# CONTENTS

1 Demo Script .......................................................... 3

2 Example Use Cases .................................................. 5

3 Tools & API Examples ................................................ 7

4 Ecosystem ................................................................. 9
   4.1 First steps with DepthAI ........................................ 10
   4.2 Spatial AI .......................................................... 38
   4.3 AI / ML / NN ...................................................... 42
   4.4 Depth perception ................................................ 59
   4.5 Computer Vision ................................................ 61
   4.6 On-device programming ....................................... 65
   4.7 FAQs & How-To .................................................. 66
   4.8 Calibration ........................................................ 107
   4.9 OAK as a webcam ............................................... 112
   4.10 Troubleshooting ............................................... 115
   4.11 SLAM with OAK ............................................... 120
   4.12 OAK on drones ................................................. 120
   4.13 OAK for Education ............................................. 121
   4.14 Support .......................................................... 122
DepthAI Docs

---

**Note:** Just received an OAK camera? Start [here](#)!

DepthAI is a *Spatial AI* platform, which allows robots and computers to perceive the world like a human can - what objects or features are - and where they are in physical world. It focuses on the combination of these 5 key features:

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

DepthAI platform is built around Movidius VPU - it’s a complete ecosystem of custom hardware, firmware, and software API. Best of all, it is modular and you can integrate this technology into your products ([docs here](#)).
Demo script is our multipurpose command line demo tool, built around pipeline builder API, that allows you to check DepthAI features straight from the command line - no coding required! It works USB and POE equally, automatically discovering any POE DepthAI on your LAN and/or USB DepthAI connected to your computer. If multiple are connected, it will prompt you on which to use for the demo.

To install and run the demo script on your OAK camera, type the following commands in the terminal.

```
git clone https://github.com/luxonis/depthai.git
cd depthai
python3 install_requirements.py
python3 depthai_demo.py
```

And then following up with the README.md for more usage examples. We have also prepared a step by step guide here with detailed instructions how to set up your DepthAI and run this script.

If you have issues during the installation, see our Installation page for additional OS-specific instructions.
In this section, you’ll find an inspiration what can you build right away with DepthAI.
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.

Table 1: Tools / Examples

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR</td>
<td>This pipeline implements text detection (EAST) followed by optical character recognition of the detected text</td>
</tr>
<tr>
<td>Multiple Devices</td>
<td>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.</td>
</tr>
<tr>
<td>Class Saver</td>
<td>This example demonstrates how to run MobilenetSSD and collect images of detected objects, grouped by detection label</td>
</tr>
<tr>
<td>NN Frame Sync</td>
<td>This example shows how to present the neural network inference results on the inferred frames</td>
</tr>
<tr>
<td>Frames Sync</td>
<td>This example demonstrates how to synchronize incoming frames using their sequence numbers. This allows displaying frames taken in exact same moment</td>
</tr>
<tr>
<td>SPI</td>
<td>This directory contains a few examples of how to use the SPI interface with the Pipeline builder API</td>
</tr>
<tr>
<td>TensorFlow Image Classification</td>
<td>This example demonstrates how to run a neural network created using TensorFlow Image Classification tutorial</td>
</tr>
<tr>
<td>WLS Filtering</td>
<td>This example demonstrates how to do host-side WLS filtering using the rectified_right and depth stream from DepthAI API.</td>
</tr>
</tbody>
</table>
Table 1: Core Repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depthai-python</td>
<td>Here you’ll find Python bindings creating the Python API of DepthAI</td>
</tr>
<tr>
<td>depthai-core</td>
<td>Our core API written in C++</td>
</tr>
<tr>
<td>depthai-shared</td>
<td>This repository contains shared data between our main firmware and depthai-core host library.</td>
</tr>
<tr>
<td>depthai-ros</td>
<td>DepthAI ROS Wrapper. This is an attempt at basic DepthAI to ROS2 interface. It’s largely leveraging the existing depthai-python examples.</td>
</tr>
<tr>
<td>depthai-unity</td>
<td>DepthAI Unity Wrapper projects and examples. Useful for synthetic dataset generation.</td>
</tr>
<tr>
<td>depthai-hardware</td>
<td>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.</td>
</tr>
<tr>
<td>depthai-spi-api</td>
<td>API of the SPI protocol</td>
</tr>
<tr>
<td>depthai-spi-library</td>
<td>DepthAI SPI Library</td>
</tr>
<tr>
<td>depthai-bootloader-shared</td>
<td>The depthai-bootloader-shared repository contains shared data between our bootloader firmware and depthai-core host library.</td>
</tr>
</tbody>
</table>
Table 2: Tools & Docs Repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depthai-ml-training</td>
<td>Here you can find repositories to help you connect your NN and create BLOBs.</td>
</tr>
<tr>
<td>depthai-ml-training</td>
<td>Here you can find repositories to help you connect your NN and create BLOBs.</td>
</tr>
<tr>
<td>blob-converter</td>
<td>Web-based tool to convert model into MyriadX blob</td>
</tr>
<tr>
<td>depthai-gui</td>
<td>DepthAI GUI is a WYSIWYG tool that allows to create a custom DepthAI pipelines, run them and see the results - all in one tool.</td>
</tr>
<tr>
<td>depthai-tutorials</td>
<td>This repo contains source code for tutorials published on docs.luxonis.com.</td>
</tr>
<tr>
<td>Factory-calibration</td>
<td>Factory Calibration (WIP); This package contains two ROS workspace one is for depthai capture and calibration node and another is for Interbotix ViperX 300 Robot Arm 6DOF (KIT-VIPX300-6DOF) arm bot control using moveit.</td>
</tr>
<tr>
<td>depthai-docker</td>
<td>This repository contains a Dockerfile, that allows you to run OpenVINO on DepthAI inside a Docker container.</td>
</tr>
<tr>
<td>sbr-util</td>
<td>Utility to view and manipulate SBR binary images</td>
</tr>
<tr>
<td>depthai-docs-website</td>
<td>If you want to contribute and update our docs, you can simply create a pull request.</td>
</tr>
</tbody>
</table>

Table 3: Demo Repositories

| Repository          | Description | |
|--------------------|-------------|
| 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. |
| esp32-spi-message-demo | ESP32 reference app for interfacing with DepthAI over SPI |
| depthai-core-example | CMake example project which serves as a template on how to quickly get started with C++ and depthai library |

We’re always happy to help with code or other questions you might have.

4.1 First steps with DepthAI

Hello DepthAI users!

In this guide, I assume you just got your OAK camera (e.g. OAK-D) and you want to give it the first try to explore what is possible with it and what you can achieve when working with it.

- First, we will run a DepthAI demo script, that will allow you to preview DepthAI functionalities.
- Following up, I will go through some of the options available in the demo script and show their usage/results
- Last, you will receive useful links to expand your knowledge further and check open-sourced use-case implementations, code examples, and tutorials, that you can use as a starting point for your projects.
Let’s start with the device setup below

### 4.1.1 Connect the OAK camera (USB)

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

**Warning:** Make sure to use **USB3 cable**! If you aren’t, *force USB2 communication* in the program.

USB3 cable is colored blue in the inside of the USB-A connector of the USB-C cable.

