DepthAI Docs

Luxonis

May 02, 2024

CONTENTS

DepthAI is a *Spatial AI* **platform** that is used for communication with and development of our devices; OAK cameras and RAE robots.

It allows you to develop projects and products that require:

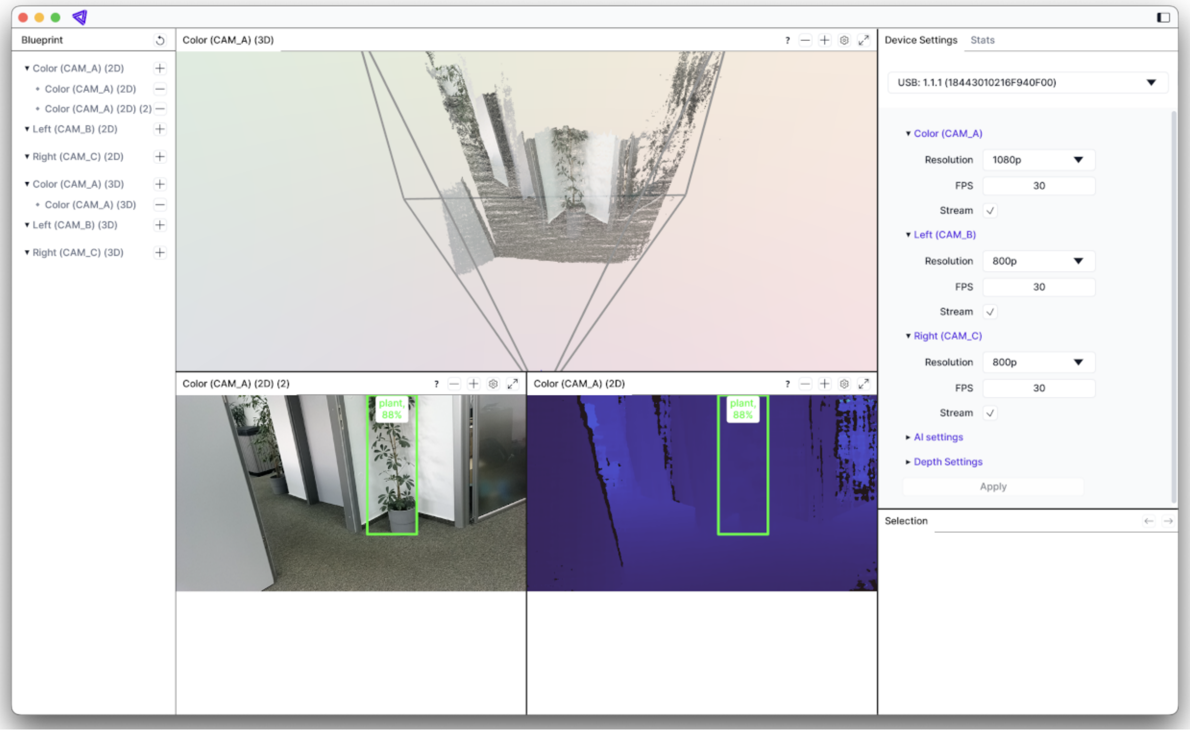
1. *Artificial Intelligence*
2. *Computer Vision*
3. *Depth perception* (Stereo, ToF)
4. Performant (high resolution and FPS, multiple sensors)
5. Embedded, low power solution

Best of all, it is modular and you can **integrate** this technology into your products.

CHAPTER

ONE

DEPTHAI VIEWER



USB: 1.1.1 (18443010216F940F00)

Color (CAM_A)
Resolution 1080p
FPS 30
Stream ✓

Left (CAM_B)
Resolution 800p
FPS 30
Stream ✓

Right (CAM_C)
Resolution 800p
FPS 30
Stream ✓

DepthAI Viewer is the visualization tool for DepthAI and OAK cameras. It's a GUI application that will run a demo app by default, which will visualize all streams and run inference on the device. It also allows you to change the configuration of the device. DepthAI viewer works for USB and POE cameras.

To install and run the DepthAI Viewer, run the following commands in the terminal:

```
python3 -m pip install depthai-viewer
# Run the DepthAI Viewer
python3 -m depthai_viewer
```

We have also prepared a **step by step guide** *here* with detailed instructions how to set up your DepthAI and run this script.

CHAPTER

TWO

EXAMPLE USE CASES

In this section, you’ll find an inspiration what can you build right away with DepthAI.

CHAPTER

THREE

TOOLS & API EXAMPLES

In this section, you'll see examples of various API usage permutations, to show what the API is capable of or to solve some meta problem, like how to stream the data, how to collect it and alike.

OCR	This pipeline implements text detection (EAST) followed by optical character recognition of the detected text
Multiple Devices	This example shows how you can use multiple DepthAI's on a single host. The demo will find all devices connected to the host and display an RGB preview from each of them
Face recognition	Detects all faces in the frame, gets face feature vectors and compares it with database to perform face recognition
Message Syncing	This example shows how to sync messages (eg. NN results with frames) with software, based on either timestamps or sequence numbers
License plate recognition	Detects license plates and performs license plate recognition operation on the camera itself
WLS Filtering	This example demonstrates how to do host-side WLS filtering using the rectified_right and depth stream from DepthAI API
QR code scanner	QR Code detection model running on the device combined with on-host QR code decoder
Deploy Roboflow models	Deploy over 10,000 pre-trained AI models from Roboflow Universe and your own Roboflow custom models

CHAPTER

FOUR

ECOSYSTEM

Table 1: Core Repositories

depthai-python	Here you'll find Python bindings creating the Python API of DepthAI
depthai-core	Our core API written in C++
depthai-ros	DepthAI ROS Wrapper. This is an attempt at basic DepthAI to ROS2 interface. It's largely leveraging the existing depthai-python examples.
depthai-unity	DepthAI Unity Wrapper projects and examples. Useful for synthetic dataset generation.
depthai-hardware	This repository contains Luxonis open sourced baseboards, and contains Altium design files, documentation, and pictures to help you understand more about the embedded hardware that powers DepthAI.
depthai-ml-training	Here you can find repositories to help you connect your NN and create BLOBs.

Table 2: Demo Repositories

depthai-experiments	In this repository, you'll find various experiments using DepthAI. You can use those examples as a basis or a reference in your application.
depthai	This repo contains a demo application, which can load different networks, create pipelines, record video, etc. This program includes an example of depth & CNN inference and ready to use models.
depthai-core-example	CMake example project which serves as a template on how to quickly get started with C++ and depthai library
depthai-tutorials	This repo contains source code for tutorials published on docs.luxonis.com.
depthai-docker	This repository contains a Dockerfile, that allows you to run OpenVINO on DepthAI inside a Docker container.

4.1 First steps with DepthAI

This guide will go through the first steps with **OAK camera** and **DepthAI library**:

1. Installing DepthAI
2. Device setup - connecting the OAK camera to your (host) computer
3. Running *DepthAI Viewer*, the visualization GUI app for DepthAI
4. Next steps; examples, demos, API docs

4.1.1 Installing DepthAI

Follow instructions below to install DepthAI and its dependencies/requirements with an installer.

macOS

Windows

Linux

Execute the script below to install DepthAI on macOS:

```
curl -fL https://docs.luxonis.com/install_dependencies.sh | bash
```

Please refer to this documentation if any issues occur.

Windows 10/11 users can **install DepthAI** with the Windows Installer.

Installer will install either the **newer** DepthAI Viewer (visualization GUI application), or DepthAI Demo (python script, older GUI application) or and all the dependencies. **We suggest using** the DepthAI Viewer.

After the installer finishes, you can directly run the DepthAI app from the list of applications, which will run the installed demo. You can skip Setup section (as Installer performs the whole setup) of this tutorial and go directly to *DepthAI Viewer*.

Execute the script below to install DepthAI on Linux systems:

```
sudo wget -qO- https://docs.luxonis.com/install_depthai.sh | bash
```

Please refer to Installation documentation if any issues occur.

If you would like to avoid using installer and would prefer manually installing dependencies, requirements and DepthAI, see Manual DepthAI installation.

4.1.2 Device setup

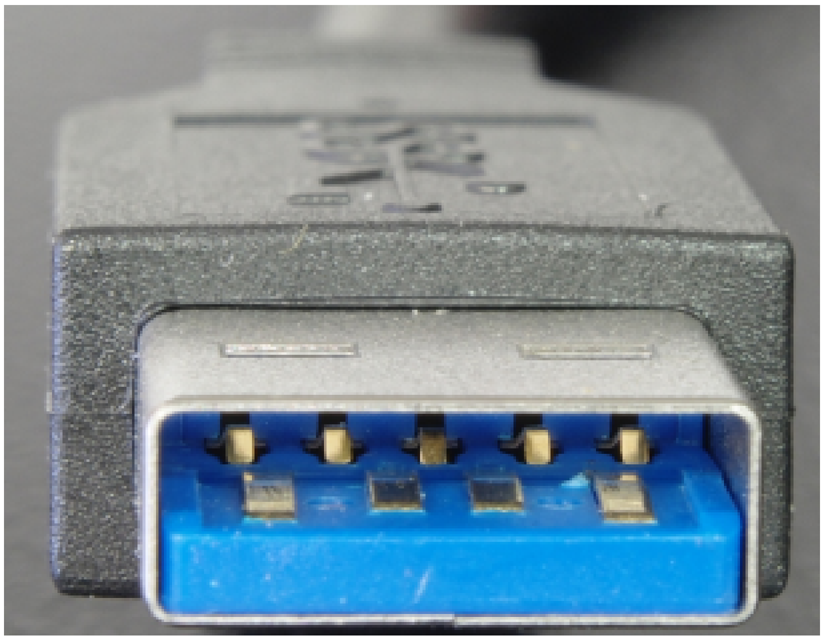
Now that we have installed requirements, we can setup the device. OAK cameras can be separated into two categories depending on how you connect to them; either via ethernet (OAK PoE cameras) or via USB (all others).

OAK USB camera

OAK PoE camera

If your OAK came with an included USB cable, we suggest using that to connect the OAK camera to the host computer.

Warning: Make sure to use **USB3 cable**, as this is has been a very common culprit of OAK connectivity issues. If you aren't using USB3 cable, *force USB2 communication*.



USB3 cable is colored blue in the inside of the USB-A connector of the USB-C cable. If it's not blue, it might be USB2 charging cable.

Make sure that the device is connected to your host (which can be a PC or Raspberry Pi or another capable computer) directly to a USB3 port, or via a powered USB hub.

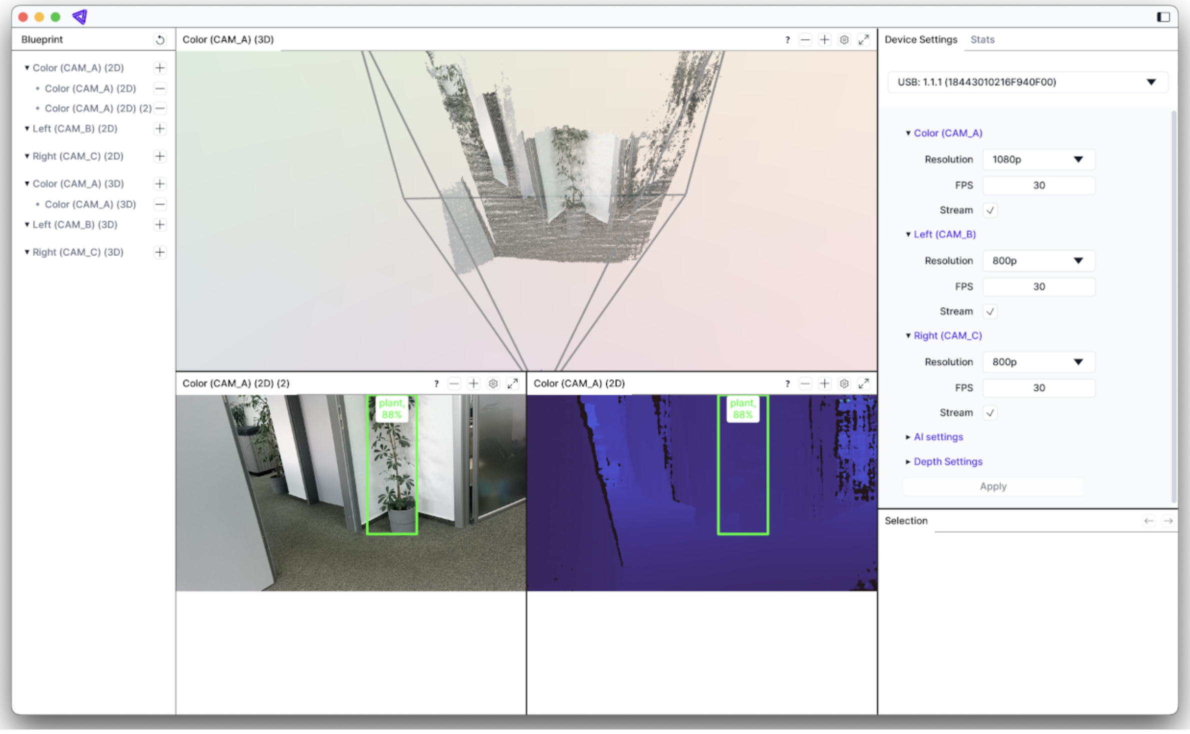
If you are using OAK PoE device, you will first need to connect the device to a PoE switch or a PoE injector. We recommend following the Getting started with OAK PoE devices for a step-by-step tutorial.

4.1.3 DepthAI Viewer

After the installer finishes, you can run the DepthAI Viewer by running:

```
depthai-viewer
# OR
python3 -m depthai_viewer
```

Running the Viewer for the first time, the app will download a default *mobilenet-ssd* model, configure the OAK camera and then show default streams from the camera.



USB: 1.1.1 (18443010216F940F00)

Color (CAM_A)
Resolution 1080p
FPS 30
Stream ✓

Left (CAM_B)
Resolution 800p
FPS 30
Stream ✓

Right (CAM_C)
Resolution 800p
FPS 30
Stream ✓

AI settings
Depth Settings
Apply

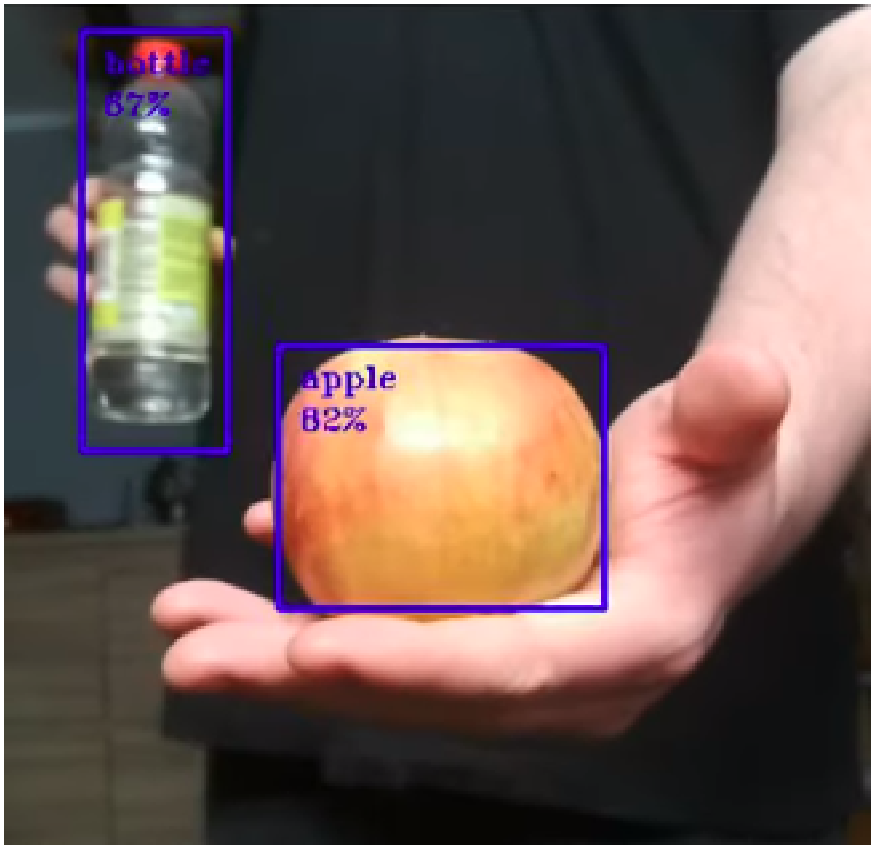
4.1.4 Default model

While the Viewer is running, you can see detection results, and if you are standing in front of the camera, you should see yourself detected as a person with a high probability.

The model that is used by default is a MobileNetv2 SSD object detector trained on the PASCAL 2007 VOC classes, which are:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: airplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, TV/monitor

So give it a try to detect different objects, like bottles or apples



4.1.5 Next steps

In the previous sections, we learned how to preview basic DepthAI features. From this point, you can explore the DepthAI world further

- **Usecases**

  Check our *Example Use Cases* for ready to use applications that solve a specific problem on DepthAI

- **Getting started** with coding

  Be sure to check hello world tutorial on API section for a step-by-step introduction to the API

- **Train and deploy a custom model** to OAK

  Visit *Custom training* page for ready to use Colab notebooks

- Already **built apps** for OAK devices

  See luxonis/depthai-experiments repository for apps built with depthai library

- Depthai **API library** repository

  See luxonis/depthai-python repository which contains Python bindings for the depthai API library, Code samples and various utility programs.

4.2 Spatial AI

Spatial AI allows robots or computers to perceive the world as a human can - what objects or features are - and where they are in the physical world. DepthAI platform **leverages Spatial AI by fusing** *AI capabilities* with *depth perception* on the OAK camera itself.

There are a few different approaches to achieve AI + depth fusion:

1. *Neural inference fused with depth map*
2. *Semantic depth*
3. *Stereo neural inference*

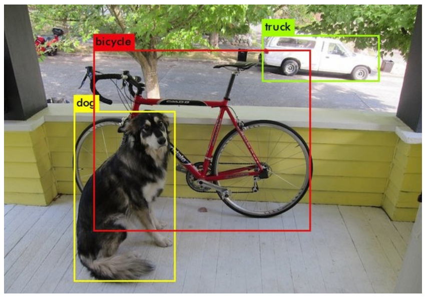
4.2.1 1. Neural inference fused with depth map

DepthAI can fuse neural inference (object/landmark detection) results with a depth map to estimate spatial coordinates (XYZ) of all objects/landmarks in the scene.

This technique is excellent for existing (pre-trained) 2D object/feature detectors as it runs inference on color/mono frames, and uses resulting bounding boxes to determine ROIs (regions-of-interest). DepthAI then averages depth from depth map inside these ROIs and calculates spatial coordinates from that (calculation here).

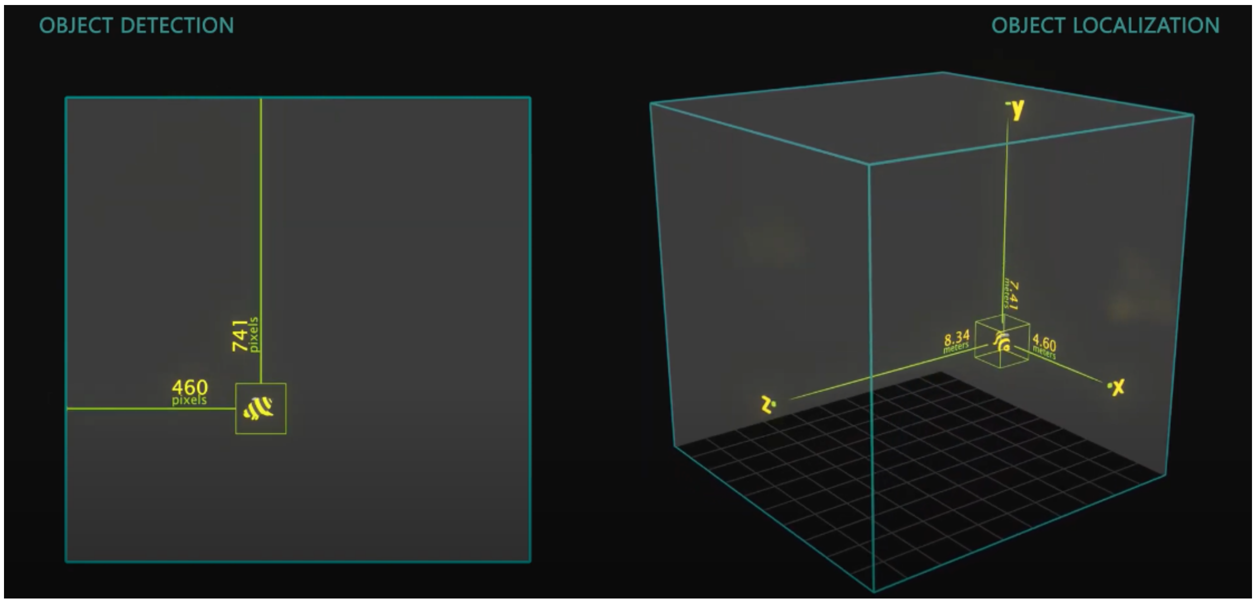
3D Object Localization

First, let us define what 'Object Detection' is. It is the technical term for finding the bounding box of an object of interest, in an image's pixel space (i.e., pixel coordinates),.

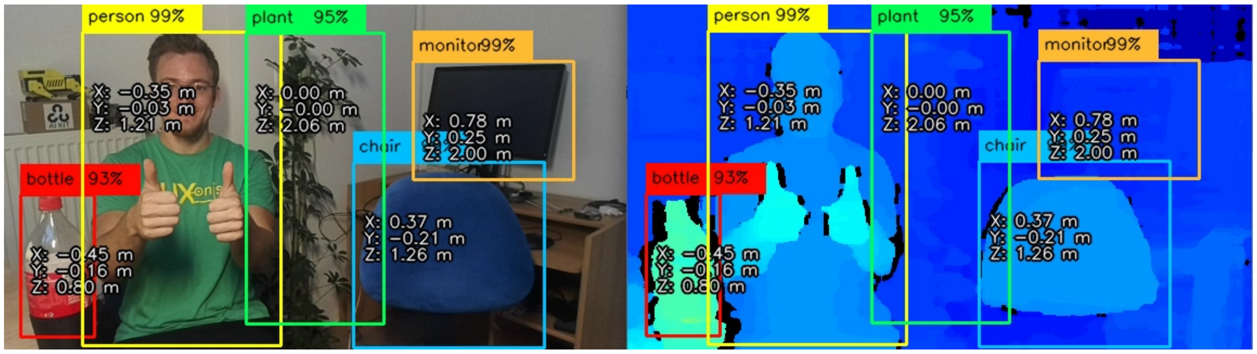


3D Object Localization (or 3D Object Detection) is all about finding objects in physical space instead of pixel space. It is useful when measuring or interacting with the physical world in real-time.

Below is a visualization to showcase the difference between Object Detection and 3D Object Localization:



DepthAI extends these 2D neural networks (eg. MobileNet, Yolo) with spatial information to give them 3D context.



On the image above, a depthai application runs MobileNet object detector and fuses object detections with a depth map to provide spatial coordinates (XYZ) of objects it sees: person, potted plant, bottle, and chair.

3D Landmark Localization

An example would be a hand landmark detector on DepthAI. With a regular camera, this network returns the 2D (XY) coordinates of all 21 hand landmarks (contours of all joints in fingers). Using this same network with DepthAI, these 21 hand landmarks are now 3D points in physical space instead of 2D points in pixel space

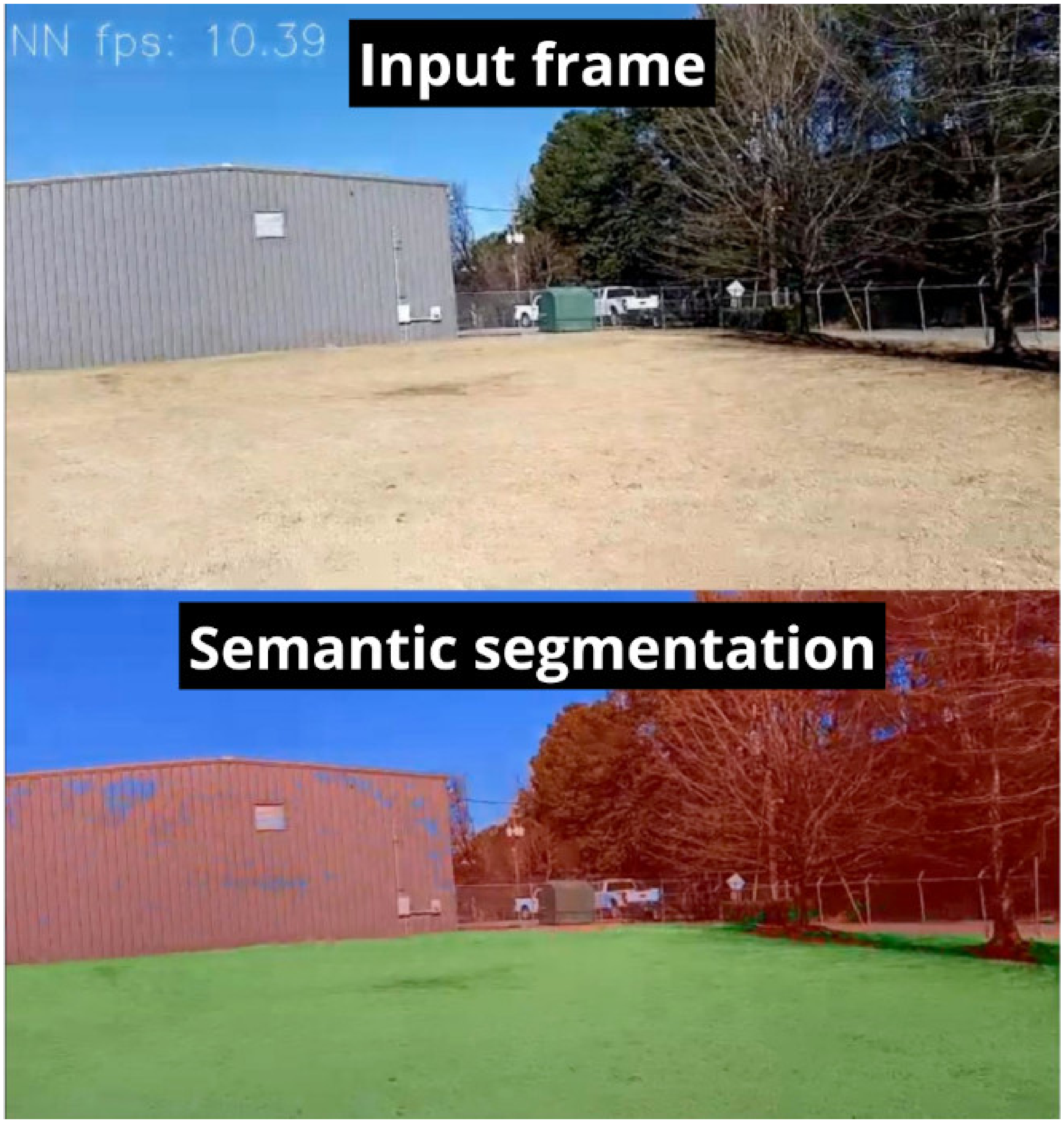
Demos: hand landmark (above), human pose landmark, and facial landmark detection demos.

4.2.2 2. Semantic depth

One of the classic problems in autonomous robotic navigation/actuation are **unknown objects**. Known objects are specified before the installation to be encountered - such as tools, other machines, workers, equipment, and facilities.

We cannot anticipate unknown objects - including those unknowable or never-before-seen. Training an object detector is sufficient for known objects as this is a "positive" form of object detection: "Pile in the path, stop." "Shovel in the path, stop." etc.

Such generic obstacle avoidance scenarios require a "negative" object detection system, and a very effective technique is to use **semantic segmentation** of RGB, Depth, or RGB+Depth.



The image above was taken from Greenzie's robotic lawnmowers (from OpenCV weekly livestream).

In such a "negative" system, the semantic segmentation system is trained on all the surfaces that are not objects. So anything that is not that surface is considered an object - allowing the navigation to know its location and take commensurate action (stop, go around, turn around, etc.). So the semantic depth is extremely valuable for **object avoidance** and **navigation planning** application.

On the image above, a person semantic segmentation model is running on RGB frames, and, based on the results, it crops depth maps only to include the person's depth.

4.2.3 3. Stereo neural inference

In this mode, the neural inference (landmark detection) is run on the left **and** right cameras to produce stereo inference results. Unlike monocular neural inference fused with stereo depth - there is no max disparity search limit - so the minimum distance is purely limited by the greater of (a) horizontal field of view (HFOV) of the stereo cameras themselves and (b) the hyperfocal distance of the cameras (minimal distance for objects to be in focus).

After we have 2D positions of landmarks from both left/right cameras, we can calculate the disparity of the results, which are then triangulated with the calibrated camera intrinsics to give the 3D position of all the detected features.

For more information, check out the Stereo neural inference demo.

Examples include finding the 3D locations of:

- Facial landmarks (eyes, ears, nose, edges of the mouth, etc.)
- Features on a product (screw holes, blemishes, etc.)
- Joints on a person (e.g., elbow, knees, hips, etc.)
- Features on a vehicle (e.g. mirrors, headlights, etc.)
- Pests or disease on a plant (i.e. features that are too small for object detection + stereo depth)

This mode does not require the neural networks to be trained with depth data. DepthAI takes standard, off-the-shelf 2D networks (which are significantly more common) and uses this stereo inference to produce accurate 3D results.

4.3 AI / ML / NN

4.3.1 Converting model to MyriadX blob

Local OpenVINO Model Conversion

In this tutorial, you'll learn how to convert OpenVINO IR models into the format required to run on DepthAI, even on a low-powered Raspberry Pi. I'll introduce you to the OpenVINO toolset, the Open Model Zoo (where we'll download the face-detection-retail-0004 model), and show you how to generate the **.blob** file needed to run model inference on your DepthAI board.

Note: Besides local model conversion (which is more time-consuming), you can also use *Blobconverter web app* or *blobconverter package*.

Haven't heard of OpenVINO or the Open Model Zoo? I'll start with a quick introduction of why we need these tools.

What is OpenVINO?

Under-the-hood, DepthAI uses the Intel technology to perform high-speed model inference. However, you can't just dump your neural net into the chip and get high-performance for free. That's where OpenVINO comes in. OpenVINO is a free toolkit that converts a deep learning model into a format that runs on Intel Hardware. Once the model is converted, it's common to see Frames Per Second (FPS) improve by 25x or more. Are a couple of small steps worth a 25x FPS increase? Often, the answer is yes!

Check the OpenVINO toolkit website for installation instructions.

What is the Open Model Zoo?

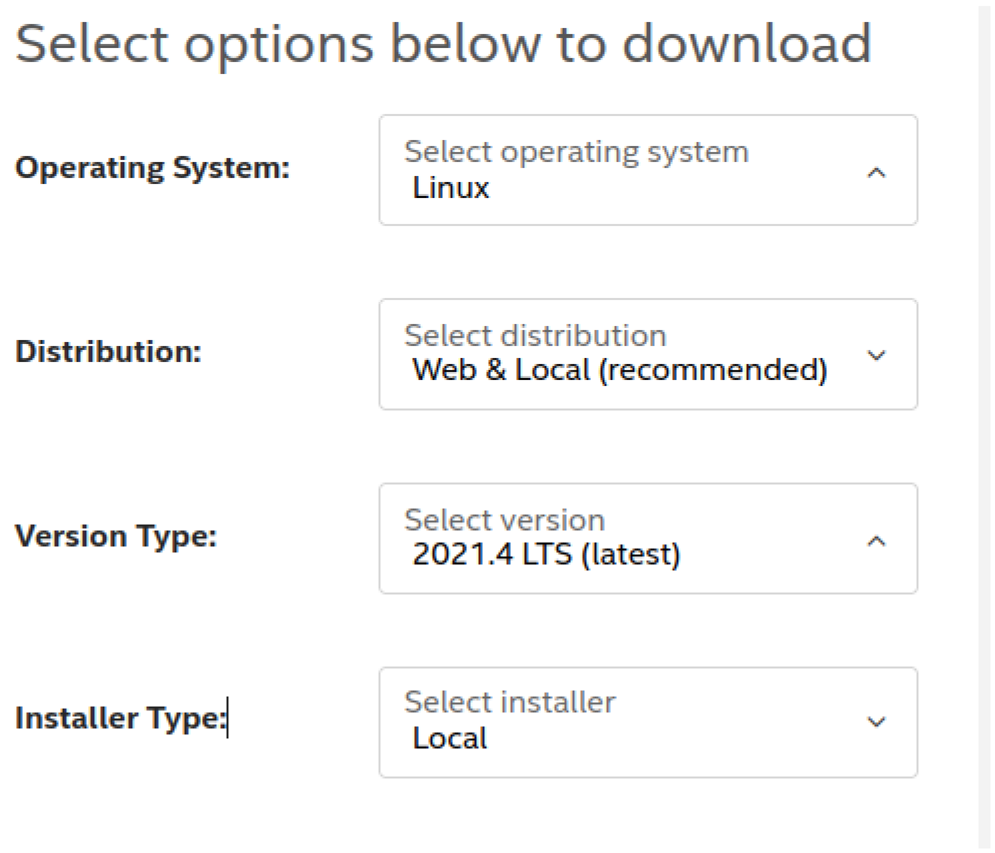
In machine learning/AI the name for a collection of pre-trained models is called a "model zoo". The Open Model Zoo is a library of freely-available pre-trained models. The Open Model Zoo also contains scripts for downloading those models into a compile-ready format to run on DepthAI.

DepthAI is able to run many of the object detection models from the Zoo. Several of those models are included in the DepthAI Github repositoy.

Install OpenVINO

Note: DepthAI gets support for the new OpenVINO version within a few days of the release, so **you should always use the latest OpenVINO version**.

You can download the OpenVINO toolkit installer from their download page, and we will use the latest version - which is 2021.4 at the time of writing.


Select options below to download

Operating System: Select operating system Linux

Distribution: Select distribution Web & Local (recommended)

Version Type: Select version 2021.4 LTS (latest)

Installer Type: Select installer Local

After downloading and extracting the compressed folder, we can run the installation:

```
~/Downloads/l_openvino_toolkit_p_2021.3.394$ sudo ./install_GUI.sh
```

All the components that we need will be installed by default. Our installation path will be `~/intel/openvino_2021` (default location), and we will use this path below.

