
hailo run-app centerpose --enable-tracking extras --enable-zmq
```

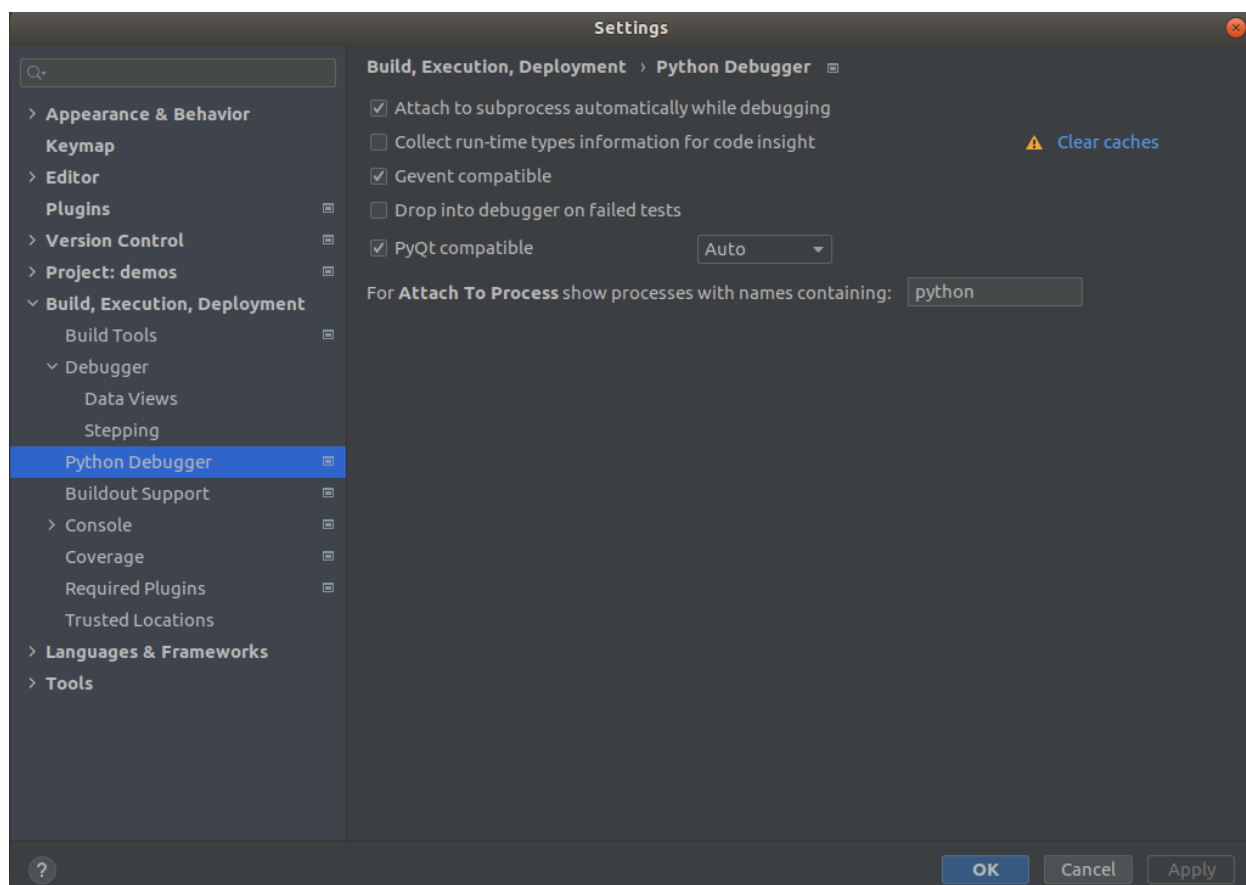
## 20 Debug and Troubleshooting

### 20.1 Breakpoints

To work with break-points we suggest to use pycharm debug utility. Using pdb in multiprocessing code is not stable. To configure your pycharm to work with breakpoints in a multiprocess environment set suspend to "all" in the breakpoints window. This will pause all processes and not only the one which hit the break-point.

In addition mark the python debugger settings to be Gevent Compatible.





