DepthAI Docs

Release 0.3.0.0

Luxonis

Jan 16, 2021

CONTENTS:

Learn how to setup your DepthAI device, view tutorials, code samples, and more.

DepthAI is the embedded spatial AI platform that helps you build products with true realtime 3D object localization (think 3D object detection) and tracking. DepthAI offloads AI, depth vision and more - processed direct from built-in cameras - freeing your host to process application-specific data. Best of all, it is modular and MIT-licenced open source, affording adding these Spatial AI/CV super powers to real commercial products.

CHAPTER

ONE

SETUP YOUR DEVICE

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

1.1 Python API

Instructions for installing, upgrading, and using the DepthAI Python API.

1.1.1 Supported Platforms

The DepthAI API python module is prebuilt for Ubuntu, macOS and Windows. For other operating systems and/or Python versions, DepthAI can be *built from source*.

1.1.2 Installing system dependencies

A couple of basic system dependencies are required to run the DepthAI library. Most of them should be already installed in most of the systems, but in case they are not, we prepared `an install script` that will make sure all dependencies are installed, along with convenient development/programming tools. There are also video guides available for macOS (here), Raspberry Pi (here), Ubuntu (here), and Windows 10 (here).

macOS

```
bash -c "$(curl -fL http://docs.luxonis.com/_static/install_dependencies.sh)"
```

Close and re-open the terminal window after this command.

The script also works on M1 Macs, Homebrew being installed under Rosetta 2, as some Python packages are still missing native M1 support. In case you already have Homebrew installed natively and things don't work, see here for some additional troubleshooting steps.

Note that if the video streaming window does not appear consider running the following:

```
python3 -m pip install opencv-python --force-reinstall --no-cache-dir
```

See the Video preview window fails to appear on macOS thread on our forum for more information.

Raspberry Pi OS

```
sudo curl -fL http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

Ubuntu

```
sudo wget -qO- http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

Windows

- Right click on Start
- Choose Windows PowerShell (Admin)
- Install Chocolatey package manager (similar to Homebrew for macOS):

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.
↪ServicePointManager]::SecurityProtocol = [System.Net.
↪ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.
↪WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

- Close the PowerShell and then re-open another PowerShell (Admin) by repeating the first two steps.
- Install Python and PyCharm

```
choco install cmake git python pycharm-community -y
```

1.1.3 Enabling the USB device (only on Linux)

Since the DepthAI is a USB device, in order to communicate with it on the systems that use `udev` tool, you need to add the udev rules in order to make the device accessible.

The following command will add a new udev rule to your system

```
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
```

1.1.4 Install from PyPi

Our packages are distributed via PyPi, to install it in your environment use

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

For other installation options, see *other installation options*.

1.1.5 Test installation

We have depthai repository on our GitHub that contains many helpful examples and prepared neural networks you can use to make your prototyping faster. It also includes the test script, maintained by our contributors, that should help you verify if your setup was correct.

First, clone the depthai repository and change directory into this repo:

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

Next install the requirements for this repo. Note that we recommend installing the dependencies in a virtual environment, so that they don't interfere with other Python tools/environments on your system.

- For development machines like Mac/Windows/Ubuntu/etc., we recommend the PyCharm IDE, as it automatically makes/manages virtual environments for you, along with a bunch of other benefits. Alternatively, *conda*, *pipenv*, or *virtualenv* could be used directly (and/or with your preferred IDE).
- For installations on resource-constrained systems, such as the Raspberry Pi or other small Linux systems, we recommend *conda*, *pipenv*, or *virtualenv*. To set up a virtual environment with *virtualenv*, run *virtualenv venv && source venv/bin/activate*.

Using a virtual environment (or system-wide, if you prefer), run the following to install the requirements for this example repository:

```
python3 install_requirements.py
```

Now, run the demo script from within depthai to make sure everything is working:

```
python3 depthai_demo.py
```

If all goes well a small window video display with overlays for any items for which the class exists in the example 20-class object detector (class list here).

1.1.6 Run Other Examples

After you have run this demo, you can either run `python3 depthai_demo.py -h` to see other neural networks that by-default can be run.

After checking this out, proceed to:

- Our tutorials, starting with how to use pre-trained neural models from OpenVINO, HERE
- Our experiments HERE to learn more ways to use DepthAI.

You can also proceed below to learn how to convert your own neural networks to run on DepthAI.

And we also have online model training below, which shows you how to train and convert models for DepthAI:

- Online ML Training and model Conversion: HERE

1.1.7 Preparing MyriadX blob file and it's config

As you can see in *example*, basic usage of `Device.create_pipeline()` method consists of specifying desired output streams and AI section, where you specify MyriadX blob and it's config.

In this section, we'll describe how to obtain both `blob_file` and `blob_file_config`.

Obtaining MyriadX blob

Since we're utilizing MyriadX VPU, your model needs to be compiled (or accurately - optimized and converted) into the MyriadX blob file, which will be sent to the device and executed.

Easiest way to obtain this blob is to use our online BlobConverter app. It has all tools needed for compilation so you don't need to setup anything - and you can even download a blob for the model from OpenVINO model zoo.

If you'd like, you can also compile the blob yourself. You'll need to install OpenVINO toolkit, then use Model Optimizer and Myriad Compiler in order to obtain MyriadX blob. We've documented example usage of these compilers here

Creating Blob configuration file

If config file is not provided or `output_format` is set to `raw`, no decoding is done on device and user must do it manually on host side.

Currently there is support to decode `Mobilenet-SSD` and `(tiny-)YOLO-v3` based networks on the device. For that config file is required with network specific parameters.

Example for *tiny-yolo-v3* network:

```
{
    "NN_config":
    {
        "output_format" : "detection",
        "NN_family" : "YOLO",
        "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":
        [
            "person",
            "bicycle",
            "car",
            "..."
        ]
```

(continues on next page)

(continued from previous page)

```
    }
}
```

- **`NN_config` - configuration for the network**
  - **`output_format`**
    - `"detection"` - decoding done on device, the received packet is in *`Detections`* format
    - `"raw"` - decoding done on host
  - `NN_family` - *"YOLO"* or *"mobilenet"*
  - **`NN_specific_metadata` - only for *"YOLO"***
    - `classes` - number of classes
    - `coordinates` - number of coordinates
    - `anchors` - anchors for YOLO network
    - `anchor_masks` - anchor mask for each output layer : `26x26`, :code`13x13` (+ *52x52* for full YOLO-v3)
    - `iou_threshold` - intersection over union threshold for detected object
    - `confidence_threshold` - score confidence threshold for detected object
- `mappings.labels` - used by `depthai_demo.py` script to decode labels from id's

Example decoding when `output_format` is set to `detection`:

```
nnet_packets, data_packets = p.get_available_nnet_and_data_packets()

for nnet_packet in nnet_packets:
  in_layers = nnet_packet.getInputLayersInfo()

  input_width  = in_layers[0].get_dimension(depthai.TensorInfo.Dimension.W)
  input_height = in_layers[0].get_dimension(depthai.TensorInfo.Dimension.H)

  detections = nnet_packet.getDetectedObjects()
  objects = list()

  for detection in detections:
      detection_dict = detection.get_dict()
      # scale normalized coordinates to image coordinates
      detection_dict["x_min"] = int(detection_dict["x_min"] * input_width)
      detection_dict["y_min"] = int(detection_dict["y_min"] * input_height)
      detection_dict["x_max"] = int(detection_dict["x_max"] * input_width)
      detection_dict["y_max"] = int(detection_dict["y_max"] * input_height)
      objects.append(detection_dict)

print(objects)
```

Example of decoding for full `yolo-v3` and `tiny-yolo-v3` on host and device is here

Example of decoding for `mobilenet` based networks on host and device is here

1.1.8 Other installation methods

To get the latest and yet unreleased features from our source code, you can go ahead and compile depthai package manually.

Dependencies to build from source

- CMake > 3.2.0
- Generation tool (Ninja, make, . . .)
- C/C++ compiler
- libusb1 development package

Ubuntu, Raspberry Pi OS, . . . (Debian based systems)

On Debian based systems (Raspberry Pi OS, Ubuntu, . . .) these can be acquired by running:

```
sudo apt-get -y install cmake libusb-1.0-0-dev build-essential
```

macOS (Mac OS X)

Assuming a stock Mac OS X install, depthai-python library needs following dependencies

- Homebrew (If it's not installed already)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install.sh)"
```

- Python, libusb, CMake, wget

```
brew install coreutils python3 cmake libusb wget
```

And now you're ready to clone the depthai-python from Github and build it for Mac OS X.

Install using GitHub commit

Pip allows users to install the packages from specific commits, even if they are not yet released on PyPi.

To do so, use the command below - and be sure to replace the `<commit_sha>` with the correct commit hash from here

```
python3 -m pip install git+https://github.com/luxonis/depthai-python.git@<commit_sha>
```

Using/Testing a Specific Branch/PR

From time to time, it may be of interest to use a specific branch. This may occur, for example, because we have listened to your feature request and implemented a quick implementation in a branch. Or it could be to get early access to a feature that is soaking in our `develop` for stability purposes before being merged into `main`.

So when working in the depthai repository, using a branch can be accomplished with the following commands. For this example, the `branch` that we will try out is `develop` (which is the branch we use to soak new features before merging them into `main`):

Prior to running the following, you can either clone the repository independently (for not over-writing any of your local changes) or simply do a `git pull` first.

```
git checkout develop
python3 install_requirements.py
```

Install from source

If desired, you can also install the package from the source code itself - it will allow you to make the changes to the API and see them live in action.

To do so, first download the repository and then add the package to your python interpreter in development mode

```
git clone https://github.com/luxonis/depthai-python.git
cd depthai-python
git submodule update --init --recursive
python3 setup.py develop  # you may need to add sudo if using system interpreter␣
↪instead of virtual environment
```

If you want to use other branch (e.g. `develop`) than default (`main`), you can do so by typing

```
git checkout develop  # replace the "develop" with a desired branch name
git submodule update --recursive
python3 setup.py develop
```

Or, if you want to checkout a specific commit, type

```
git checkout <commit_sha>
git submodule update --recursive
python3 setup.py develop
```

1.1.9 API Reference

class Device

Represents the DepthAI device with the methods to interact with it.

> **Warning:** Please be aware that all methods except *get_available_streams()* require *create_pipeline()* to be run first,

Example

```
import depthai
device = depthai.Device('', False)
pipeline = device.create_pipeline(config={
    'streams': ['previewout', 'metaout'],
    'ai': {
        "blob_file": "/path/to/model.blob",
        "blob_file_config": "/path/to/config.json",
    },
})
```

Methods

__init__ (*device_id: str*, *usb2_mode: bool*) → *Device*

Standard and recommended way to set up the object.

device_id represents the USB port id that the device is connected to. If set to specific value (e.x. `"1"`) it will look for the device in specific USB port, whereas if left empty - `''` - it will look for the device on all ports. It's useful when we have more than one DepthAI devices connected and want to specify which one to use in the code

usb2_mode, being `True/False`, allows the DepthAI to communicate using USB2 protocol, not USB3. This lowers the throughput of the pipeline, but allows to use >1m USB cables for connection

__init__ (*cmd_file: str*, *device_id: str*) → *Device*

Development and debug way to initialize the DepthAI device.

cmd_file is a path to firmware `.cmd` file that will be loaded onto the device for boot.

device_id represents the USB port id that the device is connected to. If set to specific value (e.x. `"1"`) it will look for the device in specific USB port, whereas if left empty - `''` - it will look for the device on all ports. It's useful when we have more than one DepthAI devices connected and want to specify which one to use in the code

create_pipeline (*config: dict*) → *depthai.CNNPipeline*

Initializes a DepthAI Pipeline, returning the created `CNNPipeline` if successful and `None` otherwise.

config(dict) - A `dict` of pipeline configuration settings. Example key/values for the config:

```
{
    # Possible streams:
    #   'color' - 4K color camera preview
    #   'left' - left mono camera preview
    #   'right' - right mono camera preview
    #   'rectified_left' - rectified left camera preview
    #   'rectified_right' - rectified right camera preview
    #   'previewout' - neural network input preview
    #   'metaout' - CNN output tensors
    #   'depth' - the raw depth map, disparity converted to real life distance
    #   'disparity' - disparity map, the disparity between left and right␣
↪cameras, in pixels
    #   'disparity_color' - disparity map colorized
    #   'meta_d2h' - device metadata stream
    #   'video' - H.264/H.265 encoded color camera frames
    #   'jpegout' - JPEG encoded color camera frames
    #   'object_tracker' - Object tracker results
    'streams': [
        'left',  # if left is used, it must be in the first position
        'right',
        {'name': 'previewout', 'max_fps': 12.0},  # streams can be specified␣
↪as objects with additional params
```

(continues on next page)

(continued from previous page)

```
        'metaout',
        # depth-related streams
        {'name': 'depth', 'max_fps': 12.0},
        {'name': 'disparity', 'max_fps': 12.0},
        {'name': 'disparity_color', 'max_fps': 12.0},
    ],
    'depth':
    {
        'calibration_file': consts.resource_paths.calib_fpath,
        'left_mesh_file': consts.resource_paths.left_mesh_fpath,
        'right_mesh_file': consts.resource_paths.right_mesh_fpath,
        'padding_factor': 0.3,
        'depth_limit_m': 10.0, # In meters, for filtering purpose during x,y,
↪z calc
        'median_kernel_size': 7,  # Disparity / depth median filter kernel␣
↪size (N x N) . 0 = filtering disabled
        'lr_check': True  # Enable stereo 'Left-Right check' feature.
        'warp_rectify':
        {
            'use_mesh' : True, # if False, will use homography
            'mirror_frame': True, # if False, the disparity will be mirrored␣
↪instead
            'edge_fill_color': 0, # gray 0..255, or -1 to replicate pixel␣
↪values
        },
    },
    'ai':
    {
        'blob_file': blob_file,
        'blob_file_config': blob_file_config,
        'blob_file2': blob_file2,
        'blob_file_config2': blob_file_config2,
        'calc_dist_to_bb': True, # depth calculation on CNN models with␣
↪bounding box output
        'keep_aspect_ratio': False, # Keep aspect ratio, don't use full RGB␣
↪FOV for NN
        'camera_input': "left", # 'rgb', 'left', 'right', 'left_right',
↪'rectified_left', 'rectified_right', 'rectified_left_right'
        'shaves' : 7,  # 1 - 14 Number of shaves used by NN.
        'cmx_slices' : 7,  # 1 - 14 Number of cmx slices used by NN.
        'NN_engines' : 2,  # 1 - 2 Number of NN_engines used by NN.
    },
    # object tracker
    'ot':
    {
        'max_tracklets'        : 20, #maximum 20 is supported
        'confidence_threshold' : 0.5, #object is tracked only for detections␣
↪over this threshold
    },
    'board_config':
    {
        'swap_left_and_right_cameras': True, # Swap the Left and Right␣
↪cameras.
        'left_fov_deg': 73.5, # Horizontal field of view (HFOV) for the␣
↪stereo cameras in [deg].
        'rgb_fov_deg': 68.7938, # Horizontal field of view (HFOV) for the RGB␣
↪camera in [deg]
```

(continues on next page)

(continued from previous page)

```
        'left_to_right_distance_cm': 9.0, # Left/Right camera baseline in [cm]
        'left_to_rgb_distance_cm': 2.0, # Distance the RGB camera is from the␣
↪Left camera.
        'store_to_eeprom': False, # Store the calibration and board_config␣
↪(fov, baselines, swap-lr) in the EEPROM onboard
        'clear_eeprom': False, # Invalidate the calib and board_config from␣
↪EEPROM
        'override_eeprom': False, # Use the calib and board_config from host,␣
↪ignoring the EEPROM data if programmed
    },
    'camera':
    {
        'rgb':
        {
            # 3840x2160, 1920x1080
            # only UHD/1080p/30 fps supported for now
            'resolution_h': 3040, # possible - 1080, 2160, 3040
            'fps': 30,
        },
        'mono':
        {
            # 1280x720, 1280x800, 640x400 (binning enabled)
            'resolution_h': 800, # possible - 400, 720, 800
            'fps': 30,
        },
    },
    'app':
    {
        'sync_video_meta_streams': False,  # Synchronize 'previewout' and
↪'metaout' streams
        'sync_sequence_numbers'  : False,  # Synchronize sequence numbers for␣
↪all packets. Experimental
        'usb_chunk_KiB' : 64, # USB transfer chunk on device. Higher (up to␣
↪megabytes) may improve throughput, or 0 to disable chunking
    },
    #'video_config':
    #{
    #    'rateCtrlMode': 'cbr', # Options: cbr / vbr
    #    'profile': 'h265_main', # Options: 'h264_baseline' / 'h264_main' /
↪'h264_high' / 'h265_main / 'mjpeg' '
    #    'bitrate': 8000000, # When using CBR (H264/H265 only)
    #    'maxBitrate': 8000000, # When using CBR (H264/H265 only)
    #    'keyframeFrequency': 30, (H264/H265 only)
    #    'numBFrames': 0, (H264/H265 only)
    #    'quality': 80 # (0 - 100%) When using VBR or MJPEG profile
    #}
    #'video_config':
    #{
    #    'profile': 'mjpeg',
    #    'quality': 95
    #}
}
```

get_available_streams() → List[str]

Return a list of all streams supported by the DepthAI library.

```
>>> device.get_available_streams()
['meta_d2h', 'color', 'left', 'right', 'rectified_left', 'rectified_right',
↪'disparity', 'depth', 'metaout', 'previewout', 'jpegout', 'video', 'object_
↪tracker']
```

get_nn_to_depth_bbox_mapping() → dict
Returns dict that allows to match the CNN output with the disparity info.

Since the RGB camera has a 4K resolution and the neural networks accept only images with specific resolution (like 300x300), the original image is cropped to meet the neural network requirements. On the other side, the disparity frames returned by the neural network are in full resolution available on the mono cameras.

To be able to determine where the CNN previewout image is on the disparity frame, this method should be used as it specifies the offsets and dimensions to use.

```
>>> device.get_nn_to_depth_bbox_mapping()
{'max_h': 681, 'max_w': 681, 'off_x': 299, 'off_y': 59}
```

request_af_mode()
Set the 4K RGB camera autofocus mode to one of the available *AutofocusMode*

request_af_trigger()
Manually send trigger action to AutoFocus on 4k RGB camera

request_jpeg()
Capture a JPEG frame from the RGB camera and send it to `jpegout` stream. The frame is in full available resolution, not cropped to meet the CNN input dimensions.

send_disparity_confidence_threshold(*confidence: int*)
Function to send disparity confidence threshold for StereoSGBM algorithm. If the disparity value confidence is below the threshold, the value is marked as invalid disparity and treated as background

send_disparity_confidence_threshold(*confidence: int*)
Function to send disparity confidence threshold for StereoSGBM algorithm. If the disparity value confidence is below the threshold, the value is marked as invalid disparity and treated as background

get_right_homography()

> **Warning:**
>
> Note: Requires *dual-homography calibration*.
>
> Return a 3x3 homography matrix used to rectify the right stereo camera image.

get_left_homography()

> **Warning:**
>
> Note: Requires *dual-homography calibration*.
>
> Return a 3x3 homography matrix used to rectify the left stereo camera image.

get_left_intrinsic()

Warning:

Note: Requires *dual-homography calibration*.

Return a 3x3 intrinsic calibration matrix of the left stereo camera.

get_right_intrinsic()

Warning:

Note: Requires *dual-homography calibration*.

Return a 3x3 intrinsic calibration matrix of the right stereo camera.

get_rotation()

Warning:

Note: Requires *dual-homography calibration*.

Return a 3x3 rotation matrix representing the rotation of the right stereo camera w.r.t left stereo camera.

get_translation()

Warning:

Note: Requires *dual-homography calibration*.

Return a 3x1 vector representing the position of the right stereo camera center w.r.t left stereo camera center.

class AutofocusMode

An enum with all autofocus modes available

Members

AF_MODE_AUTO

This mode sets the Autofocus to a manual mode, where you need to call *Device.request_af_trigger()* to start focusing procedure.

AF_MODE_CONTINUOUS_PICTURE

This mode adjusts the focus continually to provide the best in-focus image stream and should be used when the camera is standing still while capturing. Focusing procedure is done as fast as possible.

This is the default mode the DepthAI operates in.

AF_MODE_CONTINUOUS_VIDEO

This mode adjusts the focus continually to provide the best in-focus image stream and should be used when the camera is trying to capture a smooth video steam. Focusing procedure is slower and avoids focus overshoots

AF_MODE_EDOF

This mode disables the autofocus. EDOF stands for Enhanced Depth of Field and is a digital focus.

AF_MODE_MACRO
It's the same operating mode as *AF_MODE_AUTO*

class CNNPipeline
Pipeline object using which the device is able to send it's result to the host.

Methods

get_available_data_packets() → List[*depthai.DataPacket*]
Returns only data packets produced by the device itself, without CNN results

get_available_nnet_and_data_packets() → tuple[List[*depthai.NNetPacket*], List[*depthai.DataPacket*]]
Return both neural network results and data produced by device

class NNetPacket
For any neural network inference output `NNPacket.get_tensor()` can be used. For the specific case of `Mobilenet-SSD`, `YOLO-v3` decoding can be done in the firmware. Decoded objects can be accessed through *getDetectedObjects()* as well in addition to raw output to make the results of this commonly used networks easily accessible. See *blob config file* for more details about different neural network output formats and how to choose between these formats.

Neural network results packet. It's not a single result, but a batch of results with additional metadata attached

Methods

getMetadata() → *depthai.FrameMetadata*
Returns metadata object containing all proprietary data related to this packet

get_tensor(*name: Union[int, str]*) → numpy.ndarray

> **Warning:** Works only, when in *blob config file* `output_format` is set to `raw`.

Returns a shaped numpy array for the specific network output tensor, based on the neural network's output layer information.

For example: in case of `Mobilenet-SSD` it returns a `[1, 1, 100, 7]` shaped array, where `numpy.dtype` is `float16`.

Example of usage:

```
nnetpacket.get_tensor(0)
# or
nnetpacket.get_tensor('detection_out')
```

__getitem__(*name: Union[int, str]*) → numpy.ndarray
Same as *get_tensor()*

Example of usage for `Mobilenet-SSD`:

```
nnetpacket[0]
# or
nnetpacket['detection_out']
```

getOutputsList() → list
Returns all the output tensors in a list for the network.

getOutputsDict() → dict
Returns all the output tensors in a dictionary for the network. The key is the name of the output layer, the value is the shaped numpy array.

getOutputLayersInfo() → *depthai.TensorInfo*
Returns information about the output layer for the network.

getInputLayersInfo() → *depthai.TensorInfo*
Returns information about the input layers for the network.

getDetectedObjects() → *depthai.Detections*

> **Warning:** Works when in *blob config file* `output_format` is set to `detection` and with detection networks (`Mobilenet-SSD`, `(tiny-)YOLO-v3` based networks)

Returns the detected objects in *`Detections`* format. The network is decoded on device side.

class TensorInfo
Descriptor of the input/output layers/tensors of the network.

When network is loaded the tensor info is automatically printed.

Attributes

name: *str*
Name of the tensor.

dimensions: *list*
Shape of tensor array. E.g. : `[1, 1, 100, 7]`

strides: *list*
Strides of tensor array.

data_type: *string*
Data type of tensor. E.g. : `float16`

offset: *int*
Offset in the raw output array.

element_size: *int*
Size in bytes of one element in the array.

index: *int*
Index of the tensor. E.g. : in case of multiple inputs/outputs in the network it marks the order of input/output.

Methods

get_dict() → dict
Returns TensorInfo in a dictionary where the *key* is the name of attribute.

get_dimension() → int
Returns the specific dimension of the tensor

```
tensor_info.get_dimension(depthai.TensorInfo.Dimension.WIDTH)   # returns␣
↪width of tensor
```

class Detections
Container of neural network results decoded on device side.

Example of accessing detections

Assuming the detected objects are stored in `detections` object.

- Number of detections