Download the face-detection-retail-0004 model

Now that we have OpenVINO installed, we can use the model downloader

```
cd ~/intel/openvino_2021/deployment_tools/tools/model_downloader
python3 -mpip install -r requirements.in
python3 downloader.py --name face-detection-retail-0004 --output_dir ~/
```

This will download model files to `~/intel/`. Specifically, the model files we need are located at:

```
cd ~/intel/face-detection-retail-0004/FP16
```

We will move into this folder so we can later compile this model into the required **.blob**.

You'll see two files within the directory:

```
$ ls -lh
total 1.3M
```

(continues on next page)

(continued from previous page)

```
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

The model is in the OpenVINO Intermediate Representation (IR) format:

- `face-detection-retail-0004.xml` - Describes the network topology
- `face-detection-retail-0004.bin` - Contains the weights and biases binary data.

This means we are ready to compile the model for the MyriadX!

Compile the model

The MyriadX chip used on our DepthAI board does not use the IR format files directly. Instead, we need to generate `face-detection-retail-0004.blob` using `compile_tool` tool.

Activate OpenVINO environment

In order to use `compile_tool` tool, we need to activate our OpenVINO environment.

First, let's find `setupvars.sh` file

```
find ~/intel/ -name "setupvars.sh"
/home/root/intel/openvino_2021.4.582/data_processing/dl_streamer/bin/setupvars.sh
/home/root/intel/openvino_2021.4.582/opencv/setupvars.sh
/home/root/intel/openvino_2021.4.582/bin/setupvars.sh
```

We're interested in `bin/setupvars.sh` file, so let's go ahead and source it to activate the environment:

```
source /home/root/intel/openvino_2021.4.582/bin/setupvars.sh
[setupvars.sh] OpenVINO environment initialized
```

If you see `[setupvars.sh] OpenVINO environment initialized` then your environment should be initialized correctly

Locate compile_tool

Let's find where `compile_tool` is located. In your terminal, run:

```
find ~/intel/ -iname compile_tool
```

You should see the output similar to this

```
find ~/intel/ -iname compile_tool
/home/root/intel/openvino_2021.4.582/deployment_tools/tools/compile_tool/compile_tool
```

Save this path as you will need it in the next step, when running `compile_tool`.

Run compile_tool

From the `~/intel/face-detection-retail-0004/FP16` we will now call the `compile_tool` to compile the model into the **face-detection-retail-0004.blob**

```
~/intel/openvino_2021.4.582/deployment_tools/tools/compile_tool/compile_tool -m face-
↪detection-retail-0004.xml -ip U8 -d MYRIAD -VPU_NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_
↪CMX_SLICES 4
```

You should see:

```
Inference Engine:
  IE version ......... 2021.4.0
  Build ........... 2021.4.0-3839-cd81789d294-releases/2021/4

Network inputs:
    data : U8 / NCHW
Network outputs:
    detection_out : FP16 / NCHW
[Warning][VPU][Config] Deprecated option was used : VPU_MYRIAD_PLATFORM
Done. LoadNetwork time elapsed: 1760 ms
```

Where's the blob file? It's located in the current folder

```
/intel/face-detection-retail-0004/FP16$ ls -l
total 2.6M
-rw-rw-r-- 1 root root 1176544 jul 28 19:32 face-detection-retail-0004.bin
-rw-rw-r-- 1 root root 1344256 jul 28 19:51 face-detection-retail-0004.blob
-rw-rw-r-- 1 root root  106171 jul 28 19:32 face-detection-retail-0004.xml
```

Run and display the model output

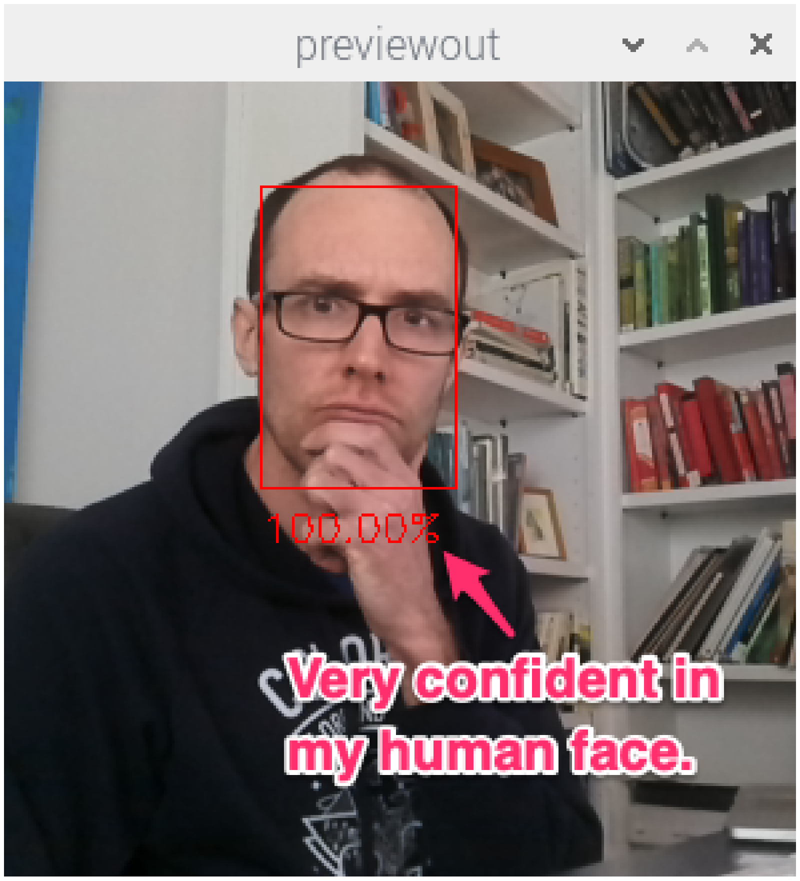
With neural network `.blob` in place, we're ready to roll! To verify that the model is running correctly, let's modify a bit the program we've created in Hello World tutorial

In particular, let's change the `setBlobPath` invocation to load our model. **Remember to replace the paths to correct ones that you have!**

```
- detection_nn.setBlobPath(str(blobconverter.from_zoo(name='mobilenet-ssd',␣
↪shaves=6)))
+ detection_nn.setBlobPath("/path/to/face-detection-retail-0004.blob")
```

And that's all!

You should see output annotated output similar to:



Reviewing the flow

The flow we walked through works for other pre-trained object detection models in the Open Model Zoo:

1. Download the model:

```
python3 downloader.py --name face-detection-retail-0004 --output_dir ~/
```

2. Create the MyriadX blob file:

```
./compile_tool -m [INSERT PATH TO MODEL XML FILE] -ip U8 -d MYRIAD -VPU_
↪NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_SLICES 4
```

Here are **all supported compile_tool arguments**.

3. Use this model in your script

You're on your way! You can find the complete code for this tutorial on GitHub.

Note: You should also check out the *Use a Pre-trained OpenVINO model* tutorial, where this process is significaly simplified.

To allow DepthAI to use your custom trained models, you need to convert them into a MyriadX blob file format - so that they are optimized for the best inference on MyriadX **VPU** processor.

There are two conversion steps that have to be taken in order to obtain a blob file:

- Use *Model Optimizer* to produce **OpenVINO IR representation** (where IR stands for Intermediate Representation)
- Use *Compile Tool* to compile IR representation model into **VPU blob**

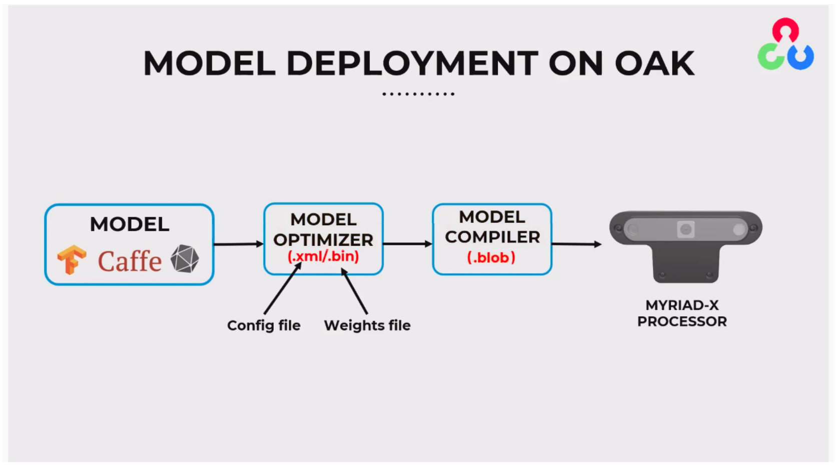


Fig. 1: from OpenCV Courses site

Model Optimizer

OpenVINO's Model optimizer converts the model from the original framework format into the OpenVINO's Intermediate Representation (IR) standard format (`.bin` and `.xml`). This format of the model can be deployed across multiple Intel devices: CPU, GPU, iGPU, **VPU** (which we are interested in), and FPGA.

Example usage of Model Optimizer with *online Blobconverter*:

```
--data_type=FP16 --mean_values=[0,0,0] --scale_values=[255,255,255]
```

Example for *local conversion*:

```
mo --input_model path/to/model.onnx --data_type=FP16 --mean_values=[0,0,0] --scale_
↪values=[255,255,255]
```

All arguments below are also documented on OpenVINO's docs here.

FP16 Data Type

Since we are converting for VPU (which supports FP16), we need to use parameter `--data_type=FP16`. More information here.

Mean and Scale parameters

OpenVINO's documentation here. **–mean_values** and **–scale_values** parameters will normalize the input image to the model: `new_value = (byte - mean) / scale`. By default, frames from ColorCamera/MonoCamera are in U8 data type (`[0,255]`).

Models are usually trained with normalized frames `[-1,1]` or `[0,1]`, so we need to normalize frames before running the inference. One (not ideal) option is to create *Custom model* that normalizes frames before inferencing (example here), but it's better (more optimized) to do it in the model itself.

Common options:

- **[0,1]** values, mean=0 and scale=255 (`([0,255] - 0) / 255 = [0,1]`)
- **[-1,1]** values, mean=127.5 and scale=127.5 (`([0,255] - 127.5) / 127.5 = [-1,1]`)
- **[-0.5,0.5]** values, mean=127.5 and scale=255 (`([0,255] - 127.5) / 255 = [-0.5,0.5]`)

Model layout parameter

OpenVINO's documentation here. Model layout can be specified with `--layout` parameter. We use **Planar / CHW** layout convention. A similar DepthAI error message will be shown if the image layout is not matching the model layout:

```
[NeuralNetwork(0)] [warning] Input image (416x416) does not match NN (3x416)
```

Note that by default, ColorCamera node will output `preview` frames in **Interleaved / HWC** layout (as it's native to OpenCV), and can be changed to Planar layout via API:

```
import depthai as dai
pipeline = dai.Pipeline()
colorCam = pipeline.createColorCamera()
colorCam.setInterleaved(False) # False = Planar layout
```

Color order

OpenVINO's documentation here. NN models can be trained on images that have either RGB or BGR color order. You can change from one to another using `--reverse_input_channels` parameter. We use **BGR** color order. For example, see Changing color order>.

Note that by default, ColorCamera node will output `preview` frames in **BGR** color order (as it's native to OpenCV), and can be changed to RGB color order via API:

```
import depthai as dai
pipeline = dai.Pipeline()
colorCam = pipeline.createColorCamera()
colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB) # RGB color order,␣
↪BGR by default
```

Compile Tool

After converting the model to OpenVINO's IR format (.bin/.xml), we need to use OpenVINO's Compile Tool to compile the model in IR format into `.blob` file, which can be deployed to the device (*tutorial here*)

Input layer precision: RVC2 only supports FP16 precision, so `-ip U8` will add conversion layer U8->FP16 on all input layers of the model - which is what we usually want. In some cases (eg. when we aren't dealing with frames), we want to use FP16 precision directly, so we can use `-ip FP16` (Cosine distance model example).

Shaves: RVC2 has a total has 16 SHAVE cores (see Hardware accelerators documentation). Compiling for more SHAVEs can make the model perform faster, but the proportion of shave cores isn't linear with performance. Firmware will warn you about a possibly optimal number of shave cores, which is `available_cores/2`. As by default, each model will run on 2 threads.

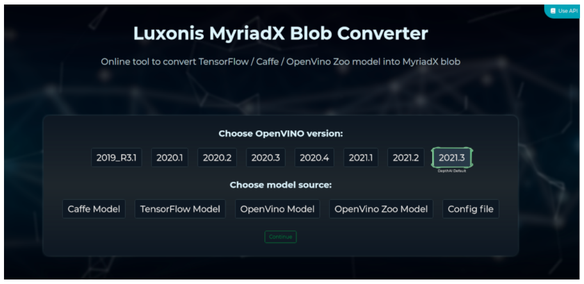
Converting and compiling models

There are a few options to perform these steps:

1. Using our *online blobconverter app*
2. Using our *blobconverter library*
3. *Converting & Compiling locally*

1. Using online blobconverter

You can visit our **online** Blobconverter app which allows you to convert and compile the NN model from **TensorFlow, Caffe, ONNX, OpenVINO IR, and OpenVINO Model Zoo**.



2. Using blobconverter package

For automated usage of our blobconverter tool, we have released a blobconverter PyPi package, that allows converting & compiling models both from the command line and from the Python script directly. Example usage below.

Install and usage instructions can be found here

```python
import blobconverter

blob_path = blobconverter.from_onnx(
    model="/path/to/model.onnx",
    data_type="FP16",
    shaves=5,
)
```

3. Local compilation

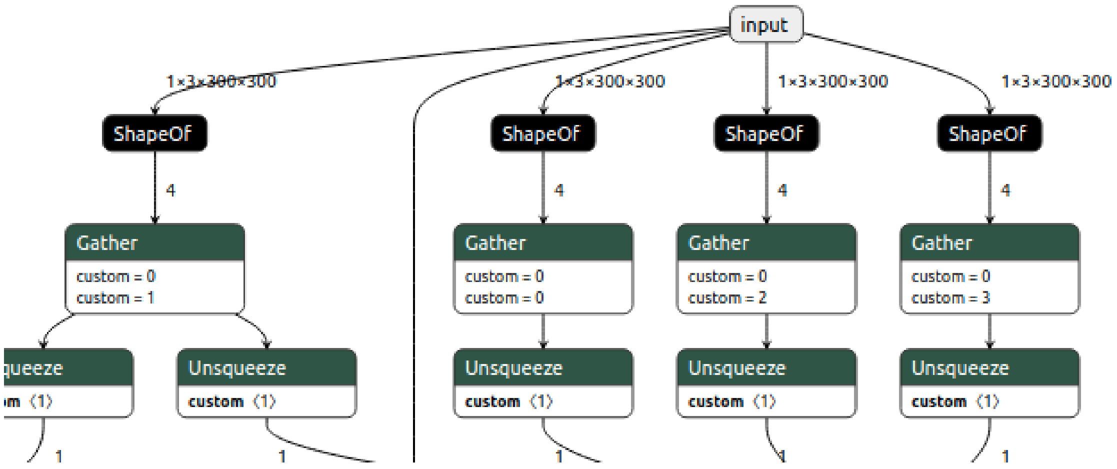
If you want to perform model conversion and compilation locally, you can follow:

- OpenVINO official instructions
- OpenVINO Python notebooks
- Our local model compilation tutorial
- Custom model conversion & compilation notes

Troubleshooting

When converting your model to the OpenVINO format or compiling it to a `.blob`, you might come across an issue. This usually means that a **connection between two layers is not supported** or that the **layer is not supported**.

For **visualization of NN models** we suggest using Netron app.



Supported layers

When converting your model to OpenVINO's IR format (`.bin` and `.xml`), you have to check if the OpenVINO supports layers that were used. Here are supported layers and their limitations for Caffee, MXNet, TensorFlow, TensorFlow 2 Keras, Kaldi, and ONNX.

Unsupported layer type "layer_type"

When using compile_tool to compile from IR (.xml/.bin) into .blob, you might get an error like this:

```
Failed to compile layer "Resize_230": unsupported layer type "Interpolate"
```

This means that the layer type is not supported by the VPU (Intels Myriad X). You can find supported OpenVINO layers by the VPU here, under the **Supported Layers** header, in the third column (**VPU**). Refer to official Intel's troubleshooting docs for more information.

Incorrect data types

If the compiler returns something along the lines of **"check error: input #0 has type S32, but one of [FP16] is expected"**, it means that you are using incorrect data types. In the case above, an INT32 layer is connected to FP16 directly. There should be a conversion in between these layers, and we can achieve that by using the OpenVINOs Convert layer between these two layers. You can do that by editing your models `.xml` and adding the Convert layer. You can find additional information on this discord thread.

4.3.2 Deploying Custom Models

This tutorial will review the process of deploying a custom trained (or pre-trained) model to the OAK camera. As mentioned in details in *Converting model to MyriadX blob* tutorial, you first need to convert the model to OpenVINO's IR format (.xml/.bin) and then compile it to .blob in order to then deploy it to the OAK camera.

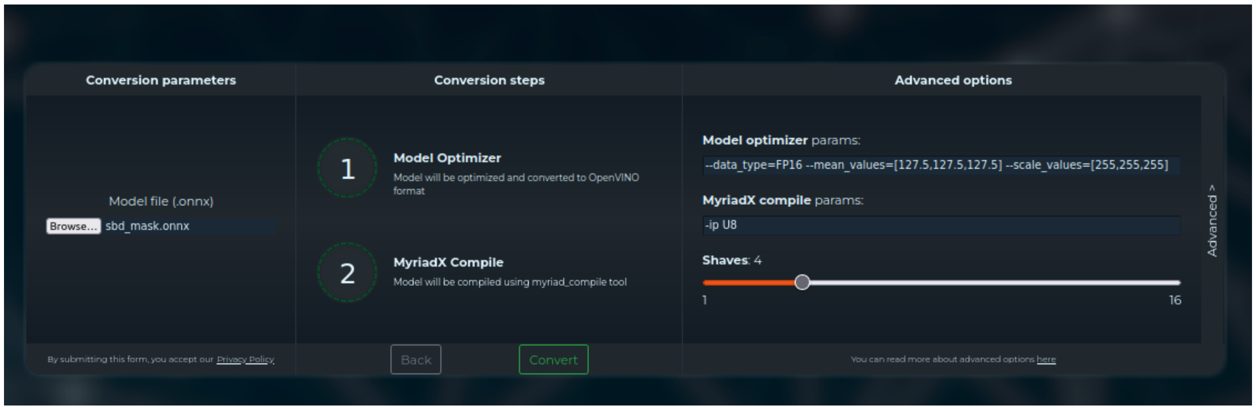
1. Face mask recognition model

For our first tutorial we will deploy the SBD_Mask classification model, as it already has pre-trained .onnx model in the /models folder.

From the examples in the repo you can see that they are using CenterFace face detection model, and then SBD_Mask classification model to recognize whether the person (more specifically the face image) is wearing a mask.

Converting to .blob

Since the .onnx model is already provided, we don't need to export the trained model (to eg. frozen TF model). After downloading the model, we can use blobconverter app to convert it to `.blob`. I will be using the latest version of OpenVINO (2021.4 as of time of writing), and I can select ONNX Model as the source. After clicking the `Continue` button, we can drag&drop the .onnx file to the `Browse...` button.



Model file (.onnx): sbd_mask.onnx

Model Optimizer
Model will be optimized and converted to OpenVINO format

MyriadX Compile
Model will be compiled using myriad_compile tool

Model optimizer params:
```
--data_type=FP16 --mean_values=[127.5,127.5,127.5] --scale_values=[255,255,255]
```

MyriadX compile params:
```
-ip U8
```

Shaves: 4

By submitting this form, you accept our Privacy Policy

You can read more about advanced options here

Before we click `Convert`, we should double-check the default values for Model Optimizer and Compile params.

Model optimizer parameters

Model Optimizer converts other model formats to OpenVINO's IR format, which produces .xml/.bin files.

To decide Model Optimizer parameters, we could either read the repository to find out the required input values, or read the code:

```
def classify(img_arr=None, thresh=0.5):
    blob = cv2.dnn.blobFromImage(img_arr, scalefactor=1 / 255, size=(csize, csize),␣
↪mean=(0, 0, 0),
                                 swapRB=True, crop=False)
    net.setInput(blob)

    heatmap = net.forward(['349'])
    match = m_func.log_softmax(torch.from_numpy(heatmap[0][0]), dim=0).data.numpy()
    index = np.argmax(match)

    return (0 if index > thresh else 1, match[0])
```

These few lines actually contain both logic for decoding (which we will use later) AND contain info about the required input values - `scalefactor=1 / 255` and `mean=(0, 0, 0)`, so the pretrained model expects `0..1` input values.

For Model Optimizer we will use the arguments below. See *Model Optimizer docs here* for **more info**.

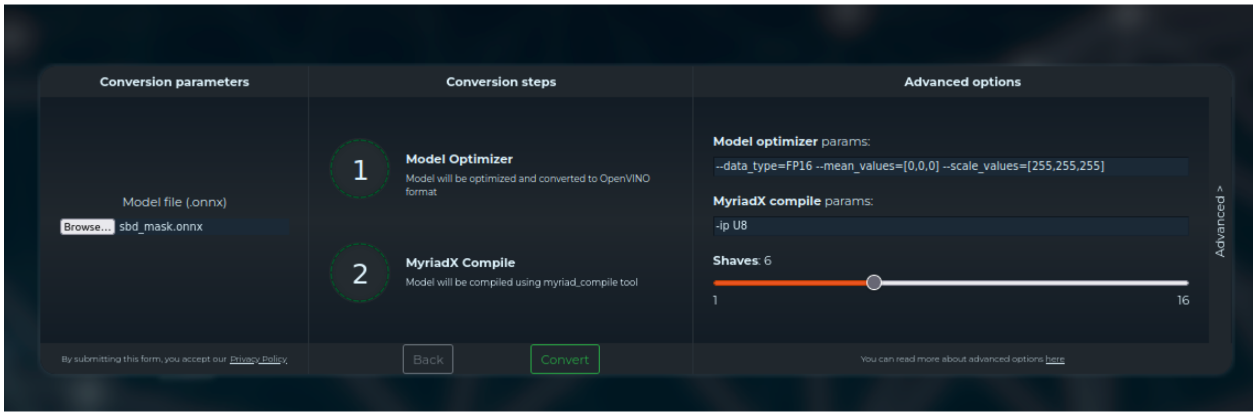
```
--data_type=FP16 --mean_values=[0,0,0] --scale_values=[255,255,255]
```

Myriad X compile parameters

After converting the model to OpenVINO's IR format (.bin/.xml), we need to use *Compile Tool* to compile the model to `.blob`, so it can be deployed to the camera.

Compiling the model

Now that we have set arguments to the blobconverter, we can click `Convert`. This will upload the .onnx model to the blobconverter server, run the Model optimizer and Compile tool, and then the blobconverter app will prompt us to save the `.blob` file.



Model file (.onnx)
sbd_mask.onnx

Model Optimizer
Model will be optimized and converted to OpenVINO format

MyriadX Compile
Model will be compiled using myriad_compile tool

Model optimizer params:
--data_type=FP16 --mean_values=[0,0,0] --scale_values=[255,255,255]

MyriadX compile params:
-ip U8

Shaves: 6

By submitting this form, you accept our Privacy Policy

You can read more about advanced options here

Fig. 2: Arguments to convert/compile the SBD Mask classification model

Deploying the model

Now that we have .blob, we can start designing depthai pipeline. This will be a standard 2-stage pipeline:

1. Run 1st NN model; object detection model (face detection in our case)
2. Crop the high-resolution image to only get the image of the object (face)
3. Use cropped image to run the 2nd NN model; image recognition (**SBD Mask classification** model)

We already have quite a few 2-stage pipeline demos:

- Age-gender recognition demo - First detect faces, then run age-gender recognition model
- Emotion recognition demo - First detect faces, then run emotion recognition model
- Face recognition demo - First detect faces, then run face recognition model (it also runs head pose estimation in between, documented here)
- Person re-identification demo - First detect people, then run person re-id model

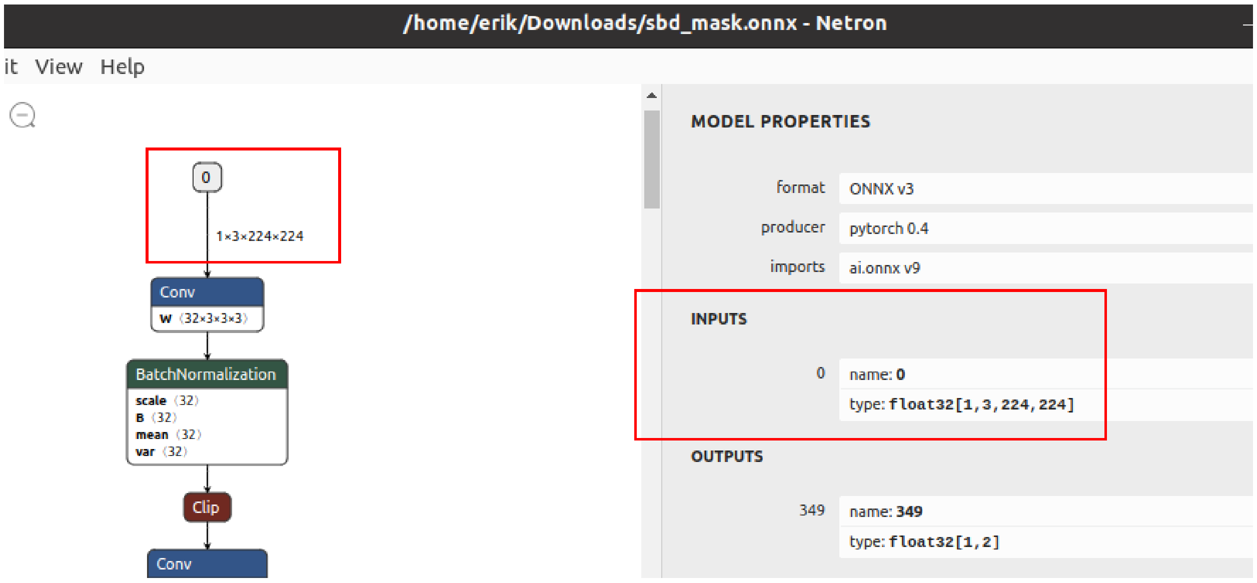
We will start with the age-gender recognition demo and simply replace the recognition model, so instead of running the age-gender model, we will run the SBD mask model.

Face detection

The age-gender demo uses face-detection-retail-0004 model, which is great in terms of accuracy/performance, so we will leave this part of the code: (lines 0-64)

Input shape

Age-gender uses age-gender-recognition-retail-0013 recognition model, which requires 62x62 frames. Our SBD-Mask model requires 224x224 as the input frame. You can see this when opening .xml/.onnx with the Netron app.



MODEL PROPERTIES

format	ONNX v3
producer	pytorch 0.4
imports	ai.onnx v9

INPUTS

0	name: 0
	type: float32[1,3,224,224]

OUTPUTS

349	name: 349
	type: float32[1,2]

Fig. 3: Input shape expected by the SBD Mask model

`recognition_manip` ImageManip node is responsible for cropping high-resolution frame to frames of faces at the required shape. We will need to change 62x62 to 224x224 shape in Script node (line 116) and as the ImageManip initial configuration (line 124).

```
    # Inside Script node
            for i, det in enumerate(face_dets.detections):
                cfg = ImageManipConfig()
                correct_bb(det)
                cfg.setCropRect(det.xmin, det.ymin, det.xmax, det.ymax)
                # node.warn(f"Sending {i + 1}. det. Seq {seq}. Det {det.xmin}, {det.
↪ymin}, {det.xmax}, {det.ymax}")
-               cfg.setResize(62, 62)
+               cfg.setResize(224, 224)
                cfg.setKeepAspectRatio(False)
                node.io['manip_cfg'].send(cfg)
                node.io['manip_img'].send(img)
    """)
    cam.preview.link(image_manip_script.inputs['preview'])

    recognition_manip = pipeline.create(dai.node.ImageManip)
-   recognition_manip.initialConfig.setResize(62, 62)
+   recognition_manip.initialConfig.setResize(224, 224)
    recognition_manip.setWaitForConfigInput(True)
    image_manip_script.outputs['manip_cfg'].link(recognition_manip.inputConfig)
    image_manip_script.outputs['manip_img'].link(recognition_manip.inputImage)
```

The pipeline will now send 224x224 cropped frames of all detected faces to the recognition NN.

Change the model

Now that `recognition_nn` will get 224x224 frames, we have to change the recognition model to the SBD-Mask model (line 132). I have placed my `sbd_mask.blob` in the same folder as the main.py script.

```
    # Second stange recognition NN
    print("Creating recognition Neural Network...")
    recognition_nn = pipeline.create(dai.node.NeuralNetwork)
-   recognition_nn.setBlobPath(blobconverter.from_zoo(name="age-gender-recognition-
↪retail-0013", shaves=6))
+   recognition_nn.setBlobPath("sbd_mask.blob") # Path to the .blob
    recognition_manip.out.link(recognition_nn.input)
```

Change decoding

The pipeline will stream SBD-Mask recognition results to the host. `MultiMsgSync.py` script will sync these recognition results with high-resolution color frames and object detection results (to display the bounding box around faces).

As *mentioned above*, SBD-Mask repository contained decoding logic as well, so we can just use that. First we need to run `log_softmax` function and then `np.argmax`. I will be using scipy's log_softmax function for simplicity. So we need to import `from scipy.special import log_softmax` in the script.

```
    bbox = frame_norm(frame, (detection.xmin, detection.ymin, detection.xmax,␣
↪detection.ymax))
```

(continues on next page)

(continued from previous page)

```
    # Decoding of recognition results
-   rec = recognitions[i]
-   age = int(float(np.squeeze(np.array(rec.getLayerFp16('age_conv3')))) * 100)
-   gender = np.squeeze(np.array(rec.getLayerFp16('prob')))
-   gender_str = "female" if gender[0] > gender[1] else "male"

+   rec = recognitions[i].getFirstLayerFp16() # Get NN results. Model only has 1␣
↪output layer
+   index = np.argmax(log_softmax(rec))
+   # Now that we have the classification result we can show it to the user
+   text = "No Mask"
+   color = (0,0,255) # Red
+   if index == 1:
+       text = "Mask"
+       color = (0,255,0)


    cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (10, 245, 10), 2)
    y = (bbox[1] + bbox[3]) // 2
```

Visualizing results

From the decoding step we got the text ("Mask"/"No Mask") which we want to display to the user and color (green/red) which we will use to color the rectangle around the detected face.

```
    text = "No Mask"
    color = (0,0,255) # Red
    if index == 1:
        text = "Mask"
        color = (0,255,0)

-   cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (10, 245, 10), 2)
+   cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 3) # Colorize␣
↪bounding box and make it thicker
    y = (bbox[1] + bbox[3]) // 2
-   cv2.putText(frame, str(age), (bbox[0], y), cv2.FONT_HERSHEY_TRIPLEX, 1.5, (0, 0,␣
↪0), 8)
-   cv2.putText(frame, str(age), (bbox[0], y), cv2.FONT_HERSHEY_TRIPLEX, 1.5, (255,␣
↪255, 255), 2)
-   cv2.putText(frame, gender_str, (bbox[0], y + 30), cv2.FONT_HERSHEY_TRIPLEX, 1.5,␣
↪(0, 0, 0), 8)
-   cv2.putText(frame, gender_str, (bbox[0], y + 30), cv2.FONT_HERSHEY_TRIPLEX, 1.5,␣
↪(255, 255, 255), 2)
+   cv2.putText(frame, text, (bbox[0], y + 30), cv2.FONT_HERSHEY_TRIPLEX, 1.5, (0, 0,␣
↪0), 8) # Display Mask/No Mask text
+   cv2.putText(frame, text, (bbox[0], y + 30), cv2.FONT_HERSHEY_TRIPLEX, 1.5, (255,␣
↪255, 255), 2)
    if stereo:
        # You could also get detection.spatialCoordinates.x and detection.
↪spatialCoordinates.y coordinates
        coords = "Z: {:.2f} m".format(detection.spatialCoordinates.z/1000)
        cv2.putText(frame, coords, (bbox[0], y + 60), cv2.FONT_HERSHEY_TRIPLEX, 1, (0,
↪ 0, 0), 8)
        cv2.putText(frame, coords, (bbox[0], y + 60), cv2.FONT_HERSHEY_TRIPLEX, 1,␣
↪(255, 255, 255), 2)
```

Changing color order

I noticed that the end result wasn't very accurate. This can be a result of variety of things (model is just inaccurate, model lost accuracy due to quantization (INT32->FP16), incorrect mean/scale values, etc.), but I like to first check color order. ColorCamera node will output BGR color order by default (on `preview` output). The model's accuracy won't be best if you send BGR frames to it and it was trained on RGB frames - which was the issue here.

You can change `preview`'s color order by adding this line:

```
    print("Creating Color Camera...")
    cam = pipeline.create(dai.node.ColorCamera)
    cam.setPreviewSize(1080, 1080)
    cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
    cam.setInterleaved(False)
    cam.setBoardSocket(dai.CameraBoardSocket.RGB)
+   cam.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)
```

Note that the **face detection model's accuracy will decrease** due to this change, as it expects BGR and will get RGB. The correct way would be to specify –reverse_input_channels (See *Color order* documentation) argument using the *Model Optimizer*, which is what was used to generate xml/bin files that were uploaded to our DepthAI Model Zoo.