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

On Ubuntu, you can check if a new USB device was detected by running

```bash
$ lsusb | grep MyriadX
Bus 003 Device 002: ID 03e7:2485 Intel Movidius MyriadX
```

**Note:** If you are running other OS than Ubuntu, or you think something has gone wrong, we have detailed OS-specific installation guides [here](#).
4.1.2 Connect the OAK camera (PoE)

If you are using PoE device, you should follow the Getting started with OAK PoE devices tutorial.

4.1.3 Use Windows Installer

If you prefer, we’ve encapsulated all setup procedures inside a single .exe file you can download here and skip the Setup section.

When downloaded and run, it will install all necessary components and package requirements. Once finished, it will run the demo script automatically.

4.1.4 Setup

In this section, I will describe how to install the demo script manually with a command line

Download demo script

To download the demo script, you can either use git or directly download a zip file

From zip file

First, download a repository package from here and then unpack the archive to a directory of preference. Next, open a terminal session in this directory.

From git

First, open the terminal session and go to a directory of preference, where you’d like to download your demo script. Then, run the following code to download the demo script

```
$ git clone https://github.com/luxonis/depthai.git
```

After the repository is downloaded, make sure to enter the downloaded repository by running

```
$ cd depthai
```

Create python virtualenv (optional)

To create and use the virtualenv, you can follow an official python guide to virtualenvs or follow os-specific guides on the web, like “How to Create Python 3 Virtual Environment on Ubuntu 20.04”

This will make sure that you are using a fresh environment and that Python 3 is the default interpreter - this can help to prevent potential issues.

I usually create and use virtualenvs by running

```
$ python3 -m venv myvenv
$ source myvenv/bin/activate
$ pip install -U pip
```

And this may require installing these packages prior
Install requirements

Once the demo source code is downloaded, and you have your terminal session set up, the next thing that has to be done is to install all additional packages that this script requires (together with the depthai Python API itself).

To install these packages, run the `install_requirements.py` script

```
$ python3 install_requirements.py
```

**Warning:** If you are using a Linux system, in most cases you have to add a new udev rule for our script to be able to access the device correctly. You can add and apply new rules by running

```
$ 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
```

Now, you should be able to start using the demo script, which we will do now

Run demo script

Having everything set up, we are now ready to use the demo script by running

```
$ python3 depthai_demo.py
```

### 4.1.5 Default run

Running the demo for the first time, the script will compile and download a default mobilenet-ssd model, configure the OAK camera and then show a default color preview that will contain a scaled preview from the RGB camera from your device.
4.1.6 Change preview

To see other previews from the device, you can use the preview switcher that is visible in the top-left section of the GUI.
### Table 4: Available previews

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Shows preview from color camera</td>
<td></td>
</tr>
<tr>
<td>nnInput</td>
<td>Shows preview from right mono camera</td>
<td>Disabled if no AI model is running</td>
</tr>
<tr>
<td>left</td>
<td>Shows preview from left mono camera</td>
<td></td>
</tr>
<tr>
<td>right</td>
<td>Shows preview from right mono camera</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>depth</td>
<td>Shows disparity map calculated from depthRaw preview and JET colored.</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>depthRaw</td>
<td>Shows raw depth map. Best for depth-based calculations</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>disparity</td>
<td>Shows disparity map produced on device</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>disparityCol</td>
<td>Shows disparity map produced on device and JET colored. Should be the same as depth preview but produced on the device.</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>rectifiedLeft</td>
<td>Rectified left camera frames</td>
<td>OAK-D needed</td>
</tr>
<tr>
<td>rectifiedRight</td>
<td>Rectified right camera frames</td>
<td>OAK-D needed</td>
</tr>
</tbody>
</table>

## 4.1.7 Default model

While the demo was running, you could see the detection results - and if you were standing in front of the camera, you should see yourself detected as a person with a pretty 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.
Or even cats
4.1.8 Using other models

We have prepared other models, which you can try and evaluate with ease. To run the demo script with e.g. `face-detection-retail-0004`, click on the CNN Model combo box and select mentioned model.

Which will allow you to detect human faces, like below:

You can use this combo box to change the model that is being run on the DepthAI. It’s also possible to use command line to select:

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

Below, there is a list of models that you can use, having just the demo script downloaded.
Table 5: Available models

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>FPS</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>deeplabv3p_person</td>
<td>People segmentation</td>
<td>22.1</td>
<td>Usage</td>
</tr>
<tr>
<td>face-detection-adas-0001</td>
<td>Face detection</td>
<td>13.4</td>
<td>Usage</td>
</tr>
<tr>
<td>face-detection-retail-0001</td>
<td>Face detection</td>
<td>30.0</td>
<td>Usage</td>
</tr>
<tr>
<td>mobilenet-ssd</td>
<td>Object detection (20 classes)</td>
<td>30.0</td>
<td>Usage</td>
</tr>
<tr>
<td>pedestrian-detection-adas-0002</td>
<td>People detection</td>
<td>13.1</td>
<td>Usage</td>
</tr>
<tr>
<td>person-detection-retail-0001</td>
<td>People detection</td>
<td>10.7</td>
<td>Usage</td>
</tr>
<tr>
<td>person-vehicle-bike-detection-0001</td>
<td>People, vehicle and bike detection</td>
<td>6.2</td>
<td>Usage</td>
</tr>
<tr>
<td>yolo-v3</td>
<td>Object detection (80 classes)</td>
<td>1.9</td>
<td>Usage</td>
</tr>
<tr>
<td>tiny-yolo-v3</td>
<td>Object detection (80 classes)</td>
<td>29.9</td>
<td>Usage</td>
</tr>
<tr>
<td>vehicle-detection-adas-0001</td>
<td>Vehicle detection</td>
<td>14.0</td>
<td>Usage</td>
</tr>
<tr>
<td>vehicle-license-plate-detection-0106</td>
<td>License plate detection</td>
<td>30.0</td>
<td>Usage</td>
</tr>
<tr>
<td>openpose2</td>
<td>Pose estimation (openpose)</td>
<td>6.5</td>
<td>Usage</td>
</tr>
<tr>
<td>human-pose-estimation-0001</td>
<td>Pose estimation (intel)</td>
<td>7.3</td>
<td>Usage</td>
</tr>
</tbody>
</table>

- **deeplabv3p_person** - Allows to highlight parts of the image where a person is detected
  
  `$ python3 depthai_demo.py -cnn deeplabv3p_person`

- **face-detection-adas-0001** - Allows to detect faces on the image (slower)
  
  `$ python3 depthai_demo.py -cnn face-detection-adas-0001`
• **face-detection-retail-0004** - Allows to detect faces on the image (faster)

```bash
$ python3 depthai_demo.py -cnn face-detection-retail-0004
```
• **mobilenet-ssd** - Object detector that detects 20 different classes (default)

```bash
$ python3 depthai_demo.py --cnn mobilenet-ssd
```
- pedestrian-detection-adas-0002 - allows to detect people on the image (slower)

$ python3 depthai_demo.py -cnn pedestrian-detection-adas-0002

- person-detection-retail-0013 - allows to detect people on the image (faster)
$ python3 depthai_demo.py -cnn person-detection-retail-0013

$ python3 depthai_demo.py -cnn person-vehicle-bike-detection-crossroad-1016

- **person-vehicle-bike-detection-crossroad-1016** - allows to detect both people, bikes and vehicles on the image
• **yolo-v3** - Object detector that detects 80 different classes (slower)

```bash
$ python3 depthai_demo.py -cnn yolo-v3
```
• **tiny-yolo-v3** - Object detector that detects 80 different classes (faster)

```
$ python3 depthai_demo.py -cnn tiny-yolo-v3
```
• `vehicle-detection-adas-0002` - allows to detect vehicles on the image

```bash
$ python3 depthai_demo.py -cnn vehicle-detection-adas-0002
```

• `vehicle-license-plate-detection-barrier-0106` - allows to detect both vehicle and license
plate on the image (only Chinese license plates)

```
$ python3 depthai_demo.py -cnn vehicle-license-plate-detection-barrier-0106
```

- **openpose2** - Human pose estimation model

```
$ python3 depthai_demo.py -cnn openpose2
```
• human-pose-estimation-0001 - Human pose estimation model from Open Model Zoo

$ python3 depthai_demo.py -cnn human-pose-estimation-0001

All of the data we use to download and compile a model can be found here.