## 20.2 Profiling

So, you got an awesome app that works great, but it runs slowly. For your optimizing pleasure, we added some profiling tools that you can use to better find and fix bottlenecks in the pipeline:

### 20.2.1 Inline Profiling

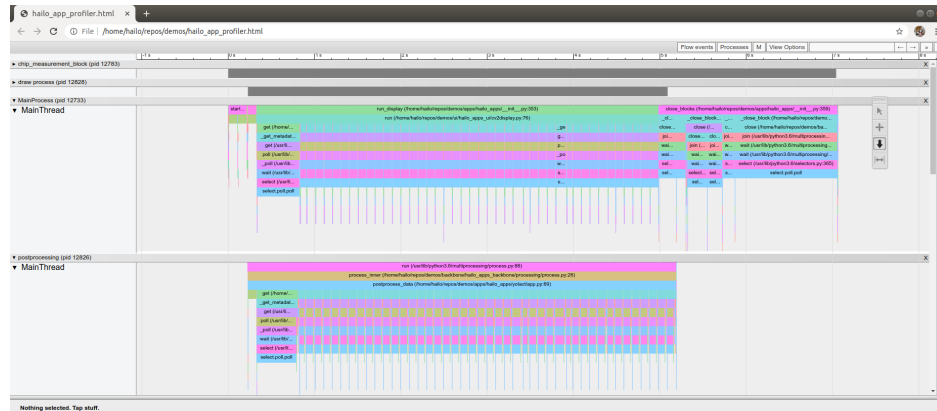
Adding prints every few lines with timestamps is a nightmare, didn't they build a tool for that?

Yes, they did... We wrapped cProfile tool for your leisure. Don't care about internals just want to know which part take the most time?

```
from hailo_apps_backbone.profiling import profiling
# get instance
pr = profiling()
pr.start() # start logging
#your code goes here
pr.display() # outputs log to screen
For some more options use:
pr = profiling(aggregate=1, sortby='cumulative', display_top_percent=0.2)
# aggregate: aggregates data over several rounds
# display_top_percent changes how much data you see
# Sortby - Check cProfile help (this default is probably OK for you)
```



## 20.2.2 Profiler Reports



If you are unsure where exactly to expect a bottleneck, and want a way to inspect the pipeline in a more graphical orientation, then profiler reports are exactly what you need! We enabled interactive profiler reports that you can open after running an application see a detailed visualization of function hierarchy on a timeline. The profiler is a low-overhead tracing tool that runs in the background of the application and saves the report to an HTML file that you can view in your web browser. To enable the profiler simply run with '-profile' from the command line:

```
hailo run-app <any_app> --profile
```

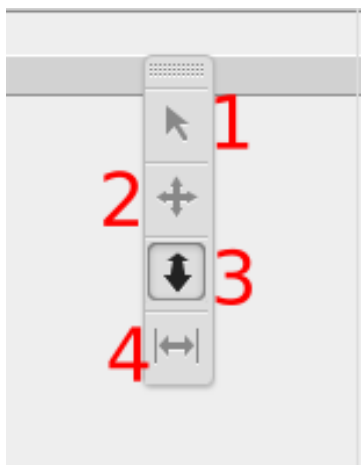
After closing the app, you will see that a report is saved:

```
Saving report to /home/hailo/repos/demos/hailo_app_profiler.html ...
Dumping trace data to json, total entries: 138006, estimated json file size: 15.8MiB
Generating HTML report
Report saved.