```
mo --input_model sbd_mask.onnx --data_type=FP16 --mean_values=[0,0,0] --scale_
↪values=[255,255,255] --reverse_input_channels
```

End result

You can view all changes we have made on Github here.

You might have noticed that face detection isn't perfect when I have a mask on the face. That's probably due to RGB/BGR issue mentioned *above*. It's also likely the accuracy drops because the face-detection-retail-0004 model wasn't trained on images that had faces covered with masks. The lighting on my face also wasn't the best. We might get better results if we used ObjectTracker node to track faces, but that's out of the scope of this tutorial.

2. QR code detector

This tutorial will focus around deploying the **WeChat** QR code detector. I found this model while going through OpenCV Model Zoo. There are 2 models in this folder:

- **detect_2021nov** - QR code detection model
- **sr_2021nov** - QR code super resolution (224x224 -> 447x447)

We will be focusing on the first one, the QR code detection model.

Converting QR code detector to OpenVINO

Compared to the previous model (SBD-Mask), I couldn't find relevant information about mean/scale values for this model. For more information on mean/scale values, see the *1. tutorial*. In such cases, I usually do the following:

1. Convert the model to OpenVINO (using model optimizer) without specifying mean/scale values
2. Use OpenVINO's Inference Engine to run IR model (.bin/.xml)
3. After getting the decoding correct, I try out different mean/scale values until I fugire out the correct combination

4. After getting the decoding and mean/scale values right, I convert the model to blob and develop a DepthAI pipeline for it

So let's first convert the model to IR format (xml/bin) using OpenVINO:

```
mo --input_model detect_2021nov.caffemodel
```

Now that we have xml/bin, we can also look at the input/output shape of the model using Netron. Input shape is `1x384x384` (so grayscale frame, not color) and output shape is `100x7`.

Using Inference Engine (IE) to evaluate the model

The code below was modified from our depthai-inference-engine. I personally like to evaluate the inference on the CPU first and get these values correct:

- Mean/Scale values,
- Color order,
- Model layout

So I can estimate **accuracy degradation due to quantization** when going from **CPU (INT32)** to **Myriad X (FP16)**.

```python
from openvino.inference_engine import IECore
import cv2
import numpy as np

def crop_to_square(frame):
    height = frame.shape[0]
    width  = frame.shape[1]
    delta = int((width-height) / 2)
    return frame[0:height, delta:width-delta]

model_xml = 'detect_2021nov.xml'
model_bin = "detect_2021nov.bin"
shape = (384, 384) # We got this value from Netron

ie = IECore()
print("Available devices:", ie.available_devices)
net = ie.read_network(model=model_xml, weights=model_bin)
input_blob = next(iter(net.input_info))
# You can select device_name="MYRIAD" to run on Myriad X VPU
exec_net = ie.load_network(network=net, device_name='CPU')

MEAN = 127.5 # Also try: 127.5
SCALE = 255 # Also try: 0, 127.5

# Frames from webcam. Could take frames from OAK (running UVC pipeline)
# or from video file.
cam = cv2.VideoCapture(0)
cam.set(cv2.CAP_PROP_FPS, 30)
while True:
    ok, og_image = cam.read()
    if not ok: continue

    og_img = crop_to_square(og_image) # Crop to 1:1 aspect ratio
    og_img = cv2.resize(og_img, shape) # Downscale to 384x384
    image = cv2.cvtColor(og_img, cv2.COLOR_BGR2GRAY) # To grayscale
```

(continues on next page)

(continued from previous page)

```
    image = (image - MEAN) / SCALE # Normalize the input frame
    image = image.astype(np.int32) # Change type

    output = exec_net.infer(inputs={input_blob: image}) # Do the NN inference
    print(output) # Print the output

    cv2.imshow('USB Camera', og_img)
    if cv2.waitKey(1) == ord('q'): break
```

Decoding QR code detector

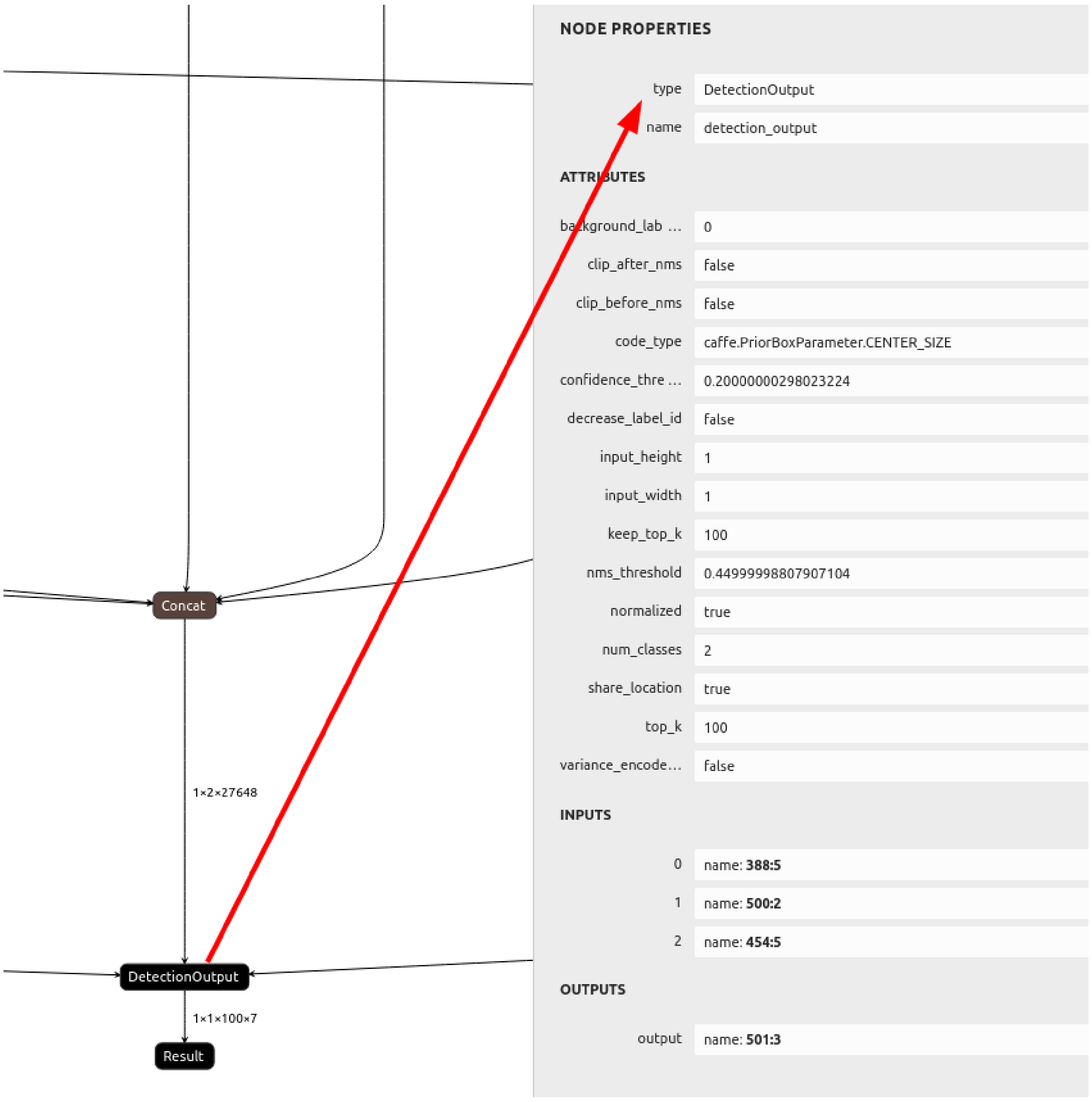
The script above will print full NN output. If you have a QR code in front of the camera, the output array should contain some values other than 0. The `(100,7)` output is the standard object detection output which contains:

```
batch_num = result[0] # Always 0, 1 frame at a time
label = result[1] # Always 1, as we only detect one object (QR code)
confidence = result[2]
bounding_box = result[3:7]
```

Initially, I thought we would need to perform NMS algorithm on the host, but after checking the model, I saw it has the DetectionOutput layer at the end. This layer performs NMS in the NN, so it's done on the camera itself. When creating a pipeline with the DepthAI we will also be able to use MobileNetDetectionNetwork node, as it was designed to decode these standard SSD detection results.

```
# Normalize the bounding box to frame resolution.
# For example, [0.5, 0.5, 1, 1] bounding box on 300x300 frame
# should return [150, 150, 300, 300]
def frame_norm(frame, bbox):
    bbox[bbox < 0] = 0
    normVals = np.full(len(bbox), frame.shape[0])
    normVals[::2] = frame.shape[1]
    return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

# ...

# Do the NN inference
output = exec_net.infer(inputs={input_blob: image})
print(output) # Print the output

results = output['detection_output'].reshape(100, 7)
for det in results:
    conf = det[2]
    bb = det[3:]
    bbox = frame_norm(og_img, bb)
    cv2.rectangle(og_img, (bbox[0], bbox[1]) , (bbox[2], bbox[3]), (255, 127, 0), 3)

cv2.imshow('USB Camera', og_img)
if cv2.waitKey(1) == ord('q'): break
```

After trying a few MEAN/SCALE values, I found that MEAN=0 and SCALE=255 works the best. We don't need to worry about color order as the model requires grayscale images.



NODE PROPERTIES

type	DetectionOutput
name	detection_output

ATTRIBUTES

background_lab …	0
clip_after_nms	false
clip_before_nms	false
code_type	caffe.PriorBoxParameter.CENTER_SIZE
confidence_thre …	0.20000000298023224
decrease_label_id	false
input_height	1
input_width	1
keep_top_k	100
nms_threshold	0.44999998807907104
normalized	true
num_classes	2
share_location	true
top_k	100
variance_encode…	false

INPUTS

0	name: **388:5**
1	name: **500:2**
2	name: **454:5**

OUTPUTS

output	name: **501:3**

Fig. 4: NMS layer in the model

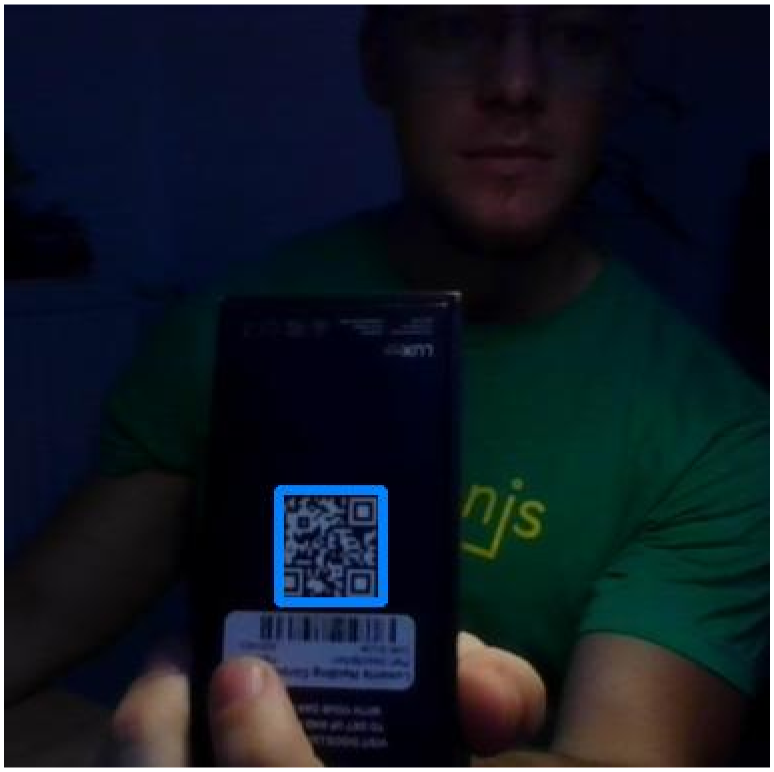


Fig. 5: Success! Light blue bounding box around the QR code!

Testing accuracy degradation due to FP16 quantization

Now let's try the model with FP16 precision instead of INT32. If you connect an OAK camera to your computer you can select `MYRIAD` as the inference device instead of CPU. **If the model works correctly with IE on Myriad X, it will work with DepthAI as well.**

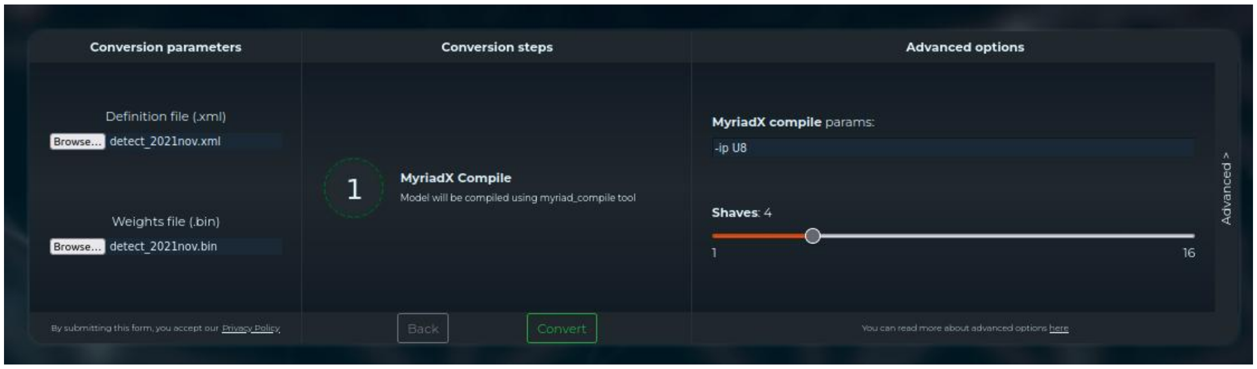
```
- exec_net = ie.load_network(network=net, device_name='CPU')
+ exec_net = ie.load_network(network=net, device_name='MYRIAD')

  # ...
  image = cv2.cvtColor(og_img, cv2.COLOR_BGR2GRAY) # To grayscale
  image = (image - MEAN) / SCALE # Normalize the input frame
- image = image.astype(np.int32) # Change type
+ image = image.astype(np.float16) # Change type

  output = exec_net.infer(inputs={input_blob: image}) # Do the NN inference
```

I haven't noticed any accuracy degradation loss, so we can use *Model Optimizer* and proceed with correct arguments this time. We will specify scale value 255 and FP16 datatype.

```
mo --input_model detect_2021nov.caffemodel --scale 255 --data_type FP16
```

Integrating QR code detector into DepthAI

We now have our normalized model in IR. I will use blobconverter app to convert it to `.blob`, which is required by the DepthAI.


Conversion parameters
Conversion steps
Advanced options
Browse...
1
Back
Convert
Advanced >
By submitting this form, you accept our Privacy Policy.
You can read more about advanced options here

Definition file (.xml): detect_2021nov.xml

Weights file (.bin): detect_2021nov.bin

MyriadX Compile
Model will be compiled using myriad_compile tool

MyriadX compile params: -ip U8

Shaves: 4 (1–16)

As mentioned above, the model outputs the standard SSD detection results, so we can use `MobileNetDetectionNetwork` node. I will start with Mono & MobilenetSSD example code and **only change blob path, label map, and frame shape**.

```
import cv2
import depthai as dai
import numpy as np

nnPath = "detect_2021nov.blob" # Change blob path
labelMap = ["background", "QR-Code"] # Change labelMap

# Create pipeline
pipeline = dai.Pipeline()

# Define sources and outputs
monoRight = pipeline.create(dai.node.MonoCamera)
manip = pipeline.create(dai.node.ImageManip)
nn = pipeline.create(dai.node.MobileNetDetectionNetwork)
manipOut = pipeline.create(dai.node.XLinkOut)
nnOut = pipeline.create(dai.node.XLinkOut)

manipOut.setStreamName("right")
nnOut.setStreamName("nn")

# Properties
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

manip.initialConfig.setResize(384, 384) # Input frame shape
manip.initialConfig.setFrameType(dai.ImgFrame.Type.GRAY8) # Model expects Grayscale␣
↪image

nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.input.setBlocking(False)

monoRight.out.link(manip.inputImage)
manip.out.link(nn.input)
manip.out.link(manipOut.input)
```

(continues on next page)

(continued from previous page)

```
nn.out.link(nnOut.input)

with dai.Device(pipeline) as device:

    qRight = device.getOutputQueue("right", maxSize=4, blocking=False)
    qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)

    frame = None
    detections = []

    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        color = (255, 255, 0)
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,␣
↪detection.ymax))
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +␣
↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,␣
↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), color, 2)
        cv2.imshow(name, frame)

    while True:
        frame = qRight.get().getCvFrame()
        frame =  cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR) # For colored visualization
        detections = inDet = qDet.get().detections
        displayFrame("right", frame)

        if cv2.waitKey(1) == ord('q'):
            break
```

QR Code end result

This is the end result of the script above. You can see that the mono camera sensor on OAK cameras performs much better in low-light environment compared to my laptop camera (screenshot above). I uploaded this demo to depthai-experiments/gen2-qr-code-scanner where I have also added blobconvewrter and displaying NN results on high-resolution frames.

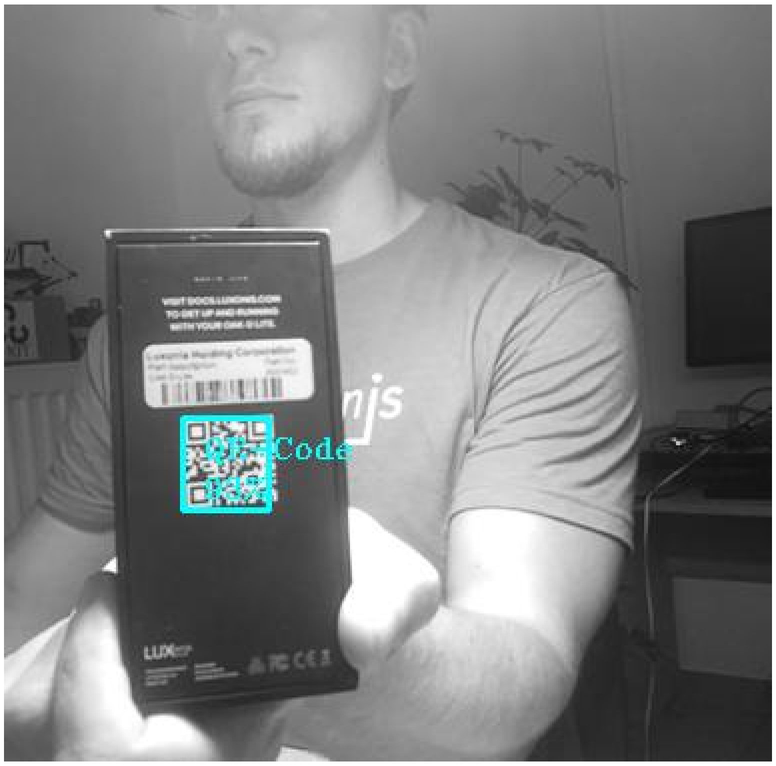
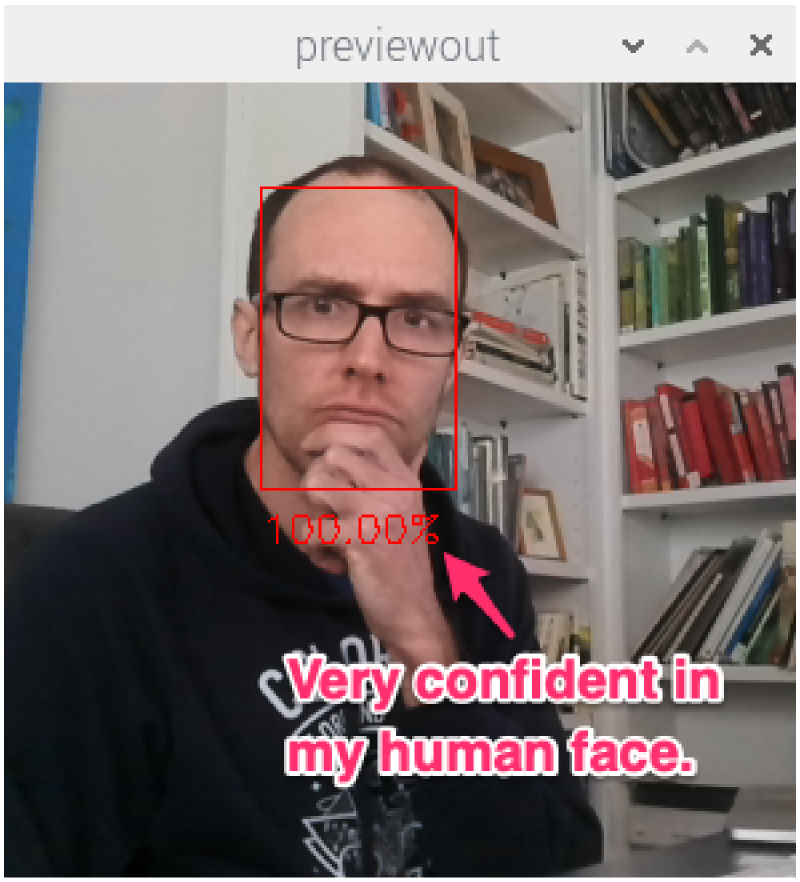


Fig. 6: On-device decoding using the script above!

4.3.3 Use a Pre-trained OpenVINO model

In this tutorial, you’ll learn how to use a pre-trained face detection model to detect faces in real-time, even on a low-powered Raspberry Pi.



If you would like to learn more about OpenVINO, Open Model Zoo and how to locally convert OpenVINO model into **.blob**, check out our *Local OpenVINO Model Conversion*

Run DepthAI Default Model

The `depthai_demo.py` file can be modified directly to you do your bidding, or you can simply pass arguments to it for which models you want to run.

For simplicity we will do the latter, simply passing arguments so that DepthAI runs the `face-detection-retail-0004` instead of the model run by default.

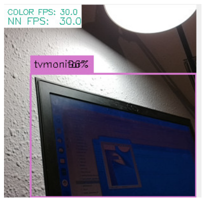
Before switching to using the `face-detection-retail-0004` let's take a baby step and give these command line options a spin. In this case we'll just pass in the same neural network that default runs when running `python3 depthai_demo.py`, just to make sure we're doing it right:

```
python3 depthai_demo.py -dd
```

This will then run the a typical demo MobileNetv2 SSD object detector trained on the PASCAL 2007 VOC classes, which are:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: airplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, TV/monitor

I ran this on my laptop with an OAK-D sitting on my desk pointing upwards randomly - and it makes out the corner of my laptop screen and correctly identifies it as `tvmonitor`:



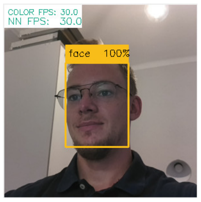
Run model

Now that we've got this verified, let's move on to trying out other models, starting with `face-detection-retail-0004`.

To use this model, simply specify the name of the model to be run with the `-cnn` flag, as below:

```
python3 depthai_demo.py -dd -cnn face-detection-retail-0004
```

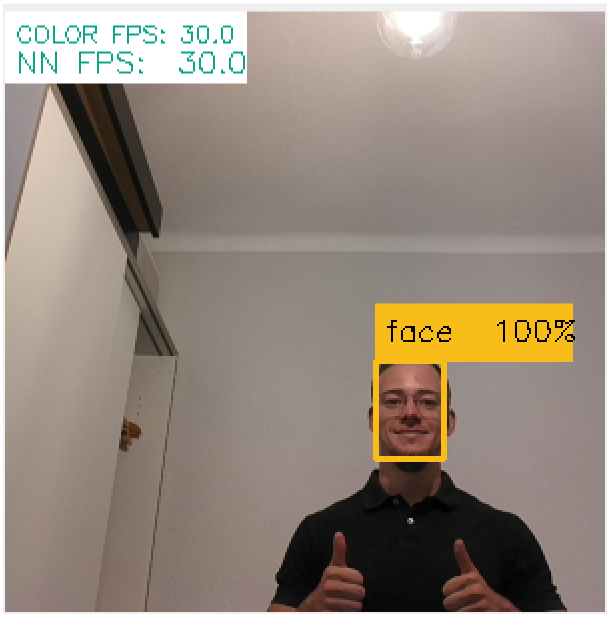
This will download the compiled `face-detection-retail-0004` NN model and use it to run inference (detect faces) on color frames:

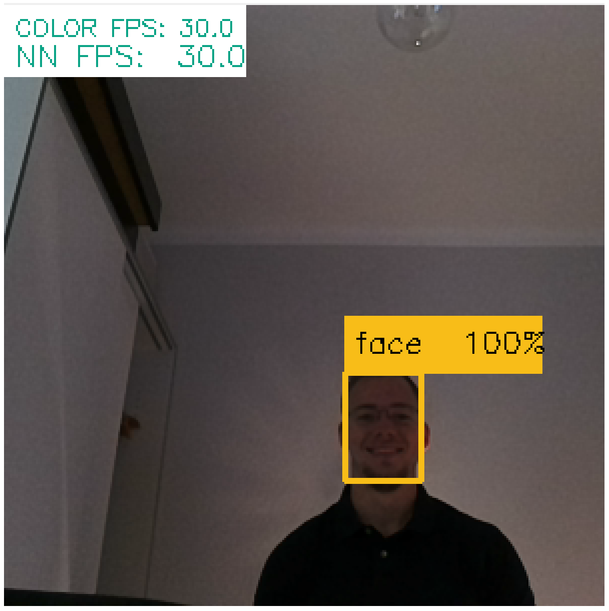


It's that easy. Substitute your face for mine, of course.

And if you'd like to try other models, just peruse here and run them by their name, just like above.

Now take some time to play around with the model. You can for example check how far away the model can detect your face:





In the latter image you can see that I'm quite back-lit, which is one of the main challenges in face detection (and other feature detection). In this case, it's likely limiting the maximum range for which a face can be detected. From the testing above, for a confidence threshold of 50%, this range appears to be about 20 feet. You could get longer range out of the same model by reducing the model confidence threshold (by changing from `0.5` here) at the cost of increased probability of false positives.

Another limiting factor is that this is a relatively low-resolution model (300x300 pixels), so faces get fairly small fairly fast at a distance. So let's try another face detection model that uses a higher resolution.

Trying Other Models

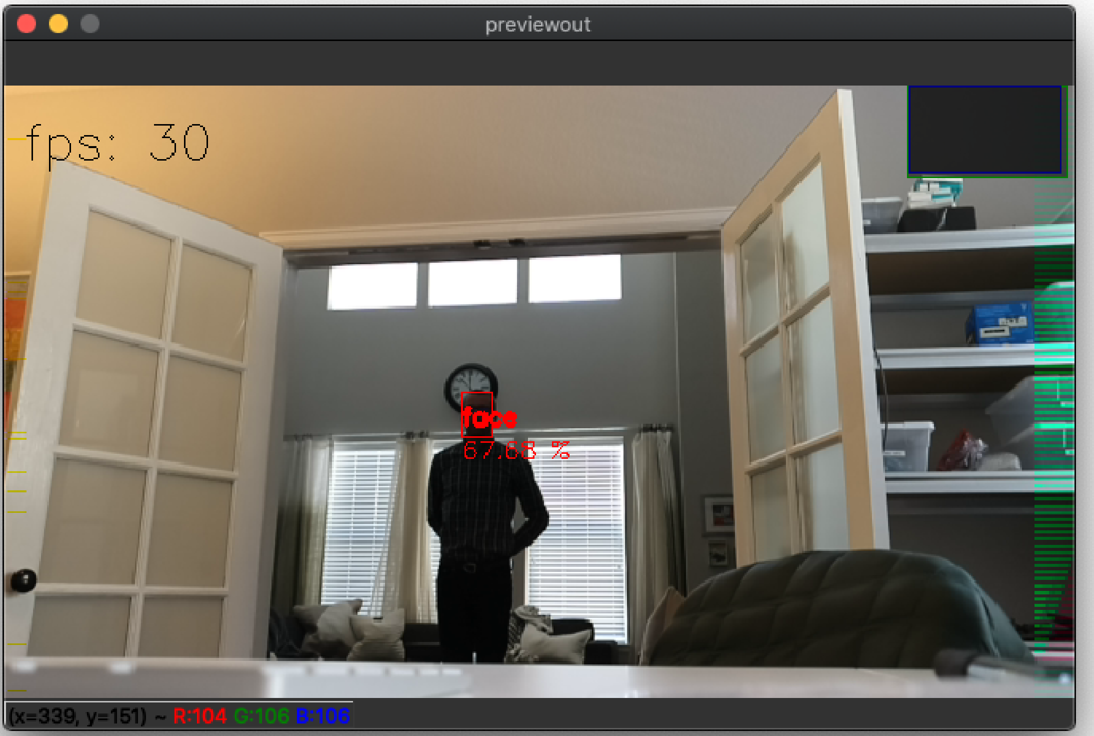
The flow we walked through works for other pre-trained models in our repository (here), which includes:

- People semantic segmentation (`deeplabv3p_person`)
- Face detection (ADAS) (`face-detection-adas-0001`)
- Face detection (retail, the one we used) (`face-detection-retail-0004`)
- Mobilenet general object detection (default model) (`mobilenet-ssd`)
- Pose estimation (`openpose2`)
- Pedestrian detection (ADAS) (`pedestrian-detection-adas-0002`)
- Person detection (retail) (`person-detection-retail-0013`)
- Person, Vehicle and Bike Detection (`person-vehicle-bike-detection-crossroad-1016`)
- tiny Yolo - general object detection (`tiny-yolo-v3`)
- Vehicle detection for driver-assistance (`vehicle-detection-adas-0002`)
- Vehicle and license plate detection (`vehicle-license-plate-detection-barrier-0106`)
- Yolo - general object detection (`yolo-v3`)

You can simply specify any of these models after the `-cnn` argument.

Let's try out `face-detection-adas-0001`, which is intended for detecting faces inside the cabin of a vehicle. (ADAS stands for Advanced Driver-Assistance Systems)

```
python3 depthai_demo.py -dd -cnn face-detection-adas-0001
```



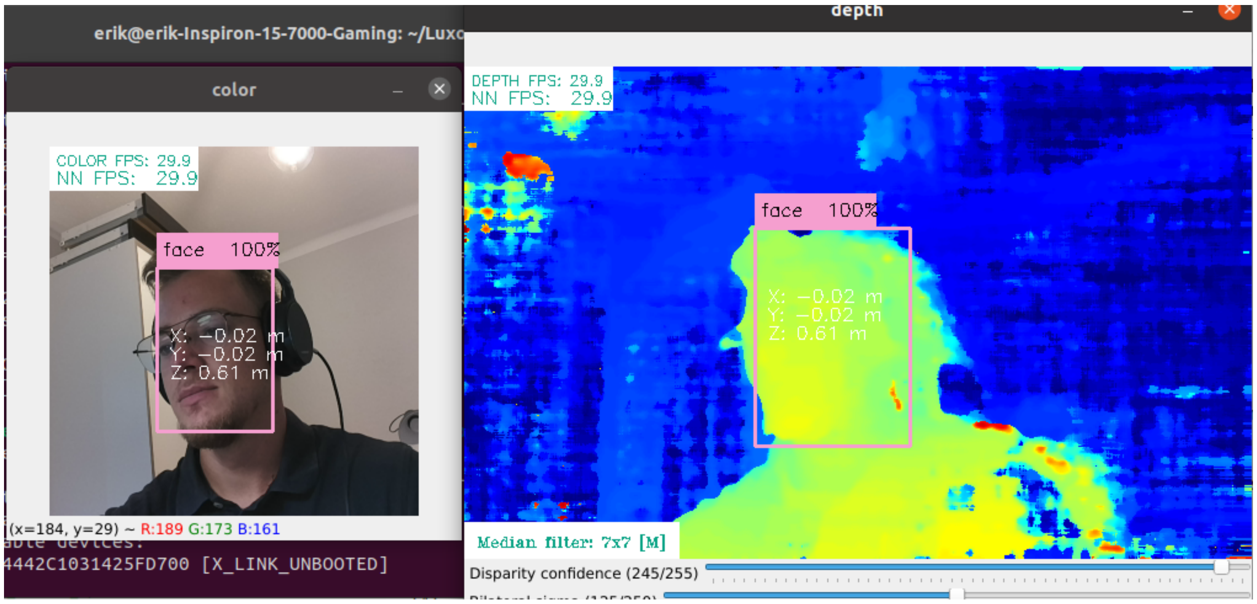
So this model actually has a shorter detection distance than the smaller model despite having a higher resolution. Why? Likely because it was intentionally trained to detect only close-in faces since it's intended to be used in the cabin of a vehicle. (You wouldn't want to be detecting the faces in cars passing by, for example.)

Spatial AI - Augmenting the Model with 3D Position

So by default DepthAI is set to return the full 3D position. So in the command above, we actually specify for it to not be calculated with `-dd` (or `--disableDepth`).

So let's run that same command, but with that line omitted, such that 3D results are returned (and displayed):