```
detections.size()
# or
len(detections)
```

- Accessing the nth detection

```
detections[0]
detections[1]  # ...
```

- Iterating through all detections

```
for detection in detections:
```

class Detection

Detected object descriptor.

Attributes

label: *int*
Label id of the detected object.

confidence: *float*
Confidence score of the detected object in interval [0, 1].

x_min: *float*
Top left `X` coordinate of the detected bounding box. Normalized, in interval [0, 1].

y_min: *float*
Top left `Y` coordinate of the detected bounding box. Normalized, in interval [0, 1].

x_max: *float*
Bottom right `X` coordinate of the detected bounding box. Normalized, in interval [0, 1].

y_max: *float*
Bottom right `Y` coordinate of the detected bounding box. Normalized, in interval [0, 1].

depth_x: *float*
Distance to detected bounding box on `X` axis. Only when depth calculation is enabled (stereo cameras are present on board).

depth_y: *float*
Distance to detected bounding box on `Y` axis. Only when depth calculation is enabled (stereo cameras are present on board).

depth_z: *float*
Distance to detected bounding box on `Z` axis. Only when depth calculation is enabled (stereo cameras are present on board).

Methods

get_dict() → dict
Returns detected object in a dictionary where the `key` is the name of attribute.

class Dimension

Dimension descriptor of tensor shape. Mostly meaningful for input tensors since not all neural network models respect the semantics of `Dimension` for output tensor

Values

W / WIDTH
Width

H / HEIGHT
Height

C / CHANNEL
Number of channels

N / NUMBER
Number of inferences

B / BATCH
Batch of inferences

class DataPacket
DepthAI data packet, containing information generated on the device. Unlike NNetPacket, it contains a single "result" with source stream info

Attributes

stream_name: str
Returns packet source stream. Used to determine the origin of the packet and therefore allows to handle the packets correctly, applying proper handling based on this value

Methods

getData() → numpy.ndarray
Returns the data as NumPy array, which you can be further transformed or displayed using OpenCV `imshow`.

Used with streams that returns frames e.x. `previewout`, `left`, `right`, or encoded data e.x. `video`, `jpegout`.

getDataAsStr() → str
Returns the data as a string, capable to be parsed further.

Used with streams that returns non-array results e.x. `meta_d2h` which returns JSON object

getMetadata() → *FrameMetadata*
Returns metadata object containing all proprietary data related to this packet

getObjectTracker() → *ObjectTracker*

> **Warning:** Works only with packets from `object_tracker` stream

Returns metadata object containing *ObjectTracker* object

size() → int
Returns packet data size

class FrameMetadata
Metadata object attached to the packets sent via pipeline.

Methods

getCameraName() → str
Returns the name of the camera that produced the frame.

getCategory() → int
Returns the type of the packet, whether it's a regular frame or arrived from taking a still

getFrameBytesPP() → int
Returns number of bytes per pixel in the packet's frame

getFrameHeight() → int
Returns the height of the packet's frame

getFrameWidth() → int
Returns the width of the packet's frame

getFrameType() → int
Returns the type of the data that this packet contains.

getInstanceNum() → int
Returns the camera id that is the source of the current packet

getSequenceNum() → int
Sequence number is assigned for each frame produced by the camera. It can be used to assure the frames are captured at the same time - e.x. if frames from left and right camera have the same sequence number, you can assume they were taken at the same time

getStride() → int
Specifies number of bytes till the next row of pixels in the packet's frame

getTimestamp() → float
When packet is created, it is assigned a creation timestamp, which can be obtained using this method

class ObjectTracker
Object representing current state of the tracker, obtained by calling *DataPacket.getObjectTracker()* method on a packet from `object_tracker` stream

Methods

getNrTracklets() → int
Return the number of available tracklets

getTracklet(*tracklet_nr: int*) → *Tracklet*
Returns the tracklet with specified `tracklet_nr`. To check how many tracklets there are, please use *getNrTracklets()* method

class Tracklet
Tracklet is representing a single tracked object, is produced by *ObjectTracker* class. To obtain it, call *ObjectTracker.getTracklet()* method.

Methods

getId() → int
Return the tracklet id

getLabel() → int
Return the tracklet label, being the neural network returned result. Used to identify a class of recognized objects

getStatus() → str
Return the tracklet status - either `NEW`, `TRACKED`, or `LOST`.

getLeftCoord() → int
Return the left coordinate of the bounding box of a tracked object

getRightCoord() → int
Return the right coordinate of the bounding box of a tracked object

getTopCoord() → int
Return the top coordinate of the bounding box of a tracked object

getBottomCoord() → int
Return the bottom coordinate of the bounding box of a tracked object

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

1.2 FAQs & How-To

1.2.1 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 an Embedded Spatial AI problem.

So we built the platform.

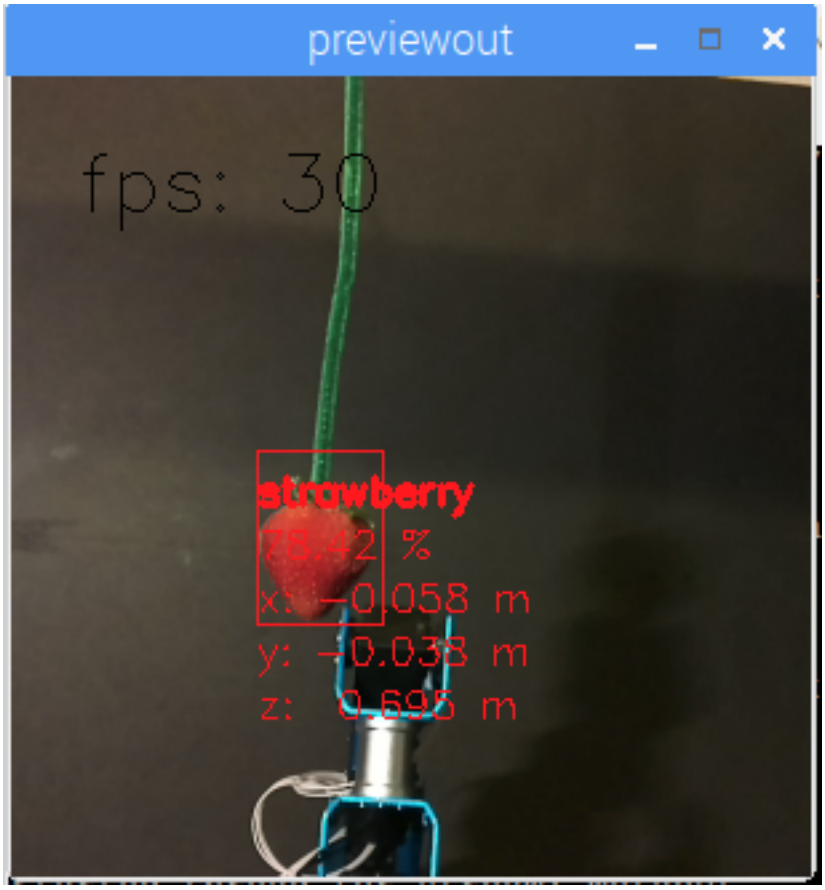
1.2.2 What is DepthAI?

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.



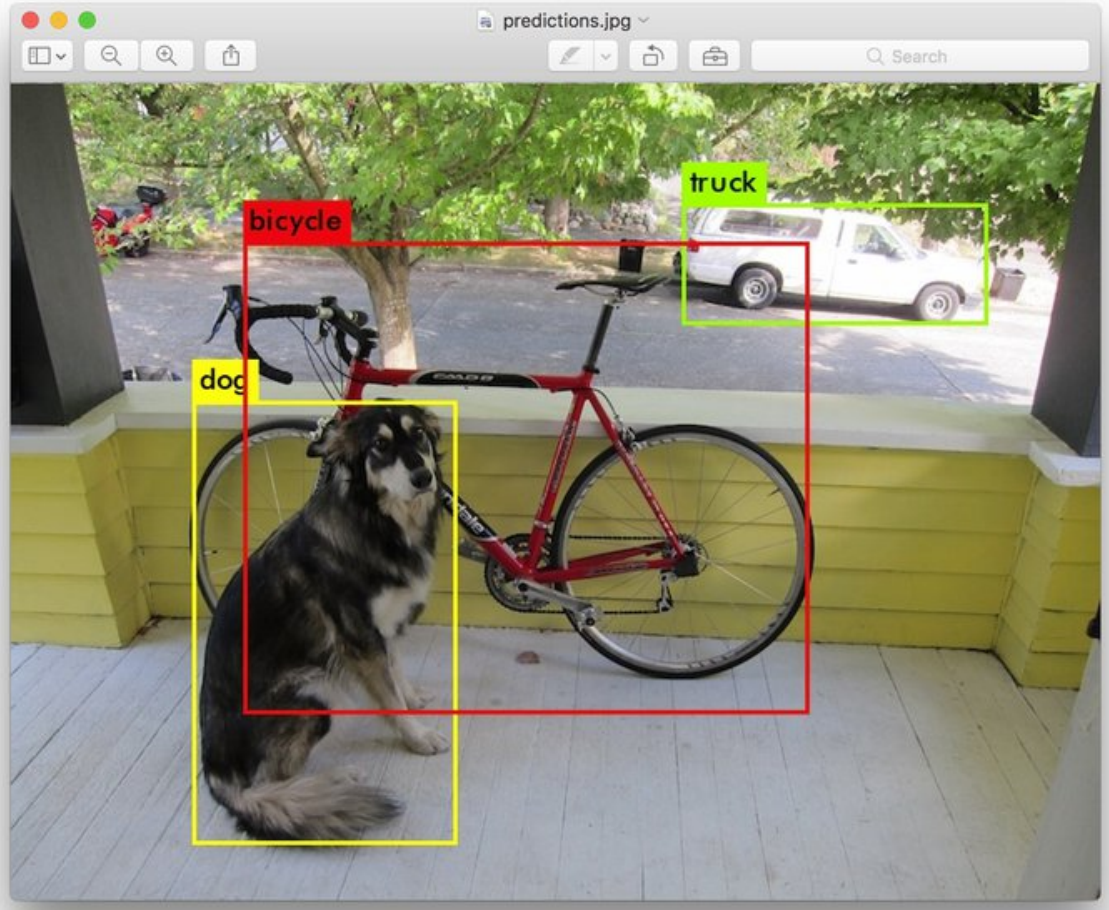
It was trained to do so over the course of a weekend, by a student (for a student project), using our free online training resources.

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) Githubs *below*, which include open-source hardware, firmware, software, and machine-learning training.

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

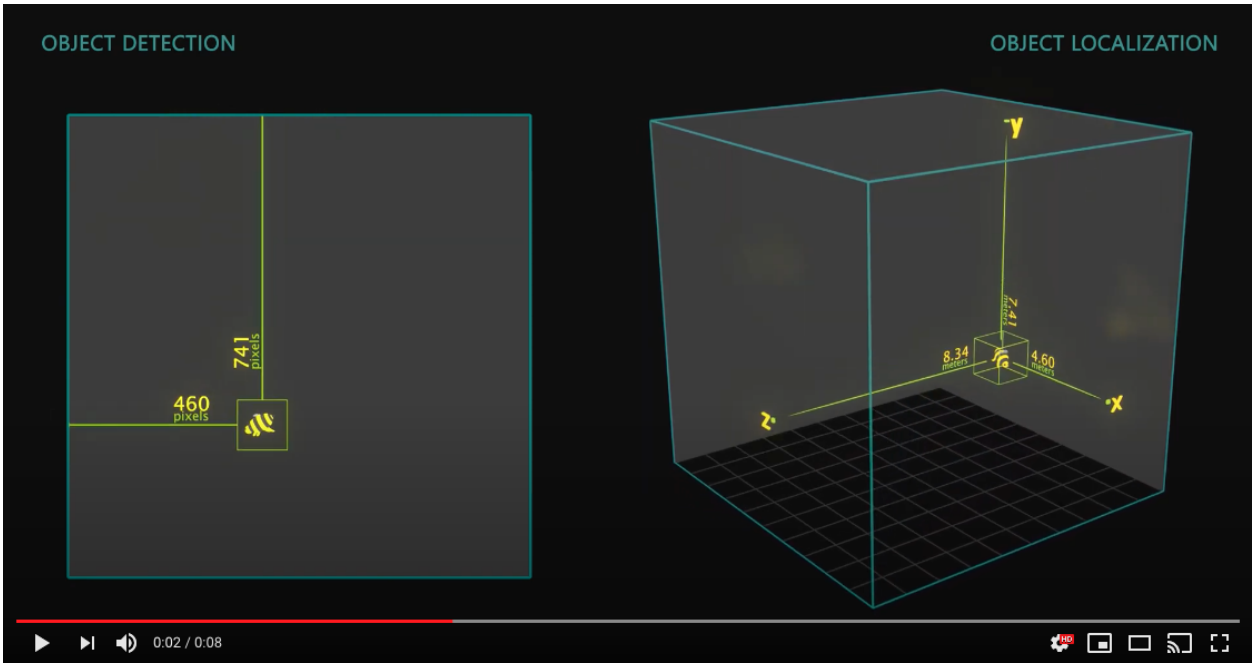
First, it is necessary to define what 'Object Detection' is:



It is the technical term for finding the bounding box of an object of interest, in pixel space (i.e. pixel coordinates), in an image.

3D Object Localization (or 3D Object Detection), is all about finding such objects in physical space, instead of pixel space. This is useful when trying to real-time measure or interact with the physical world.

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



Spatial AI is then the super-set of such 2D-equivalent neural networks being extended with spatial information to give them 3D context. So in other words, it's not limited to object detectors being extended to 3D object localizers. Other network types can be extended as well, including any network which returns results in pixel space.

An example of such an extension is using a facial landmark detector on DepthAI. With a normal camera this network returns the 2D coordinates of all 45 facial landmarks (contours of eyes, ears, mouth, eyebrows, etc.) Using this same network with DepthAI, each of these 45 facial landmarks is now a 3D point in physical space instead of 2D points in pixel space.

1.2.4 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*.

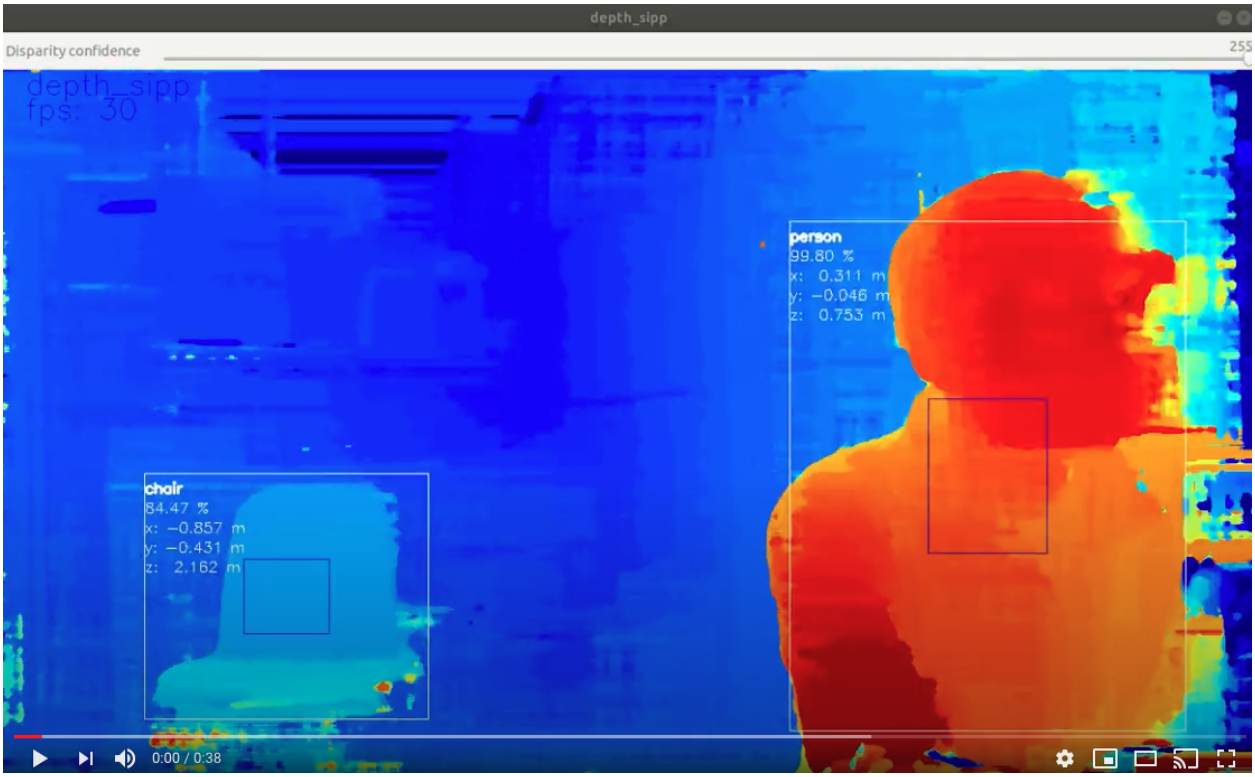
Monocular Neural Inference fused with Stereo Depth

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 -s metaout depth -bb` 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.

Stereo Neural Inference

In this mode DepthAI runs the neural network in parallel on both the left and right stereo cameras. The disparity of the results are then triangulated with the calibrated camera intrinsics (programmed into the EEPROM of each DepthAI unit) to give 3D position of all the detected features.

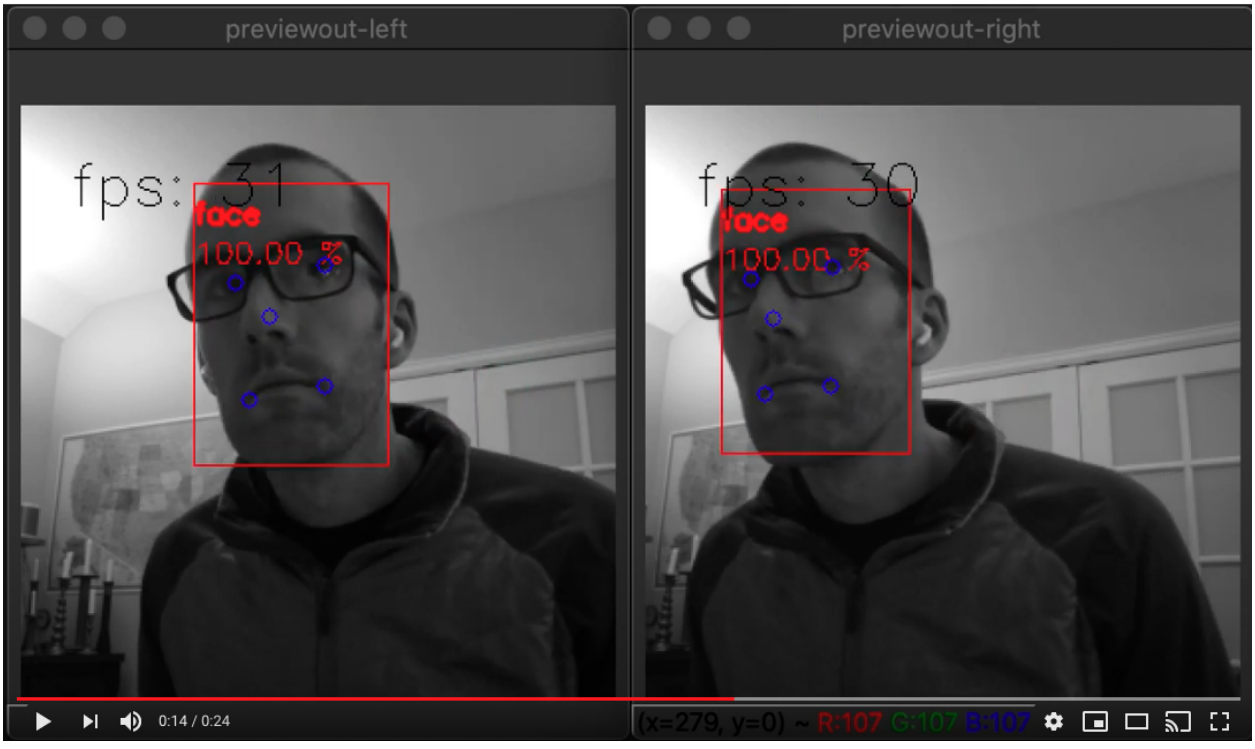
This **stereo neural inference** mode affords accurate 3D Spatial AI for networks which produce single-pixel locations of features such as facial landmark estimation, pose estimation, or other meta-data which provides feature locations like this.

Examples include finding the 3D locations of:

- Facial landmarks (eyes, ears, nose, edges of 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)

Again, 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.

An example of stereo neural inference is below.



And this is actually an interesting case as it demonstrates two things on DepthAI:

1. Stereo inference (i.e. running the neural network(s) running on both the left and right cameras in parallel)
2. Multi-stage inference (i.e. face detection flowed directly into facial landmark directly on DepthAI)

The command used to run this on DepthAI is

```
python3 depthai_demo.py -cnn face-detection-retail-0004 -cnn2 landmarks-regression-
↪retail-0009 -cam left_right -dd -sh 12 -cmx 12 -nce 2 -monor 400 -monof 30
```

Where `cam` specifies to run the neural network on both cameras, `-cnn` specifies the first-stage network to run (face detection, in this case), `-cnn2` specifies the second-stage network (facial landmark detection, in this case), and `-dd` disables running disparity depth calculations (since they are unused in this mode).

Notes

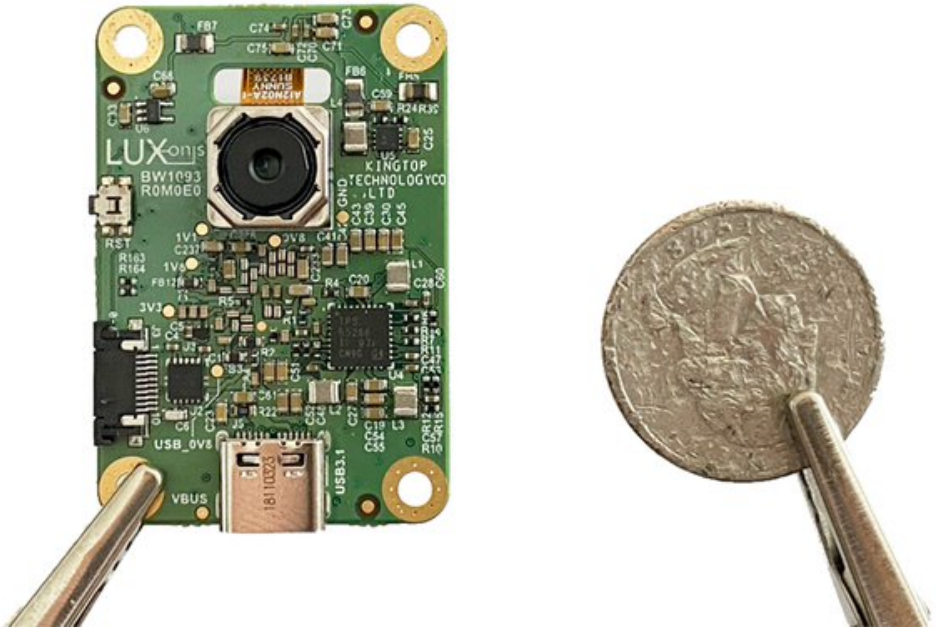
It is worth noting that monocular neural inference fused with stereo depth is possible for networks like facial-landmark detectors, pose estimators, etc. that return single-pixel locations (instead of for example bounding boxes of semantically-labeled pixels), but stereo neural inference is advised for 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.

1.2.5 What is megaAI?

The monocular (single-camera) version of DepthAI is megaAI. 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 Githubs), it is simply the tiny single-camera variant.

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

1.2.6 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. Below is a quick/dirty summary for the ~10,000-foot view of the options:

- **USB3C with Onboard Cameras** (BW1098OBC) - 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. This is the basis for OAK-D.
- **USB3C with Modular Cameras** (BW1098FFC) - 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.
- **MegaAI Single Camera** (BW1093) - This is just like the BW1098OBC, but for those who don't need depth information. Single, small, plug-and-play USB3C AI/CV camera.
- **Raspberry Pi Compute Module Edition** (BW1097) - this one has a built-in Raspberry Pi Compute Module 3B+. So you literally plug it into power and HDMI, and it boots up showing off the power of DepthAI.
- **Embedded DepthAI with WiFi/BT** (BW1092) - Currently this is in Alpha testing. So only buy it if you are comfortable with working with bleeding-edge tech and want to help us refine this product. It is the first Embedded (i.e. SPI-interface) version of DepthAI - so it has additional 128MB NOR flash, so it can boot on its own out of the NOR flash, and not host needs to be present to run. In contrast, the BW1097 can also run on its own, but it is still booting over USB from the Raspberry Pi. This BW1092, the Myriad X can run completely standalone and with no other devices. The built-in ESP32 then provides easy/convenient WiFi/BT support as well as popular integrations like plug-and-play AWS-IoT support, great iOS/Android BT examples, etc.

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:

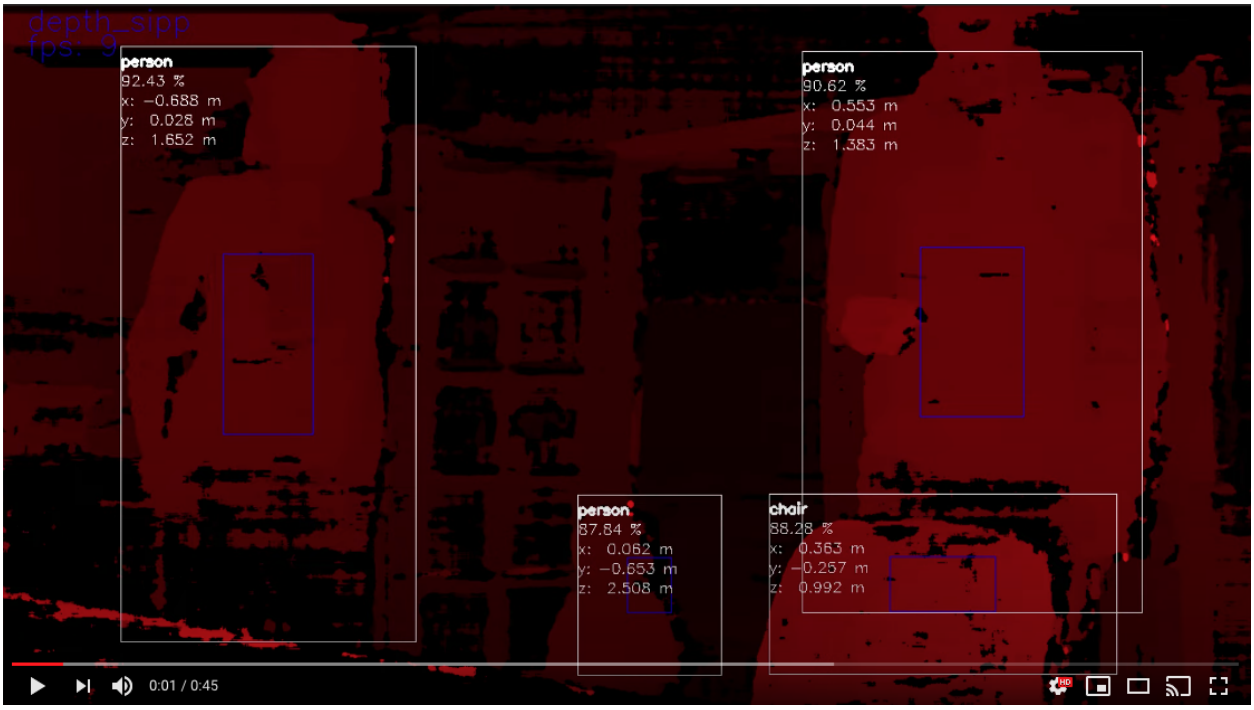
1. BW1099 - USB-boot system on module. For making devices which interface over USB to a host processor running Linux, MacOS, or Windows. In this case, the host processor stores everything, and the BW1099 boots up over USB from the host.
2. BW1099EMB - NOR-flash boot (also capable of USB-boot). For making devices that run standalone, or work with embedded MCUs like ESP32, AVR, STM32F4, etc. Can also USB-boot if/as desirable.
3. BW2099 - 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).

1.2.7 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 Install on various platforms are *here*.

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

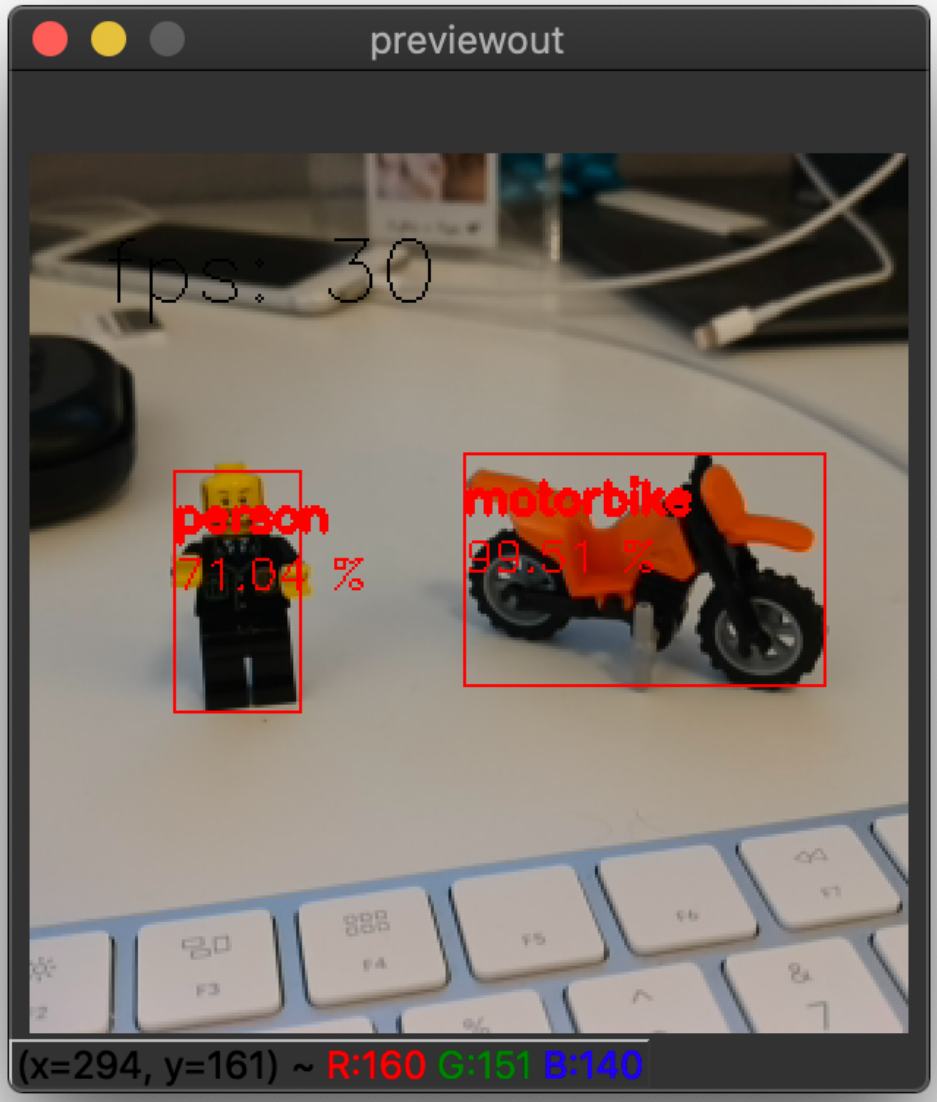


(Click on the image above to pull up the YouTube video.)

The command to get the above output is

```
python3 depthai_demo.py -s metaout previewout depth -ff -bb
```

Here is a single-camera version (megaAI) running with `python3 depthai_demo.py -dd` (to disable showing depth info):


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

1.2.8 Is DepthAI and MegaAI easy to use with Raspberry Pi?

Very. It's designed for ease of setup and use, and to keep the Pi CPU not-busy.

See *here* to get up and running quickly!

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

Yes, every model can be used, including:

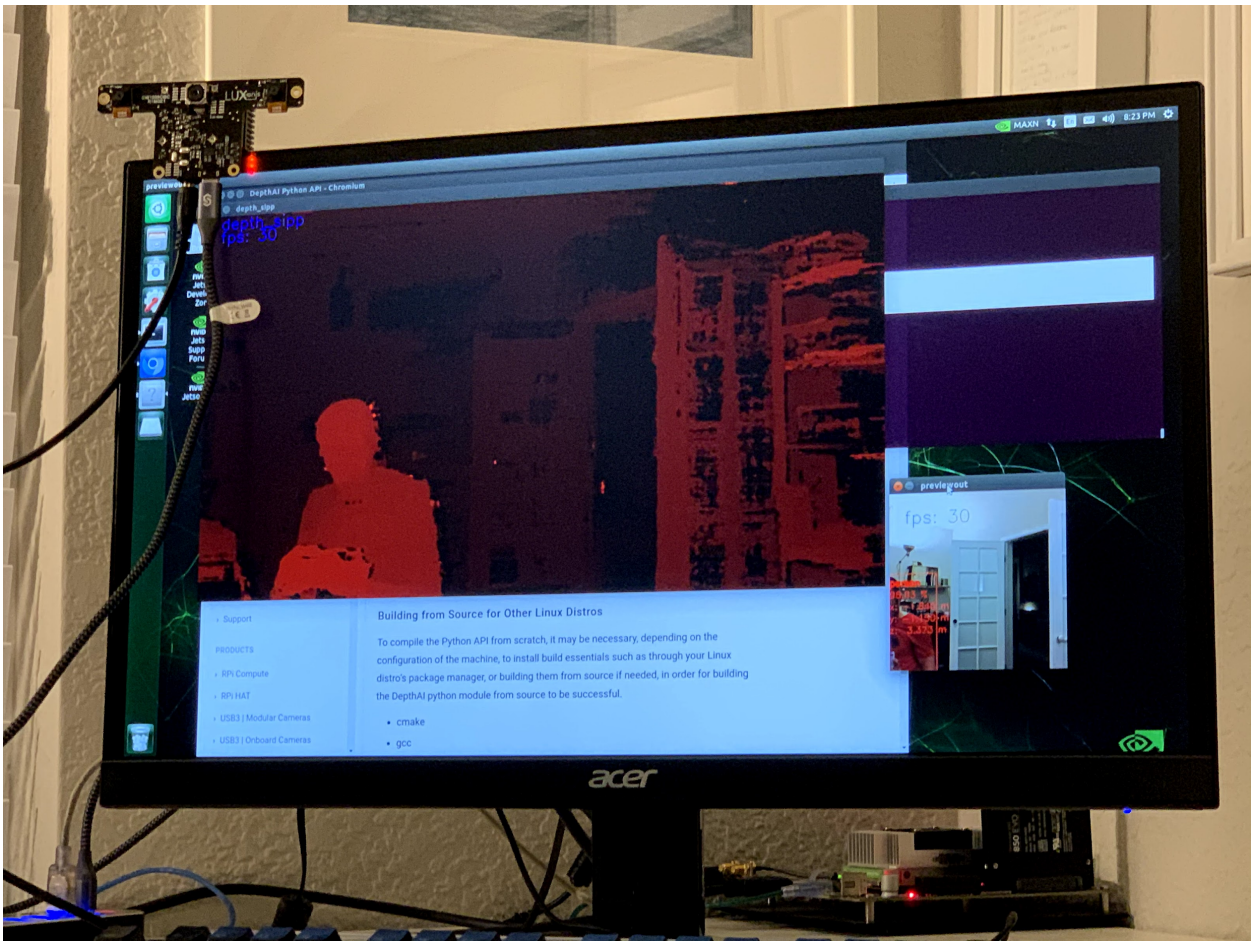
- Raspberry Pi Compute Module Edition (BW1097 - this one has a built-in Raspberry Pi Compute Module 3B+
- Raspberry Pi HAT (BW1094) - this can also be used with other hosts as its interface is USB3
- USB3C with Onboard Cameras BW1098OBC
- USB3C with Modular Cameras BW1098FFC
- MegaAI Single Camera BW1093

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

1.2.10 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:



1.2.11 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.

1.2.12 Is DepthAI OpenVINO Compatible?

Yes. As of this writing, DepthAI is fully compatible with OpenVINO 2020.1. We are in the process of upgrading to have compatibility with newer OpenVINO versions.

1.2.13 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

1.2.14 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).

1.2.15 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.

1.2.16 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.

1.2.17 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.

1.2.18 How do I Integrate DepthAI into Our Product?

How to integrate DepthAI/megaAI 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. the most mature, using our *Python API*
2. initially released by actively in development (see here),
3. supported in December 2020 (as part of Pipeline Builder Gen2 here).

In all cases, DepthAI (and megaAI) are 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/megaAI 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 BW1099.
- 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 boot off of internal flash on the BW1099EMB 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 (BW1092) available on our store. We will also be open-sourcing this design after it is fully verified (contact us if you would like the design files before we open source it).

The code-base/API for this is in active development, and a pre-release/Alpha version is available here as of this writing.

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

This will be supported through running microPython directly on the BW1099EMB as nodes in the Gen2 Pipeline Builder.

The microPython nodes are what will allow custom logic, driving I2C, SPI, GPIO, UART, etc. controls, allowing 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.

The target date for this mode is December 2020.

Hardware for Each Case:

- BW1099: USB boot. So it is intended for working with a host processor running Linux, Mac, or Windows and this host processor boots the BW1099 over USB
- BW1099EMB: USB boot or NOR-flash boot. This module can work with a host computer just like the BW1099, 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 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 BW1092 (which comes with the BW1099EMB pre-installed).

Getting Started with Development

Whether intending to use DepthAI with an *OS-capable host*, a *microcontroller over SPI* (in development), or *completely standalone* (targeted support December 2020) - 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 for now it will run only off of the image sensors (once the Gen2 pipeline builder is out, which is scheduled for December 2020, there will exist the capability to run everything off of host images/video with the DepthAI API). And this work is usually not heavy lifting… for example we had never run semantic segmentation networks before via the DepthAI API (and therefore had no reference code for doing so), but despite this one of our users actually got it working in a day without our help (e.g here).

For common object detector formats (MobileNet-SSD, tinyYOLOv1/2/3, etc.) there's effectively no work to go from NCS2 mode to DepthAI mode. You can just literally replace the classes in example MobileNet-SSD or tinyYOLO examples we have. For example for tinyYOLOv3, you can just change the labels from "mask", "no mask" and "no mask 2" to whatever your classes are from this example here and just change the blob file here to your blob file. And the same thing is true for MobileNet-SSD here.

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

Available in DepthAI API Today:

- Neural Inference (e.g. object detection, image classification, etc., including two-stage, e.g. 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
- MJPEG Encoding
- Warp/Dewarp
- Enhanced Disparity Depth Modes (Sub-Pixel, LR-Check, and Extended Disparity), here
- SPI Support, here
- Arbitrary crop/rescale/reformat and ROI return (here)

The above features are available in the Luxonis Pipeline Builder Gen1 (see example *here*). See *Pipeline Builder Gen2* for in-progress additional functionality/flexibility which will come with the next generation Luxonis pipeline builder for DepthAI.

On our Roadmap (planned delivery December 2020)

- Pipeline Builder Gen2 (arbitrary series/parallel combination of neural nets and CV functions, details here)
- Improved Stereo Neural Inference Support (here)
- microPython Support, here
- Feature Tracking (including IMU-assisted feature tracking, here)
- Integrated IMU Support (here)
- Motion Estimation (here)
- Background Subtraction (here)
- Lossless zoom (from 12MP full to 4K, 1080p, or 720p, here)
- Edge Detection (here)
- Harris Filtering (here)
- AprilTags (PR here)
- Integrated Text Detection (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

We have been working on a 2nd-generation pipeline builder which will incorporate many of the features below on our roadmap into a graphical drag/drop AI/CV pipeline which will then run entirely on DepthAI and return results of interest to the host.

This 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 without any load on the host.

1.2.20 Are CAD Files Available?

Yes.

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

1.2.21 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 192 in extended disparity modes (see *Extended Disparity* below).

To calculate the minimum distance in this mode, use the following formula, where base_line_dist and min_distance are in meters [m]: .. code-block:: python

    min_distance = focal_length * base_line_dist / 96

Where 96 is the standard maximum disparity search used by DepthAI and so for extended disparity (192 pixels), the minimum distance is:

```
min_distance = focal_length * base_line_dist / 192
```

For DepthAI, the HFOV of the the grayscale global shutter cameras is 73.5 degrees (this can be found on your board, see here, so the focal length is

```
focal_length = 1280/(2*tan(73.5/2/180*pi)) = 857.06
```

Calculation here (and for disparity depth data, the value is stored in `uint16`, where the max value of `uint16` of 65535 is a special value, meaning that that distance is unknown.)

Stereo Neural Inference

In this mode, the neural inference (object detection, landmark detection, etc.) 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.

The hyperfocal distance of the global shutter synchronized stereo pair is 19.6cm. So objects closer than 19.6cm will appear out of focus. This is effectively the minimum distance for this mode of operation, as in most cases (except for very wide stereo baselines with the *BW1098FFC*), this **effective** minimum distance is higher than the **actual** minimum distance as a result of the stereo camera field of views. For example, the objects will be fully out of the field of view of both grayscale cameras when less than 5.25cm from the BW1098OBC), but that is closer than the hyperfocal distance of the grayscale cameras (which is 19.6cm), so the actual minimum depth is this hyperfocal distance.

Accordingly, to calculate the minimum distance for this mode of operation, use the following formula:

```
min_distance = max(tan((90-HFOV/2)*pi/2)*base_line_dist/2, 19.6)
```

This formula implements the maximum of the HFOV-imposed minimum distance, and 19.6cm, which is the hyperfocal-distance-imposed minimum distance.

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:

- DepthAI RPi Compute Module Edition (*BW1097*) the minimum depth is **0.827** meters for full 1280x800 stereo resolution and **0.414** meters for 640x400 stereo resolution:

```
min_distance = 857.06.15 * 0.09 / 96 = 0.803 # m
```

calculation here

- OAK-D and USB3C Onboard Camera Edition (BW1098OBC) is **0.689** meters:

```
min_distance = 857.06*.075/96 = 0.669 # m
```

calculation here

Stereo Neural Inference Mode

For DepthAI units with onboard cameras, all models (*BW1097* and BW1098OBC) are limited by the hyperfocal distance of the stereo cameras, so their minimum depth is **0.196** meters.

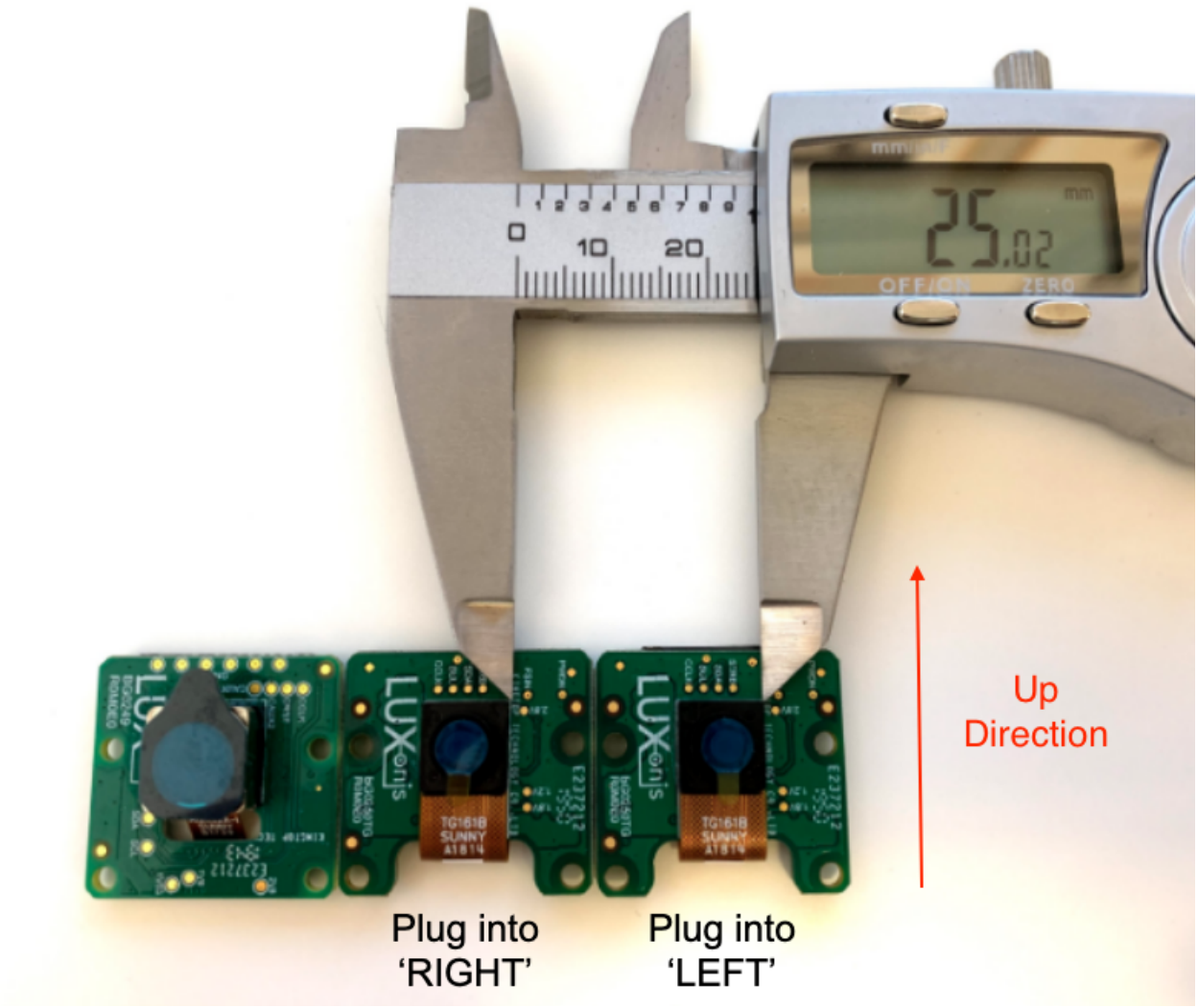
Modular Camera Minimum Depths:

Below are the minimum depth perception possible in the disparity 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:



Stereo Neural Inference Mode

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

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

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

Extended Disparity Depth Mode

If it is of interest in your application, we can implement a system called `extended disparity` which affords a closer minimum distance for the given baseline. This increases the maximum disparity search from 96 to 192. So this cuts the minimum perceivable distance in half (given that the minimum distance is now `focal_length * base_line_dist / 192` instead of `focal_length * base_line_dist / 96`).

- DepthAI RPi Compute Module Edition (BW1097): **0.414** meters
- OAK-D and USB3C Onboard Camera Edition (BW1098OBC) is **0.345** meters
- Modular Cameras at Minimum Spacing (e.g. BW1098FFC) is **0.115** meters

So if you have the need for this shorter minimum distance when using monocular neural inference fused with disparity depth, reach out to us on discord, email, or discuss.luxonis.com to let us know. It's on our roadmap but we haven't yet seen a need for it, so we haven't prioritized implementing it (yet!).

1.2.22 What Are The Maximum Depths Visible by DepthAI?

The maximum depth perception for 3D object detection is is practically limited by how far the object detector (or other neural network) can detect what it's looking for. We've found that OpenVINO people detectors work to about 22 meters or so. But generally this distance will be limited by how far away the object detector can detect objects, and then after that, the minimum angle difference between the objects.

So if the object detector is not the limit, the maximum distance will be limited by the physics of the baseline and the number of pixels. So once an object is less than 0.056 degrees (which corresponds to 1 pixel difference) difference between one camera to the other, it is past the point where full-pixel disparity can be done. The formula used to calculate this distance is an approximation, but is as follows:

```
Dm = (baseline/2) * tan_d((90 - HFOV / HPixels)*pi/2)
```

For DepthAI HFOV = 73.5(+/-0.5) degrees, and HPixels = 1280. And for the BW1098OBC, the baseline is 7.5cm.

So using this formula for existing models the *theoretical* max distance is:

- BW1098OBC (OAK-D; 7.5cm baseline): 38.4 meters
- BW1097 (9cm baseline): 46 meters
- Custom baseline: Dm = (baseline/2) * tan_d(90 - 73.5 / 1280)

But these theoretical maximums are not achievable in the real-world, as the disparity matching is not perfect, nor are the optics, image sensor, etc., so the actual maximum depth will be application-specific depending on lighting, neural model, feature sizes, baselines, etc.

After the KickStarter campaign we will also be supporting sub-pixel, which will extend this theoretical max, but again this will likely not be the -actual- limit of the max object detection distance, but rather the neural network itself will be. And this subpixel use will likely have application-specific benefits.

1.2.23 What Is the Format of the Depth Data in depth stream?

The output array is in uint16, so 0 to 65,535 with direct mapping to millimeters (mm).

So a value of 3,141 in the array is 3,141 mm, or 3.141 meters. So this whole array is the z-dimension of each pixel off of the camera plane, where the `center of the universe` is the camera marked `RIGHT`.

And the specific value of 65,535 is a special value, meaning an invalid disparity/depth result.

1.2.24 How Do I Calculate Depth from Disparity?

DepthAI does convert to depth onboard for both the `depth` stream and also for object detectors like MobileNet-SSD, YOLO, etc.

But we also allow the actual disparity results to be retrieved so that if you would like to use the disparity map directly, you can.

To calculate the depth map from the disparity map, it is (approximately) `baseline * focal / disparity`. Where the baseline is 7.5cm for BW1098OBC, 4.0cm for BW1092, and 9.0cm for BW1097, and the focal length is `883.15` (`focal_length = 1280/(2*tan(73.5/2/180*pi)) = 857.06`) for all current DepthAI models.

So for example, for a BW1092 (stereo baseline of 4.0cm), a disparity measurement of 60 is a depth of 58.8cm (`depth = 40 * 857.06 / 60 = 571 mm (0.571m)`).

1.2.25 How Do I Display Multiple Streams?

To specify which streams you would like displayed, use the `-s` option. For example for metadata (e.g. bounding box results from an object detector), the color stream (`previewout`), and for depth results (`depth`), use the following command:

```
python3 depthai_demo.py -s metaout previewout depth
```

The available streams are:

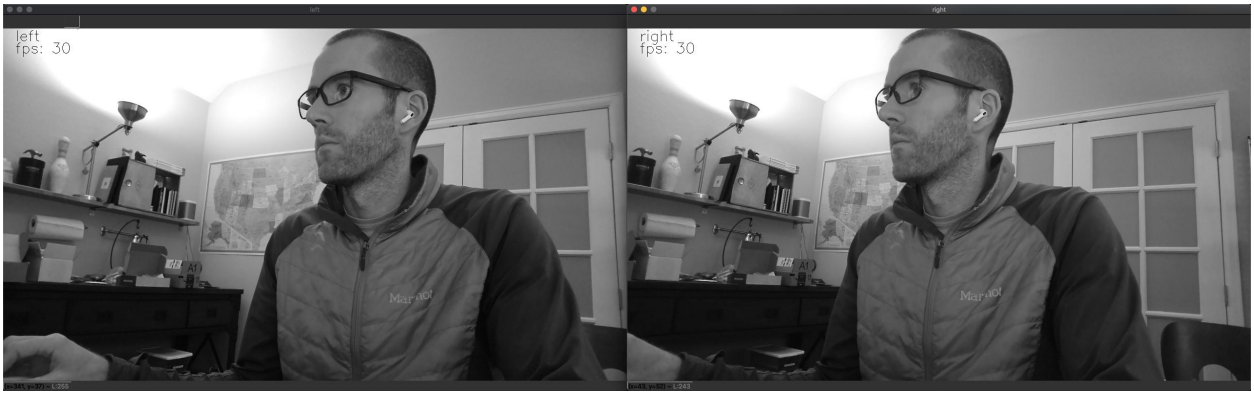
- `metaout` - Meta data results from the neural network
- `previewout` - Small preview stream from the color camera
- `color` - 4K color camera, biggest camera on the board with lens
- `left` - Left grayscale camera (marked *L* or *LEFT* on the board)
- `right` - Right grayscale camera (marked *R* or *RIGHT* on the board)
- `rectified_left` - Rectified left camera frames
- `rectified_right` - Rectified right camera frames
- `depth` - Depth in *uint16* (see here for the format.
- `disparity` - Raw disparity
- `disparity_color` - Disparity colorized on the host (`JET` colorized visualization of depth)
- `meta_d2h` - Device die temperature (max temp should be < 105C)
- `object_tracker` - Object tracker results

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

Yes, to get the raw stereo pair stream on the host use the following command:

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

This will show the full RAW (uncompressed) 1280x720 stereo synchronized pair, as below:



1.2.26 How Do I Limit The FrameRate Per Stream?

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 framerate of streams to not overwhelm the host/USB2 with too much data.

So to set streams to a specific framerate to reduce the USB2 load and host load, simply specify the stream with `-s streamname` with a comma and FPS after the stream name like `-s streamname,FPS`.

So for limiting *depth* to 5 FPS, use the following command:

```
python3 depthai_demo.py -s depth,5
```

And this works equally for multiple streams:

```
python3 depthai_demo.py -s left,2 right,2 previewout depth,5
```

It's worth noting that the framerate 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.

One can also use the following override command structure, which allows you to set the framerate per stream.

The following example sets the `depth` stream to 8 FPS and the `previewout` to 12 FPS:

```
python3 depthai_demo.py -co '{"streams": [{"name": "depth", "max_fps": 8.0},{"name":
↪"previewout", "max_fps": 12.0}]}'
```

You can pick/choose whatever streams you want, and their frame rate, but pasting in additional `{"name": "streamname", "max_fps": FPS}` into the expression above.

1.2.27 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 framerate 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.

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 framerate to match, or be just below, the framerate 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:

```
./depthai_demo.py -rgbf 24 -monof 24 -sync
```

Synchronizing on the host

The two methods *`FrameMetadata.getTimestamp()`* and *`FrameMetadata.getSequenceNum()`* can be used to guarantee the synchronization on host side.

The NNPackets and DataPackets are being sent separately from device side, and get into individual queues per stream on host side. The function *`CNNPipeline.get_available_nnet_and_data_packets()`* returns what's available in the queues at the moment the function is called (it could be that just one NN packet is unread, or just one frame packet).

With the `-sync` CLI option from depthai.py, we are doing a best effort on the device side (i.e. on the Myriad X) to synchronize NN and previewout, and send them in order: first the NN packet is being sent (and in depthai.py it gets saved as the latest), then the previewout frame is being sent (and when received in depthai_demo.py, the latest saved NN data is overlaid on).

In most cases this works well, but there is a risk (especially under high system load on host side), that the packets may still get desynchronized, as the queues are handled by different threads (in the C++ library).

So in that case, `getMetadata().getTimestamp()` returns the device time (in seconds, as float) and is also used in the stereo calibration script to synchronize the Left and Right frames.

The timestamp corresponds to the moment the frames are captured from the camera, and is forwarded through the pipeline. And the method `getMetadata().getSequenceNum()` returns an incrementing number per camera frame. The same number is associated to the NN packet, so it could be an easier option to use, rather than comparing timestamps. The NN packet and Data packet sequence numbers should match.

Also, the left and right cameras will both have the same sequence number (timestamps will not be precisely the same, but few microseconds apart – that's because the timestamp is assigned separately to each from different interrupt handlers. But the cameras are started at the same time using an I2C broadcast write, and also use the same MCLK source, so shouldn't drift).

In this case we also need to check the camera source of the NN and Data packets. Currently, depthai.py uses getMetadata().getCameraName() for this purpose, that returns a string: `rgb`, `left` or `right` .

It is also possible to use `getMetadata().getInstanceNum()`, that returns a number: 0, 1 or 2 , respectively.

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

DepthAI suppots h.264 and h.265 (HEVC) and JPEG encoding directly itself - without any host support. The *depthai_demo.py* script shows and example of how to access this functionality.

To leverage this functionality from the command line, use the *-v* (or *–video*) command line argument as below:

```
python3 depthai_demo.py -v [path/to/video.h264]
```

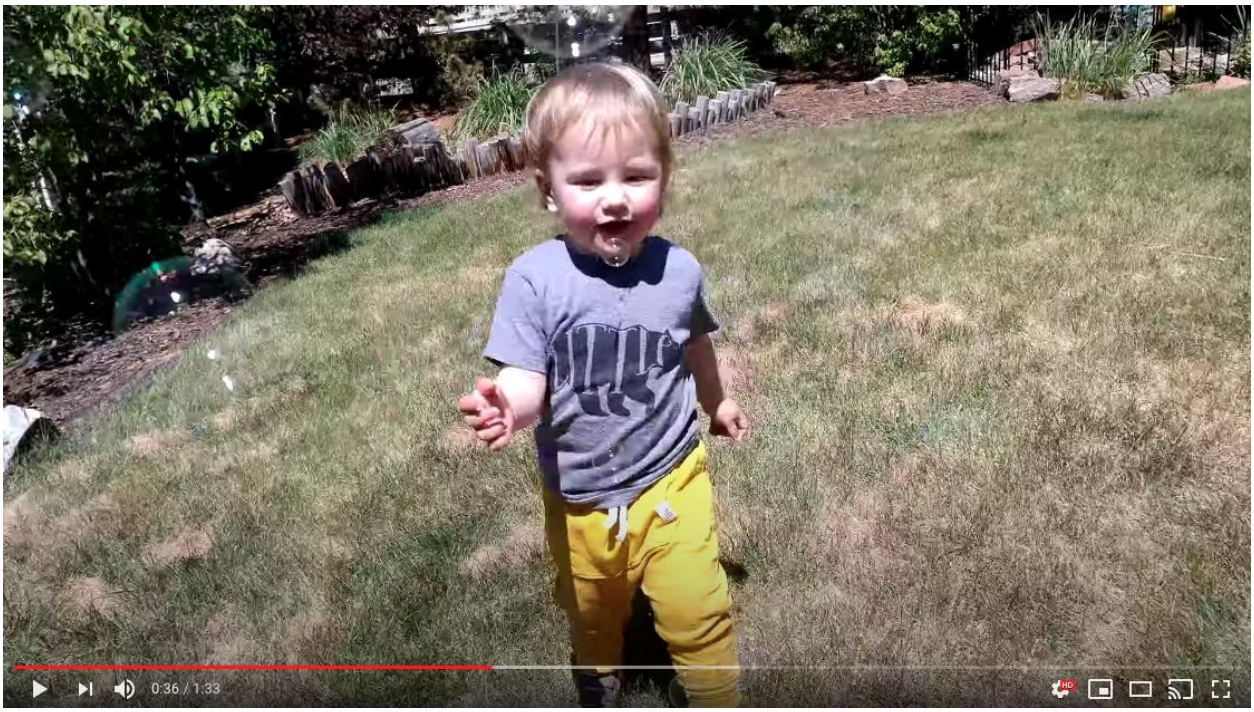
To then play the video in mp4/mkv format use the following muxing command:

```
ffmpeg -framerate 30 -i [path/to/video.h264] -c copy [outputfile.mp4/mkv]
```

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 BW1097 (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.



Video Encoding Options

Additional options can be configured in the video encoding system by adding a `video_config` section to the JSON config of the DepthAI pipeline builder, here, an example of which is here.

```
config = {
  ...
  'video_config':
  {
      'rateCtrlMode': 'cbr', # Options: 'cbr' / 'vbr' (constant bit rate or variable bit rate)
      'profile': 'h265_main', # Options: 'h264_baseline' / 'h264_main' / 'h264_high' / 'h265_main'
      'bitrate': 8000000, # When using CBR
      'maxBitrate': 8000000, # When using CBR
      'keyframeFrequency': 30, # In number of frames
      'numBFrames': 0,
      'quality': 80 # (0 - 100%) When using VBR
  }
  ...
}
```

The options above are all current options exposed for video encoding and not all must be set.

If `video_config` member is **NOT** present in config dictionary then default is used: H264_HIGH, constant bitrate 8500Kbps, keyframe every 30 frames (once per second), num B frames: 0

1.2.29 What Are The Stream Latencies?

When implementing robotic or mechatronic systems it is often quite useful to know how long it takes from a photo 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 1: Worst-case estimates of stream latencies

measured	requested	avg latency, ms
left	left	100
left	left, right	100
left	left, right, depth	100
left	left, right, depth, metaout, previewout	100
previewout	previewout	65
previewout	metaout, previewout	100
previewout	left, right, depth, metaout, previewout	100
metaout	metaout	300
metaout	metaout, previewout	300
metaout	left, right, depth, metaout, previewout	300

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

Yes, but currently not currently implemented in our API. It's on our roadmap, here

The `why` of our DepthAI API provides more flexibility in formats (unencoded, encoded, metadata, processing, framerate, etc.) and already works on any operating system (see *here*). So what we plan to do is to support UVC as part of our Gen2 Pipeline builder, so you can build a complex spatial AI/CV pipeline and then have the UVC endpoints output the results, so that DepthAI could then work on any system without drivers. For our embedded variants, this could then be flashed to the device so that the whole pipeline will automatically run on boot-up and show up to a computer a UVC device (a webcame).

Theoretically we can implement support for 3 UVC endpoints (so showing up as 3 UVC cameras), on for each of the 3 cameras.

We've prototyped 2x w/ internal proof of concept (but grayscale) but have not yet tried 3 but it would probably work. We could support a UVC stream per camera if it is of interest.

So if you would like this functionality please feel subscribe to the Github feature request here.

1.2.31 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.

To force USB2 mode, simply use the *-fusb2* (or *–force_usb2*) command line option as below:

```
python3 depthai_demo.py -fusb2
```

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 coming, 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).

1.2.32 What is "NCS2 Mode"?

All variants of DepthAI/megaAI come supporting what we call 'NCS2 mode'. This allows megaAI and DepthAI 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, DepthAI/megaAI will behave identically.

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.

DepthAI/megaAI will also support an additional host-communication mode in the Gen2 Pipeline Builder, which will be available in December 2020.

1.2.33 What Information is Stored on the DepthAI Boards

Initial Crowd Supply backers received boards which had literally nothing stored on them. All information was loaded from the host to the board. This includes the BW1097 (*BW1097*), which had the calibration stored on the included microSD card.

So each hardware model which has stereo cameras (e.g. *BW1097*, *BW1098FFC*, BW1098OBC, and *BW1094*) has the capability to store the calibration data and field-of-view, stereo baseline (`L-R distance`) and relative location of the color camera to the stereo cameras (`L-RGB distance`) as well as camera orientation (`L/R swapped`). To retrieve this information, simply run `python3 depthai_demo.py` and look for `EEPROM data:`.

Example of information pulled from a BW1098OBC is below:

```
EEPROM data: valid (v2)
  Board name      : BW1098OBC
  Board rev       : R0M0E0
  HFOV L/R        : 73.5 deg
  HFOV RGB        : 68.7938 deg
  L-R   distance : 7.5 cm
  L-RGB distance : 3.75 cm
  L/R swapped     : yes
  L/R crop region: top
  Calibration homography:
    1.002324,   -0.004016,   -0.552212,
    0.001249,    0.993829,   -1.710247,
    0.000008,   -0.000010,    1.000000,
```

Current (those April 2020 and after) DepthAI boards with on-board stereo cameras (*BW1097*, BW1098OBC, and BW1092) ship calibration and board parameters pre-programmed into DepthAI's onboard EEPROM.

1.2.34 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.

1.2.35 What is the Field of View of DepthAI and megaAI?

DepthAI and megaAI use the same 12MP RGB Camera module based on the IMX378.

- 12MP RGB Horizontal Field of View (HFOV): 68.7938 deg
- 1MP Global Shutter Grayscale Camera Horizontal Field of View (HFOV): 73.5 deg

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

ArduCam is in the process of making a variety of camera modules specifically for DepthAI and megaAI, including a variety of M12-mount options (so that the optics/view-angles/etc. are change-able by you the user).

- M12-Mount IMX378 request here
- M12-Mount OV9281 request here
- Fish-Eye OV9281 (for better SLAM) request here
- Mechanical, Optical, and Electrical equivalent OV9282 module with visible and IR capability here
- Global-Shutter Color Camera (OV9782) with same intrinsics as OV9282 grayscale here
- Original request for this here

With these, there will be a variety of options for view angle, focal length, filtering (IR, no IR, NDVI, etc.) and image sensor formats.

1.2.37 What are the Highest Resolutions and Recording FPS Possible with DepthAI and megaAI?

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

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

DepthAI and megaAI can do h.264 and h.265 (HEVC) encoding on-device. The max resolution/rate is 4K at 30FPS. With the default encoding settings in DepthAI/megaAI, 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 DepthAI BW1097 (Raspberry Pi Compute Module Edition) is below:

It's worth noting that all DepthAI and megaAI products share the same color camera specs and encoding capabilities. So footage filmed on a DepthAI unit with the color camera will be identical to that taken with a megaAI unit.

Encoded:

- 12MP (4056x3040) : JPEG Pictures/Stills
- 4K (3840x2160) : 30.00fps (3.125MB/s)

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

DepthAI and megaAI 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 framerates over USB *here*)
- 450 GB/sec memory bandwidth
- 512 MB LPDDR4 (contact us for 1GB LPDDR version if of interest)

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

DepthAI and megaAI support continuous video autofocus ('2' below, where the system is constantly autonomously searching for the best focus) and also and `auto` mode ('1' below) which waits to focus until directed by the host. (PR which adds this functionality is here.)

Example usage is shown in `depthai_demo.py`. When running `python3 depthai_demo.py` the functionality can be used by keyboard command while the program is running:

- '1' to change autofocus mode to auto
  - 'f' to trigger autofocus
- '2' to change autofocus mode to continuous video

And you can see the reference DepthAI API call here

1.2.40 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 DepthAI/megaAI's color camera module is quite close because of it's f.no and focal length.

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

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

Where:

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

So H = (4.52mm)^2/(2.0 * 0.00578mm) + 4.52mm ~= 1,772mm, or **1.772 meters** (**5.8 feet**).

We are using the effective focal length, and since we're not optics experts, we're not 100% sure if this is appropriate here, but the total height of the color module is 6.05mm, so using that as a worst-case focal length, this still puts the hyperfocal distance at **10.4 feet**.

So what does this mean for your application?

Anything further than 10 feet away from DepthAI/megaAI will be in focus when the focus is set to 10 feet or beyond. In other words, as long as you don't have something closer than 10 feet which the camera is trying to focus on, everything 10 feet or beyond will be in focus.

1.2.41 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 details on controlling auto-focus/focus.

Exposure (AE)

It is possible to set frame duration (us), exposure time (us), sensitivity (iso) via the API. And we have a small example for the color camera to show how to do this for the color camera, which is here: https://github.com/luxonis/depthai/pull/279

We are planning on making these controls more self-documenting (see here), but in the meantime, all of the available controls are here: https://github.com/luxonis/depthai-shared/blob/82435d4/include/depthai-shared/metadata/camera_control.hpp#L107

And for example to set an exposure time of 23.4ms, with the maximum sensitivity of 1600, use:

```
self.device.send_camera_control(
   depthai.CameraControl.CamId.RGB,
   depthai.CameraControl.Command.AE_MANUAL,
   "23400 1600 33333")
```

White Balance (AWB)

This will be implemented at the same time as exposure and will be included. AWB lock, AWB modes. We will post more information as we dig into this task.

1.2.42 What Are the Specifications of the Global Shutter Grayscale Cameras?

The stereo pair is composed of synchronized global shutter OV9282-based camera modules.

Specifications:

- Effective Focal Length (EFL): 2.55
- F-number (F.NO): 2.2 +/- 5%
- Field of View (FOV): - Diagonal (DFOV): 82.6(+/-0.5) deg. - Horizontal (HFOV): 73.5(+/-0.5) deg. - Vertical (VFOV): 50.0(+/-0.5) deg.
- Distortion: < 1%
- Lens Size: 1/4 inch
- Focusing: Fixed Focus, 0.196 meter (hyperfocal distance) to infinity
- Resolution: 1280 x 800 pixel
- Pixel Size: 3x3 micrometer (um)

1.2.43 Am I able to attached 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.

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 RPi HQ Camera to enable use of C- and CS-mount lenses.

1.2.44 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.

1.2.45 How to use DepthAI under VirtualBox

If you want to use VirtualBox to run the DepthAI source code, please make sure that you allow the VM to access the USB devices. Also, be aware that by default, it supports only USB 1.1 devices, and DepthAI operates in two stages:

1. For showing up when plugged in. We use this endpoint to load the firmware onto the device, which is a usb-boot technique. This device is USB2.
2. For running the actual code. This shows up after USB booting and is USB3.

In order to support the DepthAI modes, you need to download and install Oracle VM VirtualBox Extension Pack

1.2.46 How to increase NCE, SHAVES and CMX parameters?

If you want to specify how many Neural Compute Engines (NCE) to use, or how many SHAVE cores, or how many Connection MatriX blocks, you can do this with the DepthAI.

We have implemented the `-nce`, `-sh` and `-cmx` command line params in our example script. Just clone the DepthAI repository and do

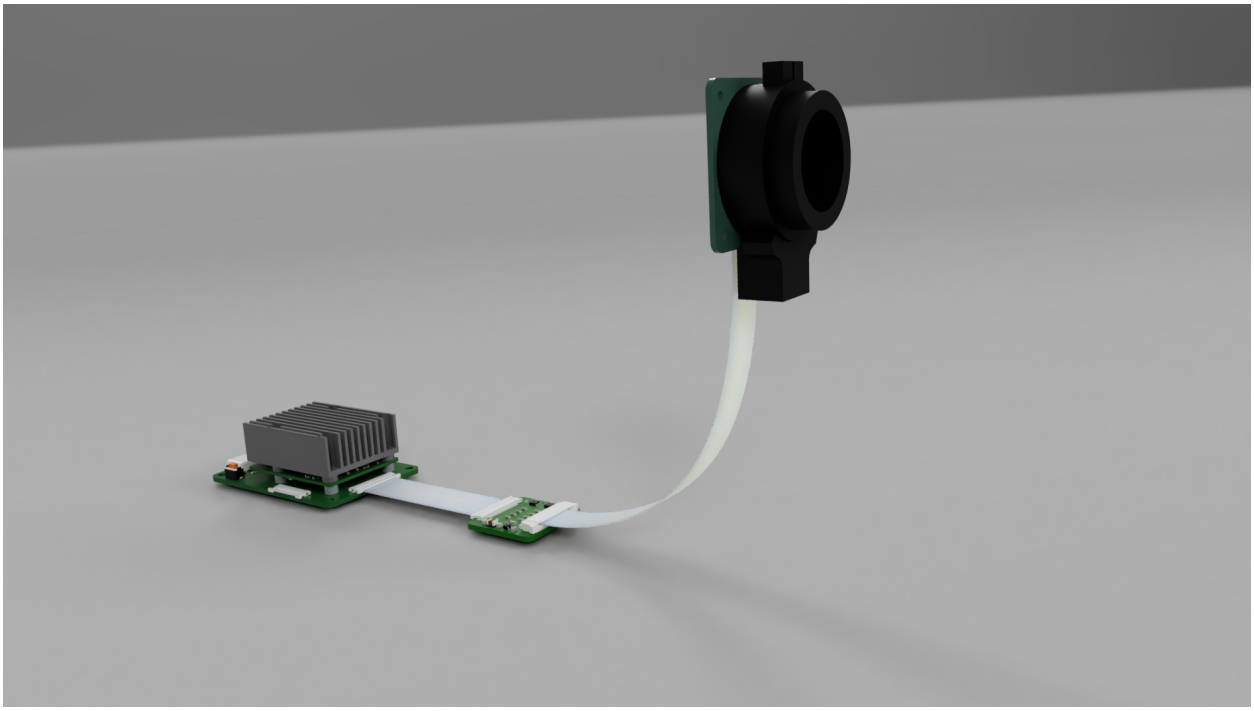
```
./depthai_demo.py -nce 2 -sh 14 -cmx 14
```

And it will run the default MobilenetSSD, compiled to use 2 NCEs, 14 SHAVEs and 14 CMXes. Note that these values **cannot be greater than the ones you can see above**, so you cannot use 15 SHAVEs or 3 NCEs. 14 is the limit for both SHAVE and CMX parameters, and 2 is the limit for NCE.

You can try it out yourself either by following *local OpenVINO model conversion tutorial* or by using our online MyriadX blob converter

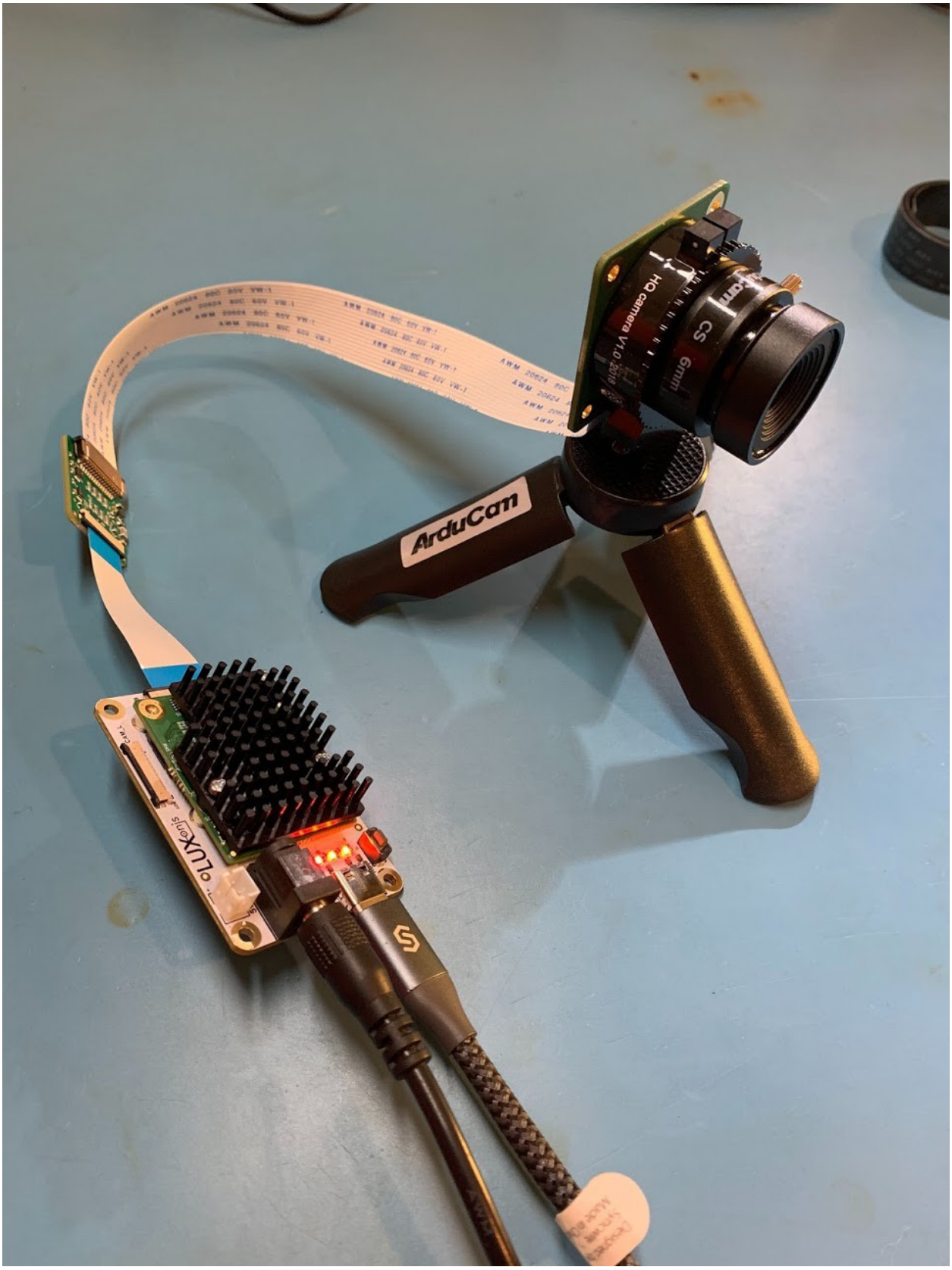
1.2.47 Can I Use DepthAI with the New RPi HQ Camera?

DepthAI FFC Edition (BW1098FFC model here) 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 DepthAI FFC Edition.



This is a particularly interesting application of DepthAI, as it allows the RPi 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).

Here are some quick images and videos of it in use:

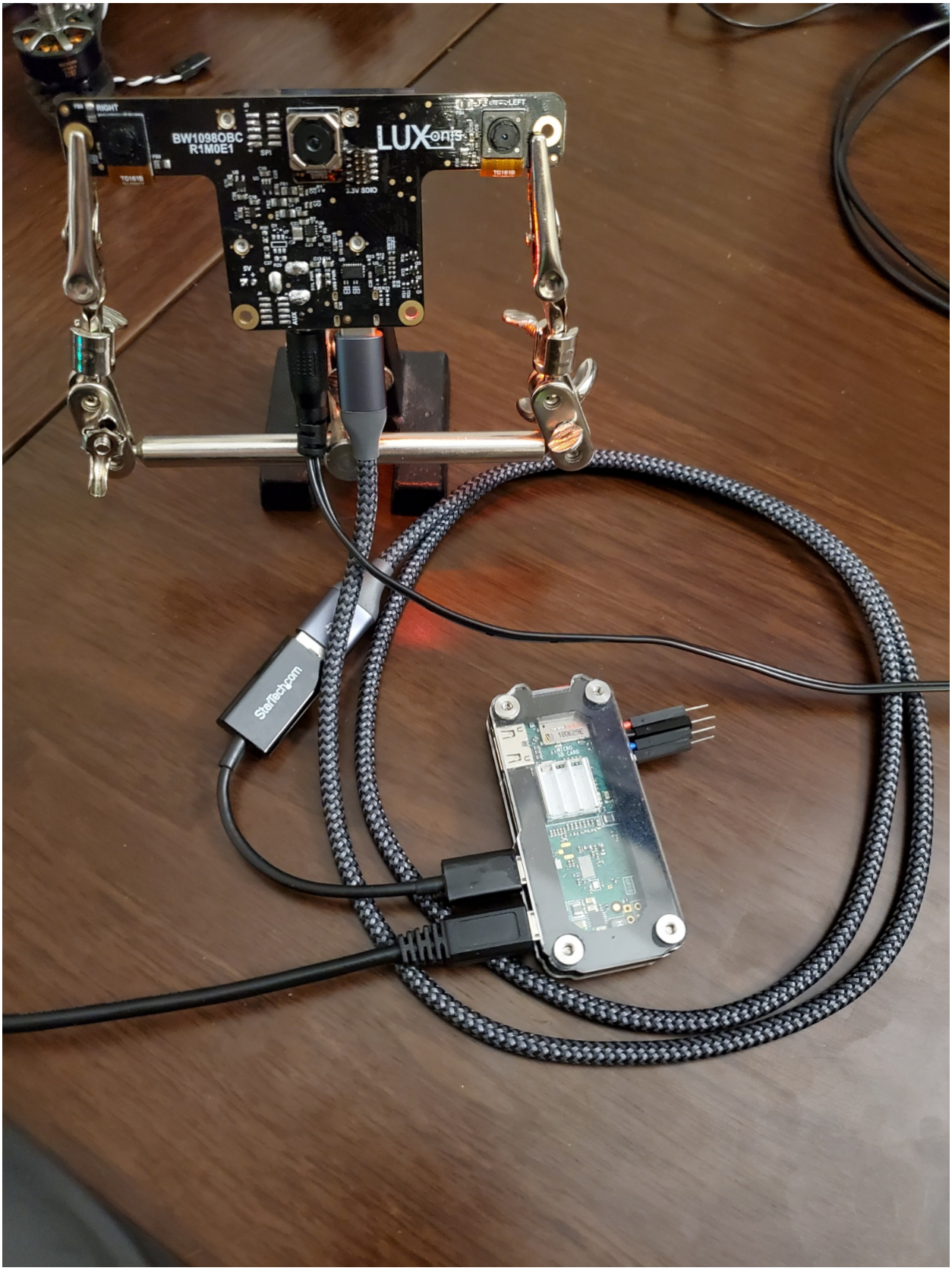

ArduCam
HQ camera V1.0 2018
6mm
CS

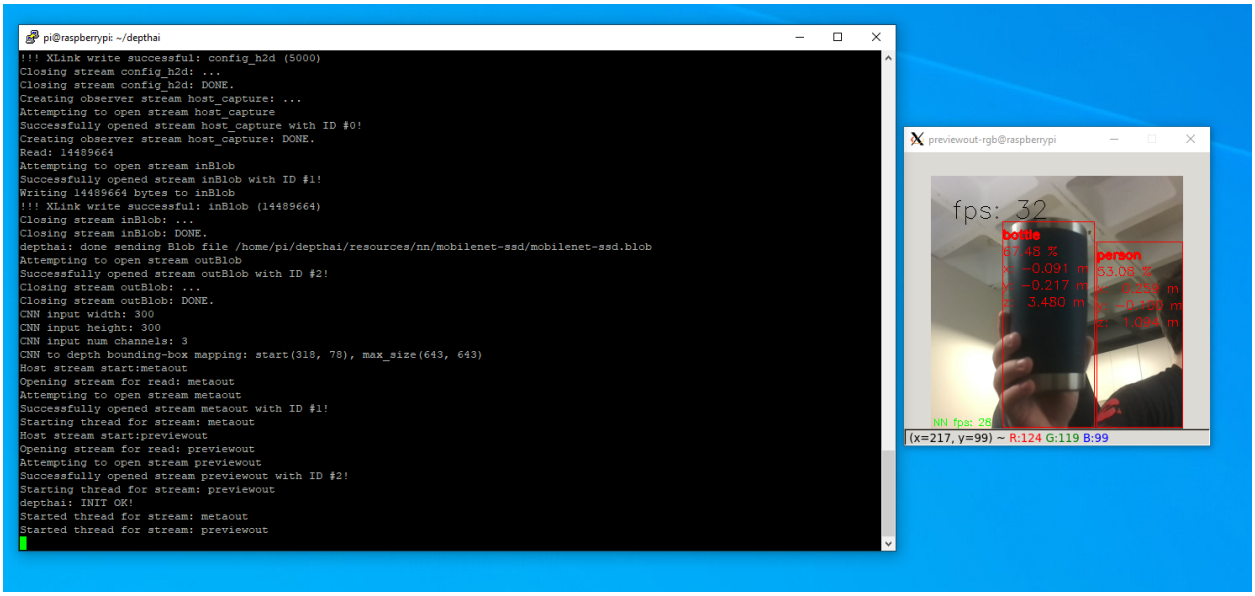

AmScope
ArduCam

You can buy this adapter kit for the DepthAI FFC Edition (BW1098FFC) here

1.2.48 Can I use DepthAI with Raspberry Pi Zero?

Yes, DepthAI is fully functional on it, you can see the example below:


BW1098OBC
R1M0E1
LUXonis
StarTech.com



Thanks to Connor Christie for his help building this setup!

1.2.49 How Much Power Does the DepthAI RPi CME Consume?

The DepthAI Raspberry Pi Compute Module Edition (RPi CME or BW1097 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:

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

- For the gray scale cameras, we use 0.5mm, 20-pin, same-side contact flex cables.
- For the RGB camera, we use a 0.5mm 26-pin, same-side contact flex cable.

One can purchase Molex's 15166 series FFCs directly to support shorter or longer lengths. Make sure you get **same-side** contacts, Molex calls this "**Type A**"

1.2.51 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

1.2.52 Where are the Githubs? 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 Githubs:

Overall

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

Embedded Use Case

- 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 - Bootloader source code which allows programming NOR flash of DepthAI to boot autonomously
- https://github.com/luxonis/depthai-spi-api - SPI interface library for Embedded (microcontroller) DepthAI application
- https://github.com/luxonis/esp32-spi-message-demo/tree/gen2_common_objdet - ESP32 Example applications for Embedded/ESP32 DepthAI use (e.g. with BW1092)

1.2.53 Can I Use and IMU With DepthAI?

Yes, our BW1099 (here) has support to talk to IMUs. And we are in the process of making a future version of the BW1098OBC (as well as BW1092) which have built-in BNO085. We do not yet have support for this IMU in the DepthAI API, but we have done proof-of-concepts and will be making this a standard feature through the API.

1.2.54 Where are Product Brochures and/or Datasheets?

Brochures:

- Editions Summary here
- System on Module (BW1099) here
- USB3 Modular Cameras Edition (BW1098FFC) here
- USB3 Onboard Cameras Edition (BW1098OBC) here
- Raspberry Pi Compute Edition Module (BW1097) here
- Raspberry Pi HAT (BW1094) here
- megaAI (BW1093) here

Datasheets:

- DepthAI System on Module (BW1099) here
- PoE Modular Cameras Edition (BW2098FFC) here

1.2.55 How can I cite Luxonis products in publications ?

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

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

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

1.2.56 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 ours (and often w/in a couple minutes). For a summary of our Githubs, 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.

1.3 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 Community Discord for live assistance from us and devs like you.
- Post a message to our forum.
- Submit an issue on DepthAI repository

1.3.1 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 Emporia Energy using USPS or Fedex. 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.

1.4 Troubleshooting

1.4.1 How can the startup demo on the RPi Compute Edition be disabled?

Delete the autostart file:

```
rm /home/pi/.config/autostart/runai.desktop
```

1.4.2 ImportError: No module named 'depthai'

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

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

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

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

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

1.4.3 Why is the Camera Calibration running slow?

Poor photo conditions can dramatically impact the image processing time) during the camera calibration. Under normal conditions, it should take 1 second or less to find the chessboard corners per-image on an RPi 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.

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

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.

1.4.5 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 (bootloader running)
- `03e7:f63b` after the application was loaded

No `/dev/video*` nodes are created.

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

1.4.6 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 do 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 `--force_usb2` option (examples below). This will allow use of the long cable still, and many DepthAI usecases do not necessitate USB3 communication bandwidth - USB2 is plenty.
3. Upgrade from Ubuntu 16.04 to Ubuntu 18.04.

Forcing USB2 Communication

If you are having trouble with communication with DepthAI/OAK, forcing USB2 can sometimes resolve the issue.

```
python3 depthai_demo.py --force_usb2
```

Or, the shorter form:

```
python3 depthai_demo.py -fusb2
```

We've also seen an unconfirmed issue of running Ubuntu-compiled libraries on Linux Mint. If running on not Ubuntu 18.04/16.04 or Raspbian, please *compile DepthAI from source*.

1.4.7 Failed to boot the device: 1.3-ma2480, err code 3

This error often can occur if the udev rules are not set on Linux. This wil coincide with depthai: Error initializing xlink

To fix this, set the udev rules using the commands below, unplugging depthai and replugging it 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
```

And in some cases, these were already set, but depthai was plugged in the entire time, so Linux could not reset the rules.

So make sure to unplug, and replug-in depthai after having run these.

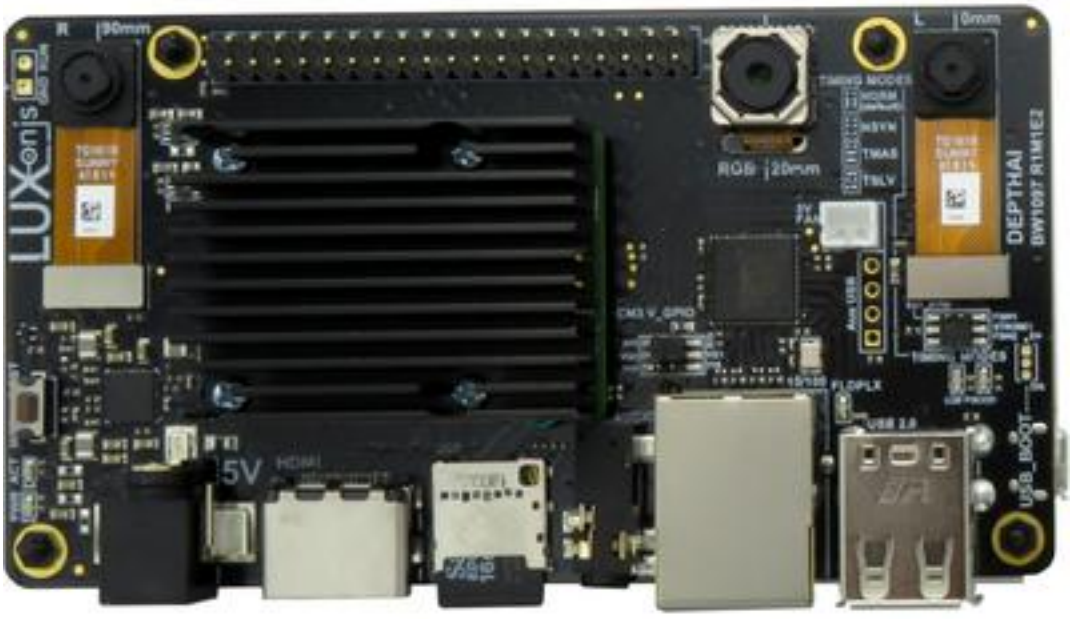
We're always happy to help with code or other questions you might have.

1.4.8 CTRL-C Is Not Stopping It!

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

1.5 Products

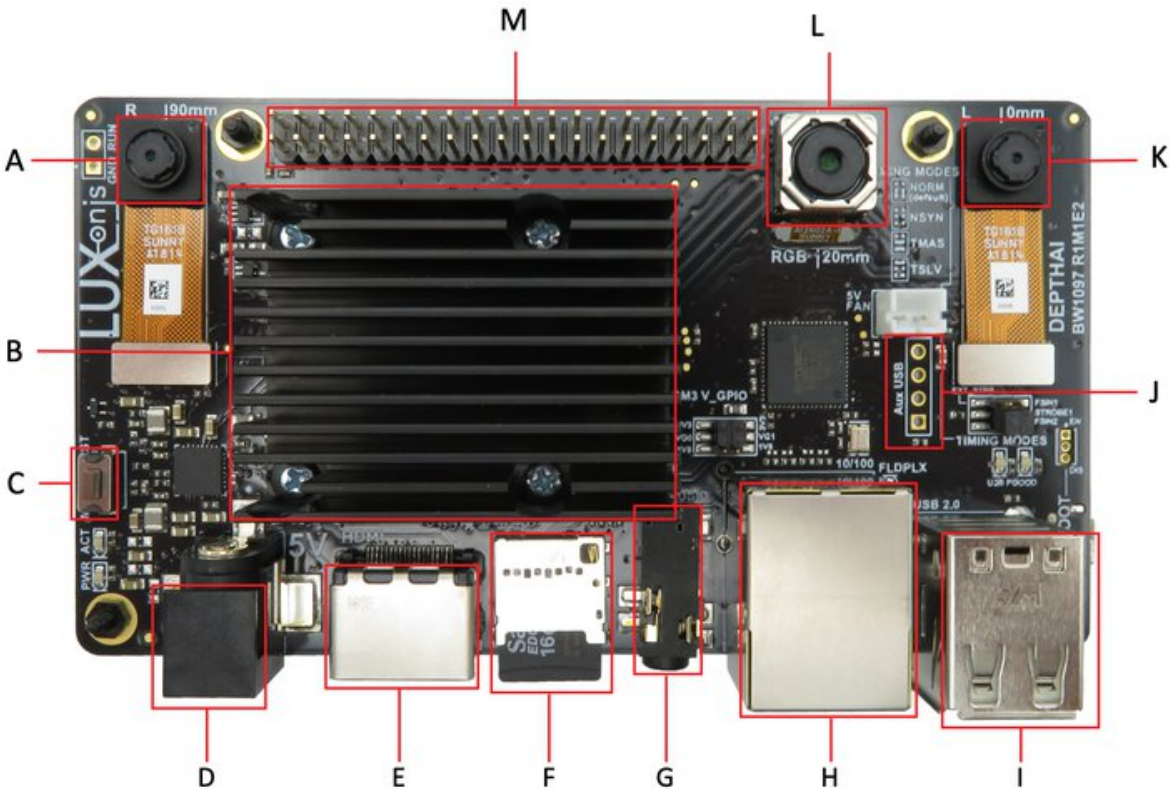
1.5.1 BW1097 - RaspberryPi Compute Module



The Raspberry Pi Compute Module Edition comes with everything needed: pre-calibrated stereo cameras on-board with a 4K, 60 Hz color camera and a µSD card with Raspbian and DepthAI Python code automatically running on bootup. This allows using the power of DepthAI with literally no typing or even clicking: it just boots up doing its thing. Then you can modify the Python code with one-line changes, replacing the neural model for the objects you would like to localize.

- Built-in RaspberryPi Compute Module
- Three integrated cameras
- Complete system; everything you need is included

Board Layout



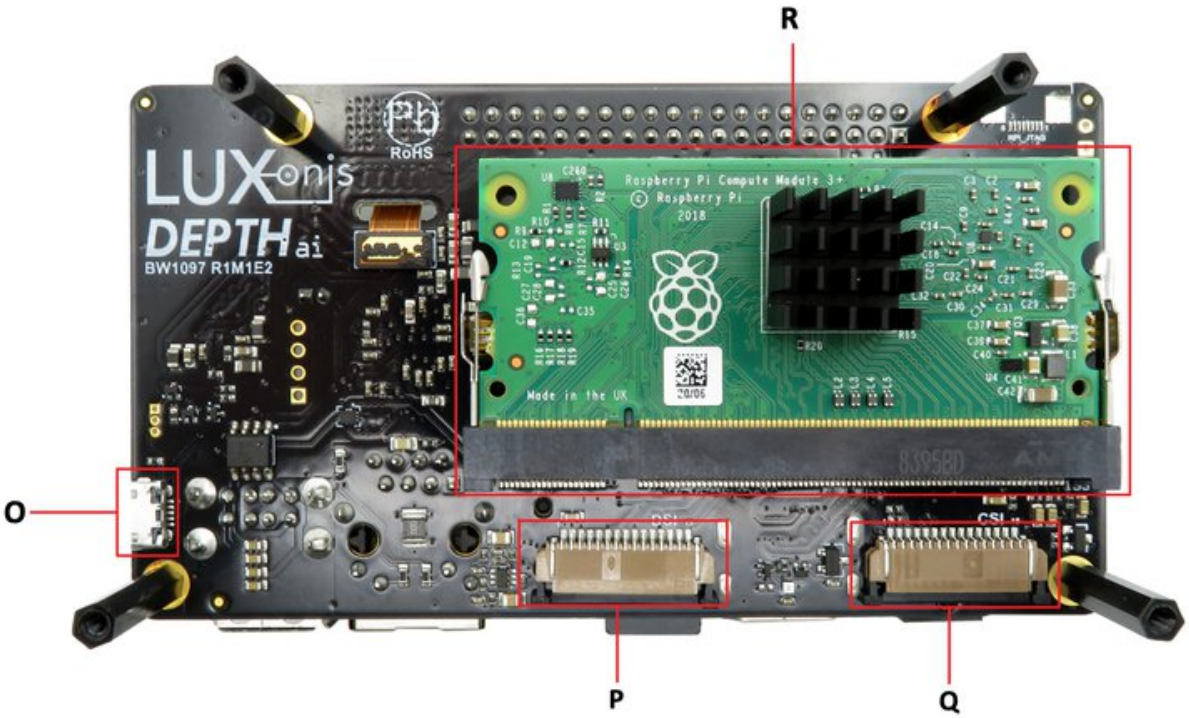


Table 2: Reference table

A. 720p 120 Hz Global Shutter (Right)	**J.** 1x Solderable USB2.0
B. DepthAI Module	**K.** 720p 120 Hz Global Shutter (Left)
C. DepthAI Reset Button	**L.** 4K 60 Hz Color
D. 5 V IN	**M.** RPi 40-Pin GPIO Header
E. HDMI	**O.** RPi USB-Boot
F. 16 GB µSD Card, Pre-configured	**P.** RPi Display Port
G. 3.5 mm Audio	**Q.** RPi Camera Port
H. Ethernet	**R.** Raspberry Pi Compute Module 3B+
I. 2x USB2.0	

What's in the box?