[hailo_apps] [warning] Profiler report saved to hailo_app_profiler.html
[info/MainProcess] process shutting down
(demo_virtualenv) hailo@hlo-demo153:~/repos/demos$
```

Double click on the file to open it, it should automatically open in your default web browser. A lot of data is presented, so the first time seeing the profiler report can be overwhelming, here are parts you should know:

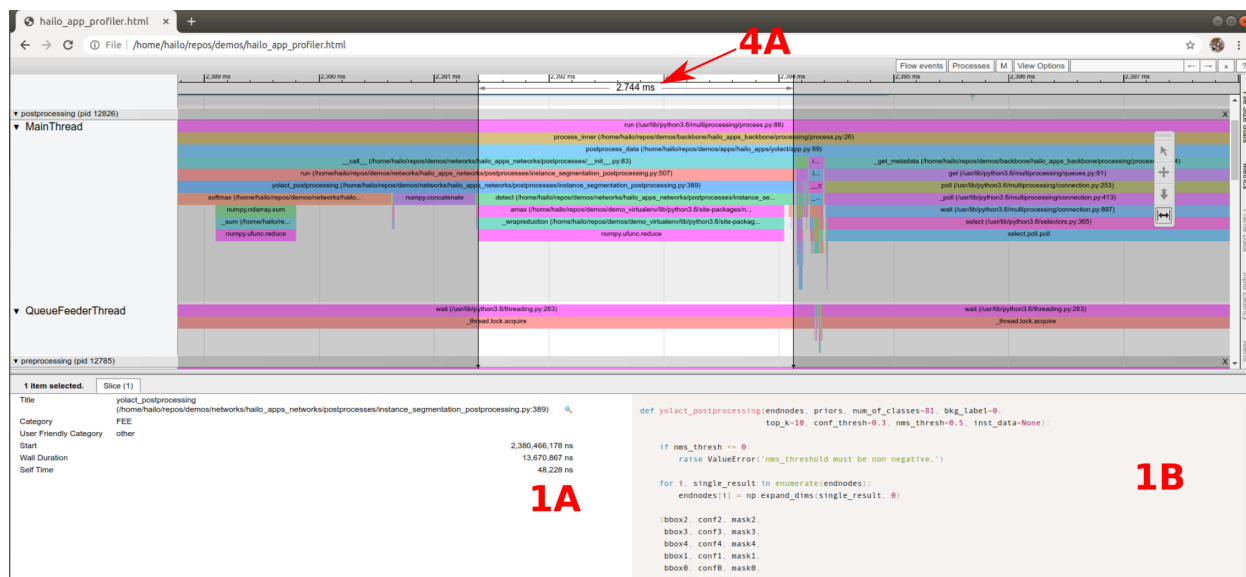


1. On the left you will see that **each process has its own tab** named for the process name and pid. Within each tab there are expandable arrows for each thread of the process. You can minimize processes you are uninterested in looking at by clicking on the arrows.
2. In box two you can see a **graph that visualizes the code execution**. Each function appears as a block, with functions that it calls appearing below it. The functions are displayed on a timeline so that you can see their execution **in the order (and duration)** that they had at runtime. The functions also list the full path to the file they are implemented in along with the line number.
3. In box three are **four tools** that you can use to interact with the profiler report. Lets take a closer look:



- (1) **Select:** Using this tool you can click on a function to see more details about it at the bottom of the page. In the bottom left (1A in Figure 17.2.2 below) you will see details including Wall Duration (time the function ran for) and to the bottom right (1B) you will see the source code of the function.
- (2) **Pan:** Using this tool you can click and drag to pan around the graph.
- (3) **Zoom:** By default the report page opens at a zoom level that fits the whole runtime into the page. You can use this tool to adjust the zoom. When it is selected, click anywhere on the graph and drag up to zoom in, or drag down to zoom out.
- (4) **Timing:** This tool lets you time selected areas. When it is selected you can click and drag to select an area to time. The duration will be written at the top of the page (4A in Figure 17.2.2 below).

4. In addition to the tools listed above, you can also **navigate using the WASD keys** on your keyboard. The A and D keys are used to navigate left and right, while the W and S keys are used to zoom in out (respectively).



With this profiling tool you should be able to visually see bottlenecks by seeing which functions take the longest (they will be wider), time them using the timing tool, and look at their source code/inner functions with the select tool. Good luck and happy hunting. This feature is based on [Viztracer](#) check it out for more details.

## 20.3 Queues and Blocks Status

This feature is already integrated to demo apps pipeline. **Use --debug to enable it when running from the command line.** This feature lets you know statistics on the queue fullness and how long the blocks had to wait in order to put / get data to / from the queue. When the 'put wait' time is long it means we got back pressure from next block. When the 'get wait' time is long it means we got a bottle neck before the queue. This debug tool will also display the average frame rate of each block.

The function is included in your app template so you can run it like this:

```
app.display_latency_status(metadata)
```

| queue                  | fullness | put wait[ms] | get wait[ms] |
|------------------------|----------|--------------|--------------|
| preprocess             | 1        | 0.02         | 3.86         |
| video_bypass           | 5        | 0.89         | 0.61         |
| infer                  | 3        | 0.03         | 0.91         |
| postprocess            | 1        | 0.03         | 2.96         |
| debug                  | 0        | 0.05         | 4.63         |
| chip_measurement_queue | 0        | 0.02         | 0.01         |
| hailo_metadata_queue_0 | 0        | 0.06         | 0.13         |

| Block      | preprocessing | send process | recv process | postprocess process |
|------------|---------------|--------------|--------------|---------------------|
| Frame rate | 322.69        | 1733.22      | 163.29       | 327.64              |

Note that the wait time are running average and the fullness is current state.