```
python3 depthai_demo.py -cnn face-detection-retail-0004
```



```
(x=184, y=29) ~ R:189 G:173 B:161
4442C1031425FD700 [X_LINK_UNBOOTED]
Disparity confidence (245/255)
```

And there you find the 3D position of my face!

You can then choose other models and get real-time 3D position for the class of interest.

4.3.4 Custom training

Overview

On our Github repo depthai-ml-training we provide several ML training notebooks trained on various data sets. You can run these notebooks on Google Colab - they provide free CPU/GPU instances, so great for prototyping and even simple production models.

We currently have these ML training tutorials:

- Tutorial - SSD MobileNetv2 training
- Tutorial - SSD MobileNetv2 training with custom data
- Tutorial - YOLOv4-tiny based Mask Detector
- Tutorial - YOLOv3-tiny based Mask Detector
- Tutorial - DeepLabV3+ MNV2 semantic segmetnation
- Tool - Google Drive image batch resizer

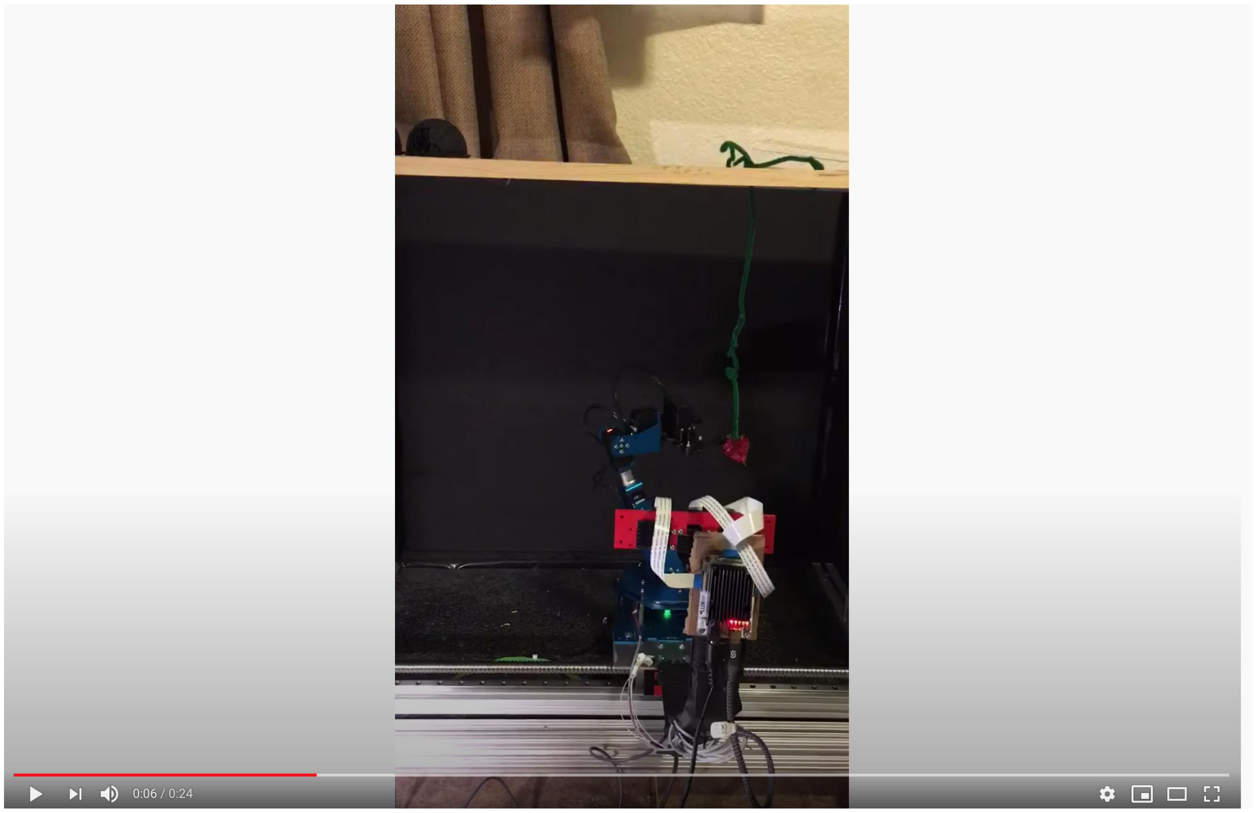
The Tutorials

The tutorial notebook *Easy_Object_Detection_With_Custom_Data_Demo_Training.ipynb* shows how to quickly train an object detector based on the MobileNet SSDv2 network.

Optionally, see our documentation around this module (here) for of a guide/walk-through on how to use this notebook. Also, feel free to jump right into the Notebook, with some experimentation it's relatively straightforward to get a model trained.

After training is complete, it also converts the model to a .blob file that runs on our DepthAI platform and modules. First the model is converted to a format usable by OpenVINO called Intermediate Representation, or IR. The IR model is then compiled to a .blob file using a server we set up for that purpose. (The IR model can also be converted locally to a blob.)

And that's it, in less than a couple of hours a fairly advanced proof of concept object detector can run on DepthAI to detect objects of your choice and their associated spatial information (i.e. X, Y, Z coordinates). For example this notebook was used to train DepthAI to locate strawberries in 3D space, see below:



The *Medical Mask Detection Demo Training.ipynb* training notebook shows another example of a more complex object detector. The training data set consists of people wearing or not wearing masks for viral protection. There are almost 700 pictures with approximately 3600 bounding box annotations. The images are complex: they vary quite a lot in scale and composition. Nonetheless, the object detector does quite a good job with this relatively small data set for such a task. Again, training takes around 2 hours. Depending on which GPU the Colab lottery assigns to the notebook instance, training 10k steps can take 2.5 hours or 1.5 hours. Either way, a short period for such a good quality proof of concept for such a difficult task. We then performed the steps above for converting to blob and then running it on our DepthAI module.

Supporting Notebooks

This notebook operates on your set of images in Google Drive to resize them to the format needed by the training notebooks.

OAK cameras can **run any AI model**, even custom architectured/built ones. You can even run multiple AI models at the same time, either in parallel or series (a demo here).

Subpages:

- *Converting model to MyriadX blob* tutorial showcases model conversion and compilation steps to .blob format so you can **run the model on the device**.

- *Deploying Custom Models* provides step-by-step tutorial on how to convert, compile and deploy the model to OAK device
- *Use one of 250+ pre-trained models*; either from OpenVINO Model Zoo or DepthAI Model Zoo

4.3.5 AI vision tasks

We have open-source examples and demos for many different AI vision tasks, such as:

- **Object detection** models provide bounding box, confidence, and label of all detected objects. Demos: MobileNet, Yolo, EfficientDet, Palm detection.
- **Landmark detection** models provide landmarks/keypoints of an object. Demos: Human pose, hand landmarks, and facial landmarks.
- **Semantic segmentation** models provide label/class for each pixel. Demos: Person segmentation, multiclass segmentation, road segmentation.
- **Classification** models provide classification label and confidence in that label. Demos: EfficientNet, Tensorflow classification, fire classification, emotions classification.
- **Recognition** models provide byte array that can be used for recognition or recognized feature itself. Demos: Face recognition, person identification, OCR, license plate recognition.

There are also many other AI vision tasks that don't fall in any of the categories above, like crowd counting, monocular depth estimation, gaze estimation, or age/gender estimation.

All of the demos above run on color/grayscale frames. Many of these vision tasks can be **fused with the depth perception** (on the OAK camera itself), which unlocks the **power of** *Spatial AI*.

4.3.6 Model Performance

AI model performance depends on the accelerator that's on the OAK device. For current devices that use RVC2 you can find the performance results here.

4.4 Depth perception

DepthAI platform supports different ways of perceiving depth:

1. *Passive stereo depth perception* - Used by non-Pro version of OAK-D cameras
2. *Active stereo depth perception* - Used by Pro version of OAK-D cameras
3. *Time-of-Flight depth perception* - Used by OAK-pToF

4.4.1 Passive stereo depth perception

Passive stereo works a lot like our eyes. Our brains (subconsciously) estimate the depth of objects and scenes based on the difference of what our left eye sees versus what our right eye sees. On OAK-D cameras, it's exactly the same; we have a stereo camera pair - left and right monocular cameras - and the VPU (brains of the OAK cameras) does the **disparity matching** to estimate the depth of objects and scenes.

The **disparity** is the distance (in pixels) between the same point in the left and right image of the stereo pair camera. A great demo of disparity is below - the disparity is visualized with a red line between points/features - which in this case are facial landmarks (eyes, nose, mouth).

OAK-D camera does that for every pixel in the mono frame - it goes through pixels on the first mono frame, finds the same point/feature on the second mono frame, and assigns a disparity value (in pixels) with some confidence for every pixel. This all happens inside the StereoDepth node. The depth map is calculated from the disparity map (on the camera) using this formula.

Disparity matching won't work well with blank, featureless surfaces (like walls or ceilings) when using passive stereo depth perception. That's because disparity matching will have a hard time matching points/features from left/right images - as there are no distinctive points/features in either frame. That's where active stereo depth perception is needed.

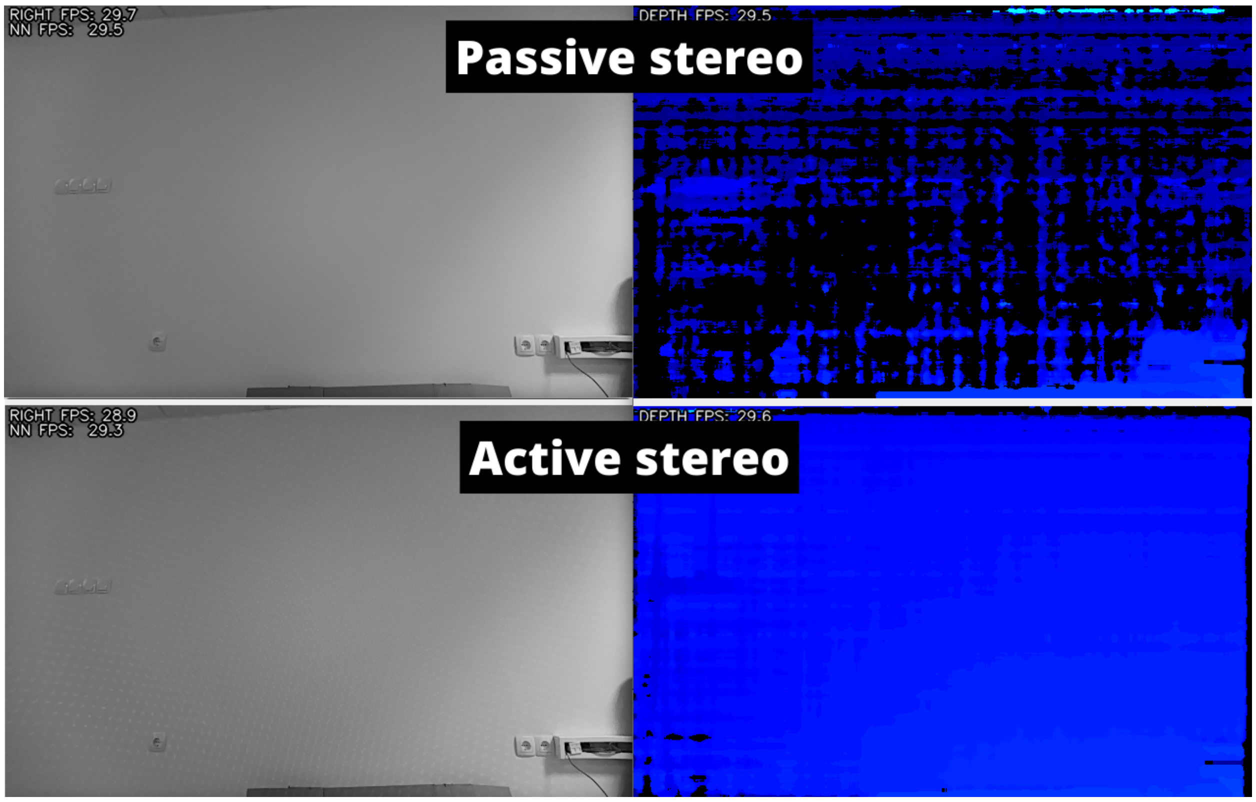
Passive stereo accuracy/smoothness depends on:

1. **Lighting/Texture**. Stereo depth depends on feature matching, and in a low light environment, features aren't as visible. As mentioned in the paragraph above, featureless surfaces (like walls) aren't suited for passive stereo depth perception. Active stereo resolves both texture and lighting requirements.
2. **Calibration**. Factory calibration should be optimal.
3. **Postprocessing filters**, documentation here (under *Depth Filters*). Additional filtering can be performed on the host-side as well, eg. WLS.

4.4.2 Active stereo depth perception

On our OAK Pro cameras, we use conventional active stereo vision (ASV). A **dot projector** is used to project many small dots in front of the device, which helps with disparity matching, especially for low-visual-interest surfaces (blank surfaces with little to no texture), such as a wall or ceiling.

The stereo matching process is performed exactly the same way as with passive stereo perception, the dots only help with the accuracy.



Here you can see passive and active stereo perception when the OAK camera is faced against a wall. If you look closely at the mono image (bottom left), you can see many small dots being projected onto the wall.

4.4.3 Time-of-Flight depth perception

Stereo perception has its pros and cons. It's cheap, can perceive depth at greater distances, and has a high resolution. On the other hand, it's not as accurate. So for high-accuracy applications, Time-of-Flight (ToF) approach is suggested, as it can provide sub-centimeter depth accuracy.

We have built a ToF FFC module that can be used with OAK-FFC and in the future, we will create a standalone camera with ToF sensor onboard.

On the gif above you can see high accuracy (especially at low FPS) point-cloud that was produced using a ToF FFC module and a color camera. ToF resolution here is 244x172.

4.5 Computer Vision

4.5.1 Run your own CV functions on-device

As mentioned in *On-device programming*, you can create **custom CV models** with your favorite NN library, convert & compile it into the `.blob` and run it on the device. This tutorial will cover how to do just that.

If you are interested in **training & deploying your own AI models**, refer to *Custom training*.

Demos:

- Frame concatenation - using PyTorch
- Laplacian edge detection - using *Kornia*
- Frame blurring - using *Kornia*
- Tutorial on running custom models on OAK by Rahul Ravikumar
- Harris corner detection in PyTorch by Kunal Tyagi

Create a custom model with PyTorch

TL;DR if you are interested in implementation code, it's here.

1. **Create PyTorch NN module**

   We first need to create a Python class that extends PyTorch's nn.Module. We can then put our NN logic into the `forward` function of the created class. In the example of frame concatenation, we can use torch.cat function to concatenate multiple frames:

```
class CatImgs(nn.Module):
    def forward(self, img1, img2, img3):
        return torch.cat((img1, img2, img3), 3)
```

   For a more complex module, please refer to Harris corner detection in PyTorch demo by Kunal Tyagi.

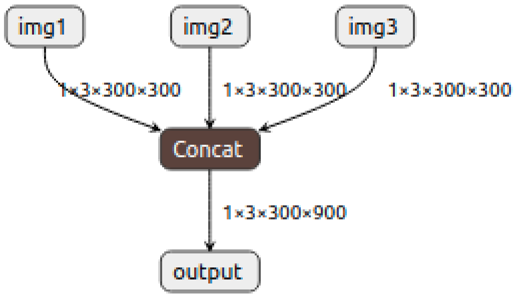
   Keep in mind that VPU supports only FP16, which means that max value is 65504. When multiplying a few values you can quickly overflow if you don't properly normalize/divide values.

2. **Export the NN module to onnx**

Since PyTorch isn't directly supported by OpenVINO, we first need to export the model to onnx format and then to OpenVINO. PyTorch has integrated support for onnx, so exporting to onnx is as simple as:

```
# For 300x300 frames
X = torch.ones((1, 3, 300, 300), dtype=torch.float32)
torch.onnx.export(
    CatImgs(),
    (X, X, X), # Dummy input for shape
    "path/to/model.onnx",
    opset_version=12,
    do_constant_folding=True,
)
```

This will export the concatenate model into onnx format. We can visualize the created model using Netron app:

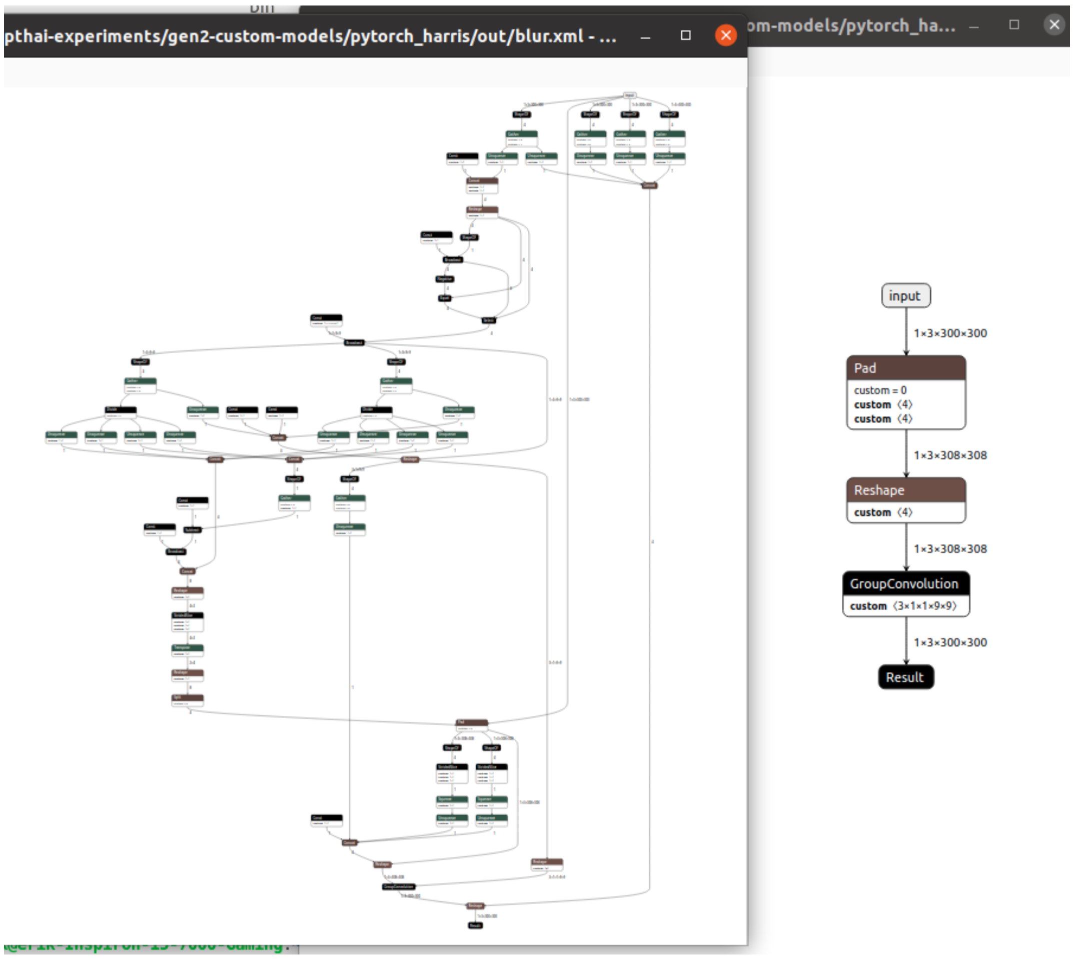


3. **Simplify onnx model**

When exporting the model to onnx, PyTorch isn't very efficient. It creates tons unnecessary operations/layers which increases the size of your network (which can lead to lower FPS). That's why we recommend using onnx-simplifier, a simple python package that removes unnecessary operations/layers.

```
import onnx
from onnxsim import simplify

onnx_model = onnx.load("path/to/model.onnx")
model_simpified, check = simplify(onnx_model)
onnx.save(model_simpified, "path/to/simplified/model.onnx")
```

Here is an example of how significant was the simplification using the onnx-simplifier. On the left, there's a blur model (from Kornia) exported directly from PyTorch, and on the right, there's a simplified network of **the same functionality**:



4. **Convert to OpenVINO/blob**

   Now that we have (simplified) onnx model, we can convert it to OpenVINO and then to the `.blob` format. For additional information about converting models, see *Converting model to MyriadX blob*.

   This would usually be done first by using OpenVINO's model optimizer to convert from onnx to IR format (.bin/.xml) and then using Compile tool to compile to `.blob`. But we could also use blobconverter to convert from onnx directly to .blob.

   Blobconverter just does both of these steps at once - without the need of installing OpenVINO. You can compile your onnx model like this:

```
import blobconverter

blobconverter.from_onnx(
    model="/path/to/model.onnx",
    output_dir="/path/to/output/model.blob",
    data_type="FP16",
    shaves=6,
    use_cache=False,
    optimizer_params=[]
)
```

5. **Use the .blob in your pipeline**

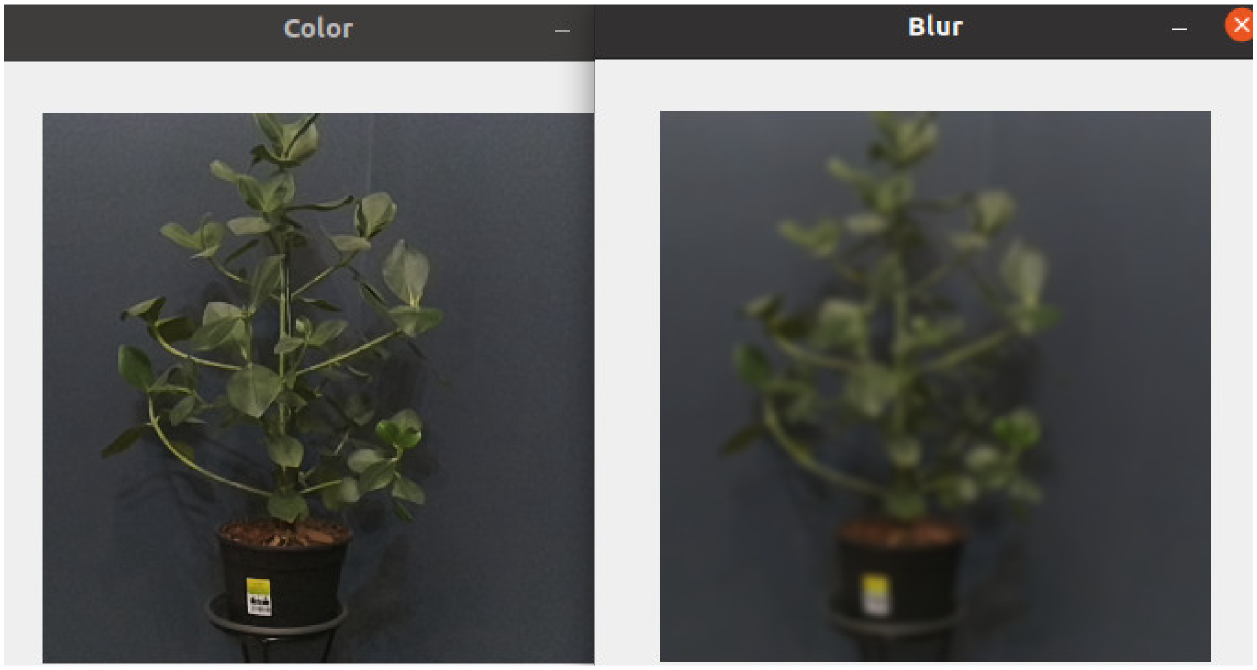
   You can now use your `.blob` model with the NeuralNetwork node. Check depthai-experiments/custom-models to run the demo applications that use these custom models.

Kornia

Kornia, “State-of-the-art and curated Computer Vision algorithms for AI.”, has a **set of common computer vision algorithms implemented in PyTorch**. This allows users to do something similar to:

```
import kornia

class Model(nn.Module):
    def forward(self, image):
        return kornia.filters.gaussian_blur2d(image, (9, 9), (2.5, 2.5))
```

and use the exact same procedure as described in *Create a custom model with PyTorch* to achieve frame blurring, as shown below:



Note: during our testing, we have found that **several algorithms aren’t supported** by either the OpenVINO framework or by the VPU. We have submitted an Issue for Sobel filter already.

Our platform supports computer vision (CV) functions to be performed on the device itself. While you can’t run OpenCV, you can use many of its supported functions. With DepthAI pipeline builder you can:

- **Crop, rotate, warp/dewarp, mirror, flip, transform perspective**, etc. with ImageManip node
- **Detect edges** (Sobel filter) with EdgeDetector node
- **Detect and track features** with FeatureTracker node
- **Track objects** (Kalman filter, Hungarian algorithm) with ObjectTracker node. Out-of-the-box support for Yolo and MobileNet object detectors.
- **Perceive stereo depth** (Census Tranform, Cost Matching and Aggregation) with StereoDepth node

If you would like to use any **other CV functions**, see *Run your own CV functions on-device* documentation on how to implement and run CV functions efficiently on the device’s hardware-accelerated blocks.

Some other CV examples:

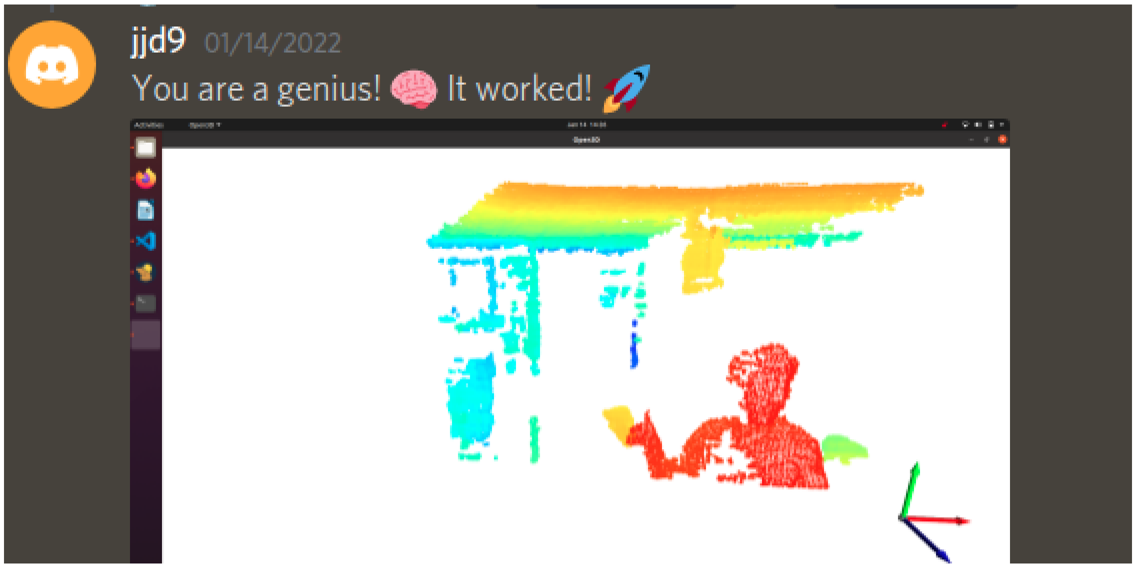
- Lossless zooming
- Host-side WLS depth filtering
- PointCloud visualization

4.6 On-device programming

4.6.1 On-device Pointcloud NN model

At the time of writing, DepthAI firmware (2.15) doesn't support converting depth to pointcloud. On the *On-device programming* page it's mentioned that Script node shouldn't be used for any kind of heavy computing, so to convert depth to pointcloud, we would need to *create a custom NN model*.

Kornia library has a function called depth_to_3d which does exactly that; it returns pointcloud from depth map and camera matrix. A smart person from our Discord community called **jjd9** created a working demo of the depth_to_3d logic running on the OAK camera. For C++ version, see code here.



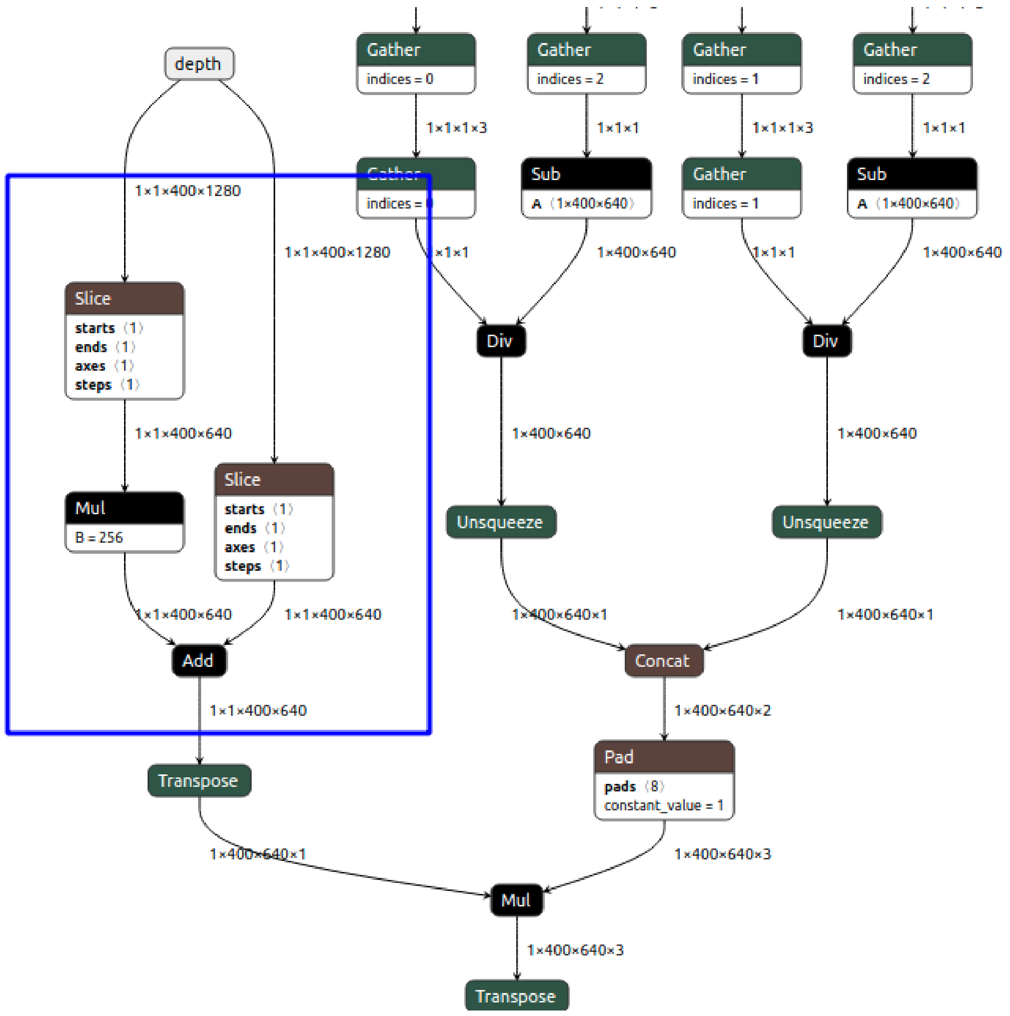
Depth to NN model

StereoDepth's `depth` output is U16 (Unsigned INT 16) datatype. Myriad X only supports FP16 datatype, but with OpenVINO's Compile Tool you can add conversion layer at the input with `-ip` or `-iop` arguments. These arguments only support **FP16** (so no conversion) or **U8** (adds U8 -> FP16 layer before the input).

Current workaround is to set conversion for U8 -> FP16 (`-ip U8`). This means the frame will be twice as wide, as each depth pixel will be represented by two (U8) integers. So instead of 640x400 depth frame, the model expects 1280x400 frame.