- BW1097 Carrier Board
- Pre-flashed µSD card loaded with Raspbian 10 and DepthAI
  - Default Password: `luxonis`
- WiFI USB dongle
- Power Supply

Setup

To get started:

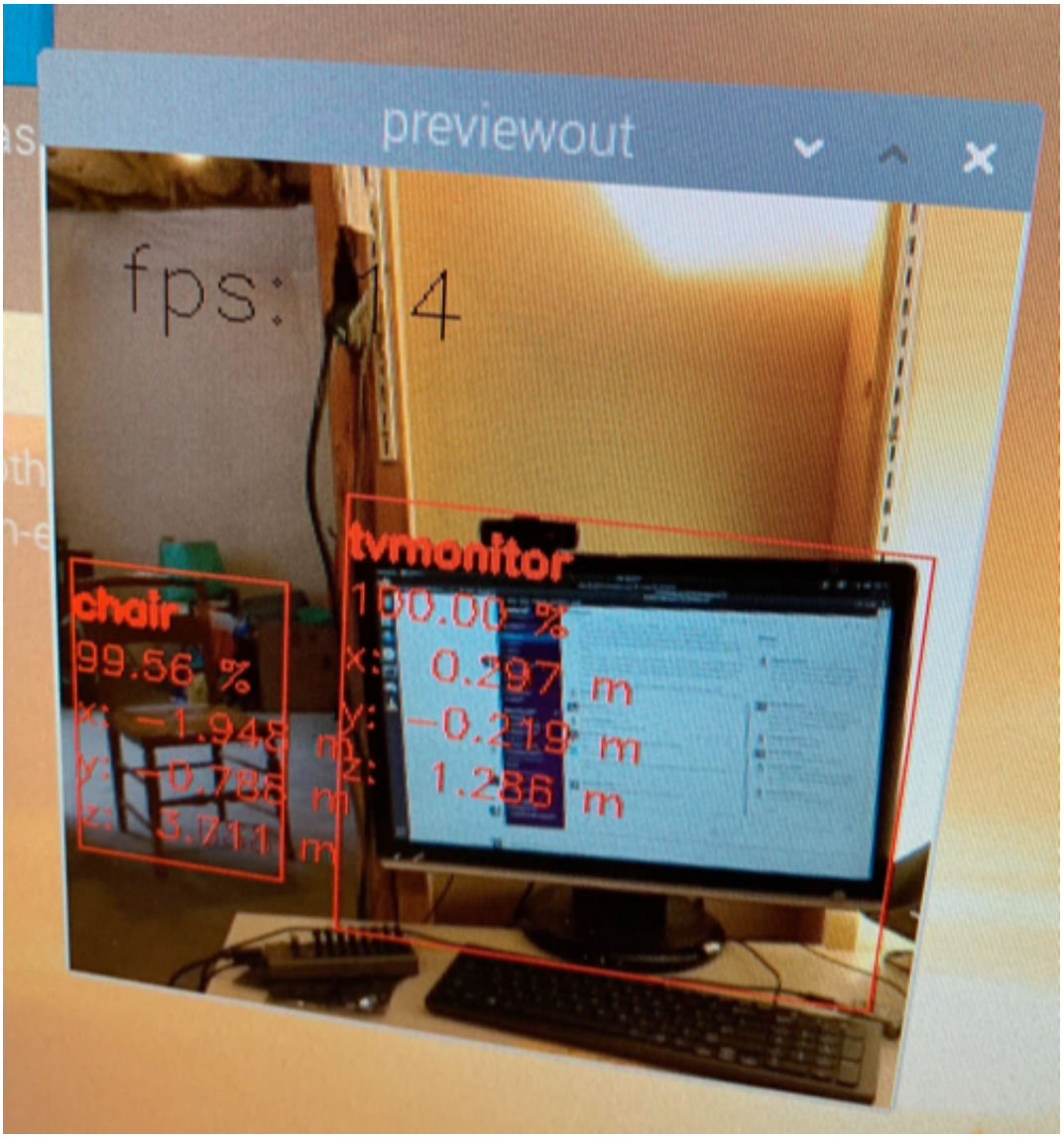
1. **Connect a display to the HDMI port.**

   Note that an HDMI cable is not included.

2. **Connect a keyboard and mouse via the USB port**

3. **Connect the power supply (included).**

   On boot, the Pi will run a Python demo script that displays a video stream annotated with object localization metadata:



   In the screenshot above, DepthAI identified a tv monitor (1.286 m from the camera) and a chair (3.711 m from the camera). See the list of object labels on GitHub.

4. **Connect to the Internet.**

   Connect the Pi to the Internet to begin trying the DepthAI tutorials and examples.

- **Connecting to a WiFi network**

  To connect to a WiFi network, use the included Linux-compatible USB WiFi dongle. The Pi should recognize the dongle and display available WiFi networks in the upper right corner of the Raspbian Desktop UI.

- **Connecting to a network via Ethernet**

  The board includes an Ethernet port. Connecting an Ethernet cable to the port will enable Internet access.

5. **Run example script**.

   See *Verify installation*

[Optional] Using your own SD-Card

If you'd like to set up DetphAI on your own (say bigger) SD-Card, there are two options:

1. Download our pre-configured Raspbian image for the BW1097 (the Raspberry Pi Compute Module Edition), here: BW1097 Raspian Image. Then, after downloading, update the DepthAI firmware/software (by doing a git pull on the DepthAI code base checked out on the Desktop).
2. Set up your own Raspbian to your liking from say a fresh Raspbian download, and then use replace dt-blob.bin and config.txt in /boot with the following two files:
   - dt-blob.bin - For enabling the Pi MIPI display
   - config.txt - For enabling the 3.5mm headphone jack

1.5.2 BW1094 - RaspberryPi Hat

The Raspberry Pi HAT Edition allows using the Raspberry Pi you already have and passes through the Pi GPIO so that these are still accessible and usable in your system(s). Its modular cameras allow mounting to your platform where you need them, up to six inches away from the HAT.

- Mounts to Raspberry Pi as a HAT for easy integration
- All Raspberry Pi GPIO still accessible through pass-through header
- Flexible Camera Mounting with 6” flexible flat cables
- Includes three FFC Camera ports

Requirements

- A RaspberryPi with an extended 40-pin GPIO Header.

Board Layout

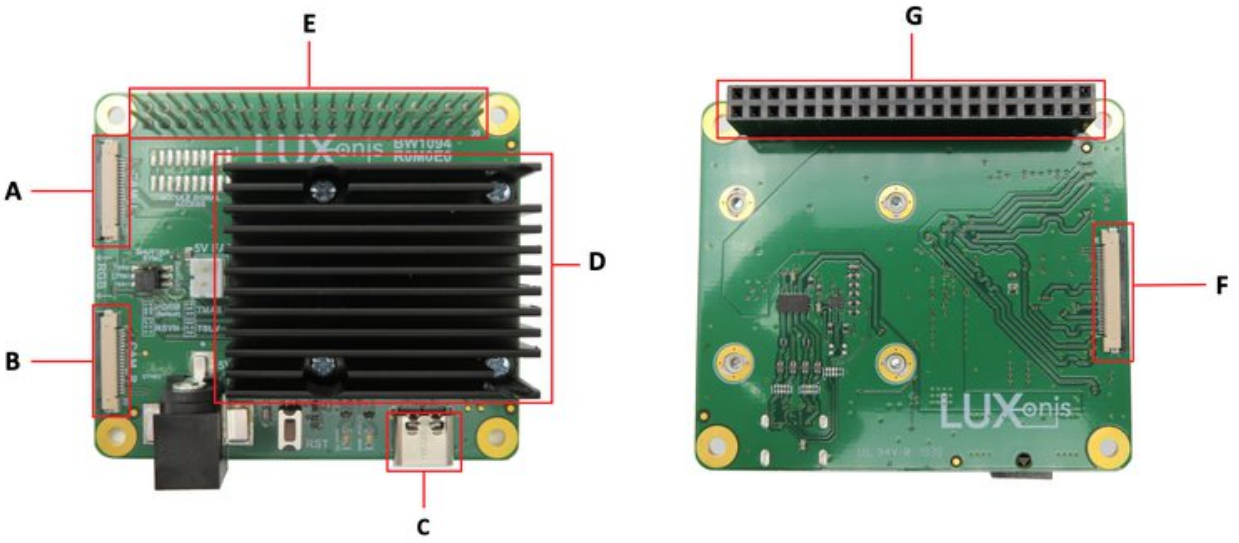


Table 3: Reference table

A. Left Camera Port	**E.** Pass-through 40-Pin Raspberry Pi Header
B. Right Camera Port	**F.** Color Camera Port
C. USB 3.0 Type-C	**G.** 40-pin Raspberry Pi Header
D. DepthAI Module	

What's in the box?

- BW1094 Carrier Board
- Pre-flashed µSD card loaded with Raspbian 10 and DepthAI
- USB3C cable (6 in.)

Setup

Follow the steps below to setup your DepthAI device.

1. **Power off your Raspberry Pi.**

   Safely power off your Raspberry Pi and unplug it from power.

2. **Insert the pre-flashed µSD card into your RPi.**

   The µSD card is pre-configured with Raspbian 10 and DepthAI.

3. **Mount the DepthAI RPi HAT.**

   Use the included hardware to mount the DepthAI RPi HAT to your Raspberry Pi.

4. **Reconnect your RPi power supply**

5. **Calibrate the cameras**.

   See *Calibration*

6. **Run example script**.

See *Verify installation*

1.5.3 BW1098FFC - USB3 with Modular Cameras

Use DepthAI on your existing host. Since the AI/vision processing is done on the Myriad X, a typical desktop could handle tens of DepthAIs plugged in (the effective limit is how many USB ports the host can handle).

Requirements

- Ubuntu 18.04 or Raspbian 10
- Cameras
  - *Modular color camera*
  - *Stereo camera pair* (if depth is required)
- USB3C cable
- USB3C port on the host

Board Layout

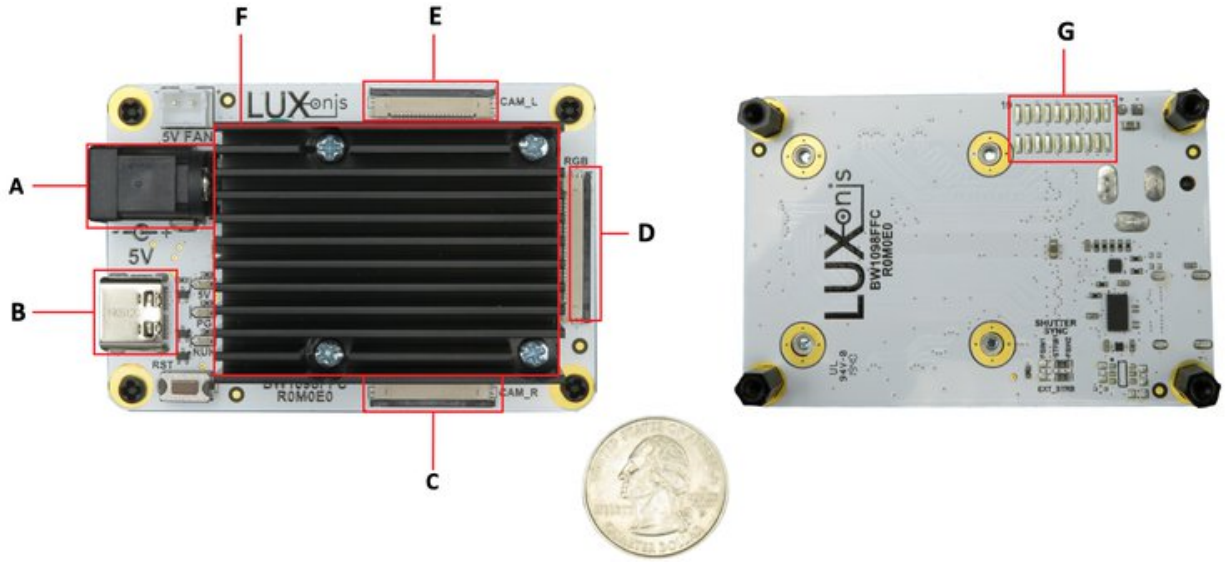


Table 4: Reference table

A. 5V IN	**E.** Left Camera Port
B. USB3C	**F.** DepthAI Module
C. Right Camera Port	**G.** Myriad X GPIO Access
D. Color Camera Port	

What's in the box?

- BW1098FFC Carrier Board
- USB3C cable (3 ft.)
- Power Supply

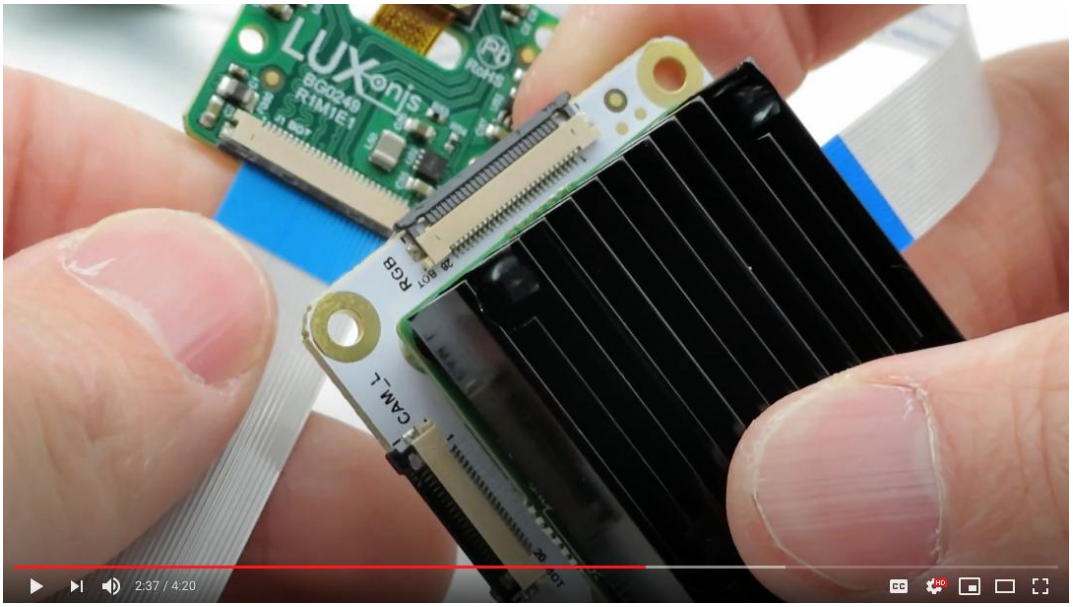
Setup

Follow the steps below to setup your DepthAI device.

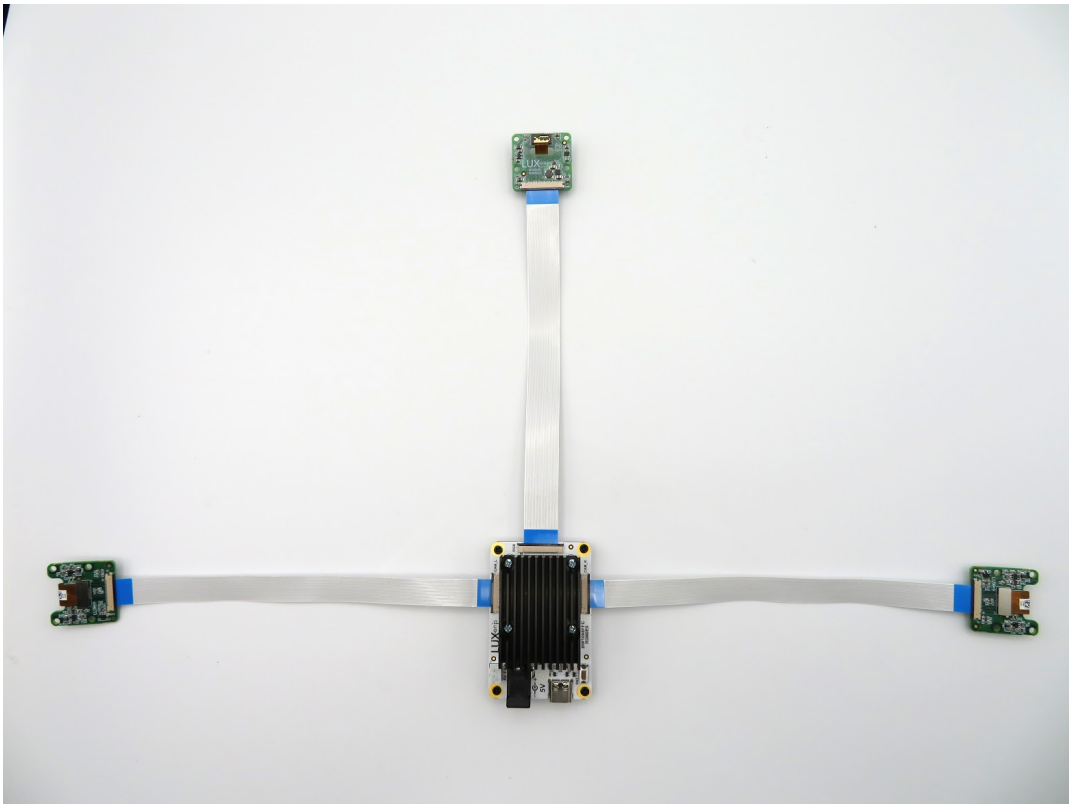
1. **Connect your modular cameras.**

   The FFC (flexible flat cable) Connectors on the BW1098FFC require care when handling. Once inserted and latched, the connectors are robust, but they are easily susceptible to damage during the de-latching process when handling the connectors, particularly if to much force is applied during this process.

   The video below shows a technique without any tool use to safely latch and delatch these connectors.



Once the flexible flat cables are securely latched, you should see something like this:



Note: Note when looking at the connectors, the blue stripe should be facing up.

> **Warning:** Make sure that the FFC cables connect to the camera is on the top side of the final setup to avoid inverted images and wrong `swap_left_and_right_cameras` setup.

2. **Connect your host to the DepthAI USB carrier board.**
3. **Connect the DepthAI USB power supply (included).**
4. **Calibrate the cameras**.

   See *Calibration*

5. **Run example script**.

   See *Verify installation*

1.5.4 OAK-D | DepthAI Onboard Cameras

The Spatial AI + CV Power House.

Requirements

- USB3C cable (included)
- USB-capable host

What's in the box?

- OAK-D / DepthAI USB3C
- USB3C cable (3 ft.)
- Power Supply
- Getting Started Card

Setup

- Install the DepthAI API, see here for how to do so.

1.5.5 BW1099 - System on Module

All DepthAI editions utilize the System on Module (SoM), which can also be used by itself to integrate into your own designs. The SoM allows the board that carries it to be a simple, easy four-layer standard-density board, as opposed to the high-density-integration (HDI) stackup (with laser-vias and stacked vias) required to directly integrate the VPU itself.

Specifications

- 2x 2-lane MIPI Camera Interface
- 1x 4-lane MIPI Camera Interface
- Quad SPI with 2 dedicated chip-selects
- I^2C
- UART
- USB2
- USB3
- Several GPIO (1.8 V and 3.3 V)
- Supports off-board eMMC or SD Card
- On-board NOR boot Flash (optional)

- On-board EEPROM (optional)
- All power regulation, clock generation, etc. on module
- All connectivity through single 100-pin connector (DF40C-100DP-0.4V(51))

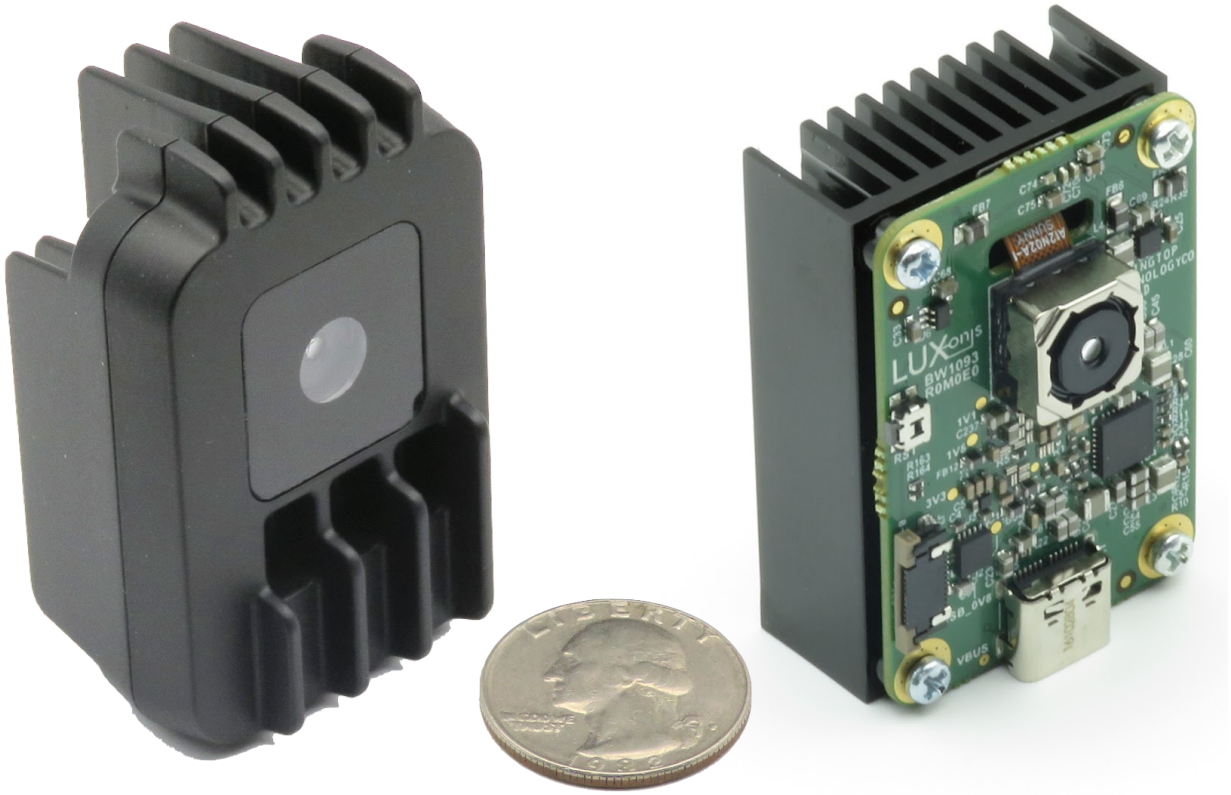
Datasheets are available here and for EMB edition here

Getting Started Integrating Into Your Products

All the boards based on the DepthAI System on Module are available on Github under MIT License here.

These are in Altium Designer format. So if you use Altium Designer, you're in luck! You can quickly/easily integrate the DepthAI SoM into your products with proven and up-to-date designs (the same designs you can buy here).

1.5.6 BW1093 - OAK-1 | MegaAI - 4K USB3 AI Camera



Use OAK-1/megaAI on your existing host. Since the AI/vision processing is done on the Myriad X, a typical desktop could handle tens of OAK-1/megaAI plugged in (the effective limit is how many USB ports the host can handle).

And since it can encode 1080p and 4K video (see *here*) you can now even save 4K video on a Pi Zero!

Requirements

- USB3C cable
- USB2 or USB3 port on the host

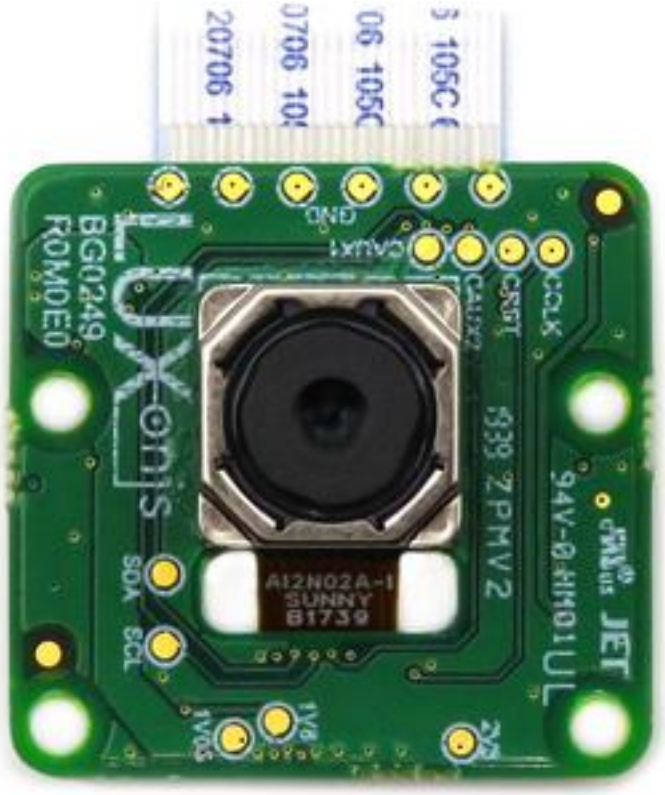
What's in the box?

- megaAI / OAK-1
- USB3C cable (3 ft.)
- Getting Started Card

Setup

- Install the DepthAI API, see here for how to do so.

1.5.7 DepthAI Color Camera



4K, 60Hz video camera with 12 MP stills and 4056 x 3040 pixel resolution.

Specifications

- 4K, 60 Hz Video
- 12 MP Stills
- Same dimensions, mounting holes, and camera center as Raspberry Pi Camera v2.1
- 4056 x 3040 pixels
- 81 DFOV°
- Lens Size: 1/2.3 inch
- AutoFocus: 8 cm - ∞
- F-number: 2.0

1.5.8 DepthAI Mono Camera

For applications where Depth + AI are needed, we have modular, high-frame-rate, excellent-depth-quality cameras which can be separated to a baseline of up to 30 cm).

Specifications

- 720p, 120 Hz Video
- Synchronized Global Shutter
- Excellent Low-light
- Same dimensions, mounting holes, and camera center as Raspberry Pi Camera v2.1
- 1280 x 720 pixels
- 83 DFOV°
- Lens Size: 1/2.3 inch
- Fixed Focus: 19.6 cm - ∞
- F-number: 2.2

1.5.9 Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

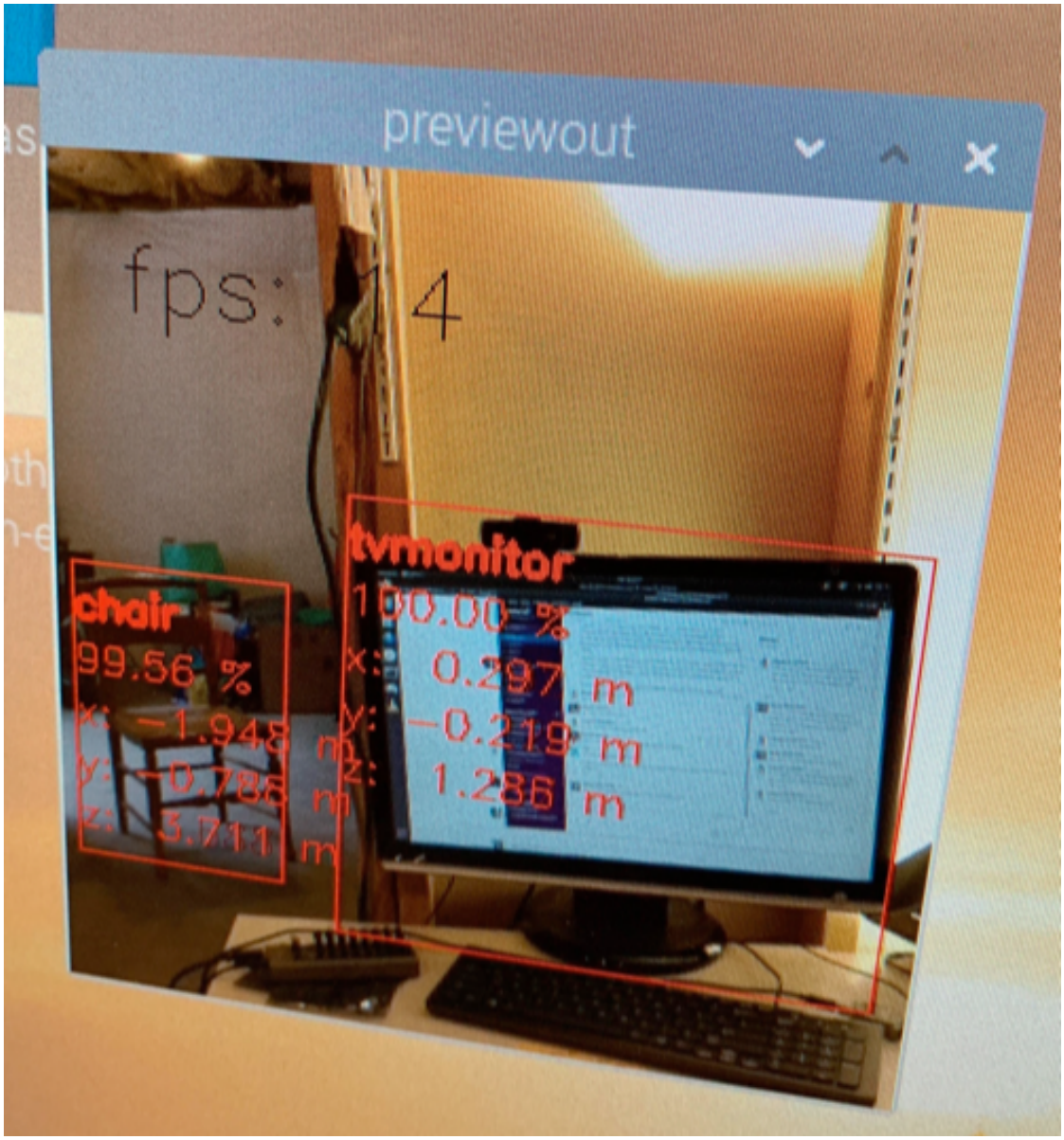
1. Start a terminal session.
2. Access your local copy of depthai.

```
cd [depthai repo]
```

1. Run demo script.

```
python3 depthai_demo.py
```

The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:



In the screenshot above, DepthAI identified a tv monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

1.5.10 Compliance and safety

All DepthAI products have undergone extensive compliance testing, and copies of the relevant certificates and conformity documents are available to download from the table below.