4.1.9 Demo usage

In this section, we'll go through the config options available in the demo script, allowing you to try out different configurations
AI Properties

Switches:

- **Enabled**: Turns AI on/off. Turning it off will prevent any neural network from being run, which will also save some of the memory. Useful for situations where we’re more focused on Depth/Encoding rather than AI processing.

Basic properties:

- **CNN Model**: Choose which model to run on DepthAI, see more at *Using other models*.
- **SHA VE s**: Determines the number of SHA VE cores used to compile the neural network. The higher the value, the faster network can run, but this also limits features that can be enabled at once.
- **Model source**: Specify which camera preview will be taken as model input, so which frames will be sent to a neural network to perform inference.
- **Full FOV**: If enabled, it will scale down the camera image to meet the nn input size. If disabled, before scaling, it will crop the image to meet the NN aspect ratio.

Advanced:

- **OpenVINO version**: Specify the OpenVINO version that will be used to compile the MyriadX blob and run the pipeline. It’s recommended to use the latest version for most cases.
- **Label to count**: Allows to display a count of a specific label visible in the demo (e.g. if you want to count the number of cats in the preview with the default model, you can do so by selecting `cat` as a count label and restarting the script).
• **Spatial bounding box**: When enabled, will draw a bounding box on depth previews that show which part of the detection area was taken into depth estimation.

• **SBB Scale Factor**: Determines the size of the spatial bounding box in comparison to the detection bounding box.

**Depth Properties**

![Depth Properties](image)

Switches:

• **Enabled**: Turns Depth on/off. Turning it off will prevent the stereo node from being created, which will also save some of the memory. Useful for situations where we're more focused on AI processing/Encoding rather than depth.

• **Use Disparity**: If not set (default), the demo script will calculate the disparity map on host, based on the depth map. If enabled, the same processing will be performed on the device, consuming some of the memory but limiting host resource usage.

Basic properties:

• **Median Filtering**: Specifies the type of noise removing median filter applied to the depth map

• **Subpixel**: Enables subpixel mode, which improves the depth precision and is especially useful for long-range measurements

• **Left Right Check**: Enables left-right check, that is used to remove incorrectly calculated disparity pixels due to occlusions at object borders

• **Extended Disparity**: Enables extended disparity mode, that allows a closer minimum distance for the given baseline

• **Depth Range**: Specifies the minimum and maximum distance calculated by the device

• **LRC Threshold**: Specifies the max difference between disparity pixels that will make the pixel invalidated (the higher threshold, the more points will get through)
Camera Properties

Basic properties:
- **FPS**: Specifies how fast should the camera capture the frames
- **Resolution**: Specifies camera sensor resolution and thus captured frame size

Advanced:
- **ISO**: Controls the light gathering ability of the camera
- **Exposure**: Controls the exposure time of the camera
- **Saturation**: Controls the intensity of colors in a frame
- **Contrast**: Controls the visual ratio of different tones in a frame
- **Brightness**: Controls how dark or light the colors are in a frame
- **Sharpness**: Controls the clarity of details in a frame
Recording:
- **Switches**: enable recording of specified camera
- **FPS inputs**: specifies recording FPS (default 30)
- **Destination**: specifies a directory path where the recordings will be stored

Reporting:
- **Switches**: enable logging of the specified feature
- **Destination**: specifies a file path where the report file will be stored

### 4.1.10 Using custom models

**Warning**: Using custom models requires a locally downloaded version of the depthai repository. If you've been using installer to download and run the script, adding custom models won't work. Please follow Setup section on how to download and set up the repository.

Let’s assume you want to run a custom model which you downloaded from the model zoo or trained yourself (or both). To prepare your model to be runnable on DepthAI, it has to be compiled into MyriadX blob format - which is an optimized version of your model, capable of utilizing MyriadX chip as a processing unit.
In our demo script, we support a few ways you can run your custom blob, which will be covered below. As an example, I’ll add a custom face detection network called `custom_model` (substitute with your preferred name) and run it with the demo script.

**Compile MyriadX blob**

To receive a MyriadX blob, the network has to be already in OpenVINO IR format (consisting of `.xml` and `.bin` files) that will be used for compilation. We won’t focus here on how to obtain this representation for your model, but be sure to check official OpenVINO conversion guide.

To convert `custom_model.xml` and `custom_model.bin`, we’ll use the `blobconverter` cli - our tool that utilizes Online MyriadX blob converter to perform the conversion. No local OpenVINO installation is needed in this case, as all of the dependencies are already installed on the server. If your model is in TensorFlow or Caffe format, you can still use our tool for conversion, just note that you’ll have to use different input flags and sometimes provide a custom model optimizer args (Read more)

First, let’s install `blobconverter` from PyPi

```bash
$ python3 -m pip install -U blobconverter
```

Now, having the `blobconverter` installed, we can compile our IR files with the following command

```bash
$ python3 -m blobconverter --openvino-xml /path/to/custom_model.xml --openvino-bin /...-path/to/custom_model.bin
```

By running this command, `blobconverter` sends a request to the BlobConverter API to perform a model compilation on provided files. After compilation, the API responds with a `.blob` file and deletes all source files that were sent with the request.

After a successful compilation, `blobconverter` returns the path to the downloaded blob file. Since this blob is required by the `depthai` repository, let’s move it there

```bash
$ mkdir <depthai_repo>/resources/nn/custom_model
$ mv <path_to_blob> <depthai_repo>/resources/nn/custom_model
```

**Configuration**

We need to provide some additional configuration for the demo script to run this blob. The demo script will look for a `custom_model.json` for details on how to configure the pipeline and parse the results.

If your model is based on MobileNetSSD or Yolo, you can use our detection output format. If it’s a different type of network, you can use raw (default) output format and provide a custom handler file to decode and display the NN results.

You can use these configuration examples to customize your `custom_model.json` inside `resources/nn/custom_model` directory

- **MobileNetSSD** (we will use this config)

```json
{
    "nn_config": {
        "output_format": "detection",
        "NN_family": "mobilenet",
        "confidence_threshold": 0.5,
        "input_size": "300x300"
    }
}
```

(continues on next page)
• Yolo

```json
{
   "nn_config": {
      "output_format": "detection",
      "NN_family": "YOLO",
      "input_size": "416x416",
      "NN_specific_metadata": {
         "classes": 80,
         "coordinates": 4,
         "anchors": [10,14, 23,27, 37,58, 81,82, 135,169, 344,319],
         "anchor_masks": {
            "side26": [1,2,3],
            "side13": [3,4,5]
         },
         "iou_threshold": 0.5,
         "confidence_threshold": 0.5
      }
   },
   "mappings": {
      "labels": [
         "unknown",
         "face"
      ]
   }
}
```

• Raw (see Custom handler on details how to create handler.py file)

```json
{
   "nn_config": {
      "output_format": "raw",
      "input_size": "300x300"
   },
   "handler": "handler.py"
}
```
Run the demo script

Having the files in place, we can now run the demo with our custom model.