```
# convert the uint8 representation of the image to uint16 (this is needed because the
# converter only allows U8 and FP16 input types)
depth = 256.0 * image[:,:,:,1::2] + image[:,:,:,::2]
```

The code above was added to the model definition. It takes two FP16 numbers and reconstructs the original depth value, so the `depth` tensor has shape 640x400 and is FP16 datatype. This logic can also be seen in the model architecture:

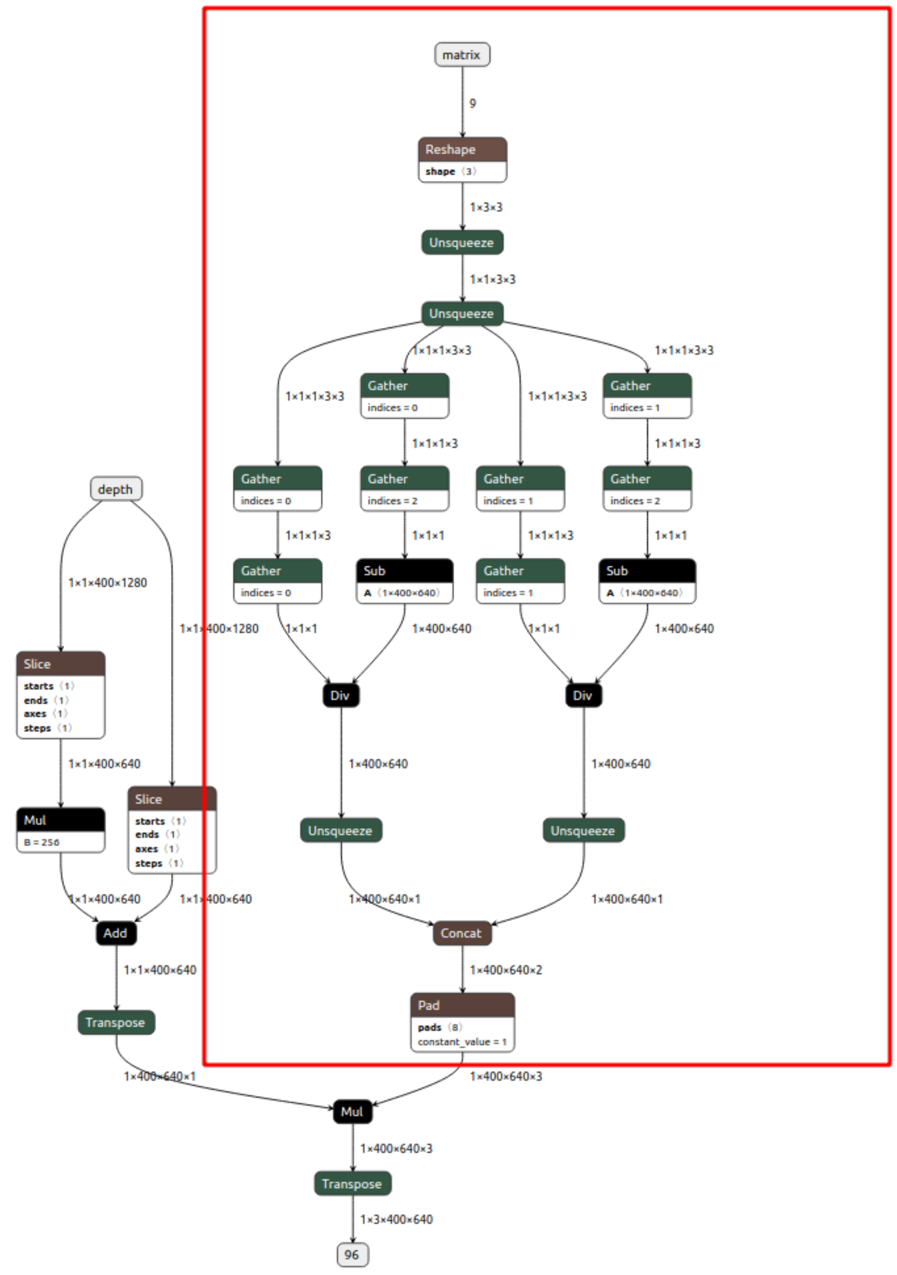


Optimizing the Pointcloud model

A few improvements could be made, as:

- The camera matrix is hard-coded into the NN model. This means users would have to create their own NN models, which adds unnecessary package dependencies (pytorch, onnx, onnxsim, blobconverter).
- It's fairly slow - pointcloud calculation (without visualization) runs at ~19FPS for 640x400 depth frames.

Since the camera matrix (intrinsics) is static, the part below in red could be calculated once instead of being calculated every single depth frame. This should reduce the complexity of the model and improve FPS.



Since Kornia library is open source, we can start with the architecture of the **depth_to_3d** function (code here) and remove the unnecessary part of the model. After moving all the logic to the same function, we end with this code.

We can remove the part of the model that calculates `xyz` vector and calculate it once on the host, then send it to the model and reuse it for every inference. I have also converted the code so it uses **numpy functions** instead of pytorch ones, to avoid pytorch dependency:

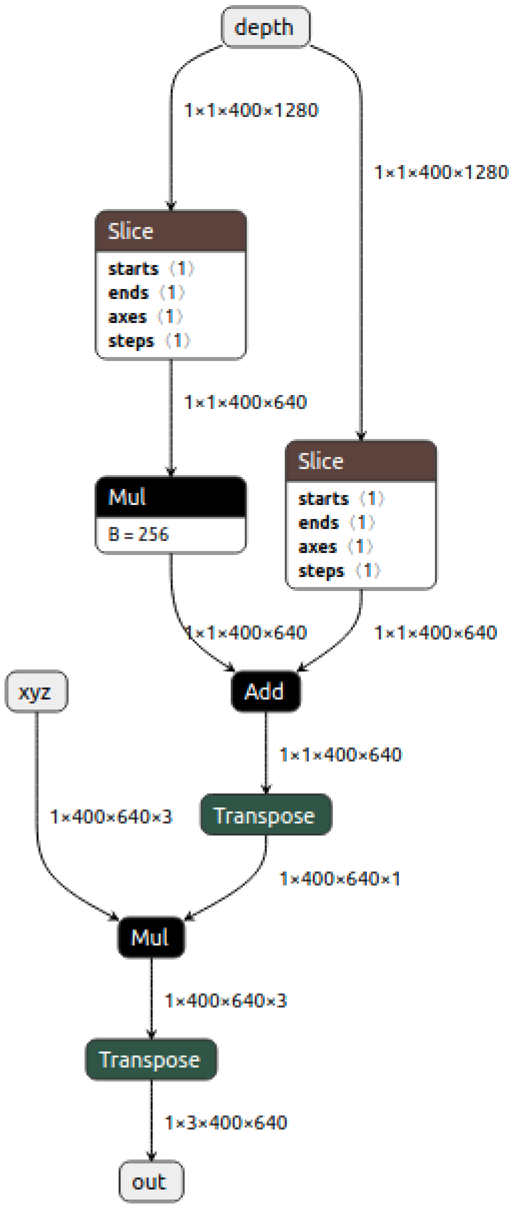
```
def create_xyz(width, height, camera_matrix):
    xs = np.linspace(0, width - 1, width, dtype=np.float32)
    ys = np.linspace(0, height - 1, height, dtype=np.float32)
```

(continues on next page)

(continued from previous page)

```
    # generate grid by stacking coordinates
    base_grid = np.stack(np.meshgrid(xs, ys)) # WxHx2
    points_2d = base_grid.transpose(1, 2, 0) # 1xHxWx2

    # unpack coordinates
    u_coord: np.array = points_2d[..., 0]
    v_coord: np.array = points_2d[..., 1]

    # unpack intrinsics
    fx: np.array = camera_matrix[0, 0]
    fy: np.array = camera_matrix[1, 1]
    cx: np.array = camera_matrix[0, 2]
    cy: np.array = camera_matrix[1, 2]

    # projective
    x_coord: np.array = (u_coord - cx) / fx
    y_coord: np.array = (v_coord - cy) / fy

    xyz = np.stack([x_coord, y_coord], axis=-1)
    return np.pad(xyz, ((0,0),(0,0),(0,1)), "constant", constant_values=1.0)
```

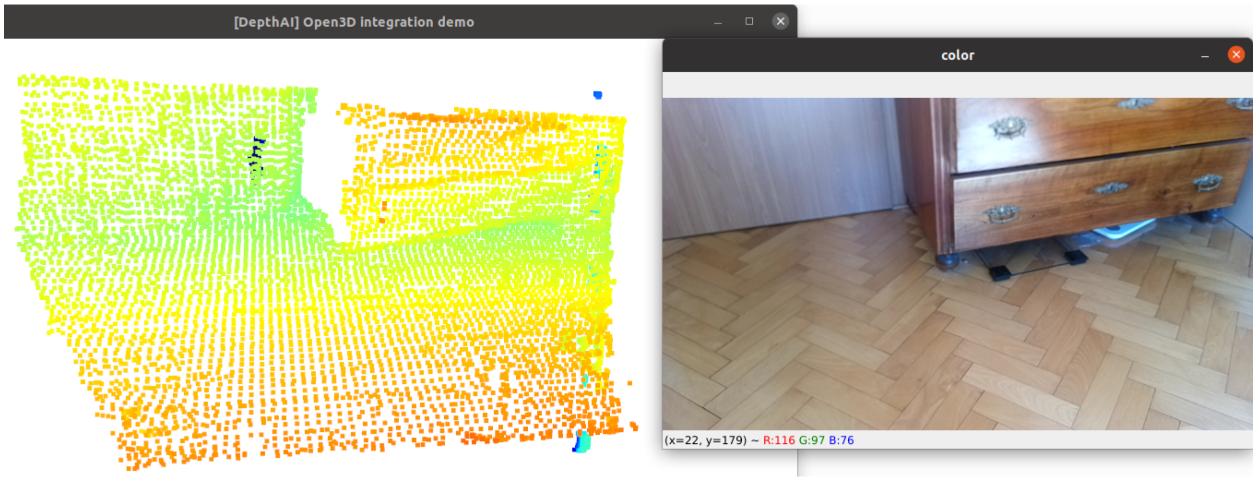
Moving this logic to the host side has significantly reduced the model's complexity, as seen below.



On-device Pointcloud demo

Altering the model has improved the performance of it, but not as much as I would have expected. The demo now runs at about 24FPS (previously 19FPS) for the 640x400 depth frames.

This **demo can be found at** depthai-experiments.


[DepthAI] Open3D integration demo
color
(x=22, y=179) ~ R:116 G:97 B:76

On the host side we only downsample (for faster visualization). For "cleaner" pointcloud we could also remove statistical outliers, but that's outside the scope of this tutorial.

While regular (firmware) on-device development is not possible due to closed nature of native tooling, we still expose a couple of alternative ways of running custom code:

1. Scripting - Using Python3.9 with Script node
2. Creating your own NN model to run more computationally heavy features
3. Creating custom OpenCL kernels

Note: With Series 3 OAK cameras, users will be able to run custom containirized apps on the OAK camera itself.

4.6.2 Using Script node

Using Script node allows you to run custom python scripts on the device itself, which allows users greater flexibility when constructing pipelines.

Script node is also **very useful when using multiple neural networks in series** and you need to process the output of the 1st one before feeding an image to the second one. **Example** here would be face age/gender recognition demo - first NN would detect faces, pass detections to the Script node which would create ImageManip configurations to crop the original frame and feed the face age/gender recognition NN only the cropped face frame.

For running computationally heavy features (eg. image filters), due to performance reasons you might want to avoid using Script node and rather go with one of the 2 options described below.

4.6.3 Creating custom NN models

You can create custom models with your favourite NN library, convert the model into OpenVINO and then compile it into the `.blob`. More information on this topic can be found on *Converting model to MyriadX blob* documentation.

Refer to *Run your own CV functions on-device* page to find out more, or check *On-device Pointcloud NN model* tutorial.

4.6.4 Creating custom OpenCL kernel

Creating custom NN models has some limitations, for example *unsupported layers* by OpenVINO/VPU. To avoid these limitations, you could consider creating custom OpenCL kernel and compile it for the VPU. This kernel will run on SHAVE core(s) on the VPU. One should also take into the account that this option is not super user friendly. We plan on creating a tutorial on how to develop these and run them on OAK cameras.

- Tutorial on how to implement custom layers with OpenCL by OpenVINO
- Custom kernel implementations in OpenCL

4.7 FAQs & How-To

4.7.1 Why Does DepthAI Exist?

In trying to solve an Embedded *Spatial AI* problem (details here), we discovered that although the perfect chip existed, there was no platform (hardware, firmware, or software) which allowed the chip to be used to solve such a Spatial AI & CV problem.

So we built the platform - known as DepthAI and the OpenCV AI Kit (OAK) - which allows folks to embed performant, spatial AI & CV into their products quickly and easily.

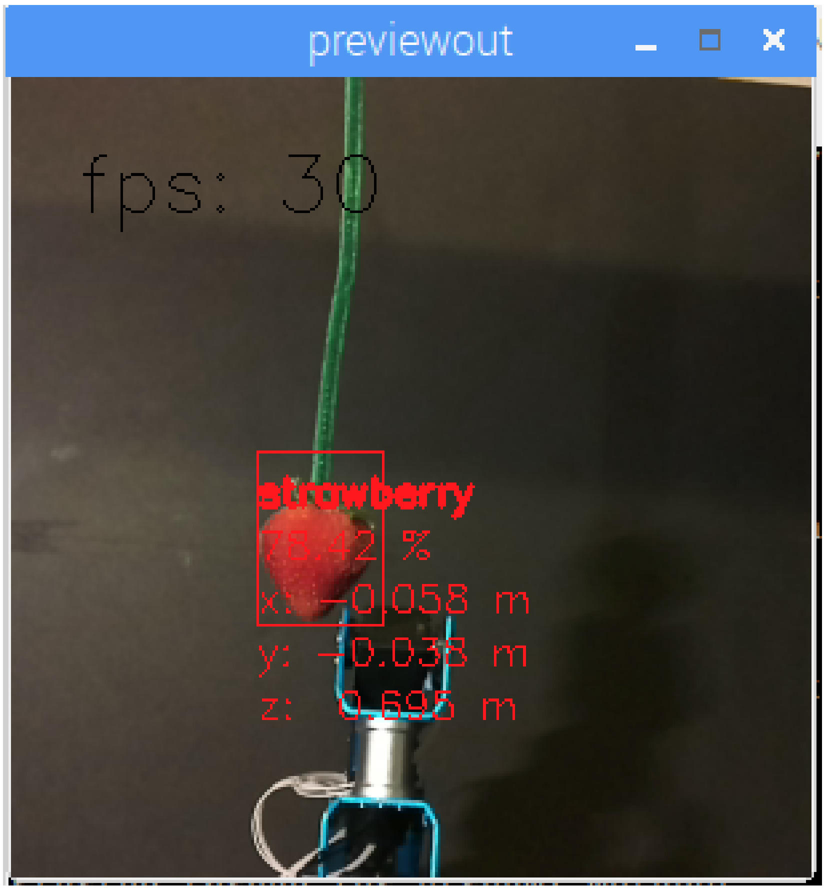
4.7.2 What is DepthAI?

DepthAI is *the* Embedded, Performant, Spatial AI+CV platform, composed of an open-source hardware, firmware, software ecosystem that provides turnkey embedded *Spatial AI+CV* and hardware-accelerated computer vision.

It gives embedded systems the super-power of human-like perception in real-time: what an object is and where it is in physical space.

It can be used with off-the-shelf AI models (how-to *here*) or with custom models using our completely-free training flow (how-to here).

An example of a custom-trained model is below, where DepthAI is used by a robot to autonomously pick and sort strawberries by ripeness.



It was trained to do so over the course of a weekend, by a student (for a student project), using our free online training resources.

DepthAI is also open-source (including hardware). This is done so that companies (and even individuals) can prototype and productize solutions quickly, autonomously, and at low risk.

See the summary of our (MIT-Licensed) Github repositories *below*, which include open-source hardware, firmware, software, and machine-learning training.

4.7.3 What is SpatialAI? What is 3D Object Localization?

Spatial AI *documentation here*. 3D Object Localization *documentation here*.

4.7.4 How is DepthAI Used? In What Industries is it Used?

DepthAI has been used in effectively every industry, from farming/ranch, to cleaning spots courts, to building personal-service robots. Here's a quick list of some common use-cases of DepthAI:

- Visual assistance (for visually impaired, or for aiding in fork-lift operation, etc.)
- Aerial / subsea drones (fault detection, AI-based guidance/detection/routing)
- E-scooter & micromobility (not allowing folks to ride rented e-scooters like jerks)
- Cargo/transport/autonomy (fullness, status, navigation, hazard avoidance)
- Sports monitoring (automatically losslessly zooming in on action)

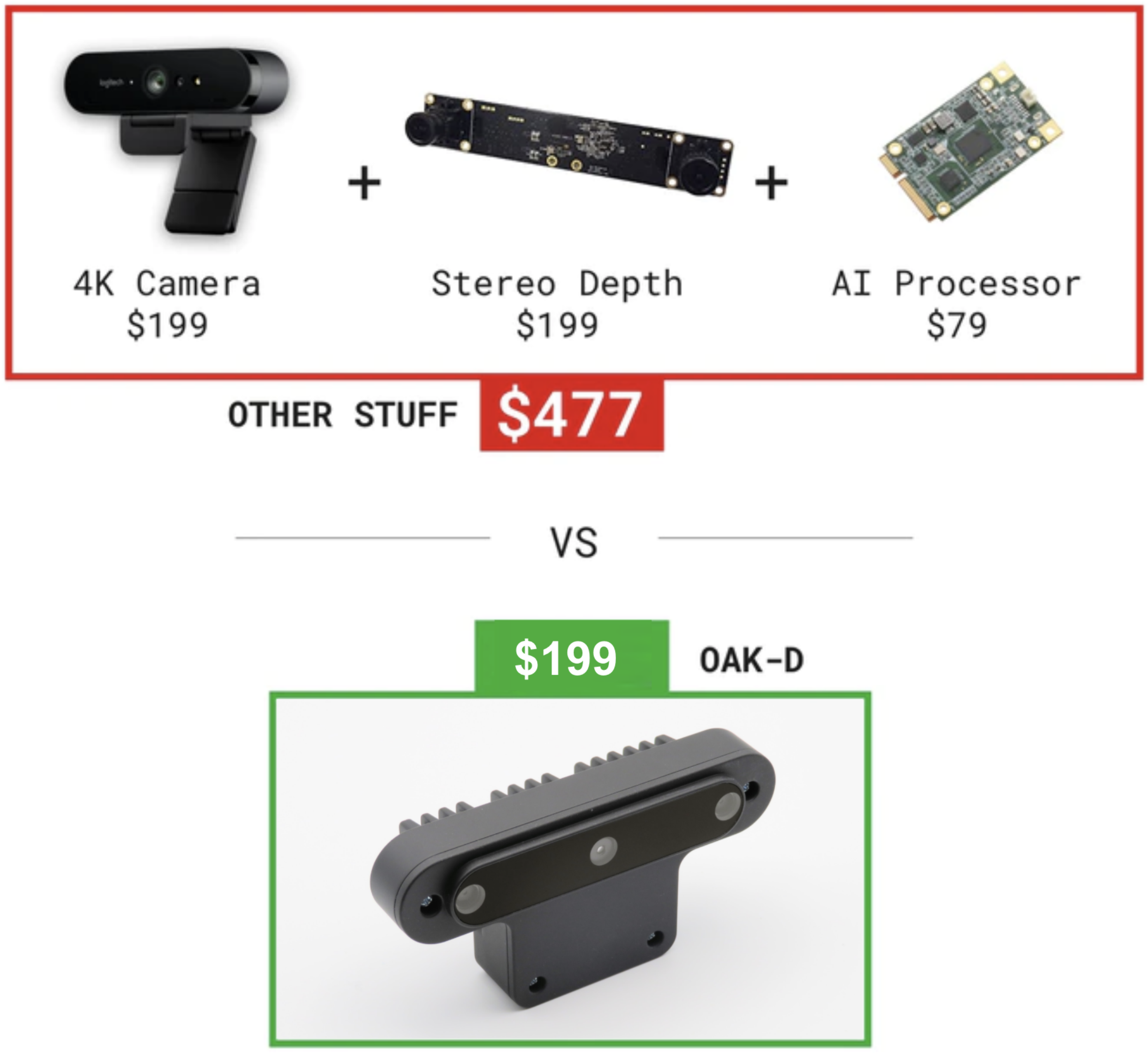
- Smart agriculture (e.g guiding lasers to kill weeds, pests, or targeting watering)

4.7.5 What Distinguishes OAK-D From Other Cameras?

DepthAI purpose is the tight fusion of real-time, hardware-accelerated depth estimation, neural inference, and computer vision into a single, simple to use interface. It is the equivalent of combining a 12MP/4K camera, a stereo depth camera, an AI processor into one product. And to boot, it has accelerated CV capabilities to tie this all together.

So this produces a smaller, lower power, more performant, significantly easier-to-use, and lower-cost solution than what would be otherwise required, which would be to purchase each of these components independently, and do the lifting to physically integrate them and also write the code to combine disparate codebases.

With DepthAI, this is all done for you, and is available in a device that you can buy and plug into a computer (as below) - and also a module (here) with all these capabilities that can be integrated into your product - to allow your products to have these capabilities built-in.



4.7.6 How Does DepthAI Provide Spatial AI Results?

There are two ways to use DepthAI to get Spatial AI results:

1. **Monocular Neural Inference fused with Stereo Depth.** In this mode the neural network is run on a single camera and fused with disparity depth results. The left, right, or RGB camera can be used to run the neural inference.
2. **Stereo Neural Inference.** In this mode the neural network is run in parallel on both the left and right stereo cameras to produce 3D position data directly with the neural network.

In both of these cases, standard neural networks can be used. There is no need for the neural networks to be trained with 3D data.

DepthAI automatically provides the 3D results in both cases using standard 2D-trained networks, as detailed *here*. These modes have differing minimum depth-perception limits, detailed *here*.

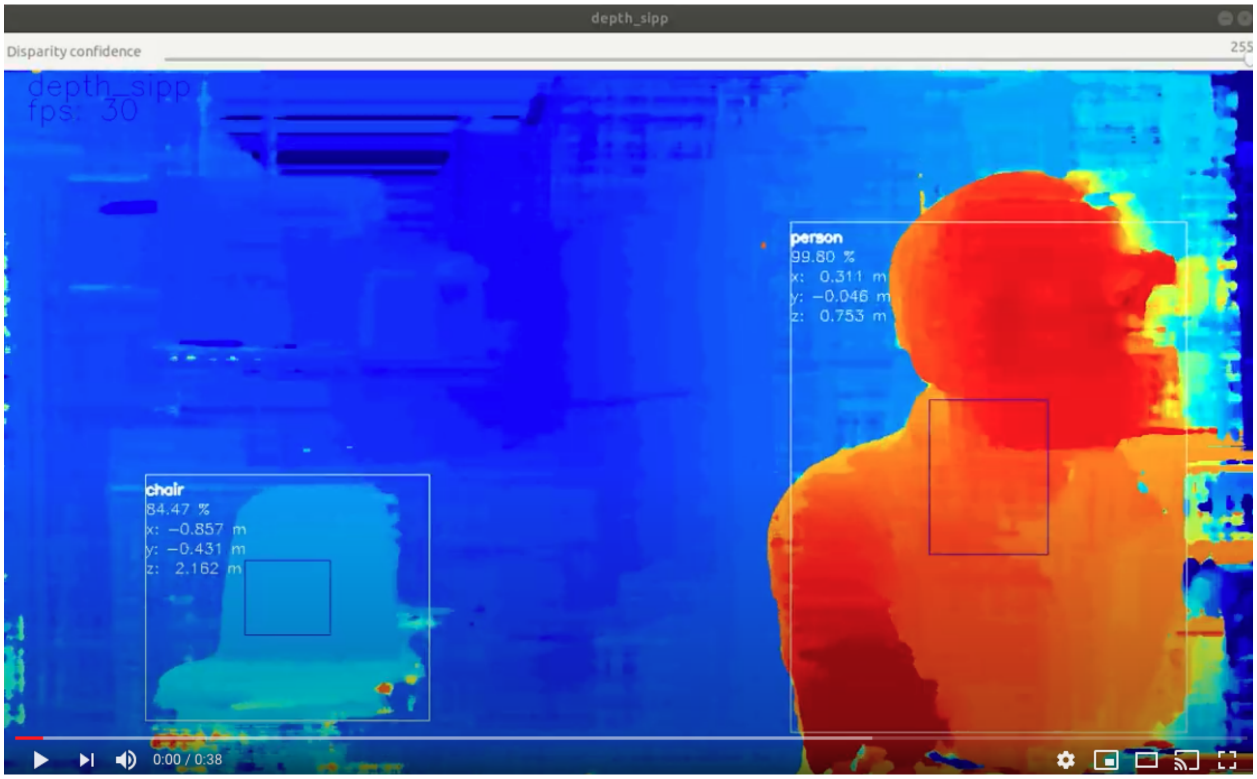
Monocular Neural Inference fused with Stereo Depth

In this mode, DepthAI runs object detection on a single cameras (user's choice: left, right, or RGB) and the results are fused with the stereo disparity depth results. The stereo disparity results are produced in parallel and in real-time on DepthAI (based on semi global matching (SGBM)).

DepthAI automatically fuses the disparity depth results with the object detector results and uses this depth data for each object in conjunction with the known intrinsics of the calibrated cameras to reproject the 3D position of the detected object in physical space (X, Y, Z coordinates in meters).

And all of these calculations are done onboard to DepthAI without any processing load to any other systems. This technique is great for object detectors as it provides the physical location of the centroid of the object - and takes advantage of the fact that most objects are usually many pixels so the disparity depth results can be averaged to produce a more accurate location.

A visualization of this mode is below.



In this case the neural inference (20-class object detection per *here*) was run on the RGB camera and the results were overlaid onto the depth stream.

And if you'd like to know more about the underlying math that DepthAI is using to perform the stereo depth, see this excellent blog post here. And if you'd like to run the same example run in that blog, on DepthAI, see this depthai-experiment.

What is the Max Stereo Disparity Depth Resolution?

The maximum resolution for the depthai depth map is 1280x800 (1MP), with either a 96-pixel (default) or 191-pixel disparity search (when *Extended Disparity* is enabled) and either a full-pixel (default) or sub-pixel matching with precision of 32 sub-pixel steps (when *Sub-Pixel Disparity* is enabled), resulting in a maximum theoretical depth precision of 191 (extended disparity search mode) * 32 (sub-pixel disparity search enabled) of 6,112. More information on the disparity depth modes are below:

1. Default (96-pixel disparity search, **range: [0..95]**): 1280x800 or 640x400, 96 depth steps
2. Extended Disparity (191-pixel disparity search, **range: [0..190]**), *here*: 1280x800 or 640x400, 191 depth steps
3. Subpixel Disparity (32 sub-pixel steps), *here*, 1280x800 or 640x400, 96 depth steps * 32 subpixel depth steps = 3,072 depth steps.
4. LR-Check Disparity, *here*: 1280x800, with disparity run in both directions for allowing recentering of the depth.

(see *Extended Disparity* below)

Notes

It is worth noting that monocular neural inference fused with stereo depth is possible for networks like facial-landmark detectors, pose estimators, etc. that return single-pixel locations (instead of for example bounding boxes of semantically-labeled pixels), but stereo neural inference is advised for these types of networks better results as unlike object detectors (where the object usually covers many pixels, typically hundreds, which can be averaged for an excellent depth/position estimation), landmark detectors typically return single-pixel locations. So if there doesn't happen to be a good stereo-disparity result for that single pixel, the position can be wrong.

And so running stereo neural inference excels in these cases, as it does not rely on stereo disparity depth at all, and instead relies purely on the results of the neural network, which are robust at providing these single pixel results. And triangulation of the parallel left/right outputs results in very-accurate real-time landmark results in 3D space.

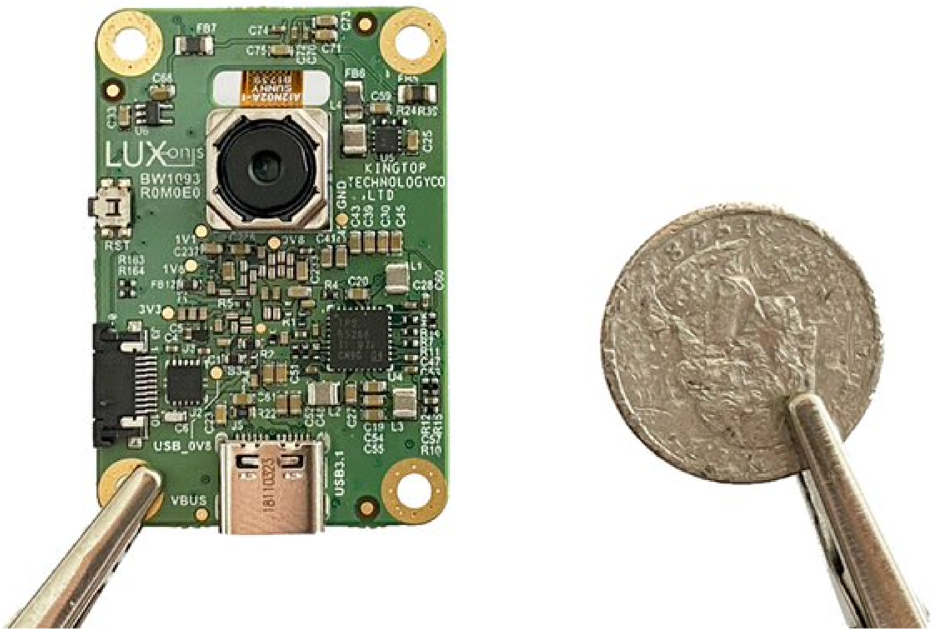
4.7.7 What is the Gen2 Pipeline Builder?

UPDATE: The Gen2 Pipeline Builder is now the standard release of DepthAI. This Gen2 API system was architected to be next-generation software suite for DepthAI and OAK. All DepthAI and OAK hardware work with Gen1 and Gen2 software, as Gen2 is purely a software re-write, no hardware changes. Gen2 is infinitely more flexible, and is the result of all that we learned from the customer deployments of Gen1. Amassing all the requests and need for flexibility from users of Gen1, we made Gen2. In short, Gen2 allows theoretically-infinite permutations of parallel and series CV + AI (neural inference) nodes, limited only by hardware capabilities, whereas Gen1 was limited for example to 2-series and 2-parallel neural inference. Full background on the Gen2 Pipeline Builder is here.

Several Gen2 Examples are here and also the docs for Gen2 are now available in the main docs page.

4.7.8 What is megaAI?

OAK-1 (previously MegaAI) is the single-camera version of OAK-D. Because not all solutions to embedded AI/CV problems require spatial information.



OAK-1 uses all the same hardware, firmware, software, and training stacks as the OAK-D (and uses the same DepthAI Github repositories), it is simply the tiny single-camera variant.

More details can be found here.

4.7.9 Which Model Should I Order?

Embedded CV/AI requires all sorts of different shapes/sizes/permutations. And so we have a variety of options to meet these needs in our store. Below is a quick/dirty summary for the ~10,000-foot view of the options:

- **USB3C with Onboard Cameras and Depth** (OAK-D) - Great for quickly using DepthAI with a computer. All cameras are onboard, and it has a USB3C connection which can be used with any USB3 or USB2 host.
- **USB3C with Single Camera** (OAK-1) - This is just like the OAK-D, but for those who don't need depth information. Single, small, plug-and-play USB3C AI/CV camera.
- **USB3C with Modular Cameras** (OAK-FFC-3P) - Great for prototyping flexibility. Since the cameras are modular, you can place them at various stereo baselines. This flexibility comes with a trade - you have to figure out how/where you will mount them, and then once mounted, do a stereo calibration. This is not a TON of work, but keep this in mind, that it's not 'plug and play' like other options - it's more for applications that require custom mounting, custom baseline, or custom orientation of the cameras.
- **PoE models** (OAK-D-PoE) - It is the equivalent of the OAK-D, with PoE instead of USB. If you don't need depth, we have OAK-1-PoE.
- **All in One Dev. Kits** (OAK-D-CM4) - this one has a built-in Raspberry Pi Compute Module 4. So you literally plug it into power and HDMI, and it boots up showing off the power of DepthAI.

More products in store.

More details - including open source 3D files and schematics, can be found in hardware documentation.

System on Modules

For designing products around DepthAI, we offer system on modules. You can then design your own variants, leveraging our open source hardware. There are three system on modules available:

1. OAK-SoM - USB-boot system on module. For making devices which interface over USB to a host processor running Linux, MacOS, or Windows. In this case, the host processor stores everything, and the OAK-SoM boots up over USB from the host.
2. OAK-SoM-IoT - NOR-flash boot (also capable of USB-boot). For making devices that run standalone, or work with embedded MCUs like ESP32, AVR, STM32F4, etc. Can also USB-boot if/as desirable.
3. OAK-SoM-Pro - NOR flash, eMMC, SD-Card, and USB-boot (selectable via IO on the 2x 100-pin connectors). For making devices that run standalone and require onboard storage (16GB eMMC) and/or Ethernet Support (the onboard PCIE interface through one of the 2x 100-pin connectors, paired with an Ethernet-capable base-board provides Ethernet support).

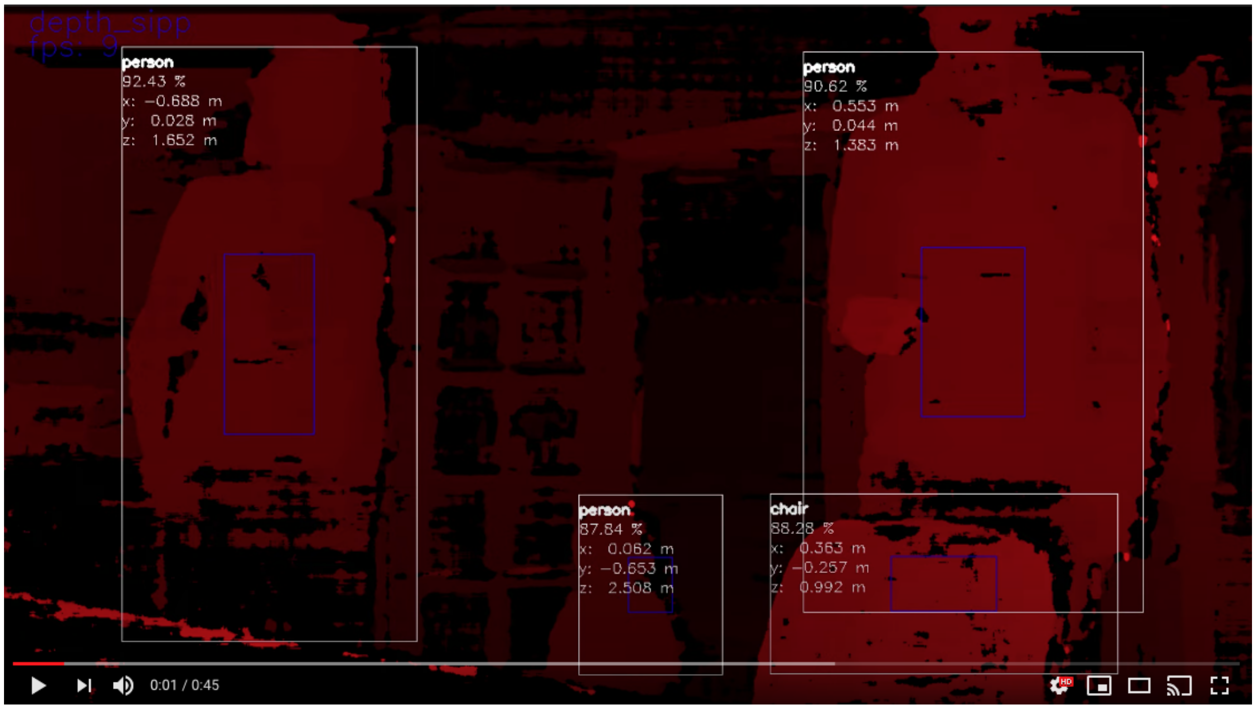
Check our hardware documentation for more details.

4.7.10 How hard is it to get DepthAI running from scratch? What Platforms are Supported?

Not hard. Usually DepthAI is up/running on your platform within a couple minutes (most of which is download time). The requirements are Python and OpenCV (which are great to have on your system anyway!). see here for supported platforms and how to get up/running with them.

Raspbian, Ubuntu, macOS, Windows, and many others are supported and are easy to get up/running. For the list of supported platforms (and instructions on how to get started), click here.

It's a matter of minutes to be up and running with the power of Spatial AI, on the platform of your choice. Below is DepthAI running on my Mac.

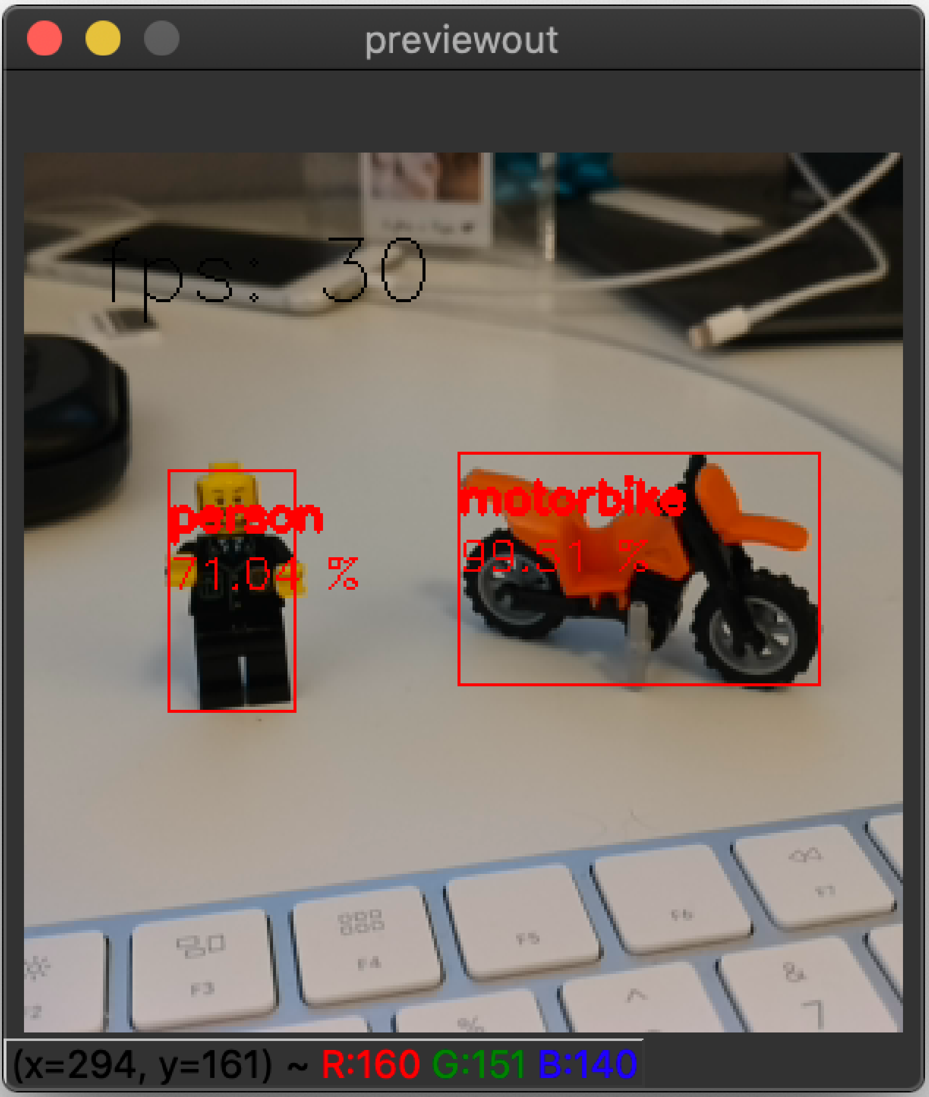


(Click on the image above to pull up the YouTube video.)

The command to get the above output is