Table 5: Compliance download & reference table

megaAI / DepthAI BW1093 / OAK	`Declaration of conformity`

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

1.6 Calibration

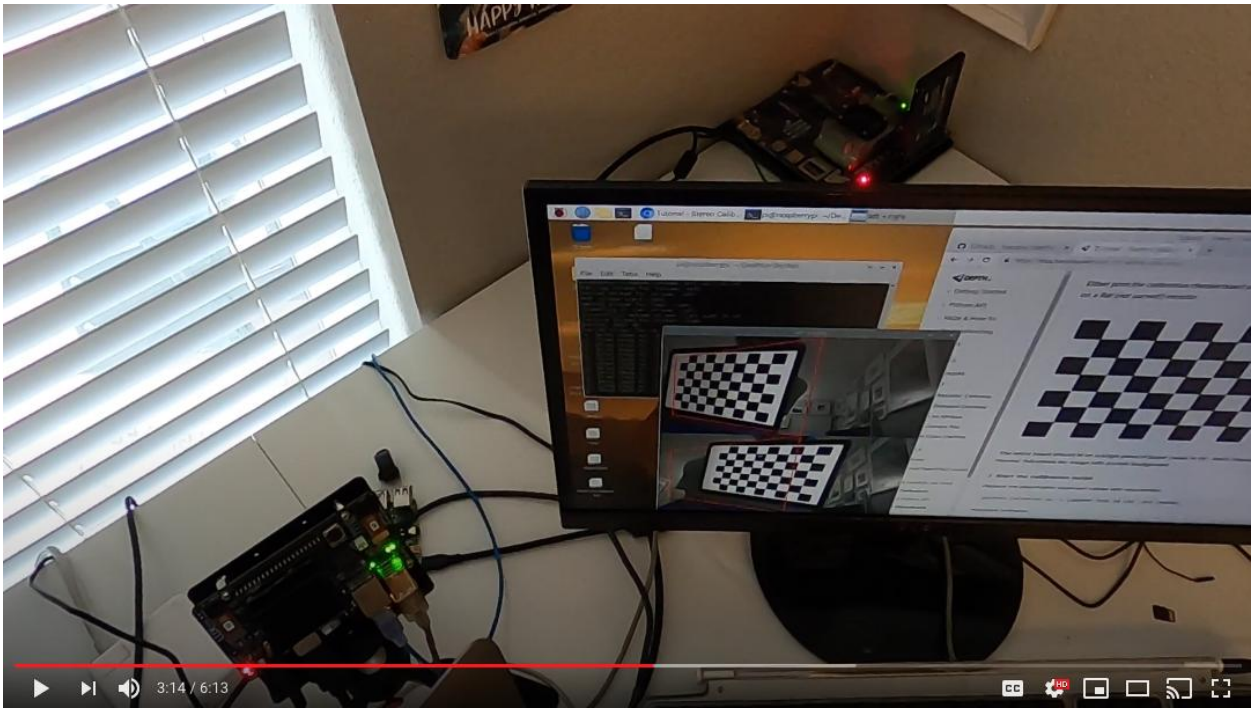
Note: Using the *BW1097 - RaspberryPi Compute Module* or BW1098OBC - USB3 with Onboard Cameras? **Your unit comes pre-calibrated**

For the modular camera editions of DepthAI (*BW1098FFC - USB3 with Modular Cameras* and *BW1094 - RaspberryPi Hat*) it is necesssary to do a stereo camera calibration after mounting the cameras in the baseline/configuration for your application.

For the *BW1097 - RaspberryPi Compute Module* and BW1098OBC - USB3 with Onboard Cameras, the units come pre-calibrated - but you may want to re-calibrate for better quality in your installation (e.g. after mounting the board to something), or if the calibration quality has started to fade over use/handling.

Below is a quick video showing the (re-) calibration of the *BW1097 - RaspberryPi Compute Module*.

Watching the video below will give you the steps needed to calibrate your own DepthAI. And 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.**

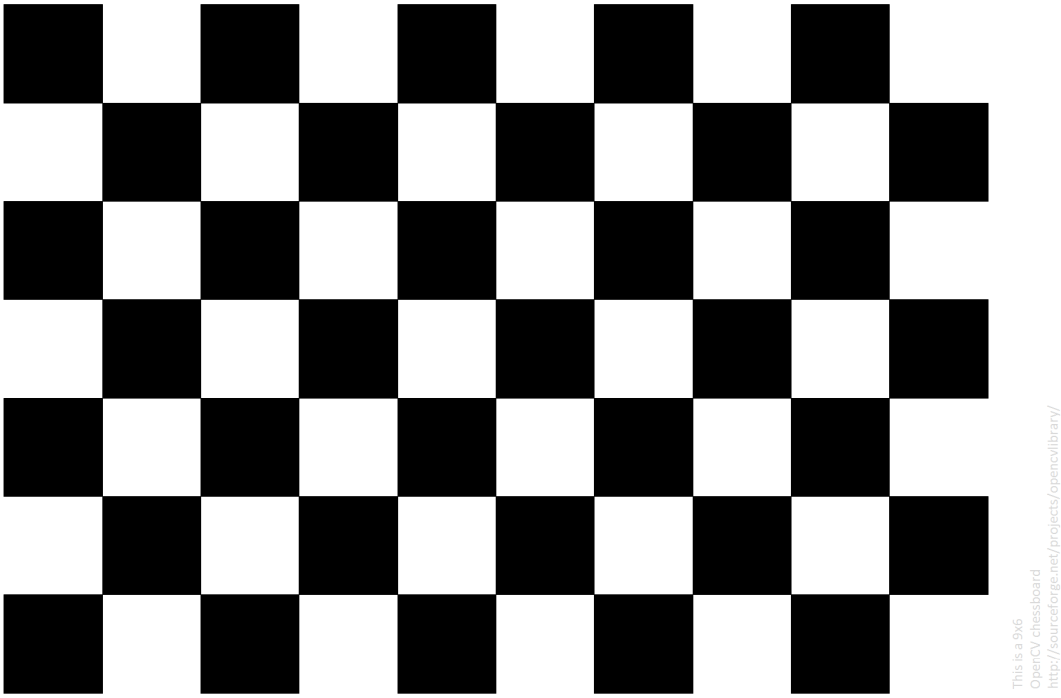
   Warning: Already checked out depthai? **Skip this step.**

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

2. Print chessboard calibration image.

Either print the calibration checkerboard onto a flat surface, or display the checkerboard 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.



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.

3. Start the calibration script.

   Replace the placeholder argument values with valid entries:

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

   Argument reference:

   - `-s SQUARE_SIZE_IN_CM`, `--square_size_cm SQUARE_SIZE_IN_CM`: Measure the square size of the printed chessboard in centimeters.
   - `-brd BOARD`, `--board BOARD`: BW1097, BW1098OBC - Board type from resources/boards/ (not case-sensitive). Or path to a custom .json board config. Mutually exclusive with [-fv -b -w], which allow manual specification of field of view, baseline, and camera orientation (swapped or not-swapped).

   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 BW1098OBC - USB3 with Onboard Cameras look like the following if the square size is 2.35 cm:

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

   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 -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:

- **BW1098OBC (USB3 Onboard Camera Edition)):**

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

- **BW1097 (RPi Compute Module Edition):**

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

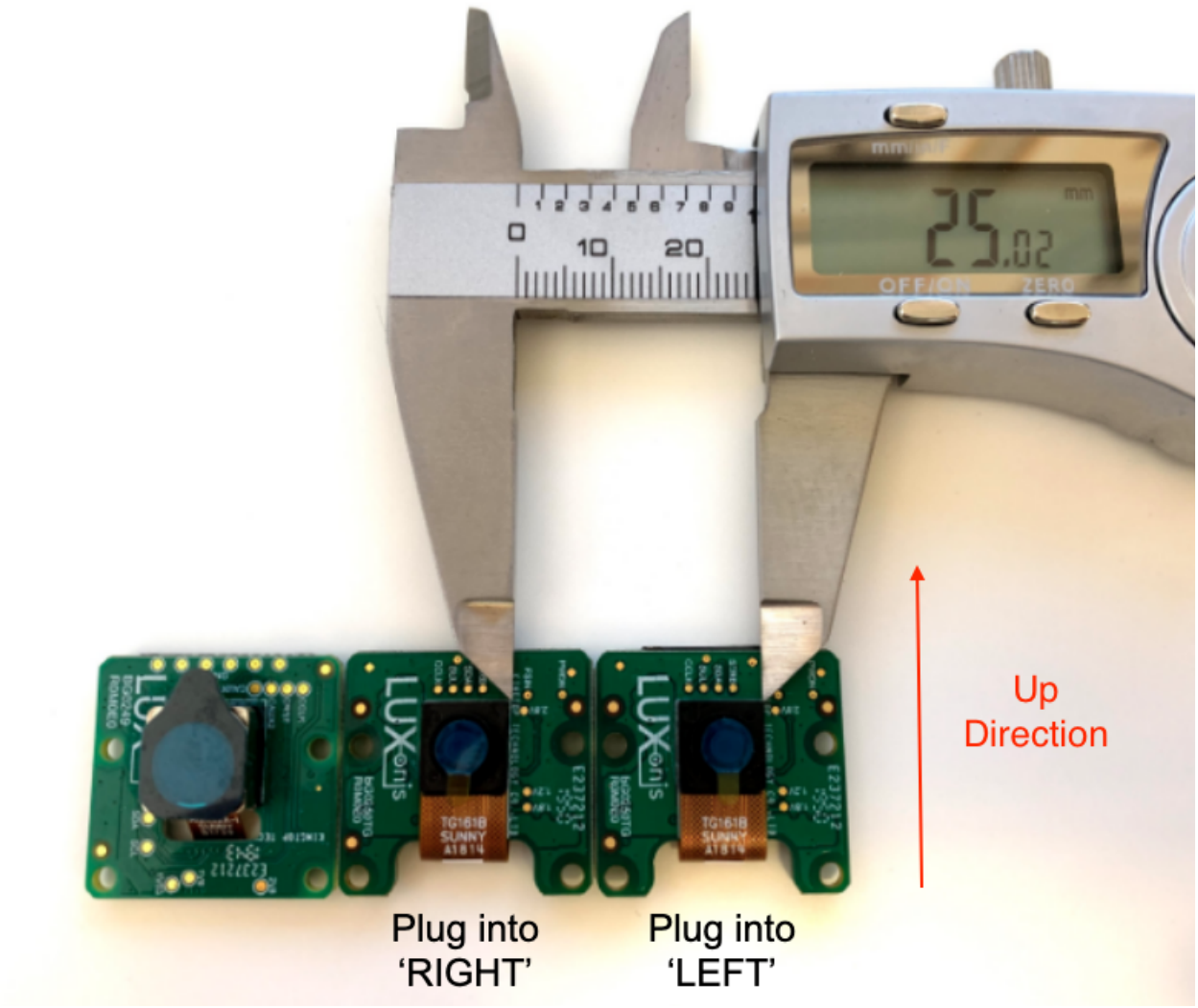
1.6.1 Modular cameras calibration

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 BW1098FFC 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:




```
{
    "board_config":
    {
        "name": "ACME01",
        "revision": "V1.2",
        "swap_left_and_right_cameras": true,
        "left_fov_deg": 73.5,
        "rgb_fov_deg": 68.7938,
        "left_to_right_distance_cm": 2.5,
        "left_to_rgb_distance_cm": 5.0
    }
}
```

Note that in this orientation of 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] -brd ACME01 -ih
```

Run `python3 calibrate.py --help` (or `-h`) for a full list of arguments and usage examples.

1.6.2 Position the chessboard and capture images.

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

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

Match the orientation of the overlayed polygon and press [SPACEBAR] to capture an image. The checkerboard 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, a calibration file will be created at `depthai/resources/depthai.calib`. This file is loaded by default via the `calib_fpath` variable within `consts/resource_paths.py`.

1.6.3 Test depth

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

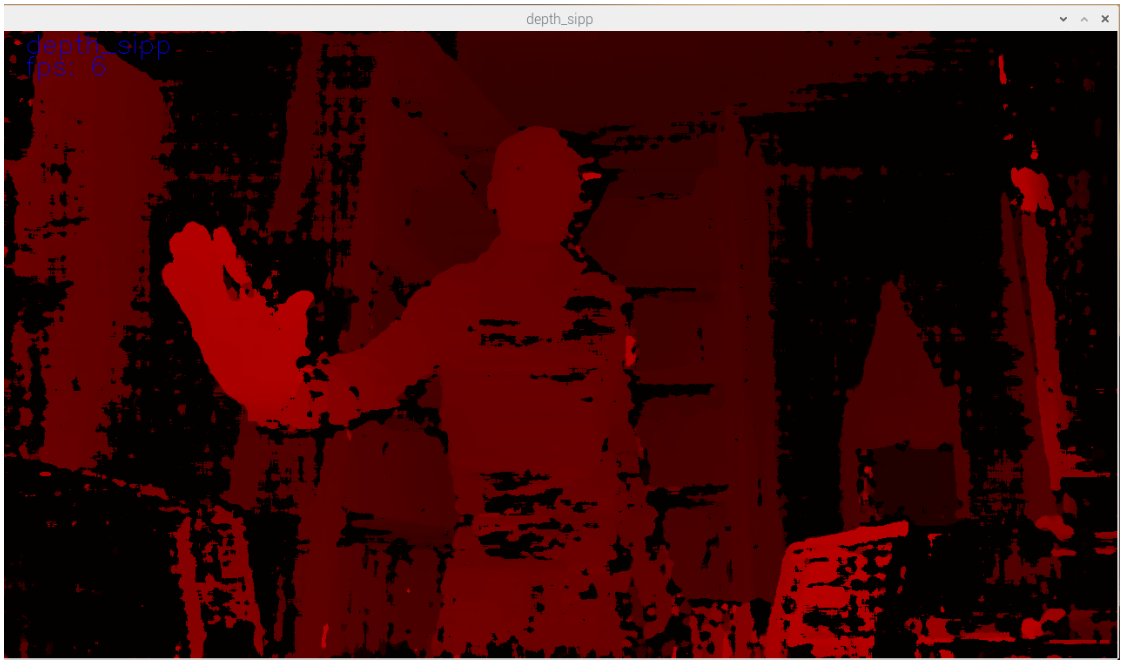
1. Start a terminal session.
2. Access your local copy of depthai.

```
cd [depthai repo]
```

3. Run test script.

```
python3 depthai_demo.py -s depth_raw -o
```

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.

1.6.4 Write calibration and board parameters to on-board eeprom

If your are happy with the depth quality above, you can write it to the on-board eeprom on DephtAI so that the calibration stick with DepthAI (all designs which have stereo-depth support have on-board eeprom for this purpose).

To write the calibration and associated board information to to EEPROM on DepthAI, use the following command:

```
python3 depthai_demo.py -brd [BOARD] -e
```

Where `[BOARD]` is either `BW1097` (Raspberry Pi Compute Module Edition), `BW1098OBC` (USB3 Onboard Camera Edition) or a custom board file (as in *here*), all case-insensitive.

So for example to write the (updated) calibration and board information to your BW1098OBC, use the following command:

```
python3 depthai_demo.py -brd bw1098obc -e
```

And to verify what is written to EEPROM on your DepthAI, you can see check the output whenever running DetphAI, simply with"

```
python3 depthai_demo.py
```

And look for `EEPROM data:` in the prints in the terminal after running the above command:

```
EEPROM data: valid (v2)
  Board name     : BW1098OBC
  Board rev      : R0M0E0
  HFOV L/R       : 73.5 deg
  HFOV RGB       : 68.7938 deg
  L-R   distance : 7.5 cm
  L-RGB distance : 3.75 cm
  L/R swapped    : yes
  L/R crop region: top
  Calibration homography:
    1.002324,   -0.004016,   -0.552212,
    0.001249,    0.993829,   -1.710247,
    0.000008,   -0.000010,    1.000000,
```

If anything looks incorrect, you can calibrate again and/or change board information and overwrite the stored eeprom information and calibration data using the `-brd` and `-e` flags as above.

1.7 Custom training

Here we have examples of Google Colaboratory (aka Colab or simply colabs) notebooks trained on various datasets. They are free GPU instances, so great for prototyping and even simple production models.

The below tutorials cover MobileNetv2-SSD, tiny-YOLOv3, tiny-YOLOv4, and Deeplabv3+ (semantic segmentation). A bunch of other object detectors and neural networks could be trained/supported on Colab and run on DepthAI, so if you have a request for a different object detector/network backend, please feel free to make a Github Issue!

And please feel free to work directly from our Github of depthai-ml-training for the latest models we support:

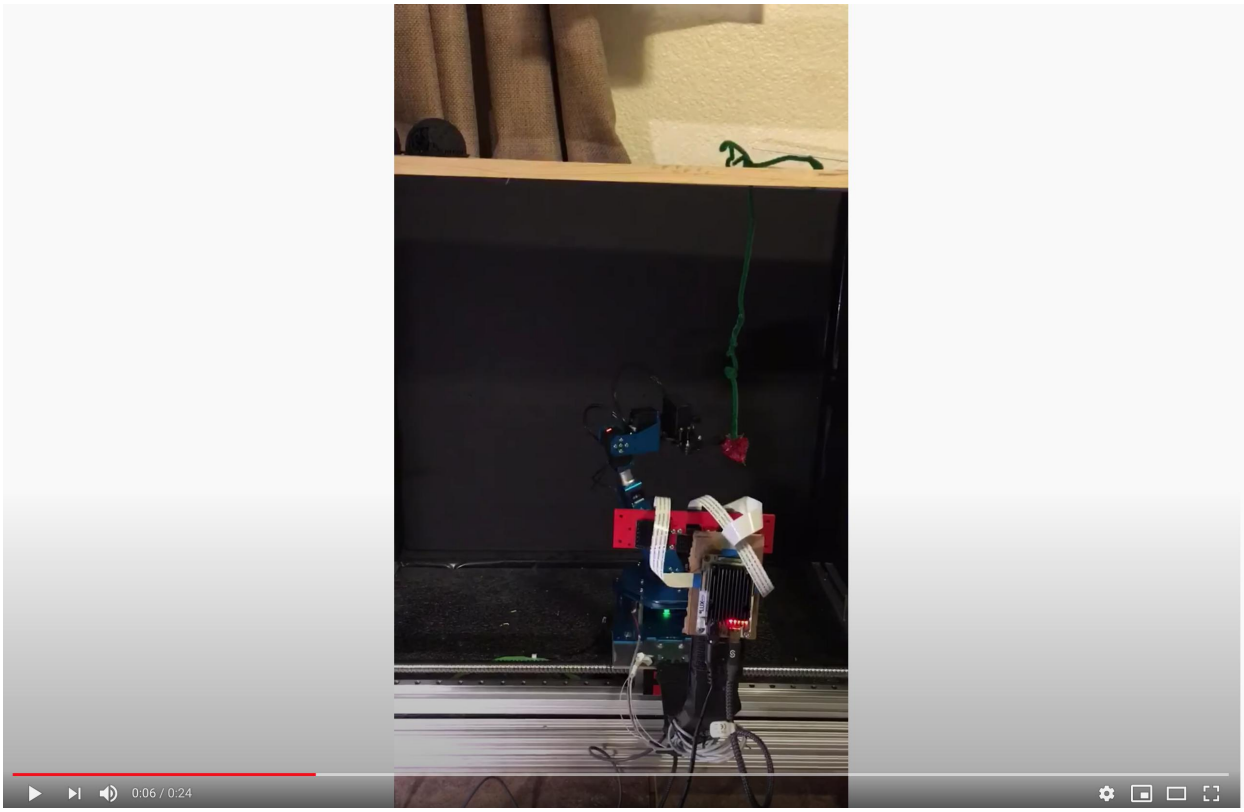
- depthai-ml-training

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

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

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

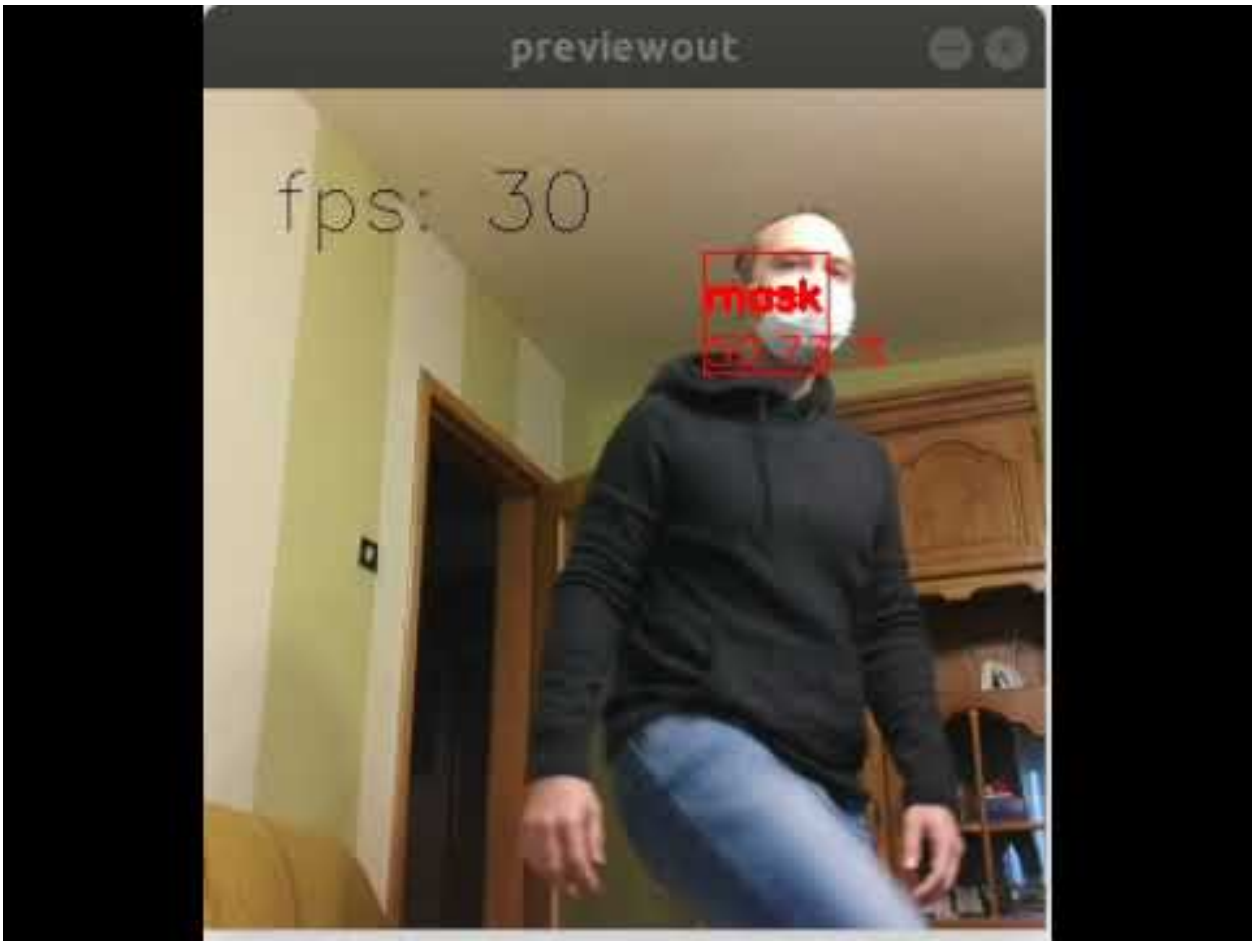
And that's it, in less than a couple of hours a fairly advanced proof of concept object detector can run on DepthAI to detect objects of your choice and their associated spatial information (i.e. xyz location). For example this notebook was used to train DepthAI to locate strawberries in 3D space, see 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 dataset 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 Luxonis DepthAI Onboard Cameras Edition (BW1098OBC):



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.

1.8 Hello World

Learn how to use the DepthAI Python API to display a color video stream.

1.8.1 Dependencies

Let's get your development environment setup first. This tutorial uses:

- Python 3.6 (Ubuntu) or Python 3.7 (Raspbian).
- The DepthAI *Python API*
- The `cv2` and `numpy` Python modules.

1.8.2 Code Overview

The `depthai` Python module provides access to your board's 4K 60 Hz color camera. We'll display a video stream from this camera to your desktop. You can find the complete source code for this tutorial on GitHub.

1.8.3 File Setup

Setup the following file structure on your computer:

```
cd ~
mkdir -p depthai-tutorials-practice/1-hello-world
touch depthai-tutorials-practice/1-hello-world/hello_world.py
cd depthai-tutorials-practice/1-hello-world
```

What's with the `-practice` suffix in parent directory name? Our tutorials are available on GitHub via the depthai-tutorials repository. We're appending `-practice` so you can distinguish between your work and our finished tutorials (should you choose to download those).

1.8.4 Install pip dependencies

To display the DepthAI color video stream we need to import a small number of packages. Download and install the requirements for this tutorial:

```
python3 -m pip install numpy opencv-python depthai --user
```

1.8.5 Test your environment

Let's verify we're able to load all of our dependencies. Open the `hello_world.py` file you *created earlier* in your code editor. Copy and paste the following into `hello_world.py`:

```
import numpy as np # numpy - manipulate the packet data returned by depthai
import cv2 # opencv - display the video stream
import depthai # access the camera and its data packets
```

Try running the script and ensure it executes without error:

```
python3 hello_world.py
```

If you see the following error:

```
ModuleNotFoundError: No module named 'depthai'
```

. . . follow *these steps in our troubleshooting section*.

1.8.6 Initialize the DepthAI Device

Start the DepthAI device:

```
device = depthai.Device('', False)
```

Try running the script. You should see output similar to:

```
No calibration file. Using Calibration Defaults.
XLink initialized.
Sending device firmware "cmd_file": /home/pi/Desktop/depthai/depthai.cmd
Successfully connected to device.
Loading config file
Attempting to open stream config_d2h
watchdog started 6000
Successfully opened stream config_d2h with ID #0!
```

If instead you see an error, please reset your DepthAI device, then try again.

1.8.7 Create the DepthAI Pipeline

Now we'll create our data pipeline using the `previewout` stream. This stream contains the data from the color camera. The model used in `ai` section is a MobileNetSSD with 20 different classes, see here for details

```
# Create the pipeline using the 'previewout' stream, establishing the first␣
↪connection to the device.
pipeline = device.create_pipeline(config={
    'streams': ['previewout', 'metaout'],
    'ai': {
        'blob_file': "/path/to/mobilenet-ssd.blob",
        'blob_file_config': "/path/to/mobilenet-ssd.json"
    }
})

if pipeline is None:
    raise RuntimeError('Pipeline creation failed!')
```

1.8.8 Display the video stream

A DepthAI Pipeline generates a stream of data packets. Each `previewout` data packet contains a 3D array representing an image frame. We change the shape of the frame into a `cv2`-compatible format and display it.

```
detections = []

while True:
    # Retrieve data packets from the device.
    # A data packet contains the video frame data.
    nnet_packets, data_packets = pipeline.get_available_nnet_and_data_packets()

    for nnet_packet in nnet_packets:
        detections = list(nnet_packet.getDetectedObjects())

    for packet in data_packets:
        # By default, DepthAI adds other streams (notably 'meta_2dh'). Only process␣
↪`previewout`.
```

(continues on next page)

(continued from previous page)

```
        if packet.stream_name == 'previewout':
            data = packet.getData()
            # the format of previewout image is CHW (Channel, Height, Width), but␣
↪OpenCV needs HWC, so we
            # change shape (3, 300, 300) -> (300, 300, 3)
            data0 = data[0,:,:]
            data1 = data[1,:,:]
            data2 = data[2,:,:]
            frame = cv2.merge([data0, data1, data2])

            img_h = frame.shape[0]
            img_w = frame.shape[1]

            for detection in detections:
                pt1 = int(detection.x_min * img_w), int(detection.y_min * img_h)
                pt2 = int(detection.x_max * img_w), int(detection.y_max * img_h)

            cv2.imshow('previewout', frame)