## 20.4 Pipeline Timestamps

This feature is already integrated to demo apps pipeline. **Use --debug to enable it when running from the command line.** This feature adds timestamps to the packet metadata. So, its 'passport' gets stamped on several points in the pipeline. It is done by default on every queue put / get and users can add more points in the code. To use it use the Metadata class to prepare your initial metadata. (This should be done anyway... see basic\_app for documented example) To extract the debug data from the metadata object use:

```
from hailo_apps_backbone.common import MetadataParser
md_parser = MetadataParser() # get instance
# somewhere in the pipeline... preferably at the end of it.
# run with the current metadata:
md_parser.metadata_parse_timestamp(metadata)
# this will display data up to the point in the pipe you ran the parser...
# To add a timestamp somewhere in the pipeline use this code:
from hailo_apps_backbone.common import metadata_update_timestamp
# some code
metadata_update_timestamp(metadata, tag='interesting')
# yet more code
```

| Latency[ms] | G-init | P-preproce | G-preproce | P-metadata | G-metadata | P-infer | G-infer | P-debug | G-debug | total |
|-------------|--------|------------|------------|------------|------------|---------|---------|---------|---------|-------|
|             | 0.00   | 4.47       | 2.65       | 0.64       | 3.71       | 4.39    | 10.69   | 3.26    | 0.17    | 29.98 |

The sample tags are in the first row. Latency is the difference between 2 tags, therefore its indented to be between 2 tags.

The last column in the latency row is total latency from the beginning of the pipeline.

## 20.5 Common Pitfalls

- Use `hailo fw-control identify` to make sure you are connected to board.
- Make sure you have access to the camera - `sudo chown <user> /dev/video*`
- You can check for kernel messages related for hailo board using - `dmesg | grep hailo`
- If PCIe driver fails to install, check that you have a valid Linux kernel supported by the platform release.
- There is a known issue with power measurements not ending correctly on FW. If you get this error:

```
hailo_platform.drivers.hailort._pyhailort.HailoRTStatusException: 14}
```

You can reset the chip using these commands:

```
hailo fw-control reset --reset-type soft
sudo modprobe -rq hailo_pci
sudo modprobe hailo_pci
```

## 21 Version Changelog

### 21.1 Version 3.11.0

- Apps Changes
  - **Deprecation Notice:** Moving to Gstreamer based framework. Old python based pipelines will be deprecated.
  - New Gstreamer based applications
    - \* `gst_detection` app. Support YOLOv5s/m, YOLOv4, YOLOv3
    - \* `gst_multi_stream` app. Support YOLOv5s/m, YOLOv4, YOLOv3
    - \* `gst_segmentation` app. Support `fcn8_resnet_v1_18`
    - \* `gst_pose_estimation` app.
  - deprecated apps:
    - \* `basic_app`
    - \* `mobiledet` support.
    - \* Old Gstreamer apps (`gstreamer_app` and `gst_hailo`)
- Core
  - Aligned all networks names to Model Zoo.
  - New Gstreamer App basic class.

### 21.2 Version 3.9.0

- Apps Changes
  - New YOLOv3 network with 416x416 resolution.
  - [Multi stream app](#).
    - \* Added support for YOLOv3 (608 and 416) and YOLOv4.
    - \* Now supporting up to 16 streams.
    - \* Mobilenet Multistream app name changed to multi stream.
  - Fixed bug in YOLO post processing which caused some small detections to be dumped.
  - [smoke\\_regnet800](#) app speedup
- New Gstreamer based application. This is a preview, and the start of our transition to Gstreamer framework. See GstHailo section for details.
- Core
  - Added handler for high temperature warning. When threshold is passed app will be shut down.
  - Fixed bug in device selection (`--use-device` flag)

### 21.3 Version 3.8.0

- Apps Changes
  - New 3D Object Detection network [smoke\\_regnet800](#)
  - New yolov4 network. Find it under Detection App [yolov4](#).

- New mobilenetSSD HEF, improved frame rate.
- monodepth\_ssd new HEF improved frame rate.
- yolov5m new HEF updated accuracy.
- Added face detection to [tiling](#).
- Polylanenet updated HEF, improved accuracy.
- yolov3 updated HEF, improved frame rate.