```
python3 depthai_demo.py -gt cv -s color depth -sbb
```

Here is a single-camera version (OAK-1) running with `python3 depthai_demo.py -gt cv -s color`:


previewout
fps: 30
person
71.04 %
motorbike
99.51 %
(x=294, y=161) ~ R:160 G:151 B:140

4.7.11 Is OAK camera easy to use with Raspberry Pi?

Very. It's designed for ease of setup and use, and to keep the Pi CPU not-busy. You can find additional information here.

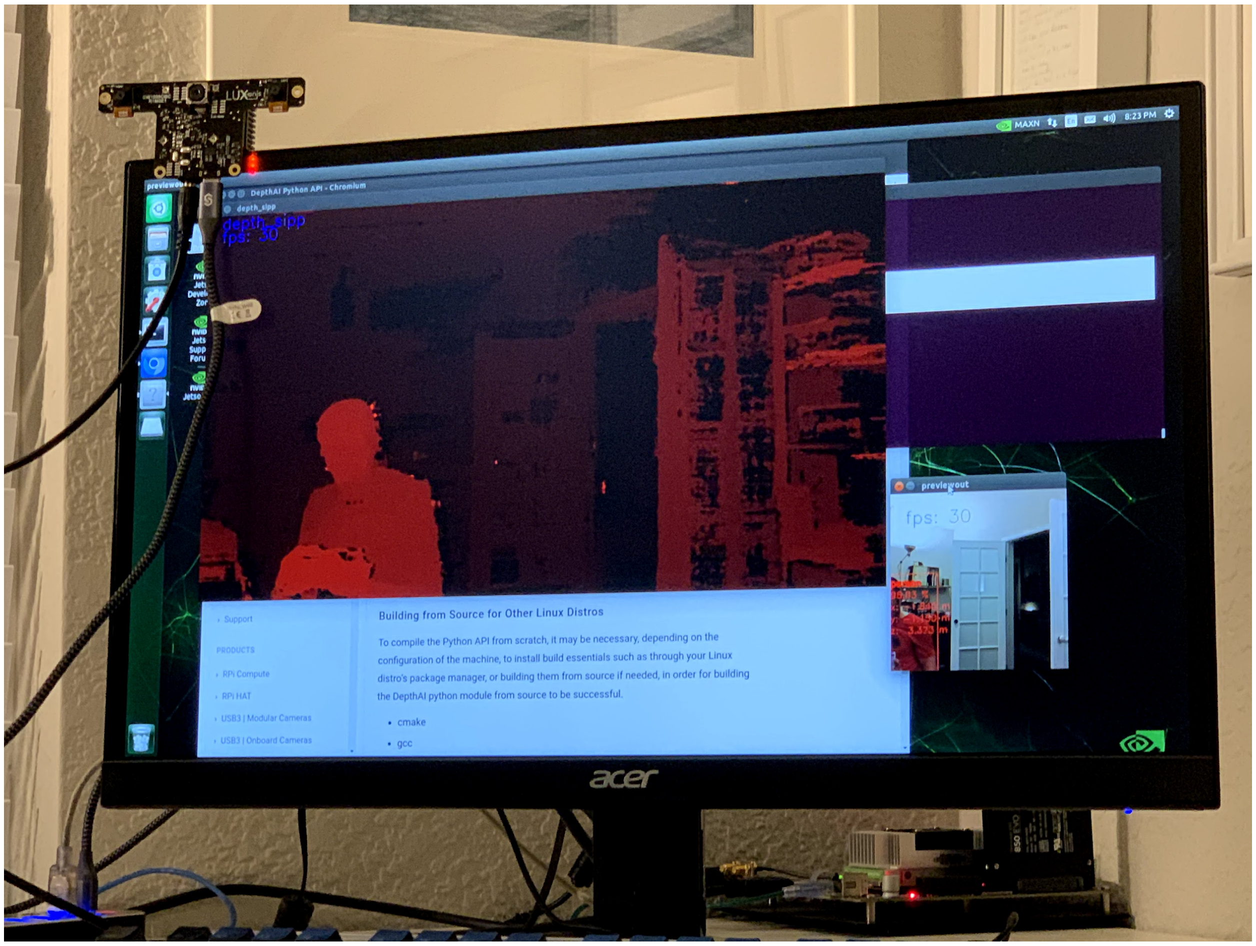
4.7.12 Can all the models be used with the Raspberry Pi?

Yep! All the models can be used with the Raspberry Pi. The only difference is that the Raspberry Pi is not as powerful as a desktop computer. You can find additional information here.

4.7.13 Does DepthAI Work on the Nvidia Jetson Series?

Yes, DepthAI works cleanly on all the Jetson/Xavier series, and installation is easy. Jetson Nano, Jetson Tx1, Jetson Tx2, Jetson Xavier NX, Jetson AGX Xavier, etc. are all supported.

See below for DepthAI running on a Jetson Tx2 I have on my desk:



Installing for NVIDIA Jetson and Xavier is now the same set of instructions as Ubuntu. See here and following the standard Ubuntu instructions.

Also don't forget about the udev rules after you have that set up. And make sure to unplug and replug your depthai after having run the following commands (this allows Linux to execute the modification of the USB rules).

```
echo 'SUBSYSTEM=="usb", ATTRS{idVendor}=="03e7", MODE="0666"' | sudo tee /etc/udev/
↪rules.d/80-movidius.rules
sudo udevadm control --reload-rules && sudo udevadm trigger
```

4.7.14 Can I Use Multiple DepthAI With One Host?

Yes. DepthAI is architected to put as-little-as-possible burden on the host. So even with a Raspberry Pi you can run a handful of DepthAI with the Pi and not burden the Pi CPU.

See here for instructions on how to do so.

4.7.15 Is DepthAI OpenVINO Compatible?

Yes, DepthAI is fully compatible with OpenVINO.

4.7.16 Can I Train My Own Models for DepthAI?

Yes.

We have a tutorial around Google Colab notebooks you can even use for this. See here

4.7.17 Do I Need Depth Data to Train My Own Custom Model for DepthAI?

No.

That's the beauty of DepthAI. It takes standard object detectors (2D, pixel space) and fuses these neural networks with stereo disparity depth to give you 3D results in physical space.

Now, could you train a model to take advantage of depth information? Yes, and it would likely be even more accurate than the 2D version. To do so, record all the streams (left, right, and color) and retrain on all of those (which would require modifying the front-end of say MobileNet-SSD to allow 5 layers instead of 3 (1 for each grayscale, 3 for the color R, G, B)).

4.7.18 If I train my own network, which Neural Operations are supported by DepthAI?

See the `VPU` section here.

Anything that's supported there under `VPU` will work on DepthAI. It's worth noting that we haven't tested all of these permutations though.

4.7.19 What network backbones are supported on DepthAI?

All the networks listed here are supported by DepthAI.

We haven't tested all of them though. So if you have a problem, contact us and we'll figure it out.

4.7.20 My Model Requires Pre-Processing (normalization, for example). How do I do that in DepthAI?

The OpenVINO toolkit allows adding these pre-processing steps to your model, and then these steps are performed automatically by DepthAI. See here for how to take advantage of this.

For instance, to scale frame pixels to the range [0,1], consider adding the following parameters to the model optimizer: `--data_type=FP16 --scale_values [255,255,255]`

To scale to the range [-1, 1], mean values should be added, e.g. for mobilenet: `--scale_values [127.5, 127.5, 127.5] --mean_values [127.5, 127.5, 127.5]`

More model converting options here

4.7.21 Can I Run Multiple Neural Models in Parallel or in Series (or Both)?

Yes. The Gen2 Pipeline Builder is what allows you to do this. And we have several example implementations of parallel, series, and parallel+series in depthai-experiments repository. A notable example is the Gaze estimation example, here, which shows series and parallel all together in one example.

4.7.22 Can DepthAI do Arbitrary Crop, Resize, Thumbnail, etc.?

Yes, see here for an example of how to do this, with WASD controls of a cropped section. And see here for extension of the cropping for non-rectangular crops, and warping those to be rectangular (which can be useful for OCR).

4.7.23 Can DepthAI Run Custom CV Code? Say CV Code From PyTorch?

Yes, see *documentation here*.

4.7.24 How do I Integrate DepthAI into Our Product?

How to integrate DepthAI depends on whether the product you are building includes:

1. a processor running an operating system (Linux, MacOS, or Windows) or
2. a microcontroller (MCU) with no operating system (or an RTOS like FreeRTOS) or
3. no other processor or microcontroller (i.e. DepthAI is the only processor in the system).

We offer hardware to support all 3 use-cases, but firmware/software maturity varies across the 3 modes:

1. Using our Python API and/or C++ API (equal capabilities)
2. Using our C++ SPI API (see here),
3. Using our standalone flashing utility to flash a depthai application for standalone mode (leveraging our SBR Util here).

In all cases, DepthAI is compatible with OpenVINO for neural models. The only thing that changes between the modalities is the communication (USB, Ethernet, SPI, etc.) and what (if any) other processor is involved.

Use-Case 1: DepthAI are a co-processor to a processor running Linux, MacOS, or Windows.

In this case, OAK camera can be used in two modalities:

- DepthAI Mode (USB, using **depthai API**, here) - this uses the onboard cameras directly into the VPU, and boots the firmware over USB from a host processor running Linux, Mac, or Windows. This is the main use-case of DepthAI/megaAI when used with a host processor capable of running an operating system (e.g Raspberry Pi, i.MX8, etc.).
- NCS2 Mode (USB, *here*) - in this mode, the device appears as an NCS2 and the onboard cameras are not used and it's as if they don't exist. This mode is often use for initial prototyping, and in some cases, where a product simply needs an 'integrated NCS2' - accomplished by integrating a OAK-SoM.

Use-Case 2: Using DepthAI with a MicroController like ESP32, ATTiny8, etc.

In this case, DepthAI boots off of internal flash on the OAK-SoM-IoT and communicates over SPI, allowing DepthAI to be used with microcontroller such as the STM32, MSP430, ESP32, ATMega/Arduino, etc.

Use-Case 3: Using DepthAI as the Only Processor on a Device.

This is supported through running Python directly on the OAK-SoM-Pro or OAK-SoM-IoT inside Script node.

The Script node allows custom logic, driving GPIO, UART, network protocols etc., letting direct controls of actuators, direct reading of sensors, etc. from/to the pipeline of CV/AI functions. A target example is making an entire autonomous, visually-controlled robotic platform with DepthAI as the only processor in the system.

Hardware for Each Case:

- OAK-SoM: USB boot. So it is intended for working with a host processor running Linux, Mac, or Windows and this host processor boots the OAK-SoM over USB
- OAK-SoM-IoT: USB boot or NOR-flash boot. This module can work with a host computer just like the OAK-SoM, but also has a 128MB NOR flash built-in and boot switches onboard - so that it can be programmed to boot off of NOR flash instead of USB. So this allows use of the DepthAI in pure-embedded applications where there is no operating system involved at all. So this module could be paired with an ATTiny8 for example, communicating over SPI.
- OAK-SoM-Pro: Supports multiple boot options: NOR (128MB), eMMC (SD-Card support), USB, Ethernet (EEPROM, 32KB). All those boot options make OAK-SoM-Pro very flexible in terms of use cases and most appropriate as a standalone device. It is designed for integration into a top-level system with a need for a low power AI vision system.

Getting Started with Development

Whether intending to use DepthAI with an *OS-capable host* or *completely standalone* - we recommend starting with the DepthAI API for prototype/test/etc. as it allows faster iteration/feedback on neural model performance/etc.

In DepthAI mode, theoretically, anything that will run in NCS2 mode will run - but sometimes it needs host-side processing if it's a network we've never run before. And this work is usually not heavy lifting. See some examples here and in out Github.

For common object detector formats (**MobileNet**-SSD, (Tiny) **YOLO** V3/V4) there's effectively no work to go from NCS2 mode to DepthAI mode because we have added the support for decoding their results on the device side. To use the device side decoding with gen2, have a look at YoloDetectionNetwork for **YOLO** (demo here) or MobileNetDetectionNetwork for **MobileNet** (demo here) decoding.

To use your own trained **Yolo** model with the DepthAI, you should start with the demo and modify its code a bit:

- Change the labels at `labelMap = ["label1", "label2", "..."]`, depending on your model
- Set the number of classes at `detectionNetwork.setNumClasses()` depending on your model
- If you haven't compiled the model with the latest OpenVINO version, set the *OpenVINO version*
- Don't forget to change the path to the model (`.blob` file)

For **MobileNet** you should follow the same steps (skip the 2nd one) but start with the MobileNet demo.

Interested in how to train an object detector with your data? You can check our **Yolo V4** training tutorial here!

4.7.25 What Hardware-Accelerated Capabilities Exist in DepthAI and/or megaAI?

The DepthAI system is a node-and-graph pipeline builder. Below are the hardware-accelerated nodes that exist in this builder.

Available in DepthAI API Today:

- Neural Inference Node, which is compatible with OpenVINO (e.g. object detection, image classification, etc., including multi-stage inference, e.g. here and here)
- Stereo Depth (including median filtering) (e.g. here)
- Stereo Inference (with two-stage, e.g. here)
- 3D Object Localization (augmenting 2D object detectors with 3D position in meters, e.g. here and here)
- Object Tracking (e.g. here, including in 3D space)
- H.264 and H.265 Encoding (HEVC, 1080p & 4K Video, e.g. here)
- JPEG Encoding (e.g. here)
- MJPEG Encoding
- Warp/Dewarp (for RGB-depth alignment/etc.)
- Enhanced Disparity Depth Modes (Sub-Pixel, LR-Check, and Extended Disparity), here
- SPI Support, here
- Arbitrary crop/rescale/reformat and ROI return (e.g. here)
- Integrated Text Detection (e.g. here)
- Pipeline Builder Gen2 (arbitrary series/parallel combination of neural nets and CV functions, background here and API documentation is here).
- Lossless zoom (from 12MP full to 4K, 1080p, or 720p, here)
- Improved Stereo Neural Inference Support (here)
- Integrated IMU Support (here)
- Edge Detection (here, video)
- On-Device Python Scripting Support, here
- Feature Tracking (here, video)

The above features are available in the Luxonis Pipeline Builder Gen2 which is now the main API for DepthAI. The Gen1 API is still supported, and can be accessed via the version switcher at the bottom left of this page. See below for in-progress additional functionality/flexibility which will be added as modular nodes to the Luxonis pipeline builder for DepthAI.

On our Roadmap (Most are in development/integration)

- Motion Estimation (here)
- Background Subtraction (here)
- OpenCL Support (supported through OpenVINO (here))

And see our Github project here to follow along with the progress of these implementations.

Pipeline Builder Gen2

The 2nd-generation DepthAI pipeline builder which incorporates all the feedback we learned from our first Generation API. It is now the mainline way to use DepthAI.

It allows multi-stage neural networks to be pieced together in conjunction with CV functions (such as motion estimation or Harris filtering) and logical rules, all of which run on DepthAI/megaAI/OAK without any load on the host.

4.7.26 Are CAD Files Available?

Yes.

The full designs (including source Altium files) for all the carrier boards are in our depthai-hardware Github.

4.7.27 How to enable depthai to perceive closer distances

If the depth results for close-in objects look weird, this is likely because they are below the minimum depth-perception distance of OAK-D.

For OAK-D, the standard-settings minimum depth is around 70cm.

This can be cut in 1/2 and 1/4 with the following options:

1. Change the resolution to 640x400, instead of the standard 1280x800.
2. Enable Extended Disparity.

See these examples for how to enable Extended Disparity.

For more information see the StereoDepth documentation.

4.7.28 What are the Minimum Depths Visible by DepthAI?

There are two ways to use DepthAI for 3D object detection and/or using neural information to get real-time 3D position of features (e.g. facial landmarks):

1. Monocular Neural Inference fused with Stereo Depth
2. Stereo Neural Inference

Monocular Neural Inference fused with Stereo Depth

In this mode, the AI (object detection) is run on the left, right, or RGB camera, and the results are fused with stereo disparity depth, based on semi global matching (SGBM). The minimum depth is limited by the maximum disparity search, which is by default 96, but is extendable to 191 in extended disparity modes (see *Extended Disparity* below).

To calculate the minimum distance in this mode, use the following formula:

```
min_distance = focal_length_in_pixels * base_line_dist / max_disparity_in_pixels
```

Where the focal_length_in_pixels is (HFOV of the grayscale global shutter cameras is 71.9 degrees):

```
focal_length_in_pixels = 1280 * 0.5 / tan(71.9 * 0.5 * PI / 180) = 882.5
```

Calculation here (and for disparity depth data, the value is stored in `uint16`, where 0 is a special value, meaning that distance is unknown.)

By using the formula above with the default settings of OAK-D (base_line_dist = **7.5cm**, max_disparity_in_pixels = **95**), we get:

```
min_distance = 882.5 * 7.5cm / 95 = 69.67cm
```

Note that this distance can be halved by either:

- Changing the resolution to 640x400, instead of the standard 1280x800.
- Enabling Extended Disparity - see these examples for how to enable Extended Disparity.

Extended disparity mode sets the max_disparity_in_pixels to **190**, thus the min_distance for the above OAK-D example is:

```
min_distance = 882.5 * 7.5cm / 190 = 34.84cm
```

Note that applying both options is possible, but at such short distances, the minimum distance is limited by focal length, which is 19.6cm, so minimum distance cannot be lower than 19.6cm.

Calculation examples for OAK-D:

- ~ 70cm with standard disparity (1280x800 resolution)
- ~ 35cm with extended disparity (1280x800 resolution)
- ~ 35cm with 640x400 resolution
- ~ 19.6cm with extended disparity and 640x400 resolution

For a more detailed explanation refer to the StereoDepth documentation.

Onboard Camera Minimum Depths

Below are the minimum depth perception possible in the disparity depth and stereo neural inference modes.

Monocular Neural Inference fused with Stereo Depth Mode

For DepthAI units with onboard cameras, this works out to the following minimum depths:

- OAK-D-CM4 the minimum depth is **0.836** meters for full 1280x800 stereo resolution and **0.418** meters for 640x400 stereo resolution:

```
min_distance = 882.5 * 0.09 / 95 = 0.836 # m
```

calculation here

- OAK-D is
  - **0.697** meters for standard disparity,
  - **0.348** meters for Extended Disparity (191 pixel) at 1280x800 resolution or standard disparity at 640x400 resolution, and
  - **0.196** meters for Extended Disparity at 640x400 resolution (this distance is limited by the focal distance of the cameras on OAK-D)

```
min_distance = 882.5 * 0.075 / 95 = 0.697 # m
```

calculation here

Stereo Neural Inference Mode

For DepthAI units with onboard cameras, all models (OAK-D-CM4 and OAK-D) are limited by the hyperfocal distance of the stereo cameras, so their minimum depth is **0.196** meters.

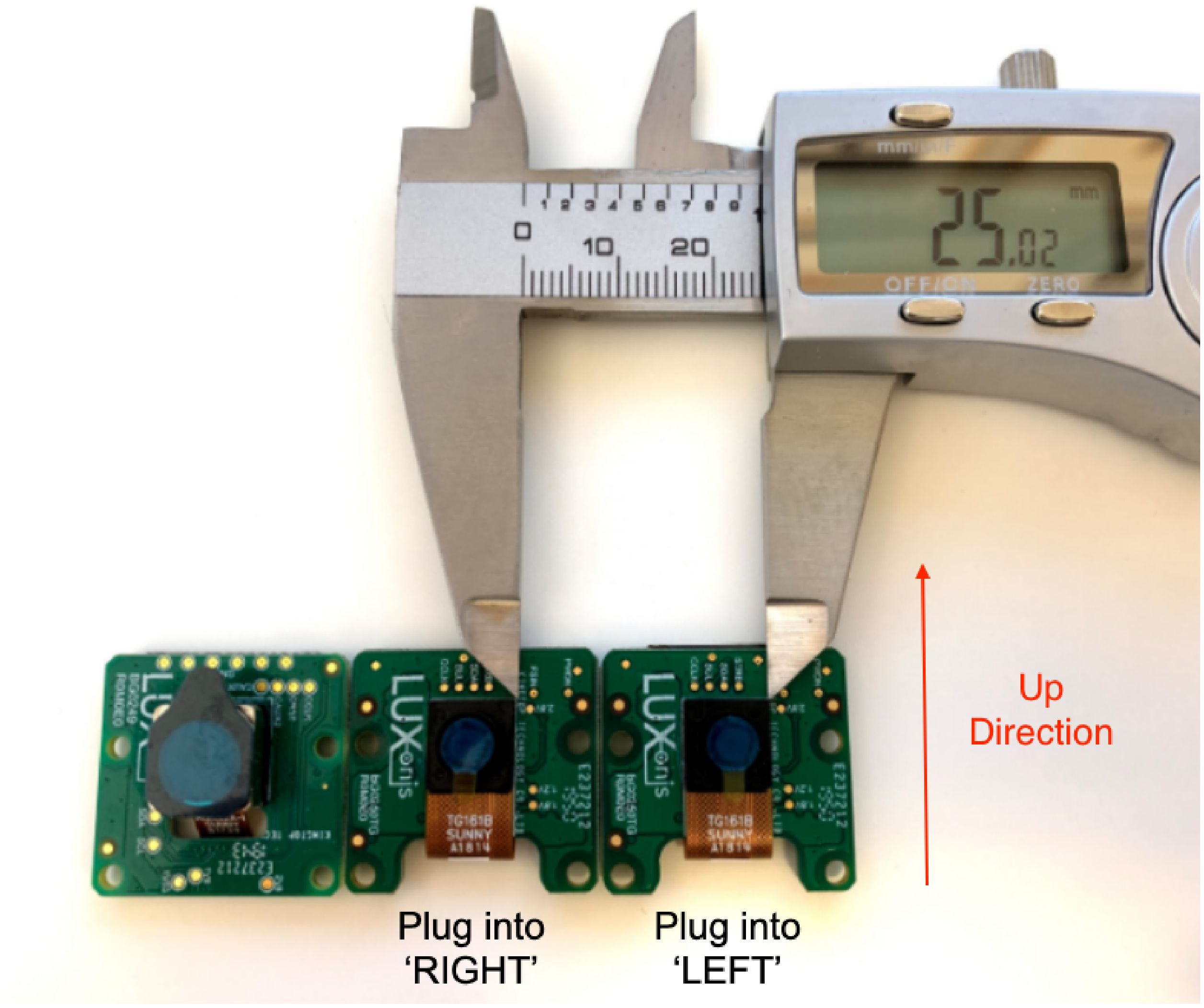
Modular Camera Minimum Depths:

Below are the minimum depth perception possible in the disparity depth and stereo neural inference modes.

Monocular Neural Inference fused with Stereo Depth Mode

For DepthAI units which use modular cameras, the minimum baseline is 2.5cm (see image below) which means the minimum perceivable depth **0.229** meters for full 1280x800 resolution and **0.196** meters for 640x400 resolution (limited by the minimum focal distance of the grayscale cameras, as in stereo neural inference mode).

The minimum baseline is set simply by how close the two boards can be spaced before they physically interfere:



For any stereo baseline under 29 cm, the minimum depth is dictated by the hyperfocal distance (the distance above which objects are in focus) of 19.6cm.

For stereo baselines wider than 29 cm, the minimum depth is limited by the horizontal field of view (HFOV):

```
min_distance = tan((90-HFOV/2)*pi/2)*base_line_dist/2
```

Extended Disparity Depth Mode

The `extended disparity` mode affords a closer minimum distance for the given baseline. This increases the maximum disparity search from 96 to 191. So this cuts the minimum perceivable distance in half (given that the minimum distance is now `focal_length * base_line_dist / 190` instead of `focal_length * base_line_dist / 95`).

- OAK-D-CM4: **0.414** meters
- OAK-D is **0.345** meters
- OAK-FFC-3P-OG is **0.115** meters

See here for examples of how to use Extended Disparity Mode.

And for a bit more background as to how this mode is supported:

Extended disparity: allows detecting closer distance objects, without compromising on long distance values (integer disparity) by running the following flow.

1. Computes disparity on the original size images (e.g. 1280x720)
2. Computes disparity on 2x downscaled images (e.g. 640x360)
3. Combines the two level disparities on Shave, effectively covering a total disparity range of 191 pixels (in relation to the original resolution).

Left-Right Check Depth Mode

Left-Right Check, or LR-Check is used to remove incorrectly calculated disparity pixels due to occlusions at object borders (Left and Right camera views are slightly different).

1. Computes disparity by matching in R->L direction
2. Computes disparity by matching in L->R direction
3. Combines results from 1 and 2, running on Shave: each pixel d = disparity_LR(x,y) is compared with disparity_RL(x-d,y). If the difference is above a threshold, the pixel at (x,y) in final disparity map is invalidated.

To run LR-Check on DepthAI/OAK, use the example here.

4.7.29 What Are The Maximum Depths Visible by DepthAI?

The max depth perception is limited by the physics of the baseline and the number of pixels (see documentation here).

Each OAK camera has max depth perception specified in its hardware documentation page.

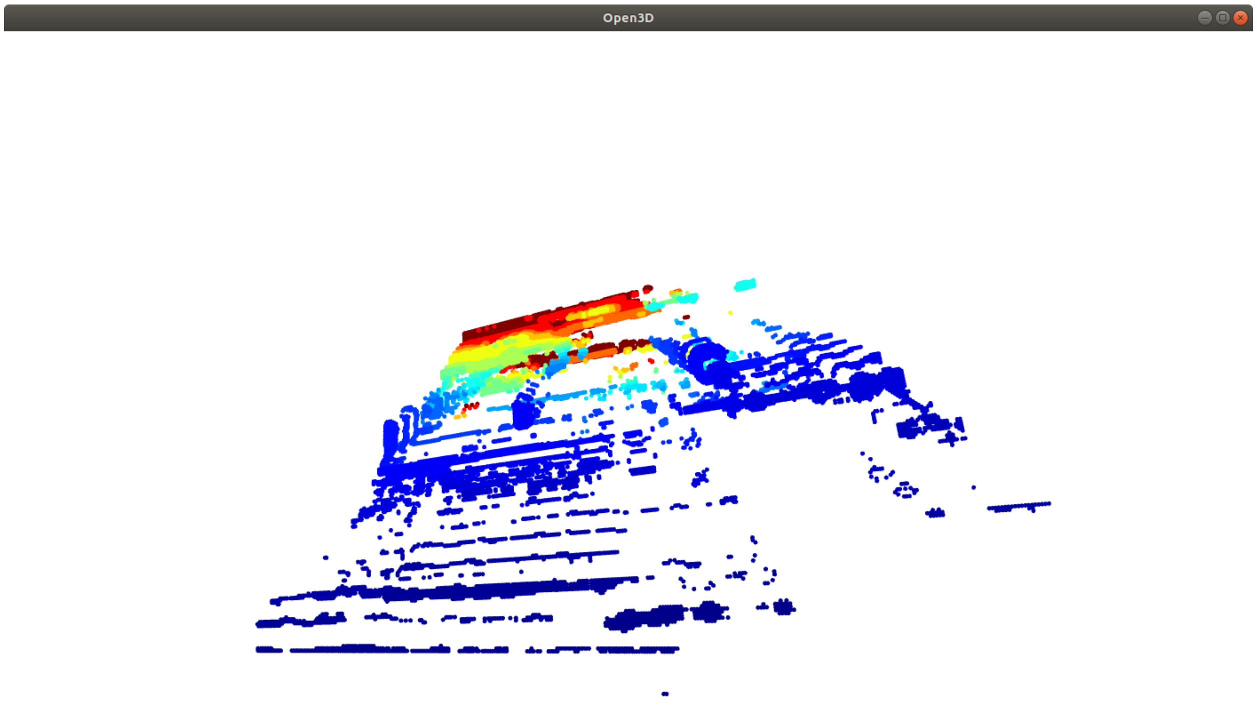
Subpixel Disparity Depth Mode

Subpixel improves the precision and is especially useful for long range measurements. It also helps for better estimating surface normals (comparison of normal disparity vs. subpixel disparity is here).

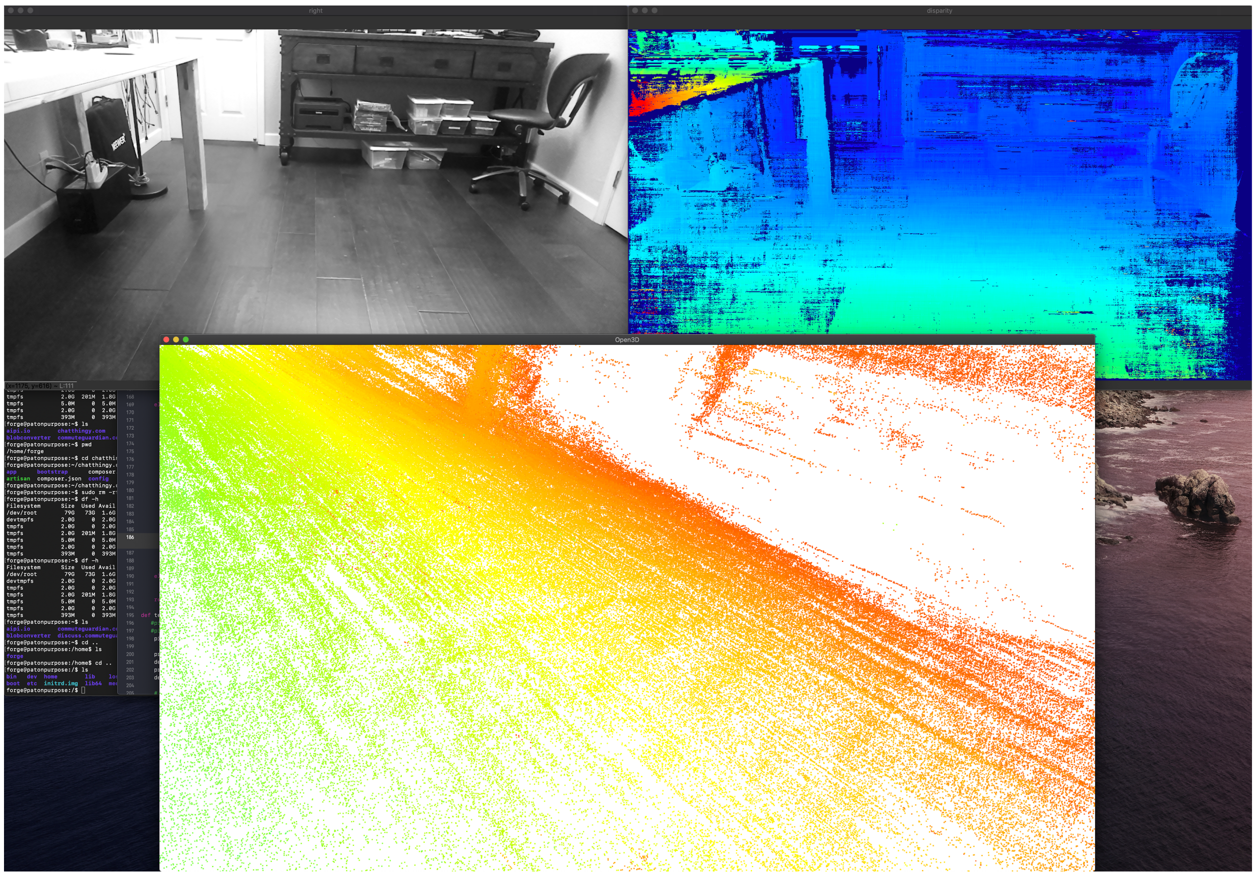
Beside the integer disparity output, the Stereo engine is programmed to dump to memory the cost volume, that is 96 bytes (disparities) per pixel, then software interpolation is done on Shave, resulting a final disparity with 5 fractional bits, resulting in significantly more granular depth steps (32 additional steps between the integer-pixel depth steps), and also theoretically, longer-distance depth viewing - as the maximum depth is no longer limited by a feature being a full integer pixel-step apart, but rather 1/32 of a pixel.

Examples of the difference in depth steps from standard disparity to subpixel disparity are shown below:

Standard Disparity (96 depth steps):



Subpixel Disparity (3,072 depth steps):