And you should see the output and your NN results displayed (or printed in the console if `raw` was selected and there is no handler file).
Be sure to check the advanced sections below or see *Next steps*

**Custom handler**

Custom handler is a file that the demo script will load and execute to parse the NN results. We specify this file with `handler` config value, specifying a path to the file of preference. It also requires `raw` output format, since it prevents the script from handling the results itself.

The `handler.py` file should contain two methods - `decode(nn_manager, packet)` and `draw(nn_manager, data, frames)`

```python
def decode(nn_manager, packet):
    pass

def draw(nn_manager, data, frames):
    pass
```

The first method, `decode`, is called whenever a NN packet arrives from the pipeline (stored as a `packet` param) also providing a `nn_manager` object that contains all nn-related info that was used by the script (like input size etc.). The goal of this function is to decode the received packets from the NN blob into meaningful results that can later be displayed.

Second one, `draw`, is called with the NN results (returned from `decode`), `nn_manager` object and `frames` array, having `[(<frame_name>, <frame>), (<frame_name>, <frame>), ...]` items. This array will contain frames that were specified with the `-s/--show` param. The goal of this function is to draw the decoded results onto received frames.

Below, you can find an example `handler.py` file that decodes and displays MobilenetSSD-based results.

```python
import cv2
import numpy as np
from depthai_helpers.utils import frame_norm

def decode(nn_manager, packet):
    bboxes = np.array(packet.getFirstLayerFp16())
    bboxes = bboxes.reshape((bboxes.size // 7, 7))
    bboxes = bboxes[bboxes[:, 2] > 0.5]
    labels = bboxes[:, 1].astype(int)
    confidences = bboxes[:, 2]
    bboxes = bboxes[:, 3:7]
    return {
        "labels": labels,
        "confidences": confidences,
        "bboxes": bboxes
    }

decoded = ["unknown", "face"]

def draw(nn_manager, data, frames):
    for name, frame in frames:
        if name == nn_manager.source:
            for label, conf, raw_bbox in zip(*data.values()):
                bbox = frame_norm(frame, raw_bbox)
                cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
```

(continues on next page)
cv2.putText(frame, decoded[label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
cv2.putText(frame, f"\{int(conf * 100)\}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

With the custom face detection model, using this code we receive the following output

![Face Detection Output](image)

We already use this handler mechanism to decode `deeplabv3p_person`, that comes as one of the available networks in the demo script to use.

**On-demand compilation**

Since files in the IR format can be large, and we’re both downloading the blob and uploading IR format to the server, we have incorporated an OpenVINO-like `model.yml` file structure that the BlobConverter server uses internally as well. You can check how this file looks like in OpenVINO model zoo or in available models in demo script.

This file is used by the OpenVINO model downloader that is used to download the required files for compilation. In our demo script, we use these files to provide a URL to the NN source files instead of uploading them along with the source code. It is also useful because on-demand compilation allows us to use the same configuration while requesting a different amount of MyriadX SHA VE cores.

To download the blob using `model.yml` file, run

```bash
$ python3 -m blobconverter --raw-config /path/to/model.yml --raw-name custom_model
```

You can also leave the `model.yml` file inside the `resources/nn/<name>` directory. This will make the demo script perform the conversion for you and run the received blob.
4.1.11 Customize the demo code

Callbacks file

If you’d like to add some custom functionalities to the demo yourself or just check how do certain variables look like, you can use a callbacks file, that should contain methods that the demo will call during execution on specific events. An example of a callbacks file that is available in the repo is below

```python
def shouldRun():
    pass  # Called to determine if the demo should be running

def onNewFrame(frame, source):
    pass  # Called when a new frame is available

def onShowFrame(frame, source):
    pass  # Called when a frame is about to be displayed

def onNn(nn_packet):
    pass  # Called when a new NN packet is available

def onReport(report):
    pass  # Called when a new report is available

def onSetup(*args, **kwargs):
    pass  # Called when the demo script is setting up

def onTeardown(*args, **kwargs):
    pass  # Called when the demo script is finishing

def onIter(*args, **kwargs):
    pass  # Called on each demo script iteration (internal loop)
```

These methods allow building custom functionalities on top of the demo script itself, whether it’s just printing or counting the data from NN or modifying how to display the frames or even make custom database/API connections to send the data to external destination.

By default, the demo script will use `callbacks.py` file in the repository, but this path can be changed using `-cb <path>` / `--callback <path>` flags.
Importing demo as a class

If preferred, the Demo script can also be imported like a regular class - this allows to control when the demo is starting up and when does it run. A simple example of how to run the demo from Python code is shown below

```python
from depthai_demo import Demo
from depthai_helpers.arg_manager import parseArgs
from depthai_helpers.config_manager import ConfigManager

args = parseArgs()
conf = ConfigManager(args)
demo = Demo(onNewFrame=<fn>, onShowFrame=<fn>, onNn=<fn>, onReport=<fn>, onSetup=<fn>,
            onTeardown=<fn>, onIter=<fn>) # all params are optional
demo.run_all(conf)
```

Remember that `PYTHONPATH` env variable must contain a path do depthai repository for the imports to work. Alternatively, you can place the script along `depthai_demo.py` in the repository.

### 4.1.12 Next steps

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

- **Looking for inspiration?**
  Check our [Example Use Cases](#) for ready to use applications that solve a specific problem on DepthAI

- **Want to start coding?**
  Be sure to check [hello world tutorial](#) on API section for a step-by-step introduction to the API

- **Want to train and deploy a custom model to DepthAI?**
  Visit [Custom training](#) page for ready to use Colab notebooks

We’re always happy to help with code or other questions you might have.

### 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, the *depthai demo* 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 landmowers (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 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.

We’re always happy to help with code or other questions you might have.

4.3 AI / ML / NN

4.3.1 Converting model to MyriadX blob

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 Model Compiler to convert IR representation into MyriadX blob

Image below (from OpenCV Courses site) shows these steps
Below, please find instructions on how to perform these steps using different methods

**Local compilation**

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

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

**Using Google Colab**

You can also train and convert models using Google Colab notebook. You can take a look at our Custom training page, where every tutorial contains also conversion & compilation steps performed directly inside the notebooks.

An example notebook with the compilation steps is here

**Using online converter**

You can also visit our online MyriadX Blob converter that allows specifying different OpenVINO target versions and supports conversions from TensorFlow, Caffe, OpenVINO IR and OpenVINO Model Zoo
Using blobconverter package

For automated usage of our blobconverter tool, we have released a blobconverter PyPi package, that allows compiling MyriadX blobs both from the command line and from the Python script directly.

Install and usage instructions can be found here

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 Caffe, 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.

We’re always happy to help with code or other questions you might have.