- [Semantic segmentation app](#) is replacing resnet18\_fcn16.
- Segmentation network changed from FCN16 to FCN8. Using 19 classes instead of 8.
- Fixed BGR/RGB issue in Retinaface and arcface age-gender.
- Updated multi-stream default to 15 streams.
- Core
  - FPS control updates see [Inference Acceleration and FPS control](#) section.
    - \* Default FPS is now 30 FPS
    - \* Added option to enable acceleration online in supported networks / apps.
    - \* FPS averaging is moved to "time based" batches instead of "frame based".
    - \* Power monitoring scheme changed to support changing power levels.
  - Face detection output aligned to detection standard.
- Tools
  - New "extras" features added. see [Extras](#) section for details.
    - \* Added ZMQ postprocess block to Centerpose.
    - \* Added Plailo postprocess, Keyboard events from centerpose.

## 21.4 Version 3.7.0

- Apps Changes
  - Face detection app
    - \* New face detection app ([8.1](#)) replacing old retinaface app.
    - \* New face detection network lightface slim. A low resolution but high FPS and low footprint network. Making it perfect to be added as an additional task for multiple task applications. Run using face detection app ([8.1](#)).
  - Yolov5 postprocessing optimization.
  - Fixed centerpose tracking.
  - updated monodepth2 and monodepthssd visualization.
- Core
  - Deprecated all JLF support.
  - Added option to change source FPS from CLI use --source-fps see [CLI options](#)
  - updated history tracker aging to be frames based (instead of time).
- Debug



- updated logger messages and severity.
- Updated breakpoint documentation. Suggestion is to use PyCharm see [Breakpoints](#).
- Profiling tool fixed.
- Install
  - Removed gst-plugins-ugly from prerequisites.

## 21.5 Version 3.6.0

- Known issue: –profile flag might cause the app to get stuck.

## 21.6 Version 3.5.0

- New OS pipe implementation for queues, to be used when a very large payload is required.
- New Add your own network / App user guide and example code.
- Debug block is now a pre-made block. Simplified integration to apps.
- New apps/networks:
  - New app MonoDepthSSD ([15.1](#)) merging 2 tasks on single chip. MonoDepth2 and MobilenetSSD300.
  - New Classification App ([17.1](#)).
    - \* This app replaces the old Resnet50 App.
    - \* Resnet50 - merged the softmax on chip
  - Fair-MOT - merged centernet NMS and Normalizatoin on chip
  - YOLOv5s - merged preprocessing space-to-depth on chip
  - YOLOv5m - merged preprocessing space-to-depth on chip
  - Resnet18-fcn16 - Changed number of classes to 8.
  - Added draw and tracking function from Openpose (deprecated) to CenterPose

## 21.7 Version 3.4.0

- Moved to new network binaries format HEF instead of JLF5.
- Added new Profiling tool (See Profiler section).
- New apps/networks:
  - YoloV5m - added to detection app.
  - New Gstreamer based App.
    - \* Still in BETA.
    - \* Supports all detection networks.
    - \* Video pipeline is moved to Gstreamer framework.
    - \* Allows for better performance especially on weak hosts.
- Removed apps/networks:
  - Openpose was deprecated. Centerpose is the pose estimation leading vehicle.

- Some features which were available in Openpose were not migrated yet, will be done next release.
- networks removed from basic app - Openpose.
- Refactored App closing sequence.
  - Now closing sequence is much faster.
  - Added timeout to Hailo queues (still blocking APP but sampling control state).
  - Processes and threads should end gracefully.
- Performance optimization of current networks. Major optimization of yolov5s/m postprocessing performance.
- Added support to define multiple inputs in MultiStream app.

## 21.8 Version 3.3.0

- Enabled power and temperature measurements for all applications.
- New apps/networks:
  - YoloV5s - added to detection app.
  - MonoDepth2 – new application for monocular depth estimation in ADAS.
  - Mobildet- added new network to tiling app.
  - Yolact-RegenetX-800MG - added to instance segmentation app and basic app.
- Removed apps/networks:
  - Yolact-MobileNetV1 - Replaced by better performing Yolact-RegenetX-800MG.
  - networks removed from basic app - mobilenet-ssd, centerNet-Resnet18
- New tool: Added Inference Acceleration tool.
- Changed the DB usage of facenet app. See [7.1](#) for details.