```
tmpfs            2.0G  201M  1.8G
tmpfs            5.0M     0  5.0M
tmpfs            2.0G     0  2.0G
tmpfs            393M     0  393M
forge@patonpurpose:~$ ls
aipi.io        chatthingy.com
blobconverter  commuteguardian.c
forge@patonpurpose:~$ pwd
/home/forge
forge@patonpurpose:~$ cd chatthi
forge@patonpurpose:~/chatthingy.
app      bootstrap     composer
artisan  composer.json  config
forge@patonpurpose:~/chatthingy.
forge@patonpurpose:~$ sudo rm -r
forge@patonpurpose:~$ df -h
Filesystem      Size  Used Avail
/dev/root        79G   73G  1.6G
devtmpfs         2.0G     0  2.0G
tmpfs            2.0G     0  2.0G
tmpfs            2.0G  201M  1.8G
tmpfs            5.0M     0  5.0M
tmpfs            2.0G     0  2.0G
tmpfs            393M     0  393M
forge@patonpurpose:~$ df -h
Filesystem      Size  Used Avail
/dev/root        79G   73G  1.6G
devtmpfs         2.0G     0  2.0G
tmpfs            2.0G     0  2.0G
tmpfs            2.0G  201M  1.8G
tmpfs            5.0M     0  5.0M
tmpfs            2.0G     0  2.0G
tmpfs            393M     0  393M
forge@patonpurpose:~$ ls
aipi.io        commuteguardian.c
blobconverter  discuss.commutegu
forge@patonpurpose:~$ cd ..
forge@patonpurpose:/home$ ls
forge
forge@patonpurpose:/home$ cd ..
forge@patonpurpose:/$ ls
bin   dev  home        lib    lo
boot  etc  initrd.img  lib64  me
forge@patonpurpose:/$
```

To run Subpixel on DepthAI/OAK, use the example here.

4.7.30 How Does DepthAI Calculate Disparity Depth?

DepthAI makes use of a combination of hardware-blocks (a semi-global-matching disparity (SGBM) hardware block) as well as accelerated vector processing code in the SHAVES of the Myriad X to produce the disparity depth.

The SGBM hardware-block can process up to 1280x800 pixels, this is its hardware limit. Using higher-resolution sensors is technically possible via downscaling. So for example, using the 12MP color camera with the 1280x800 grayscale camera is possible (and has been prototyped by some users with the Gen2 pipeline builder). Or 2x 12MP image sensors could be used for depth (theoretically). But in both cases, the image data needs to be either decimated down to 1280x800, or converted in some other way (e.g. selectively cropped/windowed).

What Disparity Depth Modes are Supported?

See **Stereo Mode** tab on SterepDepth documentation.

4.7.31 How Do I Calculate Depth from Disparity?

See StereoDepth documentation.

4.7.32 How Do I Display Multiple Streams?

To specify which streams you would like displayed, use the `-s` option. For example for the raw disparity map (`disparity`), and for depth results (`depthRaw`), use the following command:

```
python3 depthai_demo.py -gt cv -s disparity depthRaw
```

The available streams are:

- `nnInput` - Neural Network passthrough frames on which inference was made on (300x300 in case of MobileNet)
- `color` - 4K color camera, biggest camera on the board with lens
- `left` - Left grayscale camera (marked *L* or *LEFT* on the board)
- `right` - Right grayscale camera (marked *R* or *RIGHT* on the board)
- `rectifiedLeft` - Rectified left camera frames
- `rectifiedRight` - Rectified right camera frames
- `depth` - Depth in *uint16*
- `depthRaw` - Raw frames which are used to calculate depth
- `disparity` - Raw disparity
- `disparityColor` - Disparity colorized on the host (`JET` colorized visualization of depth)

Is It Possible to Have Access to the Raw Stereo Pair Stream on the Host?

Yes, you can access raw stereo pair streams using the depthai (API) library. We have an example code here.

4.7.33 How do I Synchronize Streams and/or Meta Data (Neural Inference Results)

When running heavier stereo neural inference, particularly with high host load, this system can break down, and there are two options which can keep synchronization:

1. Reduce the frame rate of the cameras running the inference to the speed of the neural inference itself, or just below it.
2. Or pull the timestamps or sequence numbers from the results (frames or metadata) and match them on the host.

See the demo here.

Reducing the Camera Frame Rate

In the case of neural models which cannot be executed at the full 30FPS, this can cause lack of synchronization, particularly if stereo neural inference is being run using these models in parallel on the left and right grayscale image sensors.

A simple/easy way to regain synchronization is to reduce the frame rate to match, or be just below, the frame rate of the neural inference. This can be accomplished via the command line with the using `-rgbf` and `-monof` commands.

So for example to run a default model with both the RGB and both grayscale cameras set to 24FPS, use the following command:

```
python3 depthai_demo.py -gt cv -rgbf 24 -monof 24
```

Synchronizing on the Host

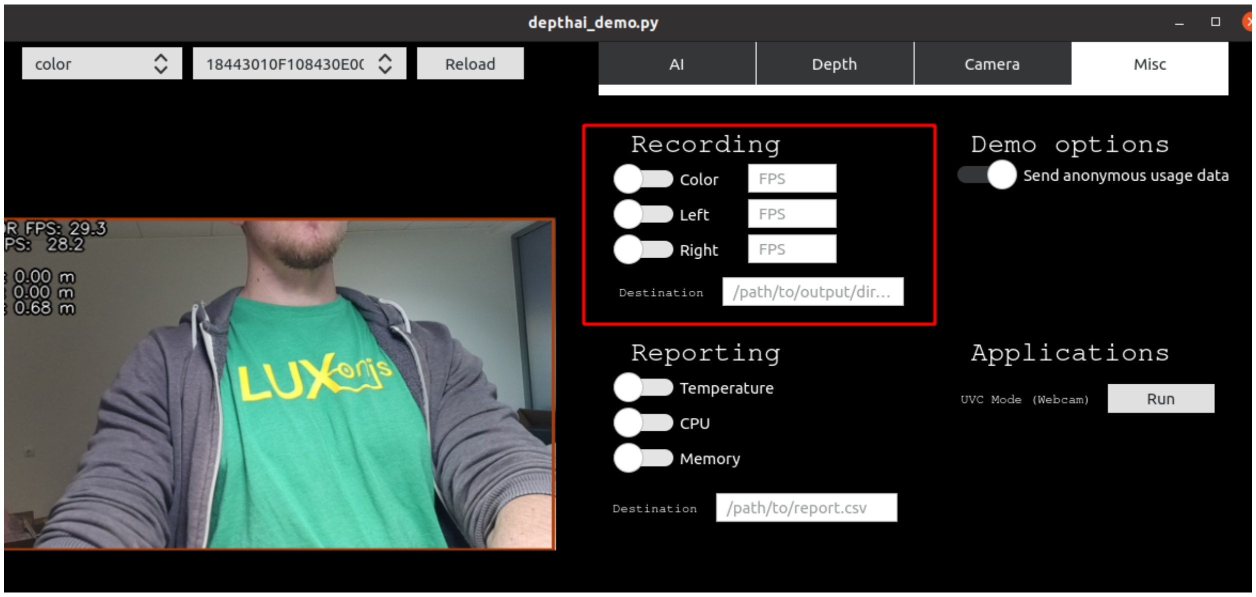
DepthAI messages has two functions - `msg.getTimestamp()` and `msg.getSequenceNum()` - which can be used for synchronization on host side or on the device using Script node.

You can use sequence number when syncing streams from one device, but when you have multiple OAK cameras and want to sync streams across multiple OAKs, you should use timestamp syncing, as host time is used (std::chrono::steady_clock) for the timestamps. Further documentation can be found here: Message Syncing.

We also have both timestamp and sequence number syncing demos here.

4.7.34 How do I Record (or Encode) Video with DepthAI?

DepthAI supports h.264 and h.265 (HEVC) and JPEG encoding directly itself - without any host support. The DepthAI demo app shows and example of how to access this functionality.



See our encoding examples which use VideoEncoder node:

- RGB and Mono Encoding, here.
- RGB Encoding and MobilenetSSD, here.
- RGB Encoding and Mono with MobilenetSSD and Depth, here.
- Encoding Max Limit, here.

Alternatively, to leverage this functionality from the `depthai_demo.py` script, use the *-enc* (or *–encode*) to specify which cameras to encode (record), with optional *-encout* argument to specify path to directory where to store encoded files. An example is below:

```
python3 depthai_demo.py -gt cv -enc left color -encout [path/to/output]
```

To then play the video in mp4/mkv format use the following muxing command:

```
ffmpeg -frame rate 30 -i [path/to/output/video.h264]
```

For more information about the script and its arguments, see our GitHub repository here.

By default there are keyframes every 1 second which resolve the previous issues with traversing the video as well as provide the capability to start recording anytime (worst case 1 second of video is lost if just missed the keyframe)

When running `depthai_demo.py`, one can record a JPEG of the current frame by hitting `c` on the keyboard.

An example video encoded on DepthAI OAK-D-CM3 (Raspberry Pi Compute Module Edition) is below. All DepthAI and megaAI units have the same 4K color camera, so will have equivalent performance to the video below.

4.7.35 What are the Capabilities of the Video Encoder on DepthAI?

See capabilities and limitations in the documentation here.

4.7.36 What Is The Stream Latency?

When implementing robotic or mechatronic systems it is often quite useful to know how long it takes from light hitting an image sensor to when the results are available to a user, the `photon-to-results` latency.

We have a documentation page here that describes the latency of the various streams available on DepthAI: DepthAI Latency.

4.7.37 How To Do a Letterboxing (Thumbnailing) on the Color Camera?

You can achieve letterboxing with the ImageManip node, see documentation here.

4.7.38 Is it Possible to Use the RGB Camera and/or the Stereo Pair as a Regular UVC Camera?

Yes, see *documentation here*.

4.7.39 How Do I Force USB2 Mode?

USB2 Communication may be desirable if you'd like to use extra-long USB cables and don't need USB3 speeds.

You can force USB2 mode by setting `maxUsbSpeed` to `dai.UsbSpeed.HIGH` when creating the device (note - it works for gen2):

```
dai.Device(pipeline, maxUsbSpeed=dai.UsbSpeed.HIGH)
```

The other way is using the `-usbs usb2` (or `--usbSpeed usb2`) command line option as below (this option only works with depthai_demo.py script):

```
python3 depthai_demo.py -usbs usb2
```

Note that if you would like to use DepthAI at distances that are even greater than what USB2 can handle, we do have DepthAI PoE variants, see here, which allow DepthAI to use up to a 328.1 foot (100 meter) cable for both data and power - at 1 gigabit per second (1gbps).

4.7.40 What is "NCS2 Mode"?

All OAK cameras come with support of what we call 'NCS2 mode'. This allows any OAK camera to pretend to be an NCS2.

So in fact, if you power your unit, plug it into a computer, and follow the instructions/examples/etc. of an NCS2 with OpenVINO, OAK camera will behave identically.

We also have an example code here. It runs facial cartoonization model (IR format) on the device using OpenVINOs Inference Engine (IE).

This allows you to try out examples from OpenVINO directly as if our hardware is an NCS2. This can be useful when experimenting with models which are designed to operate on objects/items that you may not have available locally/physically. It also allows running inference in programmatic ways for quality assurance, refining model performance, etc., as the images are pushed from the host, instead of pulled from the onboard camera in this mode.

Another common use case to run your model with IE (Inference Engine) first is to check if your model conversion to OpenVINOs IR format (eg. from TF/ONNX) was successful. After you run it successfully with the IE you can then proceed with *compiling the IR model* into the **.blob**, which is required by the DepthAI library.

4.7.41 What Information is Stored on the OAK cameras

All OAK-D (and newer OAK-1) cameras have on-board EEPROM that is used to save calibration data - things like board revision, camera intrinsics/extrinsics/distortion coefficients, FOV, IMU extrinsics, stereo rectification data etc. See Calibration reader API code on how to read this information from the OAK camera.

4.7.42 Dual-Homography vs. Single-Homography Calibration

As a result of some great feedback/insight from the OpenCV Spatial AI Competition we discovered and implemented many useful features (summary here).

Among those was the discovery that a dual-homography approach, although mathematically equivalent to a single-homography (as you can collapse the two homographies into one) actually outperforms single-homography in real-world practice.

As a result, we switched our calibration system in September 2020 to use dual-homography instead of single homography. So any units produced after September 2020 include dual homography. Any units with single homography can be recalibrated (see here) to use this updated dual-homography calibration.

4.7.43 How Do I Get Different Field of View or Lenses for DepthAI and megaAI?

ArduCam has built a variety of camera modules specifically for Luxonis' devices, including a variety of M12-mount options (so that the optics/view-angles/etc. are change-able by you the user).

See additional documentation here.

4.7.44 What are the Highest Resolutions and Recording FPS Possible with OAK cameras?

OAK cameras can be used to stream raw/uncompressed video with USB3. Gen1 USB3 is capable of 5gbps and Gen2 USB3 is capable of 10gbps. All OAK cameras are capable of both Gen1 and Gen2 USB3 - but not all USB3 hosts will support Gen2, so check your hosts specifications to see if Gen2 rates are possible.

Resolution	USB3 Gen1 (5gbps)	USB3 Gen2 (10gbps)
12MP (4056x3040)	21.09fps (390MB/s)	41.2fps (762MB/s)
4K (3840x2160)	30.01fps (373MB/s)	60.0fps (746MB/s)

OAK cameras can do h.264 and h.265 (HEVC) encoding on-device. The max resolution/rate is 4K at 30FPS. With the default encoding settings on OAK camera, this brings the throughput down from 373MB/s (raw/unencoded 4K/30) to 3.125MB/s (h.265/HEVC at 25mbps bit rate). An example video encoded on OAK-D-CM3 is below:

It's worth noting that all OAK cameras (except Lite versions) share the same color camera specs and encoding capabilities.

4.7.45 What are the theoretical maximum transmission rate for USB3 Gen1 and Gen2?

The maximum bit rate (the PHY rate) for Gen1 is 5Gbps and for Gen2 is 10Gbps. But this is the line rate - meaning purely how fast the bits can change from 0 to 1 and vice-versa. So above this, there is the USB encoding of the data, and then above this the protocol that is being used.

This FAQ answers the maximum transmission rate of USB-encoded data being sent over USB3. Keep in mind that this is prior to whatever protocol is being used over USB3 (e.g. USB Video Class (UVC), or XLink). Actual use of USB3 will always involve some form of protocol, which means the actual throughput will be lower than the following. And the CPUs involved may not be able to handle this throughput and/or the handling of the protocol used above USB3 at these rates.

So that is to say, this is the absolute maximum possible data transmission through USB3:

- Gen1 (8b/10b): 4Gbps (of 5Gbps PHY rate)
- Gen2 (128b/132b): 9.697Gbps (of 10Gbps PHY rate)

So interestingly, in Gen2 USB3, not only is the PHY rate 2x as high, the encoding overhead is significantly lower, as in USB3 Gen1 - each 8 bits get 2 bits of encoding added on top, whereas in Gen2, this can be increased to 4 bits of overhead for every 128bits of *data*. So in other words, in Gen1, 20% of what is being sent over the line is USB overhead. And in Gen2, this USB encoding overhead can be reduced down from 20% to 3.03%.

4.7.46 What is the best way to get FullHD in good quality?

See RGB Full Resolution Saver sample code to save 4K .jpeg files to the host.

4.7.47 How to run OAK-D as video device

OAK cameras do not appear as standard webcam by default. To use it as a webcam, follow the *tutorial here*.

4.7.48 How Much Compute Is Available? How Much Neural Compute is Available?

OAKs are built around the Intel Movidius Myriad X. More details/background on this part are here and also here.

A brief overview of the capabilities of DepthAI/megaAI hardware/compute capabilities:

- Overall Compute: 4 Trillion Ops/sec (4 TOPS)
- Neural Compute Engines (2x total): 1.4 TOPS (neural compute only)
- 16x SHAVES: 1 TOPS available for additional neural compute or other CV functions (e.g. through OpenCL)
- 20+ dedicated hardware-accelerated computer vision blocks including disparity-depth, feature matching/tracking, optical flow, median filtering, Harris filtering, WARP/de-warp, h.264/h.265/JPEG/MJPEG encoding, motion estimation, etc.
- 500+ million pixels/second total processing (see max resolution and frame rates over USB *here*)
- 450 GB/sec memory bandwidth
- 512 MB LPDDR4 (contact us for 1GB LPDDR version if of interest)

4.7.49 How are resources allocated? How do I see allocation?

- Resources are allocated automatically, based on the enabled nodes in the pipeline and their properties, before starting the pipeline. If there are no available resources an error will be thrown.
- After distributing the SHAVE/CMX resources between nodes (except NN), `NeuralNetwork` receives the rest of the free resources.
- There are 2 main CPUs, LeonOS and LeonRT, running Rtems OS, scheduling the tasks (USB, SHAVES, ISP etc.).
- There are a total of `16 SHAVEs` and `20 CMX` slices, each slice `128KB`, a total of `2.5MB`, together with `512MB DDR`.
- `CMX` memory is super-fast `SRAM` compared to `DRAM (DDR)`, used by Hardware CV filters, SHAVEs for highest performance and lowest latency.
- SHAVEs are accelerator processors for CV, NN algorithms.

The allocated resources can be printed with `DEPTHAI_LEVEL` environment variable set to `INFO`. For example: `DEPTHAI_LEVEL=info python3 26_1_spatial_mobilenet.py`

- [system] [info] ImageManip internal buffer size '80640'B, shave buffer size '19456'B
- [system] [info] SpatialLocationCalculator shave buffer size '11264'B
- [system] [info] SIPP (Signal Image Processing Pipeline) internal buffer size '143360'B
- [system] [info] NeuralNetwork allocated resources: shaves: [0-12] cmx slices: [0-12]

- [system] [info] ColorCamera allocated resources: no shaves; cmx slices: [13-15]
- [system] [info] MonoCamera allocated resources: no shaves; cmx slices: [13-15]
- [system] [info] StereoDepth allocated resources: shaves: [13-13] cmx slices: [13-15]
- [system] [info] ImageManip allocated resources: shaves: [15-15] no cmx slices.
- [system] [info] SpatialCalculator allocated resources: shaves: [14-14] no cmx slices.
- `ImageManip` node requires 80640+19456 bytes of CMX memory and shave 15.
- `SpatialLocationCalculator` node (used by `SpatialDetectionNetwork` requires 11264 bytes of CMX memory and shave 15).
- `SIPP (Signal Image Processing Pipeline)` requires 143360 bytes of CMX memory, which is used by stereo node, camera ISP.
- `NeuralNetwork` takes shaves [0-12] and cmx slices [0-12].
- `ColorCamera` takes cmx slices [13-15], a total of 3 at 1080p. At 4k/12MP it requires 6 slices.
- `MonoCamera` takes cmx slices [13-15].
- `StereoDepth` takes cmx slices [13-15] and shave 13.

Each node requires its own pools in the memory where data is stored. In addition to SHAVE and CMX distribution, the `CPU usage, DDR, CMX, heap` memory allocations are exposed too at runtime.

- [system] [info] Memory Usage - DDR: 74.12 / 414.56 MiB, CMX: 2.37 / 2.50 MiB, LeonOS Heap: 32.72 / 46.36 MiB, LeonRT Heap: 5.20 / 27.45 MiB
- [system] [info] Temperatures - Average: 58.40 °C, CSS: 58.94 °C, MSS 58.30 °C, UPA: 59.36 °C, DSS: 57.01 °C
- [system] [info] Cpu Usage - LeonOS 55.29%, LeonRT: 34.93%

4.7.50 What Auto-Focus Modes Are Supported? Is it Possible to Control Auto-Focus From the Host?

OAK-D, OAK-1, OAK-D-PoE, etc. all support continuous video autofocus ('2' below, where the system is constantly autonomously searching for the best focus) and also and `auto` mode which waits to focus until directed by the host, in addition to region-of-interest based focus, where the focus is automatically focused around a region provided to DepthAI (e.g. from a neural network bounding box, or some other real-time or apriori setting).

- See here for an example of switching back/forth between autofocus and manual focus, and commanding specific manual-focus positions.
- See here for autofocus controls, region of interest (to set autofocus to only consider a certain region), and triggering.
- See here for the API for manually setting the focus level.

4.7.51 What is the Hyperfocal Distance of the Auto-Focus Color Camera?

The hyperfocal distance is important, as it's the distance beyond which everything is in good focus. Some refer to this as 'infinity focus' colloquially.

The 'hyperfocal distance' (H) of OAK's color camera module is quite close because of it's f.no and focal length.

From WIKIPEDIA, here, the hyperfocal distance is as follows:

$$H = \frac{f^2}{Nc} + f$$

Where:

- f = 4.52mm (the 'effective focal length' of the camera module)
- N = 2.0 (+/- 5%, FWIW)
- c = C=0.00578mm (see here, someone spelling it out for the 1/2.3" format, which is the sensor format of the IMX378)

4.7.52 Is it Possible to Control the Exposure and White Balance and Auto-Focus (3A) Settings of the RGB Camera From the Host?

Auto-Focus (AF)

- See here for an example of switching back/forth between autofocus and manual focus, and commanding specific manual-focus positions.
- See here for autofocus controls, region of interest (to set autofocus to only consider a certain region), and triggering.
- See here for the API for manually setting the focus level.

Exposure (AE)

It is possible to set frame duration (us), exposure time (us), sensitivity (iso) via the API. See this example for controlling exposure, and setting auto or manual for exposure.

White Balance (AWB)

See here for Auto White Balance modes and controls.

4.7.53 Is it possible to control exposure and ISO with separate cameras?

In situations where the surrounding brightness differs between cameras, it can be helpful to adjust ISO to help align brightness levels.

The following settings over 3 cameras (B,C,D) have been successful for us:

```
cam['left'] .initialControl.setManualExposure(30000, 400)
cam['right'].initialControl.setManualExposure(15000, 400)
cam['camd'] .initialControl.setManualExposure( 5000, 400)
```

Or

```
cam['left'] .initialControl.setManualExposure(20000, 1600)
cam['right'].initialControl.setManualExposure(20000,  800)
cam['camd'] .initialControl.setManualExposure(20000,  400)
```

Controlling separately at runtime is possible as well. In *cam_test.py* you will need to change from linking the same control *XLinkIn* node to all cameras: *control.out.link(cam[c].inputControl)*. This will separate control nodes per camera.

Or, if you want to keep auto exposure, but just change some of the cameras to apply a different exposure compensation (EV), can set values in the range *-9* .. *+9* (default is 0):

```
cam['left'] .initialControl.setAutoExposureCompensation(-3)
cam['right'].initialControl.setAutoExposureCompensation(1)
cam['camd'] .initialControl.setAutoExposureCompensation(6)
```

4.7.54 Am I able to attach alternate lenses to the camera? What sort of mounting system? S mount? C mount?

The color camera on megaAI and DepthAI is a fully-integrated camera module, so the lens, auto-focus, auto-focus motor etc. are all self-contained and none of it is replaceable or serviceable. You'll see it's all very small. It's the same sort of camera you would find in a high-end smartphone.

So the recommended approach, if you'd like custom optics, say IR-capable, UV-capable, different field of view (FOV), etc. is to use the ArduCam M12 or CS mount series of OV9281 and/or IMX477 modules.

- IMX477 M12-Mount
- IMX477 CS-Mount
- OV9281 M12-Mount

Note that these require an adapter (here), and *below* and this adapter connects to the RGB port of the DepthAI FFC. It is possible to make other adapters such that more than one of these cameras could be used at a time, or to modify the open-source OAK-FFC-3P-OG to accept the ArduCam FFC directly, but these have not yet been made.

That said, we have seen users attach the same sort of optics that they would to smartphones to widen field of view, zoom, etc. The auto-focus seems to work appropriately through these adapters. For example a team member has tested the Occipital *Wide Vision Lens* here to work with both megaAI and DepthAI color cameras. (We have not yet tried on the grayscale cameras.)

Also, see *below* for using DepthAI FFC with the Raspberry Pi HQ Camera to enable use of C- and CS-mount lenses.

4.7.55 Can I Power DepthAI Completely from USB?

So USB3 (capable of 900mA) is capable of providing enough power for the DepthAI models. However, USB2 (capable of 500mA) is not. So on DepthAI models power is provided by the 5V barrel jack power to prevent situations where DepthAI is plugged into USB2 and intermittent behavior occurs because of insufficient power (i.e. brownout) of the USB2 supply.

To power your DepthAI completely from USB (assuming you are confident your port can provide enough power), you can use this USB-A to barrel-jack adapter cable here. And we often use DepthAI with this USB power bank here.

4.7.56 What is the Screw Mount Specification on OAK-1 and OAK-D?

It is the standard 1/4-20 "Tripod" mount used on most cameras. More information on this type of mount on Wikipedia here.

4.7.57 How to use DepthAI under VirtualBox

If you want to use VirtualBox to run the DepthAI source code, please check our tutorial here.

4.7.58 What are the SHAVES?

The SHAVES are vector processors in DepthAI/OAK. The 2x NCE (neural compute engines) were architected for a slew of operations, but there are some that are not implemented. So the SHAVES take over these operations.

These SHAVES are also used for other things in the device, like handling reformatting of images, doing some ISP, etc.

So the higher the resolution, the more SHAVES are consumed for this.

- For 1080p, 13 SHAVES (of 16) are free for neural network stuff.
- For 4K sensor resolution, 10 SHAVES are available for neural operations.

There is an internal resource manager inside DepthAI firmware that coordinates the use of SHAVES, and warns if too many resources are requested by a given pipeline configuration.

4.7.59 How to increase SHAVES parameter?

We have implemented the `-sh` command line param in our example script. Just follow the instructions on DepthAI repository and do

```
python3 depthai_demo.py -sh 9
```

And it will run the default MobilenetSSD, compiled to use 9 SHAVEs. Note that the allowed shave value **can vary depending on the amount of features enabled, but cannot be greater than 16**, so you cannot use 17 or more SHAVEs, and the more features are enabled (like ImageManips or VideoEncoders) the less SHAVEs will be available for NeuralNetwork node.

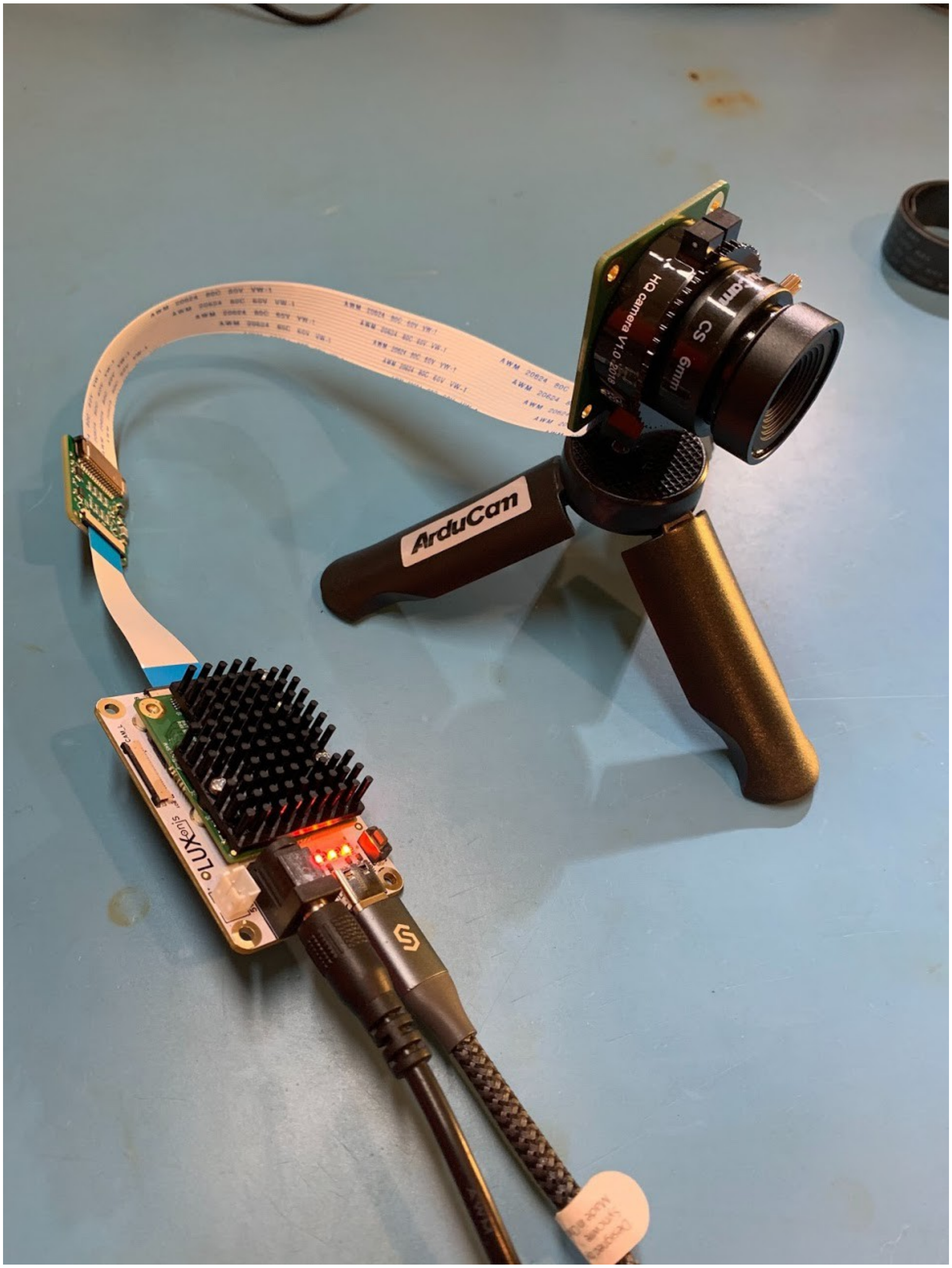
You can try compiling the model yourself either by following local OpenVINO model conversion tutorial or by using our online Myriad X blob converter. For more info, please see *Converting model to MyriadX blob*

4.7.60 Can I Use DepthAI with the New Raspberry Pi HQ Camera?

This is a particularly interesting application of DepthAI, as it allows the Arducam IMX477 HQ Camera (alternative to RPi HQ cam) to be encoded to h.265 4K video (and 12MP stills) even with a Raspberry Pi 1 or Raspberry Pi Zero - because OAK camera does all the encoding onboard - so the Pi only receives a 3.125 MB/s encoded 4K h.265 stream instead of the otherwise 373 MB/s 4K RAW stream coming off the IMX477 directly (which is too much data for the Pi to handle, and is why the Pi when used with the Arducam HQ camera directly, can only do 1080p video and not 4K video recording).

OAK-FFC-3P and OAK-FFC-4P will work with** the Arducam IMX477 HQ Camera **without an adapter board**, as you can connect the camera via the 22-26 pin adapter cable (SKU: A00403, which you get with the OAK-FFC-3P/OAK-FFC-4P) to the FFC baseboard.

OAK-FFC-3P-OG model **also works with Raspberry Pi HQ cam via an adapter board** (IMX477 based), which then does work with a ton of C- and CS-mount lenses (see here). And see here for the adapter board for OAK-FFC-3P-OG.


ArduCam
HQ camera V1.0 2018
CS
6mm
LUXonis

You can buy this adapter kit for the OAK-FFC-3P-OG here

4.7.61 Can I use DepthAI with Raspberry Pi Zero?

Yes, DepthAI is fully functional on it, additional documentation here.

4.7.62 How Much Power Does the DepthAI Raspberry Pi CME Consume?

The OAK-D-CM3 for short consumes around 2.5W idle and 5.5W to 6W when DepthAI is running full-out.

- Idle: 2.5W (0.5A @ 5V)
- DepthAI Full-Out: 6W (1.2A @ 5V)

Below is a quick video showing this:

4.7.63 A strange noise pattern appears on the OAK-D Lite (RGB), how do I resolve this?

When acquiring images with OAK-D Lite a strange noise pattern appears on RGB images. Left and right cameras are 480p, RGB image camera is 12mp with a preview size of 3840x2160. Those artifacts are related to ISP sharpness/denoise operations. These settings should reduce them:

```
camRgb.initialControl.setSharpness(0)
camRgb.initialControl.setLumaDenoise(0)
camRgb.initialControl.setChromaDenoise(4)
```

4.7.64 How To Unbind and Bind a Device?

In some cases, you may need to unbind and bind your device, i.e. a controller crashes with the following error messages:

```
[345692.730104] xhci_hcd 0000:02:00.0: xHCI host controller not responding, assume␣
↪dead
[345692.730113] xhci_hcd 0000:02:00.0: HC died; cleaning up
```

or you encounter error, such as:

```
RuntimeError: Failed to find device after booting, error message: X_LINK_DEVICE_NOT_
↪FOUND
```

or

```
Cannot enable. Maybe the USB cable is bad?
```

Instead of rebooting a host, you may unbind and bind a device.

Note! You'll need to know the PCI ID of the USB host controller to replace the "0000:00:14.0" part from the command below.

```
echo -n "0000:00:14.0" | sudo tee /sys/bus/pci/drivers/xhci_hcd/unbind; sleep 1; echo␣
↪-n "0000:00:14.0" | sudo tee /sys/bus/pci/drivers/xhci_hcd/bind
```

4.7.65 How Do I Get Shorter or Longer Flexible Flat Cables (FFC)?

For all cameras we use a 0.5mm 26-pin, same-side 152 mm contact flex cable. Follow the link for more details.

4.7.66 What are CSS MSS UPA and DSS Returned By meta_d2h?

- CSS: CPU SubSystem (main cores)
- MSS: Media SubSystem
- UPA: Microprocessor(UP) Array – Shaves
- DSS: DDR SubSystem

4.7.67 Where are the Github repositories? Is DepthAI Open Source?

DepthAI is an open-source platform across a variety of stacks, including hardware (electrical and mechanical), software, and machine-learning training using Google Colab.

See below for the pertinent Github repositories:

Overall

- https://github.com/luxonis/depthai-hardware - DepthAI hardware designs themselves.
- https://github.com/luxonis/depthai - DepthAI demo app and DepthAI SDK
- https://github.com/luxonis/depthai-python - Python API
- https://github.com/luxonis/depthai-api - C++ Core and C++ API
- https://github.com/luxonis/depthai-ml-training - Online AI/ML training leveraging Google Colab (so it's free)
- https://github.com/luxonis/depthai-experiments - Experiments showing how to use DepthAI.

Embedded Use Case

Standalone docs here.

The above examples include a few submodules of interest. You can read a bit more about them in their respective README files:

- https://github.com/luxonis/depthai-bootloader-shared - Bootloader source code which allows programming NOR flash of DepthAI to boot autonomously
- https://github.com/luxonis/depthai-spi-api - SPI interface library for Embedded (microcontroller) DepthAI application
- https://github.com/luxonis/esp32-spi-message-demo - ESP32 Example applications for Embedded/ESP32 DepthAI use

4.7.68 How Do I Build the C++ API?

Prebuilt binaries are available for Python bindings (or so called wheels).

We do not have prebuilt binaries for C++ core library.

One of the reasons is the vast number of different platforms and the second is that the library itself is quite lean so compiling along the other C++ source should not be a problem.

To compile the needed headers and a `.dll` follow this link: https://github.com/luxonis/depthai-core/tree/main#building Under - And for the dynamic version of the library

You can optionally also install it into a desired directory by appending this `cmake` flag:

```
cmake -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=[desired/installation/path]
# And then calling the install target
cmake --build . --target install
```

This should result in the headers and the library being copied to that path.

Another option is integrating into your CMake project directly, for that see: https://github.com/luxonis/depthai-core-example

And a note on building for **Windows**: Windows does not use *libusb*, but rather uses Windows internal *winusb*.

4.7.69 Can I Use an IMU With DepthAI?

Yes, all of our System on Modules have software support for both **9-axis BNO086** (and BNO080/BNO085) and **6-axis BMI270** IMU. See IMU documentation here.

4.7.70 Can I Use Microphones with DepthAI?

Yes.

- The OAK-SoM-Pro SoM supports up to 3x I2S stereo inputs (up to 6x physical microphones) and one I2S stereo output (e.g. for a stereo speaker drive).
- Any I2S mics should work, and may be possible to also use audio codecs, but those might need extra I2C config.
- It is important to note that the OAK-SoM and OAK-SoM-IoT do not have I2S support.

We have tested audio input on the OAK-SoM-Pro using 3x CMM-4030D-261-I2S-TR and have found the audio quality to be good. Theoretically many other microphones should work, however we have not tested audio output.

4.7.71 Where are Product Brochures and/or Datasheets?

These can be found at DepthAI Hardware documentation.

4.7.72 How Much Does OAK Devices Weight?

Every OAK model's weight is specified in the DepthAI Hardware documentation.

4.7.73 How Can I Cite Luxonis Products in Publications?

If DepthAI and OAK-D products has been significantly used in your research and if you would like to acknowledge the DepthAI and OAK-D in your academic publication, we suggest citing them using the following bibtex format.