### 4.3.2 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 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`:

![Image of laptop screen with detected object]

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:

![Image of face detection]

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 (deplabv3p_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):

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

Play with the feature and please share demos that you come up with (especially if you make a robot that stalks your cat) on discuss.luxonis.com and if you run into any issues, please ping us on our Discord server.

And if you find any errors in these documents, please report an issue on the Github.

We’re always happy to help with code or other questions you might have.

### 4.3.3 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 Google colab, our online converter or the 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 repository.

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.

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
  -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:

![Example output image]

**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_NUM_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 significantly simplified.

We’re always happy to help with code or other questions you might have.
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 segmentation
- 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 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 above example used a DepthAI Modular Cameras Edition (BW1098FFC).

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.

Below is a quick test of the model produced with this notebook on OAK device:
Supporting Notebooks

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

We’re always happy to help with code or other questions you might have.

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).

To run a custom AI model on the device, you need to convert it to the .blob format - documentation here.

You can also choose to use one of 250+ pretrained AI models from either OpenVINO Model Zoo or DepthAI Model Zoo, read more at Use a Pre-trained OpenVINO model.
4.3.5 Model Performance

You can estimate the performance of your model with the help of the chart below. It contains FPS estimations of models on OAK devices in dependence of FLOPs and parameters.

Model's performance in FPS dependent on MPrams and GFlops

![Graph showing performance in FPS vs. MPrams and GFlops]

FPS: 0.00 - 5.00 • 5.00 - 15.00 • 15.00 - 25.00 • 25.00 - 50.00 • 50.00 - 100.00 • 100.00+

You can find more detailed evaluation of FPS for common models in this sheet.

4.3.6 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.

We’re always happy to help with code or other questions you might have.
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.

We’re always happy to help with code or other questions you might have.
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:

   ```python
   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:

   ```python
   # 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.

```python
import onnx
from onnxsimplify import simplify

onnx_model = onnx.load("path/to/model.onnx")
model_simplified, check = simplify(onnx_model)
onnx.save(model_simplified, "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:

```python
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:

```python
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.

We’re always happy to help with code or other questions you might have.

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

We’re always happy to help with code or other questions you might have.

4.6 On-device programming

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

4.6.1 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.

4.6.2 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.

4.6.3 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

We’re always happy to help with code or other questions you might have.
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. The DepthAI reference Python script can be used to show this out (python3 depthai_demo.py -gt cv -s depth -sbb is the command used to produce the video above).

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 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. However sub-pixel and extended disparity are not yet supported simultaneously, but should be available in the near future (Pull Request). 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?

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

We named it mega because it’s tiny:
megaAI uses all the same hardware, firmware, software, and training stacks as DepthAI (and uses the same DepthAI Github repositories), it is simply the tiny single-camera variant.

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

You can buy megaAI from our distributors and also our online store 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 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.

- **Embedded with WiFi/BT (OAK-D-IoT-40 and OAK-D-IoT-75)** - We have two models that have additional 128MB NOR flash, so they can boot on their own out of the NOR flash, and no host needs to be present to run. In contrast, the OAK-D-CM4 can also run on its own, but it is still booting over USB from the Raspberry Pi. On OAK-D-IoT-40 and OAK-D-IoT-75, the Myriad X can run completely standalone and with no other
devices. The built-in ESP32 then provides easy/convenient WiFi/BT support (more info here) as well as popular integrations like plug-and-play AWS-IoT support, great iOS/Android BT examples, etc.

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.
The command to get the above output is

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

Here is a single-camera version (megaAI) running with `python3 depthai_demo.py -gt cv -s color`:
fps: 30

(person: 71.0%, motorbike: 90.1%)

(x=294, y=161) ~ R:160 G:157 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?

Yes, every model can be used, including:

- OAK-D-CM4 - this one has a built-in Raspberry Pi Compute Module 4
- OAK-D
- OAK-FFC-3P
- OAK-1
- Raspberry Pi HAT (BW1094) - this can also be used with other hosts as its interface is USB3

We even have some ROS support going as well which can be used on the Pi also.

4.7.13 Does DepthAI Work on the Nvidia Jetson Series?

Yes, DepthAI and megaAI work 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, DepthAI can be used in two modalities:

- 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.
- DepthAI Mode (USB, using our USB API, here) - this uses the onboard cameras directly into the Myriad X, 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.).
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. We even have an embedded reference design for ESP32 (OAK-D-IoT-40 (BW1092)) available on our store. And it’s open-source! You can check design files here.

We have prepared a Getting started with OAK IoT devices guide and a demo on how to work with ESP32.

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, or an ESP32 like on the OAK-D-IoT-40 (which comes with the OAK-SoM-IoT pre-installed).
- **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, a microcontroller over SPI (in development), or completely standalone - we recommend starting with either NCS2 mode or with the DepthAI USB API for prototype/test/etc. as it allows faster iteration/feedback on neural model performance/etc. And in particular, with NCS2 mode, all the images/video can be used directly from the host (so that you don’t have to point the camera at the thing you want to test).

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 our 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:

\[
\text{min_distance} = \frac{\text{focal_length_in_pixels} \times \text{base_line_dist}}{\text{max_disparity_in_pixels}}
\]

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

\[
\text{focal_length_in_pixels} = \frac{1280 \times 0.5}{\tan(71.9 \times 0.5 \times \pi / 180)} = 882.5
\]

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:

\[
\text{min_distance} = 882.5 \times 7.5\text{cm} / 95 = 69.67\text{cm}
\]

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:

\[
\text{min_distance} = 882.5 \times 7.5\text{cm} / 190 = 34.84\text{cm}
\]

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 minumum 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:

\[
\text{min_distance} = 882.5 \times 0.09 / 95 = 0.836 \text{ m}
\]

(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):

\[
\text{min_distance} = 882.5 \times 0.075 / 95 = 0.697 \text{ m}
\]

(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 19.6 cm.

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

\[
\text{min} \_ \text{distance} = \tan\left(\frac{90 - \text{HFOV}/2}{2}\right) \times \text{base} \_ \text{line} \_ \text{dist} / 2
\]

---

**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 \( \text{focal} \_ \text{length} \times \text{base} \_ \text{line} \_ \text{dist} / 190 \) instead of \( \text{focal} \_ \text{length} \times \text{base} \_ \text{line} \_ \text{dist} / 95 \)).

- **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 \( d = \text{disparity}_{LR}(x,y) \) is compared with \( \text{disparity}_{RL}(x-d,y) \). 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 it’s 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):
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 [StereoDepth 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 \(\text{disparity}\), and for depth results \(\text{depthRaw}\), use the following command:

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

The available streams are:

- \text{nnInput} - Neural Network passthrough frames on which inference was made on (300x300 in case of MobileNet)
- \text{color} - 4K color camera, biggest camera on the board with lens
- \text{left} - Left grayscale camera (marked \(L\) or LEFT on the board)
- \text{right} - Right grayscale camera (marked \(R\) or RIGHT on the board)
- \text{rectifiedLeft} - Rectified left camera frames
- \text{rectifiedRight} - Rectified right camera frames
- \text{depth} - Depth in \text{uint16}
- \text{depthRaw} - Raw frames which are used to calculate depth
- \text{disparity} - Raw disparity
- \text{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, see example code here (using API). You can also get the raw stereo pair stream with DepthAI demo using the following command:

```
python3 depthai_demo.py -gt cv -s left right
```

4.7.33 How to choose the DepthAI Demo GUI type?

Since Depthai Demo v3.0.0, we introduced two GUI types available:

- Qt-based interactive GUI \(\text{qt}\), that allows to change demo options with mouse clicks
- OpenCV-based preview \(\text{cv}\), being the default prior v3), that allows to preview requested streams and control the demo using CLI arguments

For most of the platforms, \text{qt} is selected as a default GUI type and \text{cv} serves as a fallback in case QT installation doesn’t work. You can also choose GUI type yourself with \(-gt / --guiType\) argument.

For example, to enforce OpenCV GUI, run the following command:

```
python3 depthai_demo.py -gt cv
```

Or, to enforce QT GUI:

```
python3 depthai_demo.py -gt qt
```
4.7.34 How Do I Limit The Camera FrameRate?

So the simple way to select streams is to just use the `-s` option. But in some cases (say when you have a slow host or only USB2 connection and you want to display a lot of streams) it may be necessary to limit the frame rate of streams to not overwhelm the host/USB2 with too much data.

So to set streams to a specific frame rate to reduce the USB2 load and host load, use `-rgbf / --rgbFps` to limit the RGB camera framerate and `-monof / --monoFps` to limit mono cameras framerate.

So for limiting color camera to 5 FPS, use the following command:

```bash
python3 depthai_demo.py -gt cv -rgbf 5
```

And same way to limit mono cameras to 5 FPS:

```bash
python3 depthai_demo.py -gt cv -monof 5
```

It’s worth noting that the frame rate limiting works best for lower rates. So if you’re say trying to hit 25FPS, it’s best to just leave no frame-rate specified and let the system go to full 30FPS instead.

Specifying no limit will default to 30FPS.

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

The `--sync` option is used to synchronize the neural inference results and the frames on which they were run. When this option is used, the device-side firmware makes a best effort to send metadata and frames in order of metadata first, immediately followed by the corresponding image.

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 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:

```bash
python3 depthai_demo.py -gt cv -rgbf 24 -monof 24
```
Synchronizing on the Host

**ImgFrame** message has two functions; `ImgFrame.getTimestamp()` and `ImgFrame.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.

We have both timestamp and sequence number syncing demos [here](#).

### 4.7.36 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 an 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.

![Example Video](image)

### 4.7.37 What are the Capabilities of the Video Encoder on DepthAI?

See capabilities and limitations in the documentation here.

### 4.7.38 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.

So the following results are an approximation of this photon-to-results latency, and are likely an over-estimate as we tested by actually seeing when results were updated on a monitor, and the monitor itself has some latency, so the results below are likely overestimated by whatever the latency of the monitor is that we used during the test. And we have also since done several optimizations since these measurements, so the latency could be quite a bit lower than these.
Table 6: Worst-case estimates of stream latency

<table>
<thead>
<tr>
<th>measured</th>
<th>requested</th>
<th>avg latency, ms</th>
<th>DepthAI demo arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>left</td>
<td>90</td>
<td>python3 depthai_demo.py -gt cv -s left</td>
</tr>
<tr>
<td>left</td>
<td>left, right</td>
<td>90</td>
<td>python3 depthai_demo.py -gt cv -s left</td>
</tr>
<tr>
<td>left</td>
<td>left, right, depth</td>
<td>90</td>
<td>python3 depthai_demo.py -gt cv -s left</td>
</tr>
<tr>
<td>left</td>
<td>left, right, depth, color, object detection</td>
<td>100</td>
<td>python3 depthai_demo.py -gt cv -s left depth color</td>
</tr>
<tr>
<td>left</td>
<td>left, right, depth, color, object detection (synced)</td>
<td>200</td>
<td>python3 depthai_demo.py -gt cv --sync -s left right depth color</td>
</tr>
</tbody>
</table>

Note that object detection results are not synced with the color camera if --sync isn’t added to the arguments.

4.7.39 How To do a Letterboxing (Thumbnails) on the Color Camera?

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

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

Yes, see documentation here.

4.7.41 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 usb2Mode to True when creating the device (note - it works for gen2):

```python
import depthai

dai.Device(pipeline, usb2Mode=True)
```

The other way is using the -usbs usb2 (or --usbSpeed usb2) command line option as below:

```bash
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).
4.7.42 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 OpenVINO’s 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 OpenVINO’s 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.

4.7.43 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.

4.7.44 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.

4.7.45 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.
4.7.46 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.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>USB3 Gen1 (5gbps)</th>
<th>USB3 Gen2 (10gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12MP (4056x3040)</td>
<td>21.09fps (390MB/s)</td>
<td>41.2fps (762MB/s)</td>
</tr>
<tr>
<td>4K (3840x2160)</td>
<td>30.01fps (373MB/s)</td>
<td>60.0fps (746MB/s)</td>
</tr>
</tbody>
</table>

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.

4.7.47 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%.

### 4.7.48 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.

### 4.7.49 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](#).

### 4.7.50 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)

### 4.7.51 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 the `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] 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].

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%`

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

OAK-D, OAK-1, OAK-D-IoT-40, 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.
4.7.53 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:

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

4.7.54 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.

4.7.55 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 smart phone.

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.

4.7.56 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.

4.7.57 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.

4.7.58 How to use DepthAI under VirtualBox

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

4.7.59 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 reformattting 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.
4.7.60 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*.

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

This is a particularly interesting application of DepthAI, as it allows the Raspberry Pi HQ camera to be encoded to h.265 4K video (and 12MP stills) even with a Raspberry Pi 1 or Raspberry Pi Zero - because DepthAI 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 Pi HQ camera directly, can only do 1080p video and not 4K video recording).

**OAK-FFC-3P** and **OAK-FFC-4P** will *work with* the Raspberry Pi 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 board.

**OAK-FFC-3P-OG** model **also works via an adapter board** with the Raspberry Pi HQ camera (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.

Here are some quick images and videos of it in use:
You can buy this adapter kit for the OAK-FFC-3P-OG here
4.7.62 Can I use DepthAI with Raspberry Pi Zero?

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

4.7.63 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:

4.7.64 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.

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

**Embedded Use Case**

- [https://github.com/luxonis/depthai-experiments/tree/master/gen2-spi](https://github.com/luxonis/depthai-experiments/tree/master/gen2-spi) - user examples of SPI api and standalone mode.

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](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](https://github.com/luxonis/depthai-spi-api) - SPI interface library for Embedded (microcontroller) DepthAI application
• [https://github.com/luxonis/esp32-spi-message-demo/tree/gen2_common_objdet](https://github.com/luxonis/esp32-spi-message-demo/tree/gen2_common_objdet) - ESP32 Example applications for Embedded/ESP32 DepthAI use (e.g. with OAK-D-IoT-40)

### 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](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](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 support for the BNO086 (and BNO080/BNO085) IMU. See documentation on IMU [here](https://github.com/luxonis/depthai-core-example).

### 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?

For more information about OAK devices, go to Hardware Documentation Page.