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

# The pipeline object should be deleted after exiting the loop. Otherwise device will␣
↪continue working.
# This is required if you are going to add code after exiting the loop.
del pipeline
del device
```

Run the script. Press the `Q` key with focus on the video stream (not your terminal) to exit:

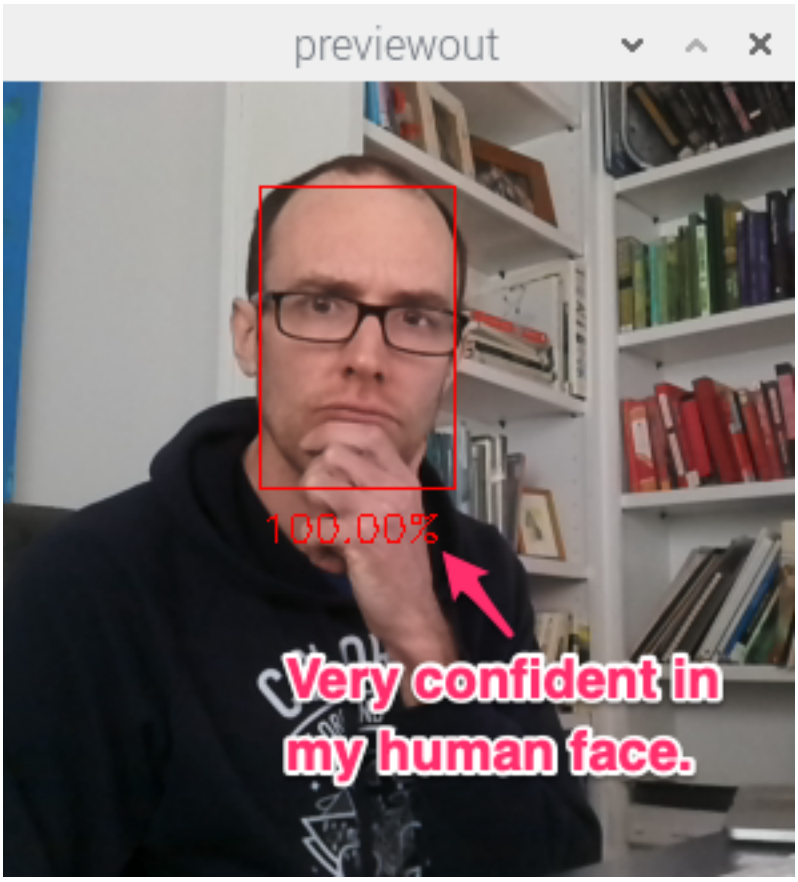
```
python3 hello_world.py
```

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

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

1.9 Use a Pre-trained OpenVINO model

In this tutorial, you'll learn how to detect faces in realtime, even on a low-powered Raspberry Pi. I'll introduce you to the OpenVINO model zoo and running models from this 'zoo'.



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

1.9.1 What is OpenVINO?

Under-the-hood, DepthAI uses the Intel MyriadX chip 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!

1.9.2 What is the Open Model Zoo?

The Open Model Zoo is a library of freely-available pre-trained models. Side note: in machine learning/AI the name for a collection of pre-trained models is called a 'model zoo'. The 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 in the Zoo, and several are pre-included in the DepthAI Github. repository. We will be using one such model in this tutorial, is face-detection-retail-0004 (pre-compiled here on our Github, and here on the OpenVINO model zoo).

We'll cover converting OpenVINO models to run on DepthAI in a later article. For now, you can find the models we've pre-converted here and brief instructions on how to do so here.

1.9.3 Dependencies

> **Warning:** Using the RPi Compute Edition or a pre-flashed DepthAI Raspberry Pi µSD card? **Skip this step**
>
> All dependencies are installed and the repository is checked out to `~/Desktop/depthai`.

This tutorial has the same dependencies as the *Hello World Tutorial* - that the DepthAI API has been installed and is accessible on the system. See *here* if you have not yet installed the API.

1.9.4 Run DepthAI Default Model

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

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

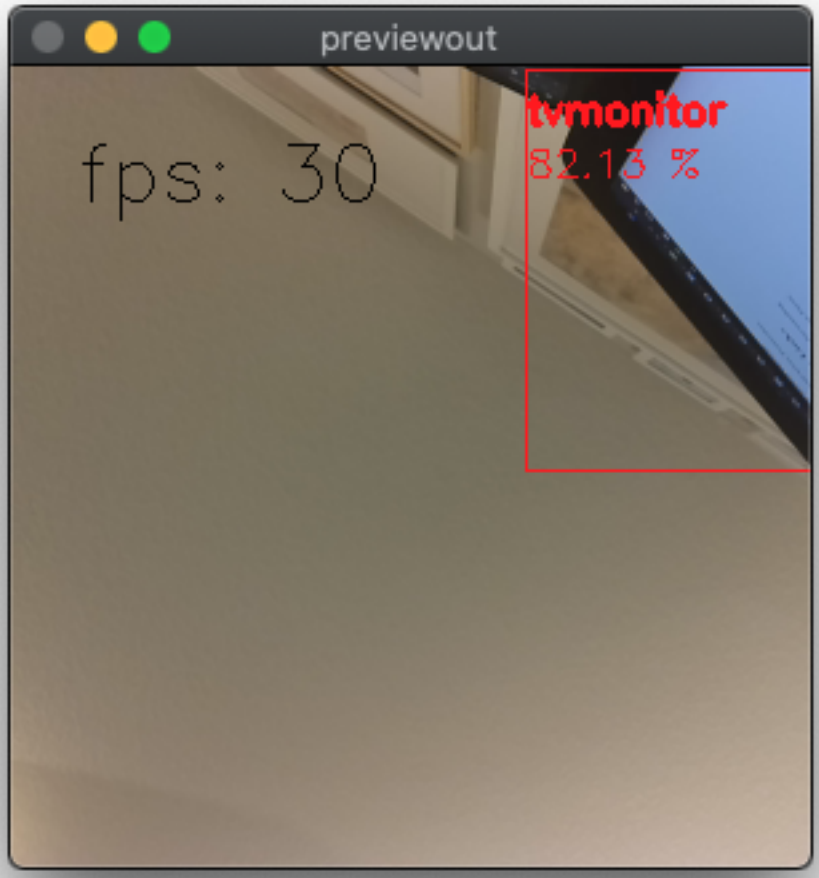
Before switching to using the `face-detection-retail-0004` let's take a baby step and give these command line options a spin. In this case we'll just pass in the same neural network that default runs when running `python3 depthai_demo.py`, just to make sure we're doing it right:

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

This will then run the a typical demo MobileNetv1 SSD object detector trained on the PASCAL 2007 VOC classes, which are:

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

I ran this on my iMac (OS X setup *here*) with a microAI sitting on my desk pointing upwards randomly - and it makes out the corner of my iMac (which is barely visible) and correctly identifies it as *tv/monitor*:



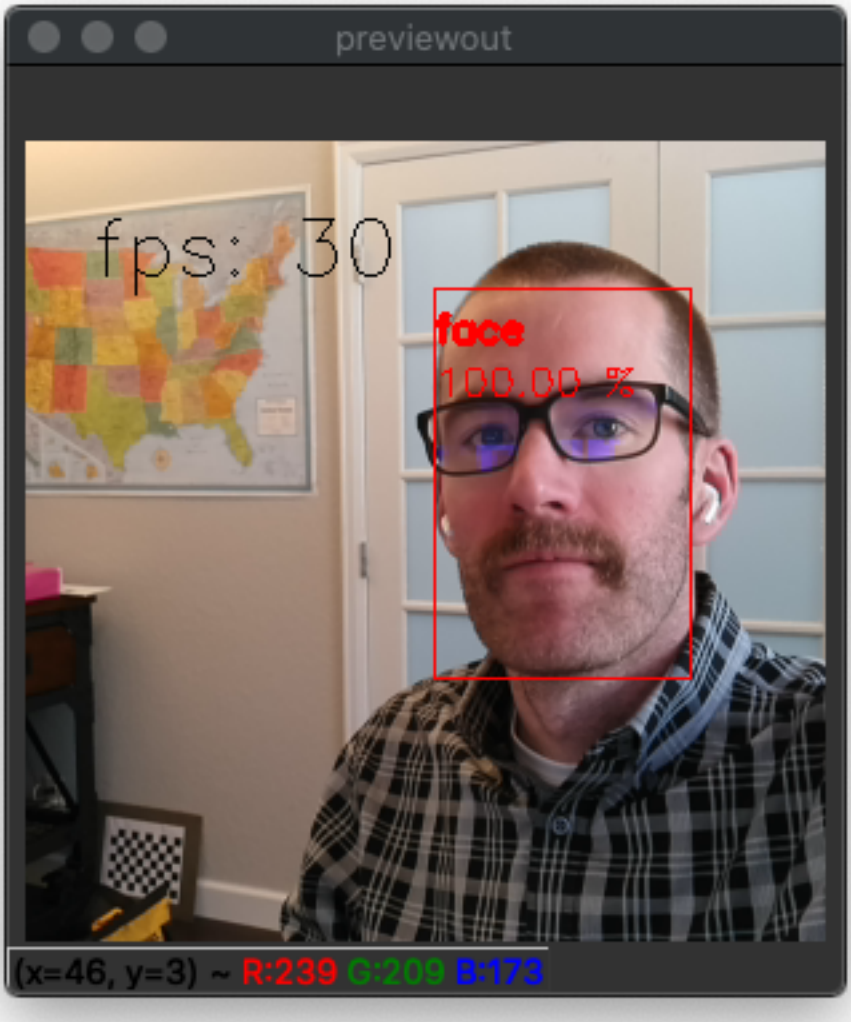
1.9.5 Run model

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
```

Execute the script to see an annotated video stream of face detections:



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:


previewout
fps: 30
face
100.00 %
(x=247, y=120) ~ R:150 G:138 B:128

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.

1.9.6 Trying Other Models

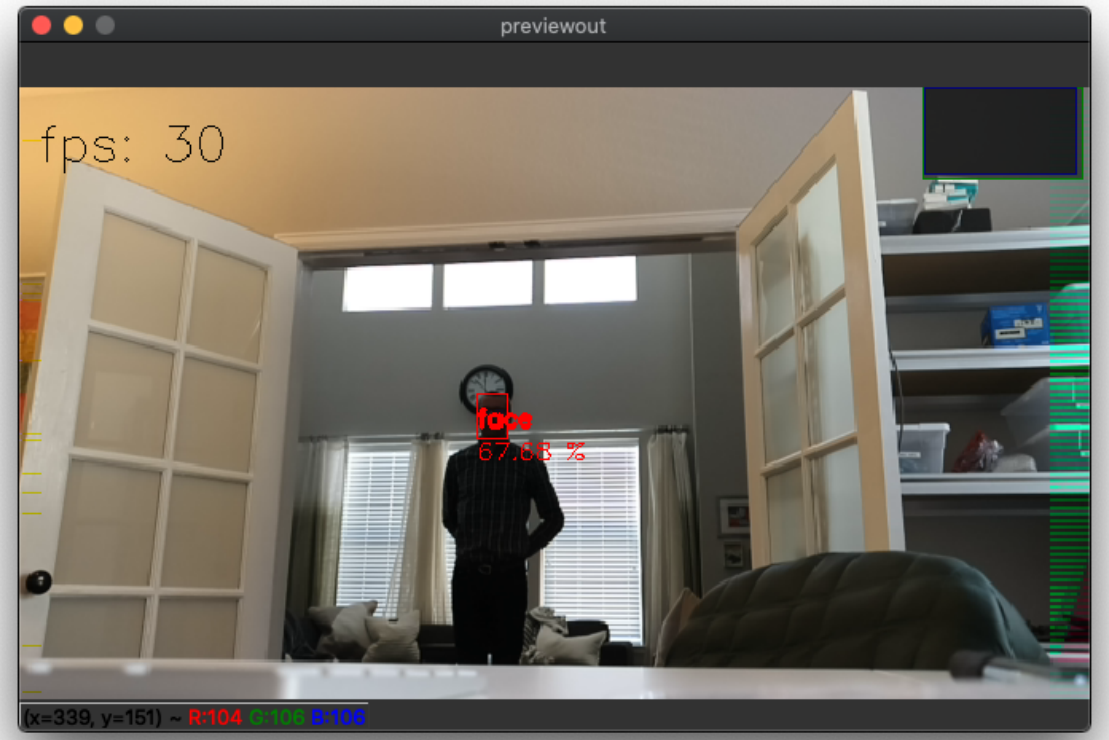
The flow we walked through works for other pre-trained object detection models in our repository (here), which includes:

- face detection for retail (`face-detection-retail-0004`)
- face detection for driver assistance (`face-detection-adas-0001`)
- facial landmarks, simple (`landmarks-regression-retail-0009`)
- facial landmarks, advanced (`facial-landmarks-35-adas-0002`)
- emotions recognition (`emotions-recognition-retail-0003`)
- pedestrian detection for driver-assistance (`pedestrian-detection-adas-0002`)
- person detection for retail environments (`person-detection-retail-0013`)
- vehicle detection for driver-assistance (`vehicle-detection-adas-0002`)
- vehicle and license plate detection (`vehicle-license-plate-detection-barrier-0106`)

Simply change the paths above to run the other models there, adding the correct labels (or funny ones, should you choose).

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.)

And also you may notice networks like emotion recognition... those networks are actually intended to be run as a second stage network (as they are meant to be applied only to images that contain only faces). So to use the emotions recognitions network, use the command below to tell DepthAI/megaAI to run it as the second stage:

```
./depthai_demo.py -cnn face-detection-retail-0004 -cnn2 emotions-recognition-retail-
↪0003 -dd -sh 12 -cmx 12 -nce 2
```

And what is this `-dd` option we've been running? Why is that there?

It's there because we wanted to save the best for last. It stands for disable depth (and has the long-form option `--disable_depth`). So if you remove that, DepthAI will now calculate the 3D position of the object being detected (a face in this example, but it works for any object detector.) (And if you're using microAI, leave it there, as microAI is monocular only - no depth information.)

So you get the **full 3D position** of the **detected object**, in this case, my face.

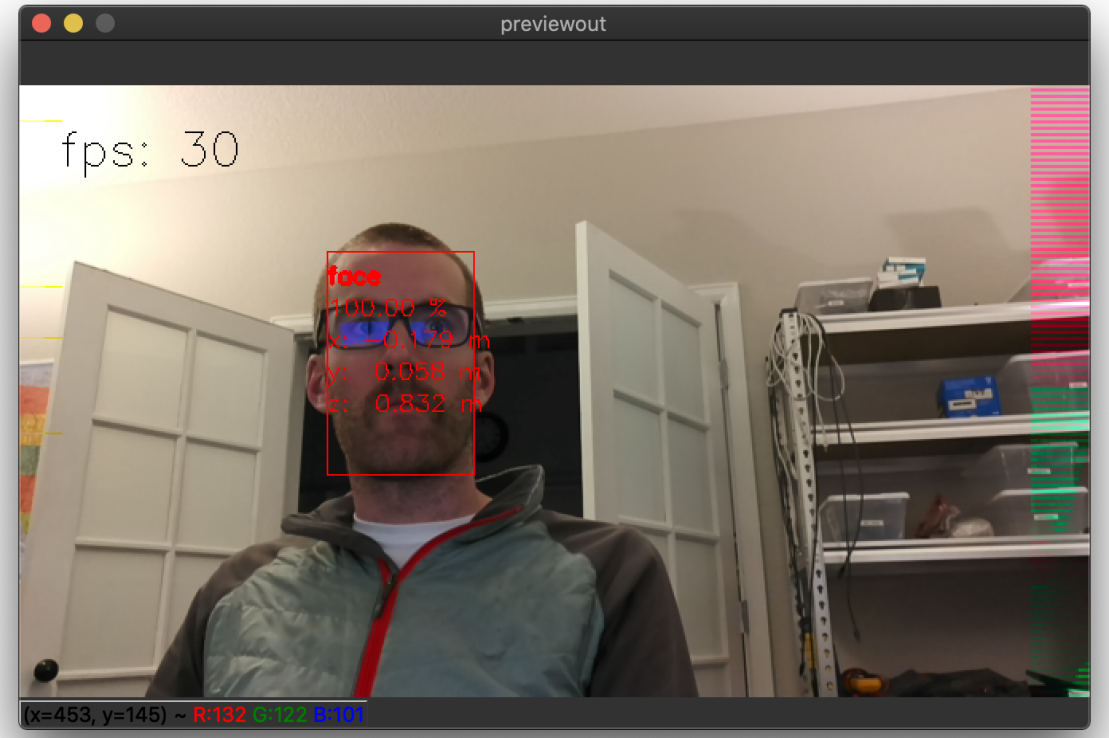
So that the full xyz position in meters is returned. See below.

1.9.7 Spatial AI - Augmenting the Model with 3D Postion

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 `--disable_depth`).

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

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



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

You can than choose other models, change the labels, and you're off - getting 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 Github.

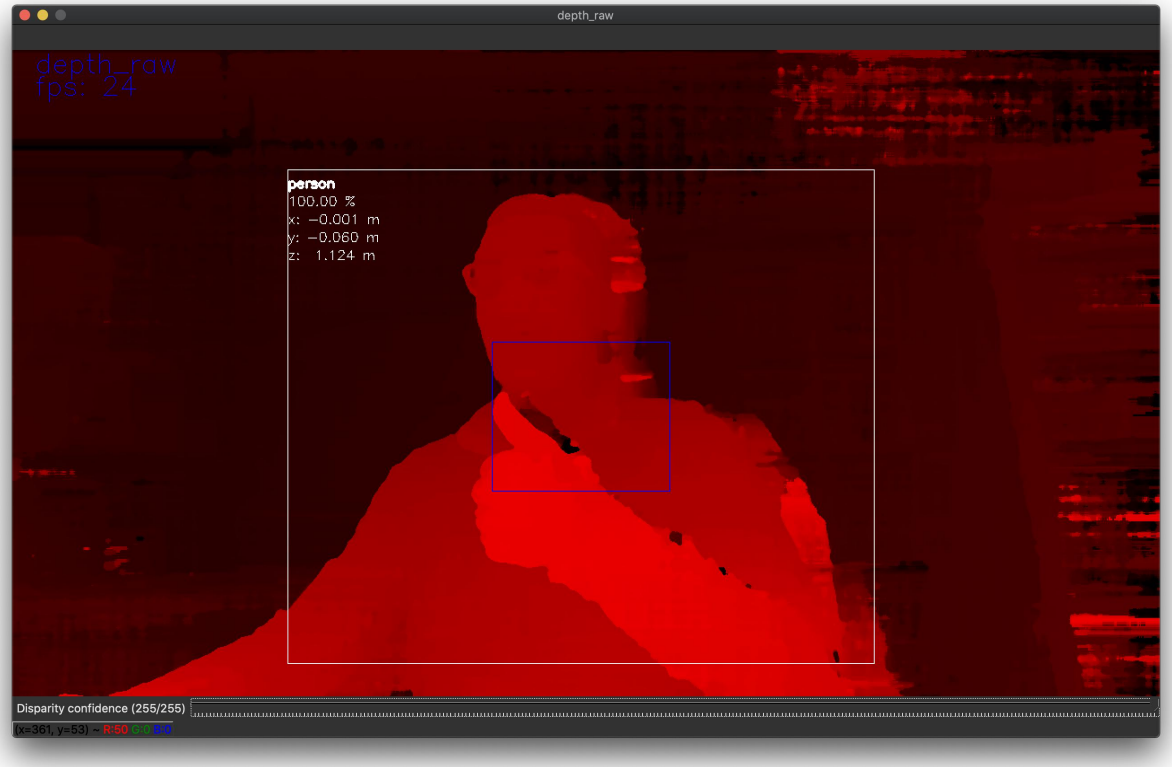
And if you find any errors in these documents, please report an issue on the docs github) on the bottom of this page to give us the correction!

Monocular Neural Inference fused with Stereo Depth

We call this mode of spatial AI 'Monocular Neural Inference fused with Stereo Depth'. To visualize how this mode works, it is helpful to overlay the neural inference bounding box over the depth results directly.

To visualize this, let's overlay the results directly onto the raw depth information (visualized in OpenCV HOT colormap):

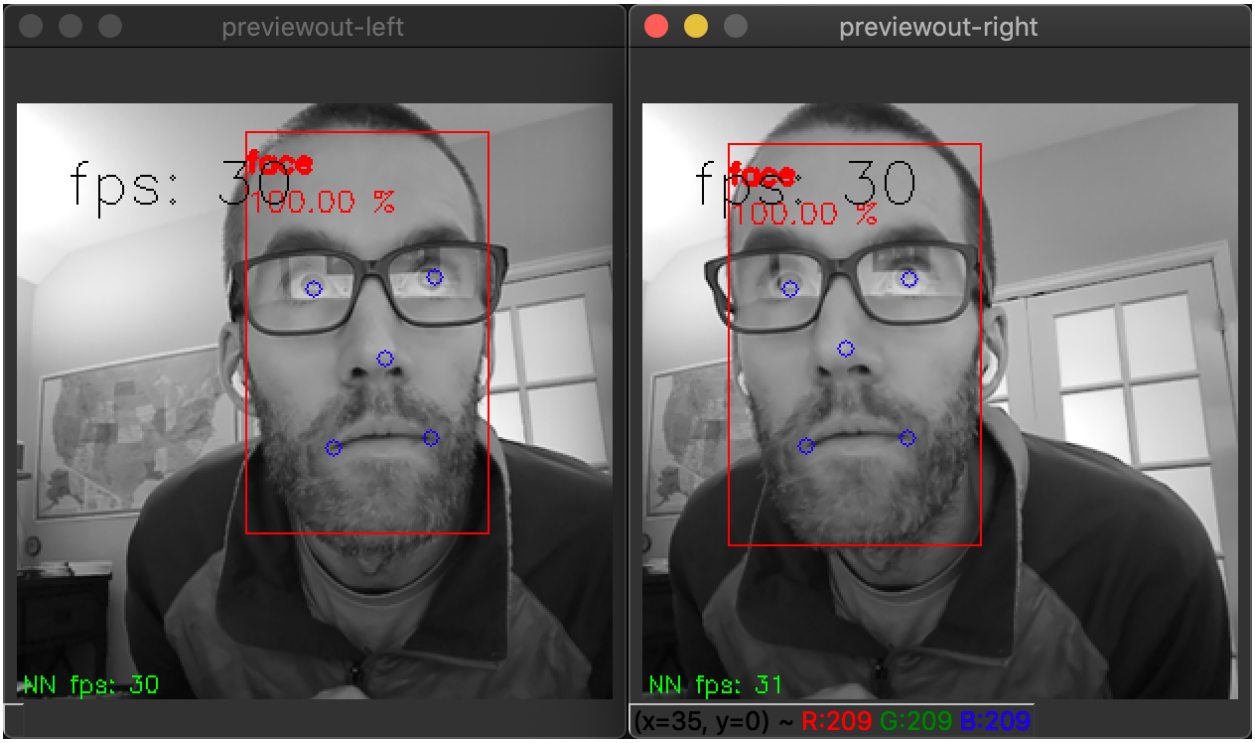
```
python3 depthai_demo.py -s metaout depth_raw -bb
```



So this 'monocular neural inference fused with stereo disparity depth' technique works well for objects, particularly bigger objects (like people, faces, etc.).

Stereo Neural Inference

Below we'll use another technique, which we dub 'stereo neural inference' (or 'Stereo AI') which works well for smaller objects and also pixel-point features like facial landmarks and pose-estimator results, etc.



This can be run with the following command:

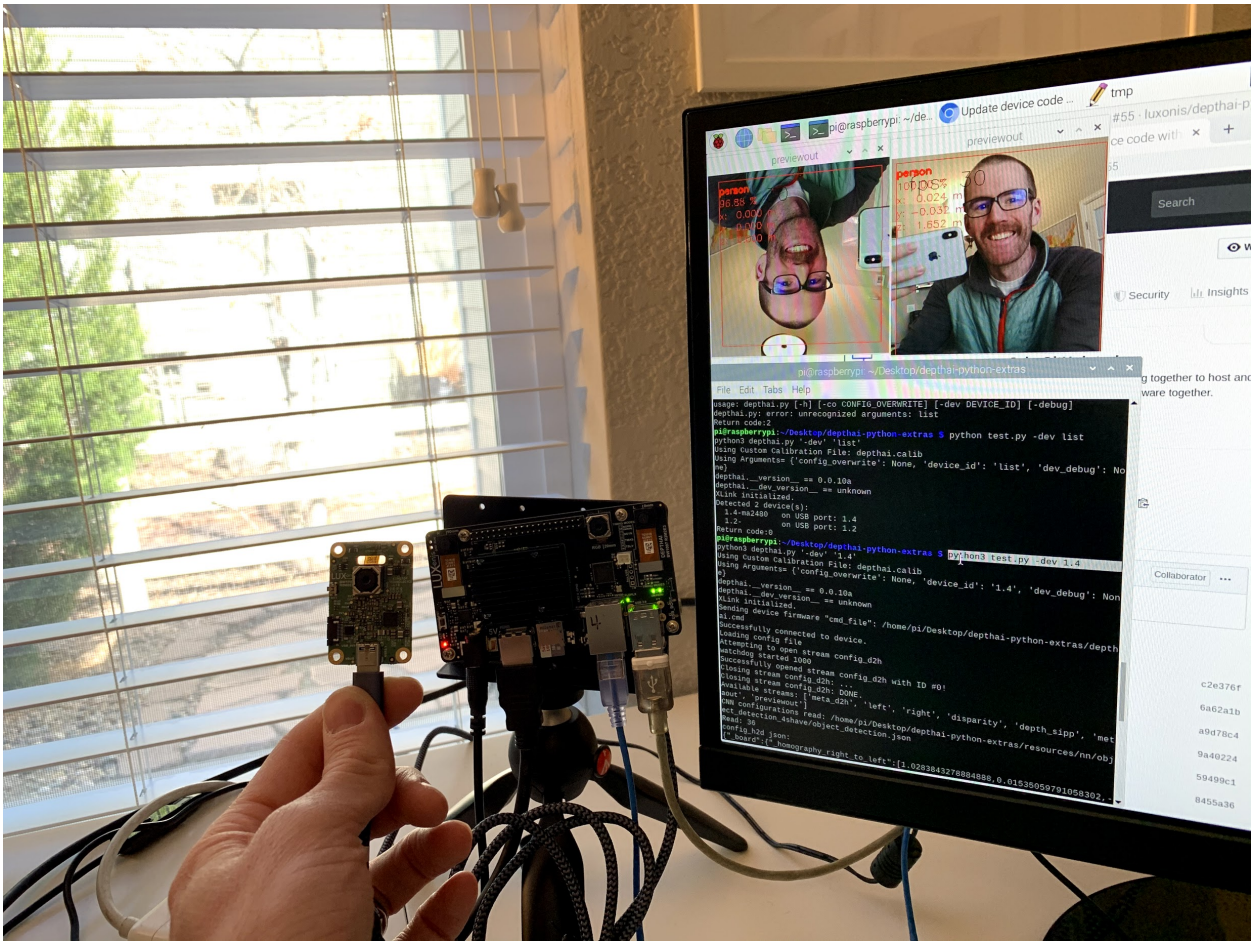
```
./depthai_demo.py -cnn face-detection-retail-0004 -cnn2 landmarks-regression-retail-
↪0009 -cam left_right -dd -sh 12 -cmx 12 -nce 2 -monor 400 -monof 30
```

And note this is running both parallel neural inference (i.e. on both cameras) and also series neural inference (the landmarks-regression network is running on the results of the face detector).

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

1.10 Multiple DepthAI per Host

Learn how to use the DepthAI `-dev` option to discover the DepthAI connected to your system, and use them individually.



Shown on the left is Luxonis uAI (BW1093) which is actually plugged into a Raspberry Pi Compute Module Edition (BW1097).

So in this case, everything is running on the (single) Raspberry Pi 3B+ which is in the back of the BW1097.

1.10.1 Dependencies

You have already set up the Python API on your system (if you have a Raspberry Pi Compute Module it came pre-setup). See *here* if you have not yet installed the DepthAI Python API on your system.

1.10.2 Discover DepthAI-USB Port Mapping

The DepthAI multi-device support is currently done by selecting the USB port into which the DepthAI is plugged in.

If you'd like to associate a given DepthAI device with specific code (e.g. neural model) to be run on it, it is recommended to plug in one device at a time, and then use the following command to determine which device is on which port:

```
python3 depthai_demo.py -dev list
```

Example results for 2x DepthAI on a system:

```
...
XLink initialized.
Detected 2 device(s):
  2-ma2480     on USB port: 1
  1.1-ma2480   on USB port: 2.1
```

1.10.3 Selecting a Specific DepthAI device to be used.

From the Detected devices(s) above, use the following command to select the device you would like to use with your code. For example, if the first device is desirable from above (the device on USB port 1), use the following command:

```
python3 depthai_demo.py -dev 1
```

And then similarly, to run the same script again on the other second device, run it with:

```
python3 depthai_demo.py -dev 2.1
```

And you can use these scripts as a basis for your own modified versions, such that you can run differing neural models on different DepthAI/uAI models.

1.10.4 Summary and Overview of Host-Side Burden

Now use as many DepthAI devices as you need!

And since DepthAI does all the heavy lifting, you can usually use quite a few of them with very little burden to the host.

And it's worth noting that you can always disable the video stream by only only requesting `metaout` here.

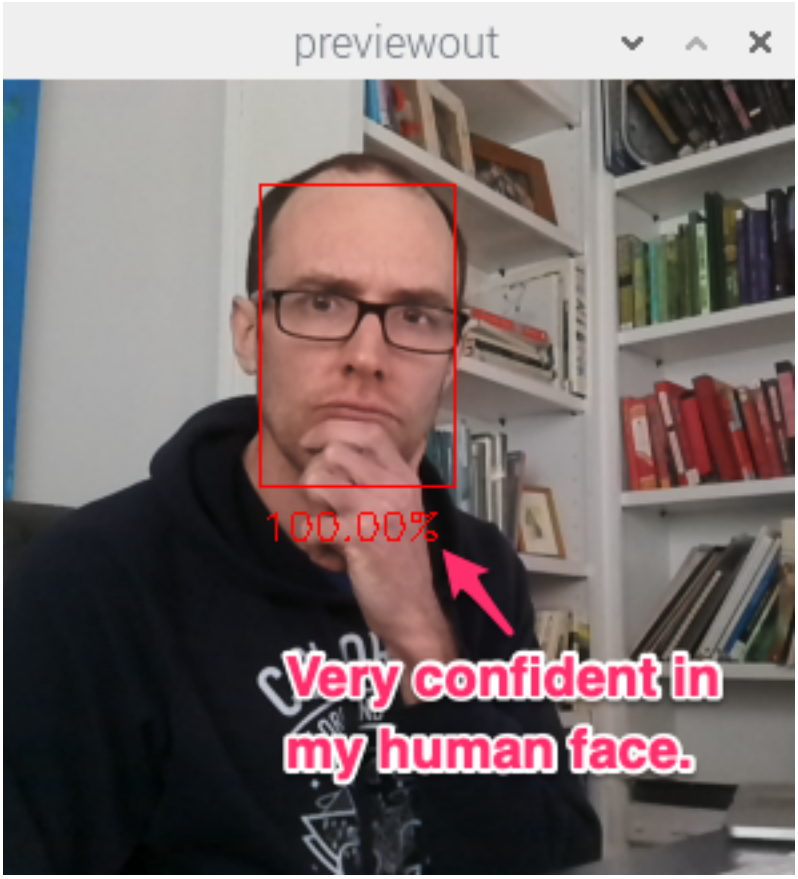
So if you're using the metadata to say, drive a robot or make decisions with code, and the video isn't needed, you can do this to substantially reduce the burden on the host - as since all the neural inference work is done on DepthAI before getting to the host - almost all the host burden is just from displaying the video.

So with the video disabled, the host only has to handle a couple kilobytes a second in terms of metadata.

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

1.11 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 files needed to run model inference on your DepthAI board.



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

1.11.1 What is OpenVINO?

Under-the-hood, DepthAI uses the Intel tecnnology 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!

1.11.2 What is the Open Model Zoo?

The Open Model Zoo is a library of freely-available pre-trained models. The 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 in the Zoo.

1.11.3 Install OpenVINO

> **Warning:** If you have OpenVINO installed or want to follow official installation, **skip this step**.
>
> Please note that the following install instructions are for **Ubuntu 18.04** OS, if you intend to use other OS, follow the official OpenVINO installation

DepthAI requires OpenVINO version `2020.1`. Let's get a package for our OS and meeting this version with the following command:

```
apt-get update
apt-get install -y software-properties-common
add-apt-repository -y ppa:deadsnakes/ppa
apt-get update
apt-get install -y wget pciutils python3.8 libpng-dev libcairo2-dev libpango1.0-dev␣
↪libglib2.0-dev libgtk2.0-dev libswscale-dev libavcodec-dev libavformat-dev
cd
mkdir openvino_install && cd openvino_install
wget http://registrationcenter-download.intel.com/akdlm/irc_nas/16345/l_openvino_
↪toolkit_p_2020.1.023.tgz
tar --strip-components=1 -zxvf l_openvino_toolkit_p_2020.1.023.tgz
./install_openvino_dependencies.sh
./install.sh # when finished, you can go ahead and do "rm -r ~/openvino_install"
```

Now, first screen we'll wee is EULA, just hit `Enter`, scroll through and type `accept`.

Next one is agreement to Intel Software Improvement Program, it's not relevant so you can choose whether consent (`1`) or not (`2`)

Next, you may see the Missing Prerequisites screen showing that `Intel® Graphics Compute Runtime for OpenCL™ Driver is missing` - you can go ahead and ignore this warning.

Finally, we'll see the install summary - please verify that it has a correct location pointed out - `/opt/intel`. If all looks good, go ahead and proceed (`1`). If the missing prerequisites screen appears again, feel free to skip it.