```
@misc{DepthAI,
title={ {DepthAI}: Embedded Machine learning and Computer vision api},
url={https://luxonis.com/},
note={Software available from luxonis.com},
author={luxonis},
year={2020},
}

@misc{OAK-D,
title={ {OAK-D}: Stereo camera with Edge AI},
url={https://luxonis.com/},
note={Stereo Camera with Edge AI capabilities from Luxonis and OpenCV},
author={luxonis},
year={2020},
}
```

4.7.74 Where can I find your Logo?

You can find official **Luxonis** logo here.

4.7.75 How Do I Talk to an Engineer?

At Luxonis we firmly believe in the value of customers being able to communicate directly with our engineers. It helps our engineering efficiency. And it does so by making us make the things that matter, in the ways that matter (i.e. usability in the right ways) to solve real problems.

As such, we have many mechanisms to allow direct communication:

- discuss.luxonis.com. Use this for starting any public discussions, ideas, product requests, support requests etc. or generally to engage with the Luxonis Community. While you're there, check out this awesome visual-assistance device being made with DepthAI for the visually-impaired, here.
- Luxonis Github. Feel free to make Github issues in any/all of the pertinent repositories with questions, feature requests, or issue reports. We usually respond within a couple hours (and often w/in a couple minutes). For a summary of our Github repositories, see *here*.

4.8 OAK as a webcam

OAK devices can be used as webcams as well. Make sure to use **USB3 cable**, as we have noticed that in some cases, **USB2 won't work**.

4.8.1 Using UVC

```
# Skip cloning if you already have depthai-python repo
git clone https://github.com/luxonis/depthai-python.git
cd depthai-python
python3 examples/install_requirements.py
python3 examples/UVC/uvc_rgb.py
```

Now you can open up your favorite meeting app, like Zoom or Slack, and select **Luxonis Device: UVC Video Contr** in the webcam selection menu.

4.8.2 Webcam workarounds

There are currently a few issues with the approach above. Even on Linux, **UVC node currently doesn't work for all apps**. Since UVC stands for `USB Video Class`, using UVC pipeline on **OAK POE models won't work**. Another known issue is using UVC pipeline on **Windows**, as it **doesn't work** due to UVC descriptors. Here are workarounds:

1. Python virtual camera

One option is to use virtual camera, such as the pyvirtualcam module. You would need to pip install the package and install it's dependencies (as mentioned in the link). Here's a demo code:

```
import pyvirtualcam
import depthai as dai
# Create pipeline
pipeline = dai.Pipeline()
cam = pipeline.create(dai.node.ColorCamera)
```

(continues on next page)

(continued from previous page)

```
cam.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)
cam.setPreviewSize(1280,720)
xout = pipeline.create(dai.node.XLinkOut)
xout.setStreamName("rgb")
cam.preview.link(xout.input)
# Connect to device and start pipeline
with dai.Device(pipeline) as device, pyvirtualcam.Camera(width=1280, height=720,␣
↪fps=20) as uvc:
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    print("UVC running")
    while True:
        frame = qRgb.get().getFrame()
        uvc.send(frame)
```

2. OBS forwarding UVC stream

We have noticed that on some apps, like Discord or Google Meet, **Luxonis Device: UVC** won't work. One workaround is to use OBS to proxy the stream and use the virtual camera inside the OBS. I am running Linux so I had to install `sudo apt install v4l2loopback-dkms` for the virtual camera to work (this is also mentioned in install instructions).

This video will show you how to do just that.

3. OBS capturing cv2 window

Another solution is to stream the video to the host, and capture the `cv2.imshow` window inside the OBS:

- Inside depthai-python repo, run `python3 examples/ColorCamera/rgb_video.py`. This will open a new window where 1080P video stream will be shown.
- Inside OBS, under Sources menu, click +, Add new source
- Click on **Window Capture (Xcomposite)** option. Then select `video`
- You can then click on **Start Video Camera** inside OBS (just like in the video above), to use the video from OAK POE model as a webcam source.

4.9 Troubleshooting

4.9.1 DepthAI can't connect to an OAK camera

For **USB OAK cameras**, DepthAI can throw an error code like `X_LINK_COMMUNICATION_NOT_OPEN` or `X_LINK_ERROR`, which is usually a sign of a **bad USB3 cable** (or a USB2 cable). If you are using USB2 cable (and want USB2 bandwidth), you have to specify USB2 protocol, see *Forcing USB2 Communication* for more information. Another common issue is that users haven't set *udev rules* on their Linux machine.

If you still can't connect to the OAK camera, you should execute `lsusb | grep 03e7`. You should see a similar line:

```
$ lsusb | grep 03e7
Bus 001 Device 120: ID 03e7:2485 Intel Movidius MyriadX
```

Another thing to check is the `dmesg -w`. After executing that and pressing enter a few times (for separator), connect your OAK camera to the host. You should see a similar output in the terminal:

```
/~$ dmesg -w

[223940.862544] usb 1-3.2: new high-speed USB device number 120 using xhci_hcd
[223940.963357] usb 1-3.2: New USB device found, idVendor=03e7, idProduct=2485,␣
↪bcdDevice= 0.01
[223940.963364] usb 1-3.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[223940.963368] usb 1-3.2: Product: Movidius MyriadX
[223940.963371] usb 1-3.2: Manufacturer: Movidius Ltd.
[223940.963373] usb 1-3.2: SerialNumber: 03e72485
```

For **PoE OAK cameras**, see PoE Troubleshooting page.

If these commands didn't return the expected log, see *Support* page.

4.9.2 Reporting firmware crash dump

Steps to create a crash dump:

1. Run a code that causes the firmware crash
2. Run Crash report script

Afterwards, please send the generated .txt file to us (email/github/forum), and our development team will try to fix the cause of the crash as soon as possible.

4.9.3 Ping was missed, closing the device connection

```
[host]   [warning] Monitor thread (device: 194435108198757340 [169.254.1.222]) - ping␣
↪was missed, closing the device connection
```

This error is mostly seen on POE cameras, and can sometimes occur after a few hours or even days of running the pipeline. It happens because device doesn't reply to ping messages, which usually happens if device is too busy (high CPU consumption), or there's a networking issue. Potential solutions include:

1. **Lower OAK CPU consumption**

   If CPU usage is above 95% that can be a bad sign. You can measure CPU usage by running the depthai in debug mode. A few options to reduce CPU consumption are:

   - lower 3A FPS
   - Update networking settings on host computer - using `ethtool`, some have reported that increasing `rx-usecs` from 0 to 400 decreased Leon CPU usage from 99% to 93%
   - Reducing the pipeline / resolution / FPS, so there's less computation happening on the camera

2. **Increasing watchdog timeout**

   Another solution is to increase watchdog timeout, so it isn't as "trigger happy". Default value for POE devices is 4000 ms, and user can set it to 4500ms (max value). This can be done by setting environmental variable `DEPTHAI_WATCHDOG`, so for example on linux: `DEPTHAI_WATCHDOG=4500 python3 my_app.py`.

3. **Implementing re-connection**

This would be quite recommended, as it would allow the application to recover from the error. So whenever there's an error, re-connect to the device and continue running the pipeline.

For depthai API, this could be implemented as:

```
pipeline = dai.Pipeline()
# ...
while True:
  # Every time it crashes, re-initialize the device and upload the pipeline to it
  with dai.Device(pipeline) as device:
    queue = device.getOutputQueue("name")
    while True:
      q.get()
```

4. **Standalone mode**

If you run your device in Standalone mode and communicate with it via networking protocols (TCP/UDP/HTTP/MQTT) and not XLink, there's no ping mechanism, so this error won't occur. If there were an issue on device side (which is why we have watchdog in the first place), device would auto-restart.

4.9.4 ImportError: No module named 'depthai'

This indicates that the `depthai` was not found by your python interpreter. There are a handful of reasons this can fail:

1. Is the Python API installed? Verify that it appears when you type:

   ```
   python3 -m pip list | grep depthai
   ```

2. Are you using a supported platform for your operating system? If not, you can always install from source:

   ```
   cat /etc/os-release
   ```

4.9.5 Why is the Camera Calibration running slow?

Poor photo conditions can dramatically impact the image processing time) during the camera calibration. Under normal conditions, it should take 1 second or less to find the chessboard corners per-image on an Raspberry Pi but this exceed 20 seconds per-image in poor conditions. Tips on setting up proper photo conditions:

- Ensure the checkerboard is not warped and is truly a flat surface. A high-quality option: print the checkerboard on a foam board.
- Reduce glare on the checkerboard (for example, ensure there are no light sources close to the board like a desk lamp).
- Reduce the amount of motion blur by trying to hold the checkerboard as still as possible.

4.9.6 Permission denied error

If `python3 -m pip install` fails with a `Permission denied` error, your user likely doesn't have permission to install packages in the system-wide path.

```
[Errno 13] Permission denied: '/usr/local/lib/python3.7/dist-packages/...'
```

Try installing in your user's home directory instead by adding the `--user` option. For example:

```
python3 -m pip install depthai --user
```

More information on Stackoverflow.

4.9.7 DepthAI does not show up under `/dev/video*` like web cameras do. Why?

The USB device enumeration could be checked with lsusb | grep 03e7 . It should print:

- `03e7:2485` after reset (boot loader running)
- `03e7:f63b` after the application was loaded

No `/dev/video*` nodes are created. See *OAK as a webcam* if you would like to use OAK camera as a webcam.

4.9.8 Intermittent Connectivity with Long (2 meter) USB3 Cables

We've found that some hosts have trouble with long USB3 cables (above 6ft/2m). It seems to have something to do with the **USB controller on the host side**. For example, all Apple computers we've tested with have never exhibited the problem, as Apple computers have powerful USB controllers.

So, if you experience this problem with your host, there are potentially **3 options**:

1. Switching to a shorter USB3 cable (say 1 meter) will very likely make the problem disappear. These 1 meter (3.3 ft.) cables are a nice length and are shipped with OAK USB3 variants.
2. *Force USB2 mode*. This will allow use of the long cable still, and many DepthAI use cases do not necessitate USB3 communication bandwidth - USB2 is plenty.
3. Use Active USB3 cable. We have tested this 10m active cable and USB3 works as expected (even without powering the repeater).

Note that Ubuntu 16.04 has an independent USB3 issue, seemingly only on new machines though. We think this has to do w/ Ubuntu 16.04 being EOLed prior to or around when these new machines hit the market. For example, this computer (here) has rampant USB3 disconnect issues under Ubuntu 16.04 (with a 1 meter cable), but has none under Ubuntu 18.04 (with a 1 meter cable).

4.9.9 Forcing USB2 Communication

If you aren't using a (working) USB3 cable or your host computer doesn't support USB3, you should force the USB2 communication. It's also recommended to use USB2 communication if you are using a longer USB cable (2m+).

For API usage, set the **maxUsbSpeed=dai.UsbSpeed.HIGH** when creating the dai.Device object

```
# Force USB2 communication
with dai.Device(pipeline, maxUsbSpeed=dai.UsbSpeed.HIGH) as device:
  # ...
```

If you are using depthai_demo you can specify USB speed with `-usbs` argument:

```
python3 depthai_demo.py -usbs usb2
```

4.9.10 Output from DepthAI keeps freezing

If the output from the device keeps freezing every few seconds, there may be a problem with the USB3 connection and forcing the device into USB2 mode could resolve this issue - instructions are in the chapter above.

When connection speed is USB2 (due to some hosts - Windows in particular - or USB controller/port/cable being USB2) - initialization of USB3-enabled firmware or streaming after a few frames may fail. The workaround here is to force the device to use the USB2-only firmware (mentioned in the chapter above).

4.9.11 DepthAI freezes after a few frames

If your app freezes and you don't get any new messages from the device after a few messages (eg. 4 frames) after booting, it's likely that queues filled up and were set to blocking mode. Additional details on node queues (on the OAK device) can be found here. We also recommend using Pipeline Graph tool to quikcly check if there is something wrong with the pipeline (eg. a node that isn't connected to anything).

4.9.12 Udev rules on Linux

- `Failed to boot the device: 1.3-ma2480, err code 3`
- `Failed to find device (ma2480), error message: X_LINK_DEVICE_NOT_FOUND`
- `[warning] skipping X_LINK_UNBOOTED device having name "<error>"`
- `Insufficient permissions to communicate with X_LINK_UNBOOTED device with name "1.1". Make sure udev rules are set`

If you are getting any of the errors above, it's most likely that udev rules are not set on your Linux machine.

To fix this, set the udev rules using the commands below, unplugging DepthAI and then plugging it back into USB afterwards.

```
echo 'SUBSYSTEM=="usb", ATTRS{idVendor}=="03e7", MODE="0666"' | sudo tee /etc/udev/
↪rules.d/80-movidius.rules
sudo udevadm control --reload-rules && sudo udevadm trigger
```

4.9.13 CTRL-C Is Not Stopping It!

If you are trying to kill a program with `CTLR-C`, and it's not working, try `CTRL-\` instead. Usually this will work.

4.9.14 Nothing happening when running a DepthAI script

If upon running a DepthAI script, nothing happens (the script just hangs) and you cannot `CTRL-C` out of it. Try running `cam_test.py` script from the `depthai-python/utilities` folder. The script should error out with something like:

```
QObject::moveToThread: Current thread (0x55b9d0a00320) is not the object's thread␣
↪(0x55b9d0cad7e0).
Cannot move to target thread (0x55b9d0a00320)


        .
        .
        .
```

(continues on next page)

(continued from previous page)

```
Aborted (Signal sent by tkill() 32104 1001)
Aborted (core dumped)
```

This means there is a problem with PyQt5 installation. Follow this issue for a resolution.

4.9.15 "DLL load failed while importing cv2" on Windows

If you are seeing the following error after installing DepthAI for Windows:

```
(venv) C:\Users\Context\depthai>python depthai_demo.py
 Traceback (most recent call last):
   File "C:\Users\Context\depthai\depthai_demo.py", line 7, in <module>
     import cv2
   File "C:\Users\Context\depthai\venv\lib\site-packages\cv2\__init__.py", line 5, in
↪<module>
     from .cv2 import *
 ImportError: DLL load failed while importing cv2: The specified module could not be␣
↪found.
```

Then installing the Windows Media Feature Pack (here) is often the resolution, as Media Feature Pack must be installed for Windows 10 N editions.

(And more background from OpenCV directly is here)

4.9.16 `python3 depthai_demo.py` returns Illegal instruction

This so far has always meant there is a problem with the OpenCV install on the host (and not actually with the depthai library). To check this, run:

```
python3 -c "import cv2; import numpy as np; blank_image = np.zeros((500,500,3), np.
↪uint8); cv2.imshow('s', blank_image); cv2.waitKey(0)"
```

If a window is not displayed, or if you get the `Illegal instruction` result, this means there is a problem with the OpenCV install. The installation scripts here often will fix the OpenCV issues. But if they do not, running

```
python3 -m pip install opencv-python --force-reinstall
```

will often fix the OpenCV problem.

4.9.17 Neural network blob compiled with incompatible openvino version

```
[NeuralNetwork(2)] [error] Neural network blob compiled with incompatible openvino␣
↪version. Selected openvino version 2020.3. If you want to select an explicit␣
↪openvino version use: setOpenVINOVersion while creating pipeline
```

The reason for this error is that depthai can't resolve the OpenVINO version from the blob. The solution is simple, the user has to specify the OpenVINO version with which the blob was compiled (as mentioned in the error message):

```
pipeline = depthai.Pipeline()
# Set the correct version:
pipeline.setOpenVINOVersion(depthai.OpenVINO.Version.VERSION_2021_1)
```

4.9.18 “realloc(): invalid pointern Aborted” on RPi

On RPi, after running `sudo apt upgrade`, you might get the error `realloc(): invalid pointer\n Aborted` when importing cv2 after depthai library. We have observed the same issue, and have found a **solution**:

- Downgrade libc6 by running `sudo apt install -y --allow-downgrades libc6=2.28-10+rpi1`, OR
- Re-install DepthAI dependencies by running `sudo curl -fL http://docs.luxonis.com/_static/install_dependencies.sh | bash`

4.9.19 [error] Attempted to start camera - NOT detected!

If you are facing any of the errors above for either Mono Left/Right or Color camera, first try using the latest depthai version (`python3 -mpip install depthai -U`). If that doesn’t help, there are 2 probable causes:

- You are using OAK FFC and a camera sensor that isn’t supported by default, so you should use a different branch, see docs here.
- A camera got disconnected during the shipping. This has been reported only a handful of times, but it’s possible.

The solution here is to open up the enclosure and re-attach the connector to the camera, see the image here for the OAK-D (left mono camera not detected).

4.9.20 [error] input tensor exceeds available data range

```
[NeuralNetwork(3)] [error] Input tensor '0' (0) exceeds available data range. Data␣
↪size (6336B), tensor offset (0), size (6912B) - skipping inference
```

This error is usually thrown when we use `NNData` message and we don’t provide the amount of bytes that the NN model expects for the inference. For example, in the error above, the NN model expects 6912 bytes (48x48x3), but only 6336 bytes were sent to it.

4.9.21 Converting YUV420 to CV2 frame

If you try to convert YUV420 frame to CV using ImgFrame’s `.getCvFrame()` method, you might come accross the error below:

```
cv2.error: OpenCV(4.6.0) d:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\
↪color.simd_helpers.hpp:108: error:
(-215:Assertion failed) sz.width % 2 == 0 && sz.height % 3 == 0 in function
'cv::impl::`anonymous-namespace'::CvtHelper<struct cv::impl::`anonymous namespace'::Set
↪<1,-1,-1>,
struct cv::impl::A0xe823dd8f::Set<3,4,-1>,struct cv::impl::A0xe823dd8f::Set<0,-1,-1>,
↪1>::CvtHelper'
```

The culprit of the error is that OpenCV requires YUV420 width to be divisible by 2, and height to be divisible by 3. A simple example that will crash:

```python
import depthai as dai
import cv2

pipeline = dai.Pipeline()
cam = pipeline.createColorCamera()
```

(continues on next page)

(continued from previous page)

```
6
7  manip = pipeline.createImageManip()
8  # manip.initialConfig.setFrameType(dai.RawImgFrame.Type.BGR888p)
9  manip.initialConfig.setResize(600, 451)
10 cam.isp.link(manip.inputImage)
11
12 xout = pipeline.createXLinkOut()
13 xout.setStreamName("out")
14 manip.out.link(xout.input)
15
16 with dai.Device(pipeline) as device:
17     f = device.getOutputQueue('out').get().getCvFrame()
18     cv2.imshow("frame", f)
19     cv2.waitKey(0)
```

Since `isp` output is **YUV420**, it will crash when calling `.getCvFrame()`. You could either resize the frame to 600x450 (so height is divisible by 3) on line 9, or uncomment the line 8, so frame gets converted to **BGR** on the device itself.

4.10 SLAM with OAK

On-board localization (VIO) and SLAM (Simultaneous Localization And Mapping) on current OAK cameras (RVC2) aren't yet supported.

Our upcoming Series 3 OAK cameras with RVC3 have Quad-core ARM A53 1.5GHz integrated into the VPU. There will be an **open-source SLAM implementation on the RVC3**. Users are be able to run custom containarized apps on the ARM, which will allow **other companies** (which specialize in VIO/SLAM) to **port their software stacks** to our cameras and license it.

Several SLAM and localization projects that support OAK-D cameras:

- ORB SLAM3 with an OAK-D and ROS1 by `@nimda`
- RTAB-Map recently (PR here) added support for depthai and OAK cameras (via ROS)
- SpectacularAI's SLAM with OAK-D - Free for non-commercial use
- ArduCam Visual SLAM tutorial
- DepthAI-SLAM
- *On-device NN inferencing for localization*

4.10.1 On-device SuperPoint for localization and SLAM

A customer shared with us a solution that was running the SuperPoint (Github repo, Arxiv paper) feature extraction NN on-device (on RVC2-based OAK-D) and then used the features for localization and SLAM (on the host computer).

4.10.2 RAE on-device VIO & SLAM

The demo below shows how you can run on-device VIO & SLAM on the RAE robot using Spectacular AI SDK. Disparity matching and feature extraction + tracking are done on accelerated blocks on the RVC3 chip. Features are then combined with disparity to provide tracked 3D points used by VO, and Spectacular AI SDK fuses that with IMU data to provide accurate localization of the robot.

4.10.3 Syncing frames and IMU messages

For VIO/SLAM solutions, you would want to sync IMU messages with the middle of the exposure. For exposure timings and timestamps, see Frame capture graphs for details. See here for **IMU/frame syncing demo**.

Some more advance algorithms weight multiple IMU messages (before/after exposure) and interpolate the final value.

4.11 OAK on drones

OAK cameras are light-weight, low-power and performant *Spatial AI* devices for edge applications, which make them the perfect solution for drone applications, such as:

- **Localization** of the drone and detected objects around it, as presented by the CVAR-U.P.Madrid team (video here). See *SLAM with OAK* for additional information.
- **Precision landing** on the target, as demonstrated here by Rishabh Singh. This is possible as object detection runs on the OAK camera at about 30 FPS and has below 150ms delay from camera to the controller.
- **Emergency landing** when your battery is running low. *Semantic depth* provides 3D location (depth points) of suitable areas to land (eg. grass fields). Stephan Sturges has developed OpenLander repo for this application.
- **Follow-me drone** with the help of 3D object detection and tracking, drone is able to follow you around, as demonstrated here by Rishabh Singh (code here). One could make this solution more robust when combining this with either face recognition or person reidentification AI model.

A few other demos:

- Team **QuetzalC++** (OpenCV AI comp) - Warehouse inspection with autonomous drones - video
- Team **QUTEagles** (OpenCV AI comp) - Drone-based biosignatures detection system for planetary exploration - video
- **Augmented Startups** built a gesture controlled drone and has 3-part tutorial on YouTube - video

4.11.1 Drone on-device NN-based localization

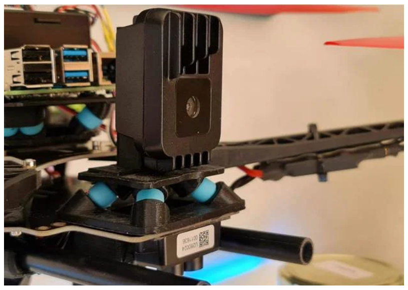
A novel AI approach for spatial localization using an OAK camera. The drone runs on-device neural inferencing to provide positioning of the drone. Paper here, Github repo here.

OAK ArduPilot integration

Rishabh Singh wrote a few **OAK-ArduPilot integration applications** like a follow-me app, obstacle avoidance app, and precision landing app. He also wrote 2 blog posts about the integration; Part 1 and Part 2.

Camera vibration

Camera vibrations can be a big challenge in applications such as drones, especially if you are using Auto-Focus color camera. To decrease the camera vibrations, we suggest firmly mounting the device on the drone. One could also consider adding shock absorption rubbers (eg. these) to decreate the vibrations. AugmentedStartup has also designed an OAK-1 anti-vibration mount for his drone project:



For drone applications we would also suggest Fixed-Focus color camera, more info here.

4.12 OAK for Education

By creating OAK devices, our goal is to make Spatial AI accessible and easy to use by anyone. That's why we want to share some of the community projects built around it that help us in achieving our common goal.

4.12.1 Cortic AI Toolkit

The Cortic A.I. Toolkit is a software package that enables makers and students to learn and experiment with A.I. algorithms on the popular Raspberry Pi 4B single-board computer. Provides a gentle introduction to A.I. for beginners while offering a lot of customization possibilities and depth perception so that users can progressively grow with it

You can read more about this project on the release post or by visiting this project on GitHub

Kudos to the Cortic AI team for sharing their awesome tool!

4.12.2 Looking for more?

You can also check out our Discuss Forum, where you can meet people behind these projects or share your idea/demo - we're looking forward to meeting you there!

4.13 Support

Running into issues or have questions? We're here to help. **Before requesting support, please check** *Troubleshooting documentation page*. To help you debug the issue you have the fastest and most efficient way, please provide all the details of the issue you are experiencing.

4.13.1 Requesting support

To **request support from our engineers**, **create a new post at** our Forum. Please provide as much information as possible, and follow the guidelines below.

DepthAI issue

DepthAI (pipeline) issue

If you are experiencing depthai pipeline issues (freezing, crashing, etc.), please provide a Minimal Reproducible Example (**MRE**) docs on how to create MRE here. This means everything, including **minimal script, required model .blobs, and any other assets**, should be compressed into single archive. Make sure that:

1. Assets/model blobs are located at the right path.
2. Remove any unnecessary code: commented out code, and code that isn't relevant to the depthai/pipeline code (so host-side code).
3. Please provide **minimal reproducible code**. Main script should be as short as possible.

Besides MRE, please also provide the following information when you are requesting support:

- Name of the OAK camera (all camera names here).
- The **version** of the **depthai library** and **bootloader** you are using (check with OAK Device Manager). If it's not the latest release, please also try updating the depthai version to the latest (`python -mpip install depthai -U`).
- If there was a crash, provide debug log by setting depthai log level to `debug`.
- Screenshot of your pipeline using the Pipeline Graph tool.

IP related issues

If you are having an issue with an app that contains your (company's) **Intellectual Property**, eg. NN model or business logic, you should first remove this IP before creating MRE:

- For **NN model**, replace your model with a public model. So if you trained an object detection NN, replace it with eg. public pretrained Mobilenet-SSD.
- For **business logic**, simply remove the code. MRE shouldn't contain much host-side code where your business logic would be.

Connectivity issue

Can't connect to an OAK USB camera

Check first:

1. If you're on Linux, check whether you have setup *udev rules*
2. Check whether you are using a **working USB3 cable** - this is a common issue. If you are using USB2 cable, please see *Forcing USB2 Communication* tutorial
3. If you are using a longer USB3 cable (above 1 meter), we'd recommend first trying a shorter USB3 cable, and checking *Using longer USB cables*

If you are experiencing connectivity issues with your OAK USB device, please provide the following information:

- OAK device model
- DepthAI version
- Cable specification you are using

Can't connect to an OAK PoE camera

Check first:

1. Getting started with OAK PoE devices
2. From the same page, please check PoE Troubleshooting page
3. Try Manually specify device IP of your OAK device
4. If you have flashed invalid static IP and device isn't accessible anymore, we suggest performing a Factory reset

If you are experiencing connectivity issues with your OAK PoE device, please provide the following information:

- OAK device model
- DepthAI version and Bootloader version (ideally OAK Device Manager screenshot)
- Terminal output of `ipconfig` / `ifconfig` command
- IP address of the device. This can be obtained either from pinging the device, by checking the DHCP server logs, or using IP scanner application.

Hardware issue

Hardware (OAK) issue

Provide detailed description of the problem, describe the device behavior, how and when it fails. When contacting support, please include the following information:

- Device model, and batch number (from the barcode label). If you don't have the box, provide order number
- Photos of the whole hardware setup, close captures of region of an issue
- Board revision (if SoM based, also HW revision of SoM)

Example barcode label with marked batch number:

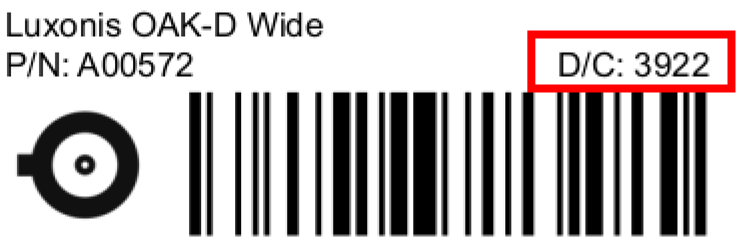


Image Quality issue

Image Quality (IQ) issue

If you are experiencing image quality issues (blurry, noisy, etc.), please first check Improving Image Quality docs. For reporting an issue, please provide detailed description of the problem, how and when the device fails. Please include the following information:

- Device model, and batch number (from the barcode label). If you don't have the box, please let us know your order number and when did you purchase the device.
- Image captures (high resolution), please add remarks to the images if needed

Calibration issue

Calibration issue

If you encountered an issue while calibrating an OAK, please provide a detailed description of the problem. When contacting support, please include the following information:

- Device model
- OS name
- DepthAI branch used
- Full command used for calibration
- Dataset folder
- Json file from calibration
- Image outputs

Converting NN model issue

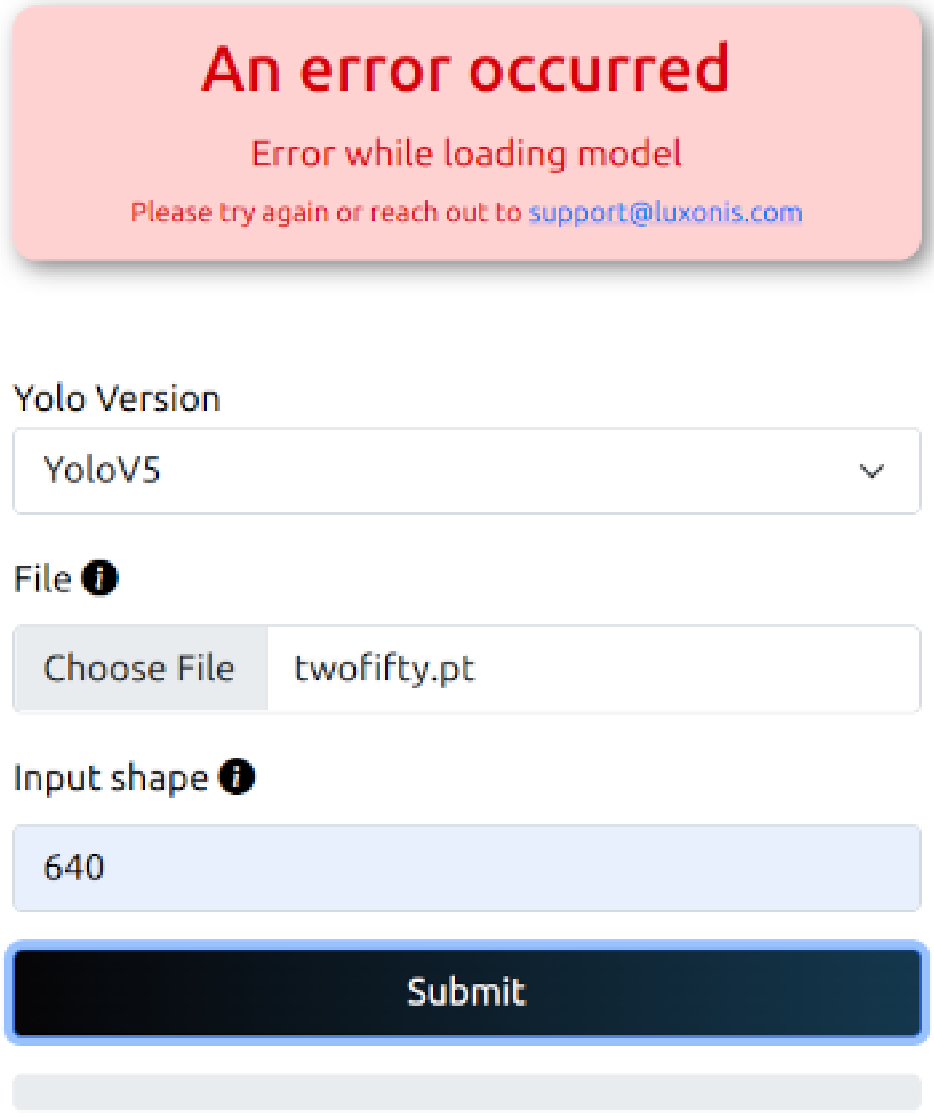
Issue when converting a Neural Network model

We officially support models for which we have notebooks at depthai-ml-training. If you encounter any error during converting blob via tools.luxonis.com or blobconverter, please provide the following information:

- Screenshot of your setup, including error message
- .pt file used
- Training procedure - notebook/repo/library name, version, commit
- Exact input parameters
- Current output, expected output, and input images for testing

For support, we suggest creating an Issue on depthai-ml-training repository.

Example screenshot:


An error occurred

Error while loading model

Please try again or reach out to support@luxonis.com

Yolo Version: YoloV5

File: twofifty.pt

Input shape: 640

4.13.2 Refunds and returns policy

At Luxonis, we are customer-focused. Our success is only possible if our customers believe in the value of our products. If for any reason you are not satisfied with your purchase, please let us know and we will make it right.

If you desire a refund, please contact support@luxonis.com with your order number and reason for the return. Refund requests within 60 days of the purchase date will be honored in full.

Shipping costs for returns within 60 days of purchase will be covered by Luxonis. Shipping costs for returns after 60 days from the purchase date will be born by the customer.

If a return is initiated because of damaged, defective, or incorrect goods, Luxonis will provide a replacement order at no cost to the customer.

Refunds will be processed within 14 days after the product has been returned.