**Brochures:**

- Editions Summary [here](#)
- OAK-SoM (System on Module) [here](#)
- OAK-FFC-3P-OG (USB3 Modular Cameras Edition) [here](#)
- OAK-D-PCBA [here](#)
- OAK-D-CM3 (Raspberry Pi Compute Edition Module) [here](#)
- Raspberry Pi HAT (BW1094) [here](#)
- OAK-1 [here](#)

**Datasheets:**

- DepthAI System on Module (OAK-SoM) [here](#)
- DepthAI System on Module Pro (OAK-SoM-Pro) [here](#)
- DepthAI System on Module IoT (OAK-SoM-IoT) [here](#)
- PoE Modular Cameras Edition (BW2098FFC) [here](#)

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.

```bibtex
@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, DepthAI, and megaAI logos 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:

- Luxonis Community Discord. Use this for real-time communication with our engineers. We can even make dedicated channels for your project/effort public or private in here for discussions as needed.

- 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.

- 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.

We’re always happy to help with code or other questions you might have.

4.8 Calibration

Note: All non-modular OAK devices are calibrated before shipment. It is not required to (re-)calibrate any of these, but for PCBA-only models like OAK-D-IoT-75, OAK-D-IoT-40, OAK-D-CM4, OAK-D-CM3, OAK-D-PCBA it can be desirable to do so if the depth quality degrades from mounting the PCBA (into an enclosure).

For the modular camera editions of DepthAI (OAK-FFC-3P and DepthAI RaspberryPi Hat) it is necessary to do a stereo camera calibration after mounting the cameras in the baseline/configuration for your application.

Below is a quick video showing the (re-) calibration of the OAK-D.

In short, the calibration uses the intersections to determine the orientation and distance of the charuco board. So the greatest accuracy will be obtained by a clear print or display of the provided board image on a flat plane.

The flatness of the calibration board is very important. Do not use curved monitors, or calibration targets that have any ‘waviness’. So if you print the charuco board, please make sure to affix the sheet to a known flat surface, without any waves. That said, using a laptop with a flat monitor is usually the easiest technique.

Watching the video below will give you the steps needed to calibrate your own DepthAI. For more information/details on calibration options, please see the steps below and also ./calibrate.py --help which will print out all of the calibration options.

1. Checkout the depthai GitHub repo.

Warning: Already checked out depthai? Skip this step.
2. Print charuco board calibration image.

Either print the calibration charuco board onto a flat surface, or display the calibration board on a flat (not curved!) monitor. Note that if you do print the calibration target, take care to make sure it is attached to a flat surface and is flat and free of wrinkles and/or ‘waves’.

Often, using a monitor to display the calibration target is easier/faster.

3. Start the calibration script.

Replace the placeholder argument values with valid entries:

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -brd [BOARD] -db
```

Argument reference:

- **-s SQUARE_SIZE_IN_CM, --squareSizeCm SQUARE_SIZE_IN_CM**: Measure the square size of the printed charuco board in centimeters.
- **-db, --defaultBoard**: flag that specifies that we’re using a calibration board provided in depthai repository (from calib.io). If you’re using other board, please instead specify **-ms MARKER_SIZE_IN_CM, --markerSizeCm MARKER_SIZE_IN_CM**: Marker size on the printed charuco board in centimeters.

![Charuco board image](image_url)

The entire board should fit on a single piece of paper (scale to fit). And if displaying on a monitor, full-screen the image with a white background.
• `-brd BOARD, --board BOARD`: BW1097, BW1098OBC - Device board type from resources/boards/ (not case-sensitive, if you’re using OAK-D please choose BW1098OBC) or path to a custom .json board config.

Retrieve the size of the squares from the calibration target by measuring them with a ruler or calipers and enter that number (in cm) in place of `[SQUARE_SIZE_IN_CM]`.

For example, the arguments for the OAK-D look like the following if the square size is 2.35 cm:

```
python3 calibrate.py -s 2.35 -brd bw1098obc -db
```

And note that mirroring the display when calibrating is often useful (so that the directions of motion don’t seem backwards). When seeing ourselves, we’re used to seeing ourselves backwards (because that’s what we see in a mirror), so do so, use the `-ih` option as below:

```
python3 calibrate.py -s 2.35 -brd bw1098obc -db -ih
```

So when we’re running calibration internally we almost always use the `-ih` option, so we’ll include it on all the following example commands:

• OAK-D:

  ```
  python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -db -brd bw1098obc -ih
  ```

• OAK-D-CM4:

  ```
  python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -db -brd bw1097 -ih
  ```

### 4.8.1 Modular cameras calibration

Use one of the board `*.json` files from here to define the baseline between the stereo cameras, and between the left camera and the color camera, replacing the items in brackets below.

• Swap left/right (i.e. which way are the cameras facing, set to `true` or `false`)
• The BASELINE in centimeters between grayscale left/right cameras
• The distance RGBLEFT separation between the Left grayscale camera and the color camera, in centimeters.

```
{
    "board_config":
    {
        "name": "ACME01",
        "revision": "V1.2",
        "swap_left_and_right_cameras": [true | false],
        "left_fov_deg": 73.5,
        "rgb_fov_deg": 68.7938,
        "left_to_right_distance_cm": [BASELINE],
        "left_to_rgb_distance_cm": [RGBLEFT]
    }
}
```

So for example if you setup your OAK-FFC-3P with a stereo baseline of 2.5cm, with the color camera exactly between the two grayscale cameras, as shown below, use the JSON further below:
Note that in this orientation of the cameras, "swap_left_and_right_cameras" is set to true.

Then, run calibration with this board name:

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -db -brd ACME01 -ih
```

Run `python3 calibrate.py --help` (or `-h`) for a full list of arguments and usage examples.
4.8.2 Position the charuco board and capture images

Left and right video streams are displayed, each containing a polygon overlay.

Hold up the printed charuco board (or laptop with the image displayed on the screen) so that the whole of the calibration board is displayed within both video streams.

Match the orientation of the overlayed polygon and press [SPACEBAR] to capture an image. The charuco board pattern does not need to match the polygon exactly, but it is important to use the polygon as a guideline for angling and location relative to the camera. There are 13 required polygon positions.

After capturing images for all of the polygon positions, the calibration image processing step will begin. If successful, the calibration data will be written to EEPROM and a copy of it will be created in files under depthai/resources/ as <Device Mx ID>.json if device is connected or depthai_calib.json otherwise.

It will also create the mesh files named left_mesh.calib and right_mesh.calib under depthai/resources/ which can be used to overcome distortions in stereo node for camera modules with distortions.

4.8.3 Test depth

We’ll view the depth stream to ensure the cameras are calibrated correctly:

1. Start a terminal session.
2. Access your local copy of depthai.
   
   ```
   cd [depthai repo]
   ```
3. Run test script.
   
   ```
   python3 depthai_demo.py
   ```

   The script launches a window, starts the cameras, and displays a depth video stream:
In the screenshot above, the hand is closer to the camera.

We’re always happy to help with code or other questions you might have.

### 4.9 OAK as a webcam

OAK devices can be used as webcams as well. To enable this feature, you can run the DepthAI demo UVC application or use uvc feature branch.
4.9.1 Using DepthAI Demo Script

First, follow the steps required to install DepthAI Demo script here.

If using QT GUI, please select the UVC app here.