- bug fix: tracking mismatch in face recognition application caused poor quality.
- bug fix: Resnet50 application working on only a subset of images.

## 21.9 Version 3.2.0

- Centernet app renamed to detection app.
- Added sliders to detection app to fine tune tracking.
- New App: Added CenterNet with ResNet18 backbone.
- New App: Added support to multi scale object detection in tiling app (COCO flavor)
- Centerpose added on chip postprocessing.
- Added auto complete to hailo CLI tool.
- Fixed basic\_app CenterNet-ResNet50 labels.
- Fixed JLFs for FaceNet application.
- Fixed PolyLaneNet app crush when input video has no lanes.
- Updated recognition threshold for FaceNet application.

- Added Gstreamer dependencies for framework upgrade.

## **21.10 Version 3.1.0**

- Main I/F is changed to PCIe. (UDP is not yet supported).
- New networks added:
  - PolyLaneNet - Lane detection network
  - FAIR MOT – Multiple objects tracking network.
  - CenterPose - New pose estimation network.



## 22 Movies Licensing

| Video Name          | Source  | License   |
|---------------------|---|---|
| Tiling Visdrone     | <a href="https://www.videvo.net/video/cars-crossing-the-river-tiber/456696/">https://www.videvo.net/video/cars-crossing-the-river-tiber/456696/</a>   | Credit the author is required. "Stock footage provided by Videvo, downloaded from www.videvo.net"   |
| Tiling COCO         | <a href="https://www.istockphoto.com/video/shibuya-crossing-in-tokyo-japan-gm1067289398-285413523">https://www.istockphoto.com/video/shibuya-crossing-in-tokyo-japan-gm1067289398-285413523</a>   | Licensed by iStock.com, Customer may not use content in any way that allows others to download, extract, or redistribute content as a standalone file (meaning just the content file itself, separate from the project or end use). |
| Tiling lightface    | <a href="https://www.pexels.com/video/a-speaker-talking-to-the-crowd-in-a-school-meeting-5644125/">https://www.pexels.com/video/a-speaker-talking-to-the-crowd-in-a-school-meeting-5644125/</a>   | Free  |
| ResNet18_FCN16      | Taken by Hailo  | Free  |
| Polylanenet         | <a href="https://www.istockphoto.com/video/denmark-xiv-synched-series-front-driving-studioprocess-plate-background-gm1041959252-278963925">https://www.istockphoto.com/video/denmark-xiv-synched-series-front-driving-studioprocess-plate-background-gm1041959252-278963925</a> and <a href="https://www.istockphoto.com/video/driving-in-new-york-city-gm531954524-94296643">https://www.istockphoto.com/video/driving-in-new-york-city-gm531954524-94296643</a> | Licensed by iStock.com, Customer may not use content in any way that allows others to download, extract, or redistribute content as a standalone file (meaning just the content file itself, separate from the project or end use). |
| FAIR MOT RegNet800m | <a href="https://www.istockphoto.com/video/people-crossing-day-time-gm1141803334-305983551">https://www.istockphoto.com/video/people-crossing-day-time-gm1141803334-305983551</a>   | Licensed by iStock.com, Customer may not use content in any way that allows others to download, extract, or redistribute content as a standalone file (meaning just the content file itself, separate from the project or end use). |
| Mutli stream app    | <a href="https://www.pexels.com/video/biking-by-the-sea-bay-4787319, airplane-take-off-855130, motorbikes-stunt-show-857129, rainy-weather-video-854204">https://www.pexels.com/video/biking-by-the-sea-bay-4787319, airplane-take-off-855130, motorbikes-stunt-show-857129, rainy-weather-video-854204</a>   | Free  |
| Mutli stream app    | iStock-531954524,iStock-1067289398  | Licensed by iStock.com, Customer may not use content in any way that allows others to download, extract, or redistribute content as a standalone file (meaning just the content file itself, separate from the project or end use)  |
| Mono Depth 2        | <a href="https://www.istockphoto.com/video/driving-in-new-york-city-gm531954524-94296643">https://www.istockphoto.com/video/driving-in-new-york-city-gm531954524-94296643</a>   | Licensed by iStock.com, Customer may not use content in any way that allows others to download, extract, or redistribute content as a standalone file (meaning just the content file itself, separate from the project or end use)  |
| Page 60             | V3.11.0   | Confidential and Proprietary   Copyright © 2021 Hailo Technologies Ltd.   |