```
Or, running directly from terminal

    python3 depthai_demo.py --app uvc
```

4.9.2 Using UVC Feature Branch

```
# Skip cloning if you already have depthai-python repo
    git clone https://github.com/luxonis/depthai-python.git
    cd depthai-python

# Navigate to the custom branch
    git checkout gen2_uvc
    python3 examples/install_requirements.py
    python3 examples/rgb_uvc.py
```

Now you can open up your favourite meeting app, like Zoom or Slack, and select Luxonis Device: UVC Video Contr in the webcam selection menu.
4.9.3 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:

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

We’re always happy to help with code or other questions you might have.
4.10 Troubleshooting

4.10.1 DepthAI can’t connect to an OAK camera

For USB OAK cameras, DepthAI can throw an error code like \texttt{X\_LINK\_COMMUNICATION\_NOT\_OPEN} or \texttt{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 \textit{Forcing USB2 Communication} for more information. Another common issue is that users haven’t set \texttt{udev rules} on their Linux machine.

If you still can’t connect to the OAK camera, you should execute \texttt{lsusb} | \texttt{grep 03e7}. You should see a similar line:

\begin{verbatim}
$ lsusb | grep 03e7
Bus 001 Device 120: ID 03e7:2485 Intel Movidius MyriadX
\end{verbatim}

Another thing to check is the \texttt{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:

\begin{verbatim}
/$ 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
\end{verbatim}

If these commands didn’t return the expected log, please send us an email to support@luxonis.com.

4.10.2 How can the start-up demo on the Raspberry Pi Compute Edition be disabled?

Delete the \texttt{autostart} file:

\begin{verbatim}
rm /home/pi/.config/autostart/runai.desktop
\end{verbatim}

4.10.3 ImportError: No module named ‘depthai’

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

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

\begin{verbatim}
python3 -m pip list | grep depthai
\end{verbatim}

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

\begin{verbatim}
cat /etc/os-release
\end{verbatim}
4.10.4 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 a 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.

4.10.5 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.

```
```

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.

4.10.6 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.

DepthAI implements the VSC (Vendor Specific Class) protocol and `libusb` is used for communication.

4.10.7 Intermittent Connectivity with Long (2 meter) USB3 Cables

- We’ve found that some hosts have trouble with USB3 + long cables (2 meter). It seems to have something to do with the USB controller on the host side.
- Other hosts have no problem at all and run for days (tested well over 3 days on some), even with long cables (tested w/ a total length of a bit over 8 feet). For example, all Apple computers we’ve tested with have never exhibited the problem.
- 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 or around when these new machines having hit the market. For example, on 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).

So unfortunately we discovered this after we shipped with long USB3 cables (2 meter cables) with DepthAI units. So if you have see this problem with your host, potentially 3 options:
1. Switch 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 now what we ship with DepthAI USB3 variants.

2. Force USB2 mode with `--usbSpeed usb2` option (examples below). This will allow use of the long cable still, and many DepthAI use cases do not necessitate USB3 communication bandwidth - USB2 is plenty.

3. Upgrade from Ubuntu 16.04 to Ubuntu 18.04.

### 4.10.8 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 `usb2Mode=True` when creating the device:

```python
# Force USB2 communication
with dai.Device(pipeline, usb2Mode=True) as device:
```

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

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

### 4.10.9 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).

### 4.10.10 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>"

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
```
4.10.11 CTRL-C Is Not Stopping It!

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

4.10.12 Is Your Raspberry Pi Locking Up or is DepthAI Crashing on Raspberry Pi?

The Raspberry Pi has a max limit of 1.2A across all its USB ports, and DepthAI/megaAI/OAK can take up to 1A (at max power, usually closer to 500mA).

So if you are seeing lockups, it could be that you are over this 1.2A limit as a result of the total power of the USB devices drawing from the Pi. Using a powered hub can prevent this, or powering fewer other things off of the Pi over USB.

This may also manifest in DepthAI randomly crashing on the Pi. This can become particularly often if DepthAI is configured to do many things at once. This is becoming increasingly likely as we enable DepthAI to do more and more things at once - and thereby increase the maximum power that DepthAI can pull. It is seeming like the peak power (current) spikes from DepthAI can go over what the Pi can handle, causing DepthAI to brown-out and return an error.

So if you are experiencing issues with DepthAI stability on Raspberry, try powering DepthAI via a power-supply and/or a powered USB hub to see if the error goes away.

4.10.13 “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)

4.10.14 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._<int8); cv2.imshow('s', blank_image); cv2.waitKey(0)"
```

If a window is not displayed, or if you get the `:bash: 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 `:bash: python3 -m pip install opencv-python --force-reinstall` will often fix the OpenCV problem.
4.10.15 Neural network blob compiled with incompatible openvino version

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):

```python
pipeline = depthai.Pipeline()
# Set the correct version:
pipeline setOpenVINOVersion(depthai.OpenVINO.Version.VERSION_2020_1)
```

4.10.16 “realloc(): invalid pointer Aborted” on RPi

On RPi, after running `sudo apt upgrade`, you might get the error `realloc(): invalid pointer
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+rp11, OR`
- Re-install DepthAI dependencies by running `sudo curl -fL http://docs.luxonis.com/_static/install_dependencies.sh | bash`

4.10.17 [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).

4.10.18 [error] input tesnor exceeds available data range

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.

We’re always happy to help with code or other questions you might have.
4.11 SLAM with OAK

SLAM (Simultaneous Localization And Mapping) can be performed using our OAK devices.

We plan on creating our own visual-inertial SLAM project that would use on-device feature tracking (using Feature-Tracker node) and inertial data (using IMU node), if the device has an IMU sensor onboard.

Our awesome community has also created several projects that use OAK for SLAM:

- ArduCam Visual SLAM tutorial
- OAKD_ORBSLAM3
- DepthAI-SLAM

You can also check out our #slam channel on our Discord server, where there are tons of useful information about how to perform SLAM on OAK devices.

We’re always happy to help with code or other questions you might have.

4.12 OAK on drones

Our amazing community has created several OAK on drones projects:

- Team QuetzalC++ (OpenCV AI comp) - Warehouse inspection with autonomous drones - video
- Team CVAR-U.P.Madrid (OpenCV AI comp) - Researching the autonomous Micro Aerial Vehicle (MAV) solution for catastrophic situations - 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
- @FPSychotic on discord - Simple ArduPilot avoidance demo with OAK-D - video

To use OAK devices with systems like PX4 or ArduPilot, one would need to develop integration of our devices/hardware with such systems.

Here are a couple of examples of PX4 sensors that could be interfaced by ESP32/other MCU and communicated to OAK to get needed data or trigger a certain action:

- Optical flow (requires a downward facing camera and a distance sensor)
- Camera
- Distance Sensors

For additional useful information about OAK on drones projects you can check out our #drone channel on our Discord server.
4.12.1 Camera vibration

Camera vibrations can be a big challenge in applications such as drones, especially for color cameras that doesn’t have fixed lens. 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:

![OAK-1 anti-vibration mount](image)

We are also planning on adding EIS (Electronic Image Stabilization) feature to the OAK - you can read more about it on our Roadmap.

We’re always happy to help with code or other questions you might have.

4.13 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.

4.13.1 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!

4.13.2 Looking for more?

You can also check out our #community channel on our Discord server, where you can meet people behind these projects or share your idea/demo - we’re looking forward to meeting you there!

We’re always happy to help with code or other questions you might have.
4.14 Support

Running into issues or have questions? We’re here to help.

You can get help in a number of ways:

- Email support@luxonis.com
- Join our Discord Community for live assistance from us and developers like you
- Post a message to our forum
- Open an issue our DepthAI Github repository

4.14.1 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.

We’re always happy to help with code or other questions you might have.