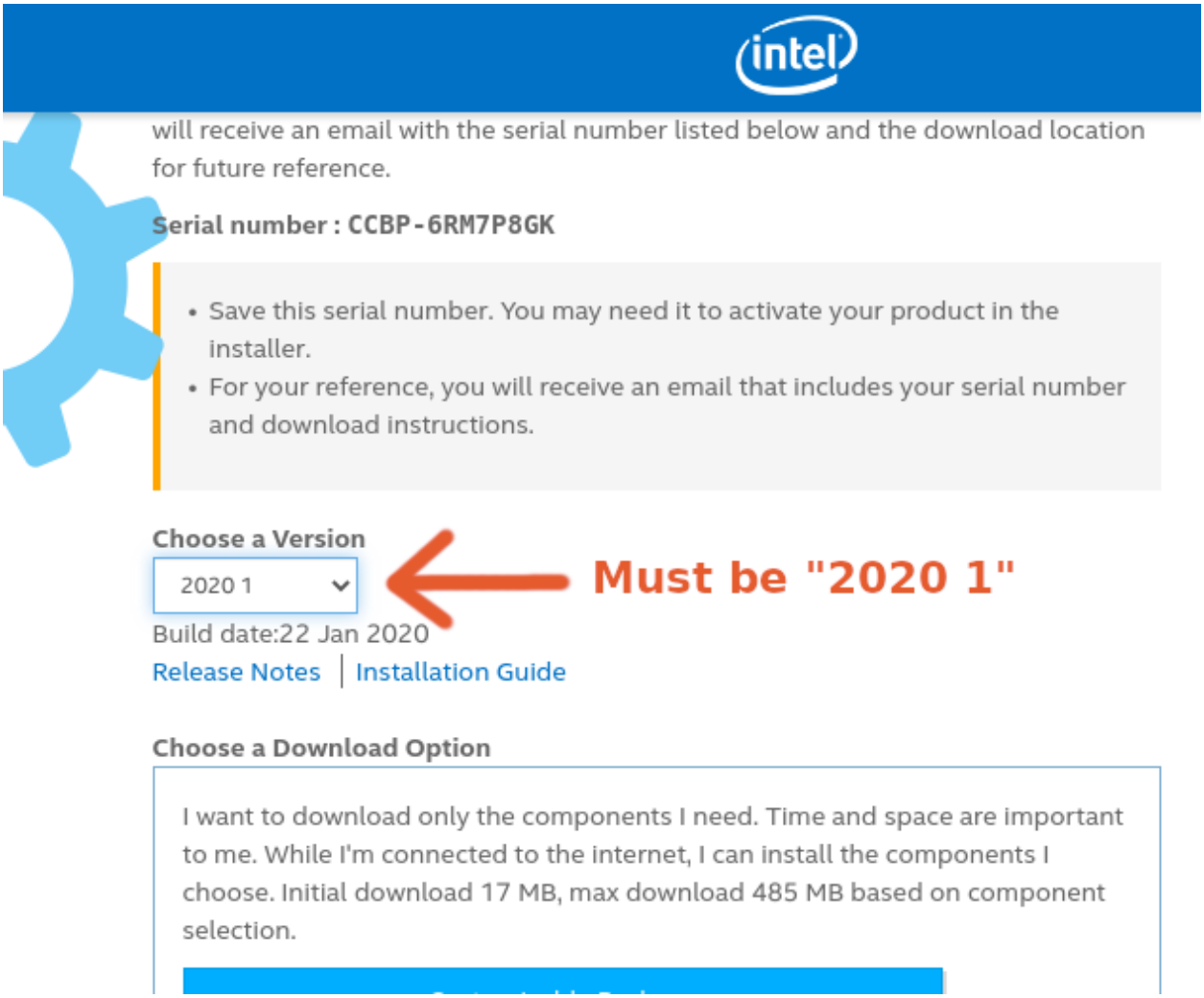
Let's verify that a correct version is installed on your host. Check your version by running the following from a terminal session:

```
cat /opt/intel/openvino/inference_engine/version.txt
```

You should see output similar to:

```
Thu Jan 23 19:14:14 MSK 2020
d349c3ba4a2508be72f413fa4dee92cc0e4bc0e1
releases_2020_1_InferenceEngine_37988
```

Verify that you see `releases_2020_1` in your output. If you do, move on. If you are on a different version, goto the OpenVINO site and download the `2020.1` version for your OS:



1.11.4 Check if the Model Downloader is installed

When installing OpenVINO, you can choose to perform a smaller install to save disk space. This custom install may not include the model downloader script. Lets check if the downloader was installed. In a terminal session, type the following:

```
find /opt/intel/ -iname downloader.py
```

Move on if you see the output below:

```
/opt/intel/openvino_2020.1.023/deployment_tools/open_model_zoo/tools/downloader/
↪downloader.py
```

Didn't see any output? Don't fret if `downloader.py` isn't found. We'll install this below.

Install Open Model Zoo Downloader

If the downloader tools weren't found, we'll install the tools by cloning the Open Model Zoo Repo and installing the tool dependencies.

Start a terminal session and run the following commands in your terminal:

```
apt-get install -y git curl
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
rm get-pip.py
cd ~
git clone https://github.com/opencv/open_model_zoo.git
cd open_model_zoo
git checkout tags/2020.1
cd tools/downloader
python3 -m pip install --user -r ./requirements.in
```

This clones the repo into a `~/open_model_zoo` directory, checks out the required `2020.1` version, and installs the downloader dependencies.

1.11.5 Create an OPEN_MODEL_DOWNLOADER environment variable

Typing the full path to `downloader.py` can use a lot of keystrokes. In an effort to extend your keyboard life, let's store the path to this script in an environment variable.

Run the following in your terminal:

```
export OPEN_MODEL_DOWNLOADER='INSERT PATH TO YOUR downloader.py SCRIPT'
```

Where `INSERT PATH TO YOUR downloader.py SCRIPT` can be found via:

```
find /opt/intel/ -iname downloader.py
find ~ -iname downloader.py
```

For example, if you installed `open_model_zoo` yourself:

```
export OPEN_MODEL_DOWNLOADER="$HOME/open_model_zoo/tools/downloader/downloader.py"
```

1.11.6 Download the face-detection-retail-0004 model

We've installed everything we need to download models from the Open Model Zoo! We'll now use the Model Downloader to download the `face-detection-retail-0004` model files. Run the following in your terminal:

```
$OPEN_MODEL_DOWNLOADER --name face-detection-retail-0004 --output_dir ~/open_model_
↪zoo_downloads/
```

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

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

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!

1.11.7 Compile the model

The MyriadX chip used on our DepthAI board does not use the IR format files directly. Instead, we need to generate two files:

- `face-detection-retail-0004.blob` - We'll create this file with the `myriad_compile` command.
- `face-detection-retail-0004.json` - A `blob_file_config` file in JSON format. This describes the format of the output tensors. You can read more about this file structure and examples *here*

We'll start by creating the `blob` file.

Locate myriad_compile

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

```
find /opt/intel/ -iname myriad_compile
```

You should see the output similar to this

```
find /opt/intel/ -iname myriad_compile
/opt/intel/openvino_2020.1.023/deployment_tools/inference_engine/lib/intel64/myriad_
↪compile
```

Since it's such a long path, let's store the `myriad_compile` executable in an environment variable (just like `OPEN_MODEL_DOWNLOADER`):

```
export MYRIAD_COMPILE=$(find /opt/intel/ -iname myriad_compile)
```

Activate OpenVINO environment

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

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

```
find /opt/intel/ -name "setupvars.sh"
/opt/intel/openvino_2020.1.023/opencv/setupvars.sh
/opt/intel/openvino_2020.1.023/bin/setupvars.sh
```

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

```
source /opt/intel/openvino_2020.1.023/bin/setupvars.sh
[setupvars.sh] OpenVINO environment initialized
```

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

Run myriad_compile

```
$MYRIAD_COMPILE -m ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/
↪face-detection-retail-0004.xml -ip U8 -VPU_MYRIAD_PLATFORM VPU_MYRIAD_2480 -VPU_
↪NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_SLICES 4
```

You should see:

```
Inference Engine:
  API version ............ 2.1
  Build .................. 37988
  Description ....... API
Done
```

Where's the blob file? It's located in the same folder as `face-detection-retail-0004.xml`:

```
ls -lh ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/
total 2.6M
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin
-rw-r--r-- 1 root root 1.3M Jul 28 12:50 face-detection-retail-0004.blob
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

1.11.8 Create the blob config file

The MyriadX needs both a `blob` file (which we just created) and a *blob_file_config* in JSON format. We'll create this config file manually. In your terminal:

```
cd ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/
touch face-detection-retail-0004.json
```

Copy and paste the following into `face-detection-retail-0004.json`:

```
{
    "NN_config": {
        "output_format" : "detection",
        "NN_family" : "mobilenet",
        "confidence_threshold" : 0.5
    }
}
```

What do these values mean?

- `output_format` - is either `detection` or `raw`. Detection tells the DepthAI to parse the NN results and make a detection objects out of them, which you can access by running `getDetectedObjects` method. Raw means "do not parse, I will handle the parsing on host", requiring you to parse raw tensors
- `NN_family` - only needed when `output_format` is `detection`. Two supported NN families are right now `mobilenet` and `YOLO`
- `confidence_threshold` - DepthAI will only return the results which confidence is above the threshold.

1.11.9 Run and display the model output

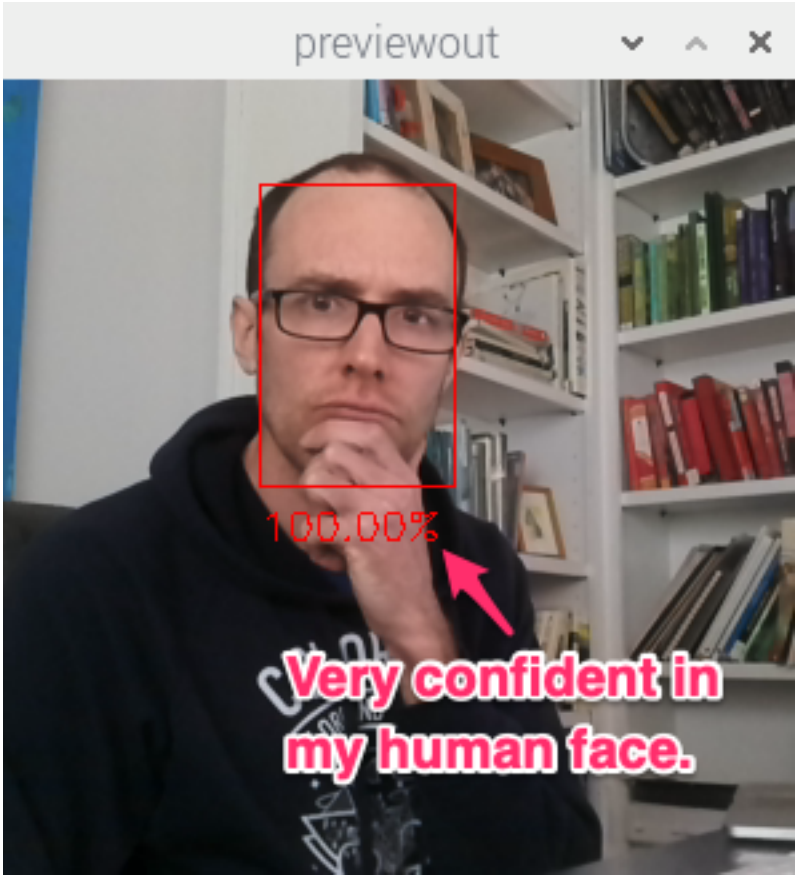
With both `blob` and a `json` blob config file, we're ready to roll! To verify that the model is running correctly, let's modify a bit the program we've created in *Hello World* tutorial

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

```
pipeline = device.create_pipeline(config={
    'streams': ['previewout', 'metaout'],
    'ai': {
-         'blob_file': "/path/to/mobilenet-ssd.blob",
+         'blob_file': "/path/to/open_model_zoo_downloads/intel/face-detection-retail-
↪0004/FP16/face-detection-retail-0004.blob",
-         'blob_file_config': "/path/to/mobilenet-ssd.json"
+         'blob_file_config': "/path/to/open_model_zoo_downloads/intel/face-detection-
↪retail-0004/FP16/face-detection-retail-0004.json"
    }
})
```

And that's all!

You should see output annotated output similar to:



1.11.10 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:

```
$OPEN_MODEL_DOWNLOADER --name [INSERT MODEL NAME] --output_dir ~/open_
↪model_zoo_downloads/
```

2. Create the MyriadX blob file:

```
$MYRIAD_COMPILE -m [INSERT PATH TO MODEL XML FILE] -ip U8 -VPU_MYRIAD_
↪PLATFORM VPU_MYRIAD_2480 -VPU_NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_
↪SLICES 4
```

3. *Create the blob config file* based on the model output.
4. Use this model in your script

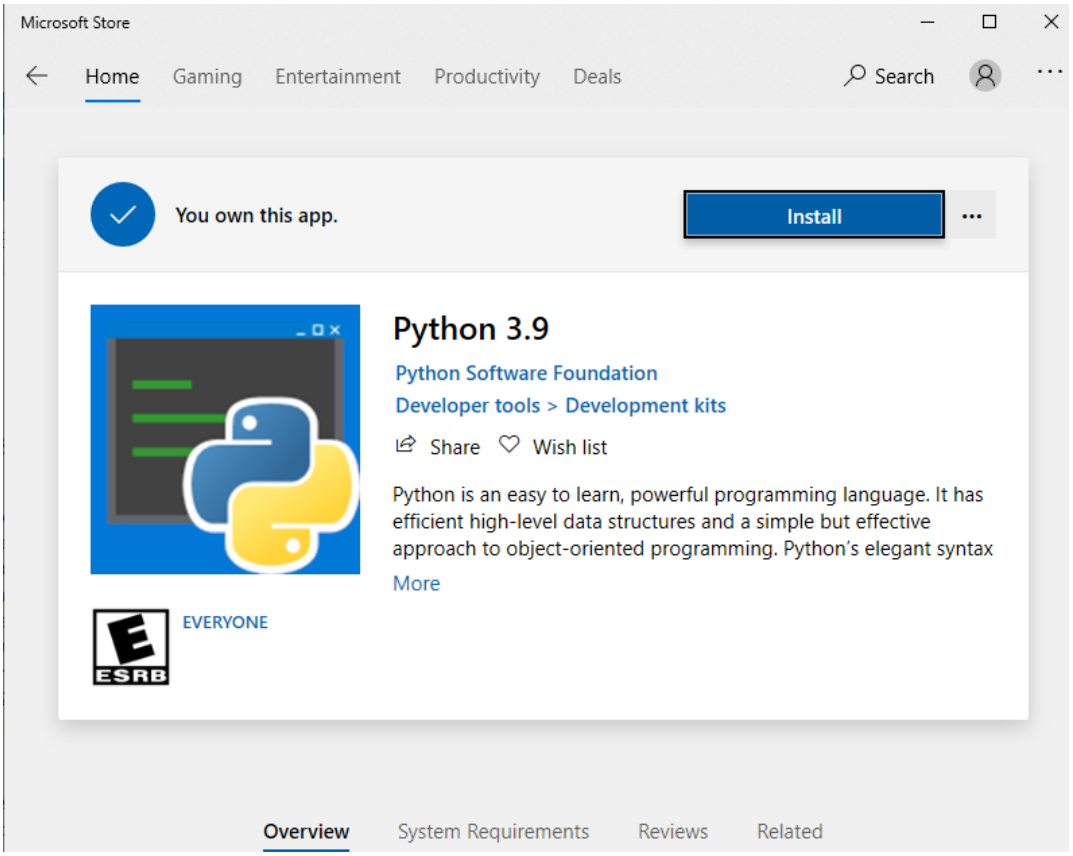
You're on your way! You can find the complete code for this tutorial on GitHub.

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

1.12 Install requirements on Windows

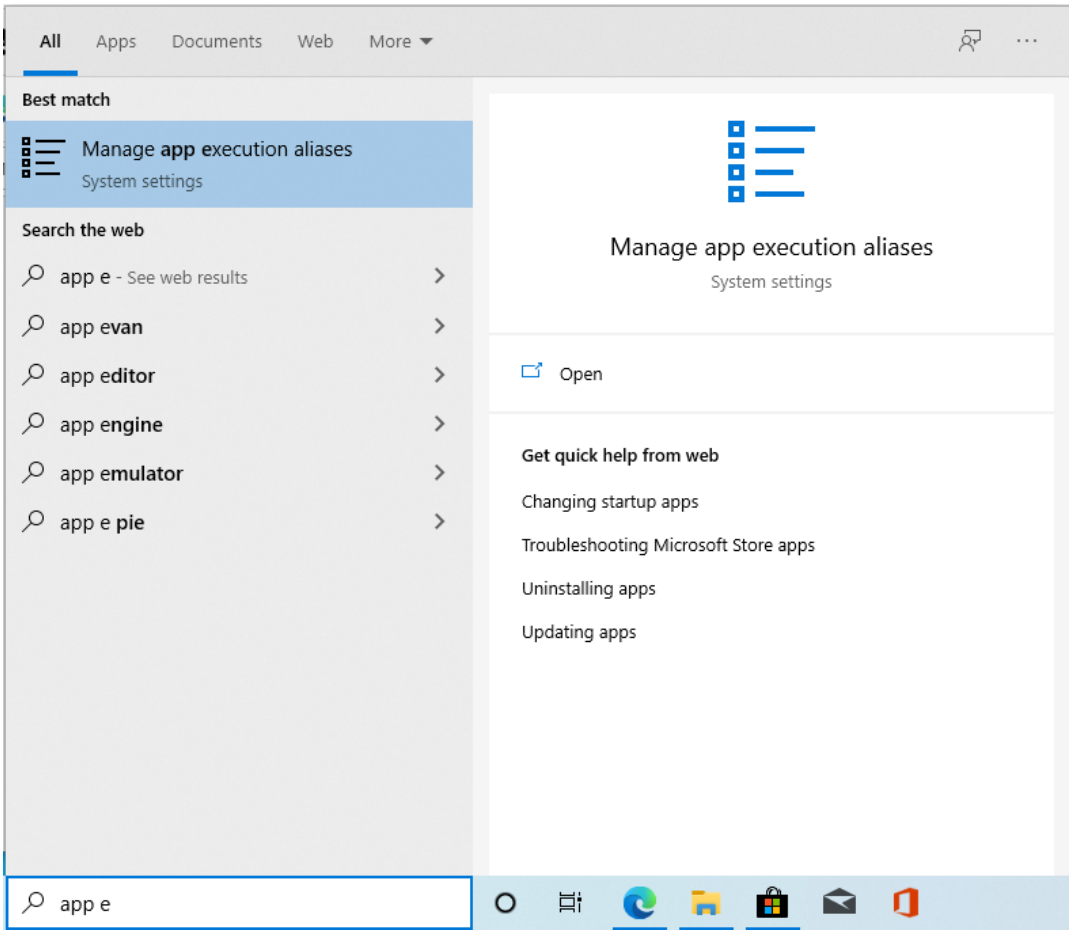
This tutorial provides steps to install DepthAI on a Windows 10 system

1. Install Python 3.9 from the Microsoft Store (direct link).

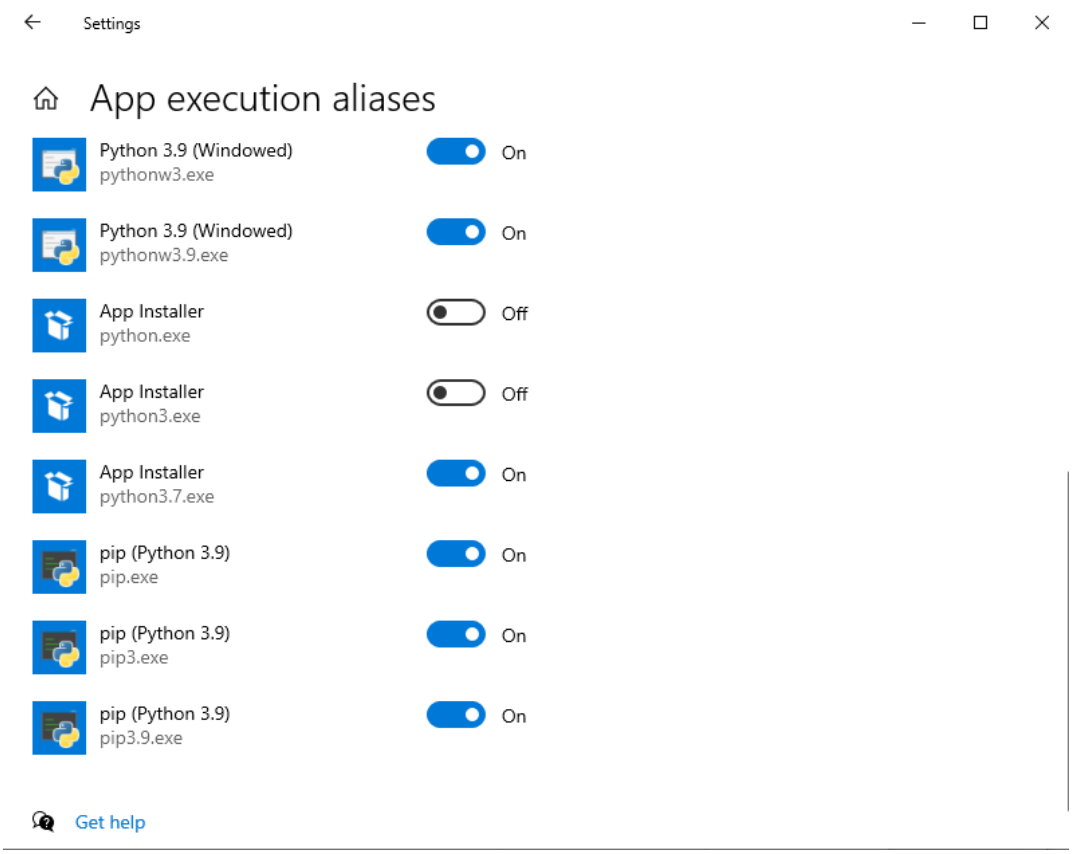


Once the installation is complete you need to disable the **App Execution Aliases** for both **python.exe** and **python3.exe**.

This can be done by going to start and typing `Manage app` and choose the `Manage app execution aliases` entry.

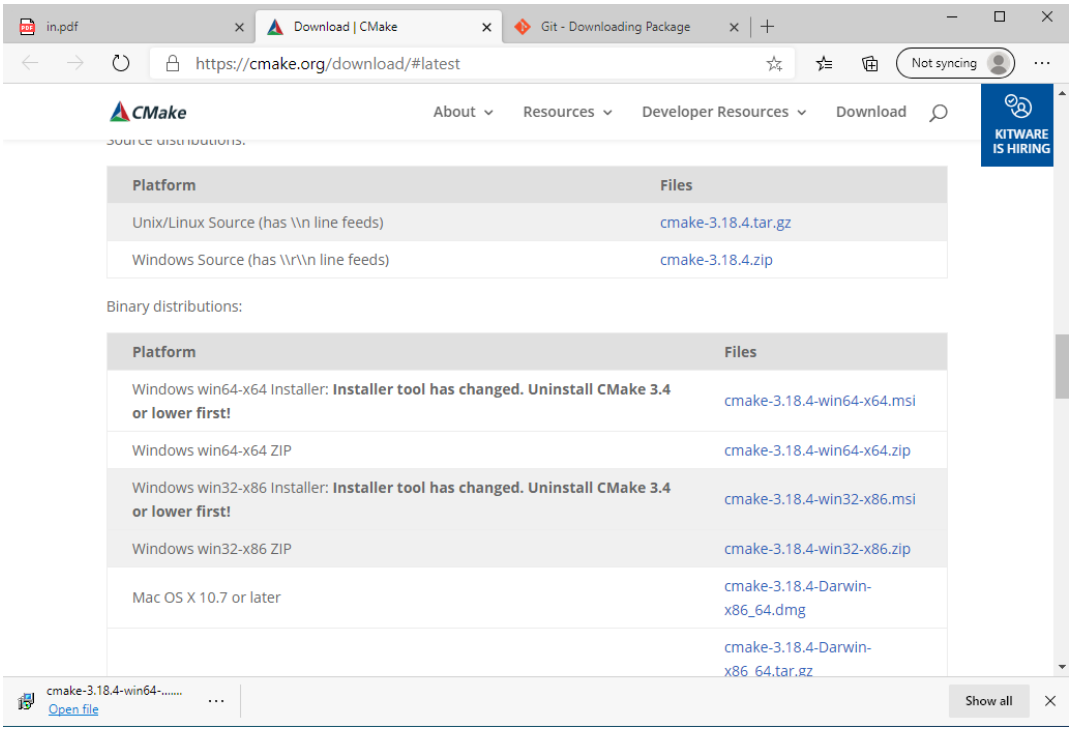


From there you turn off the `App Installer` alias for both **python.exe** and **python3.exe**.



2. Install Git. You can find the latest version here.

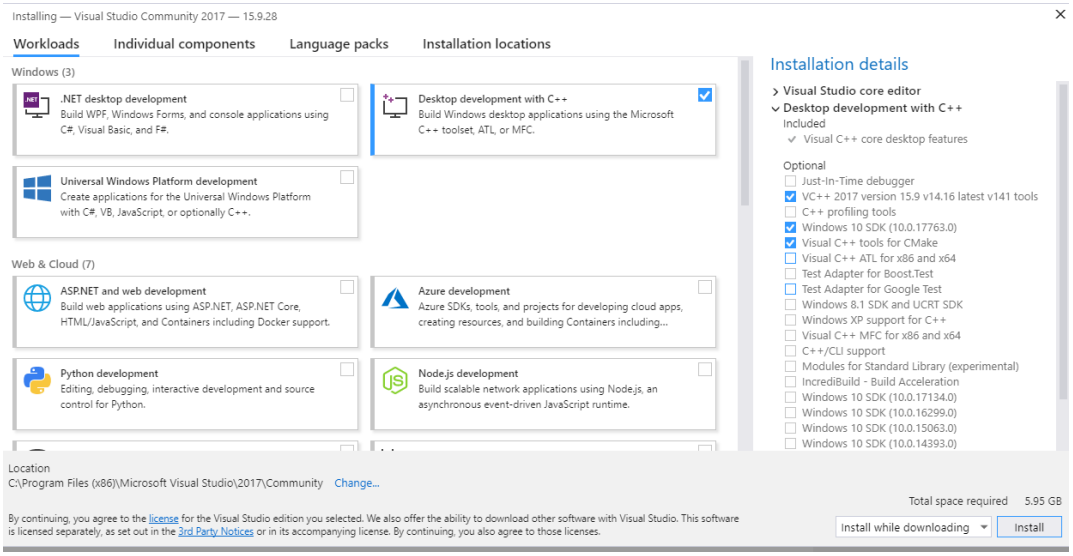
3. Install CMake. You can find the latest version here.



4. Install Microsoft Visual Studio 2017 (direct download link here)

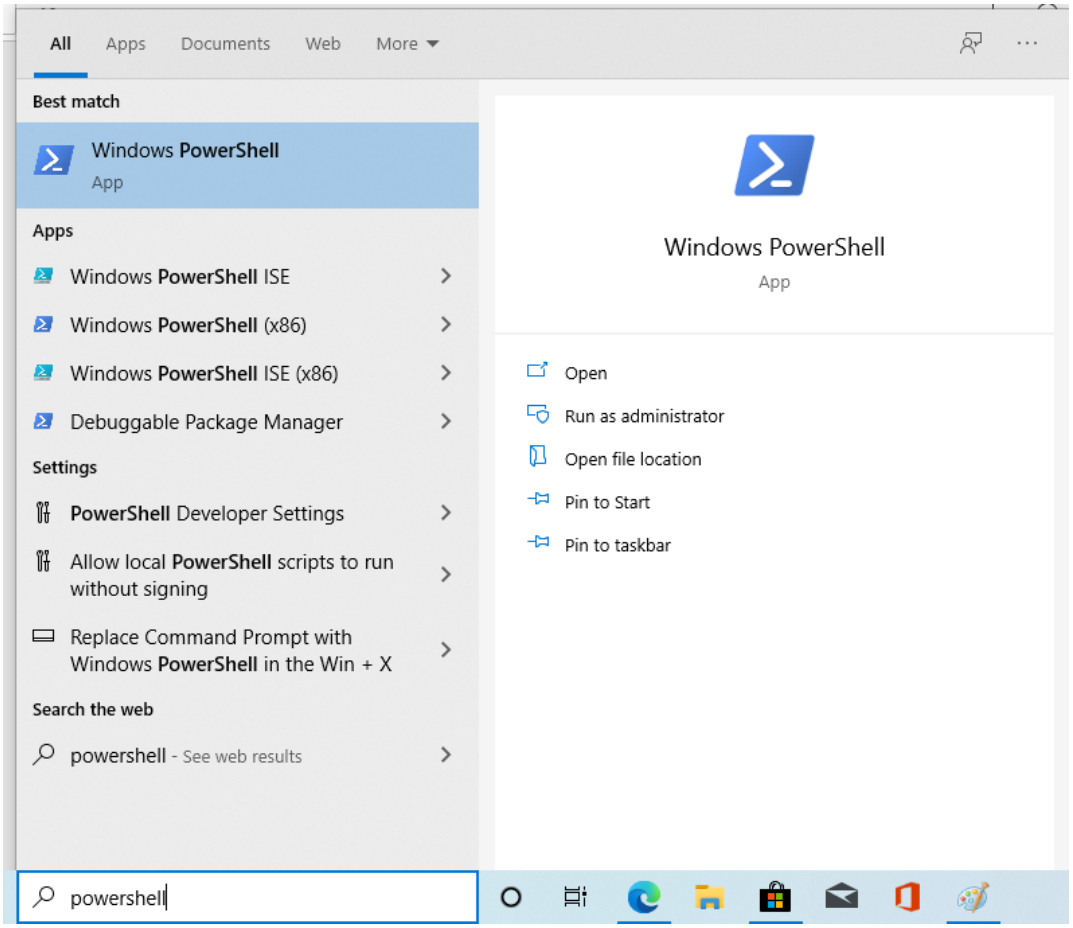
   The 2017 version is specifically required to build the OpenCV Python package from PyPI

   Note that you only need **Windows 10 SDK**, **Visual C++ for CMake** and **VC++ 2017**, not the whole package (see an image below)



5. Open Windows PowerShell

   You can do that by typing **PowerShell** in the searchbar

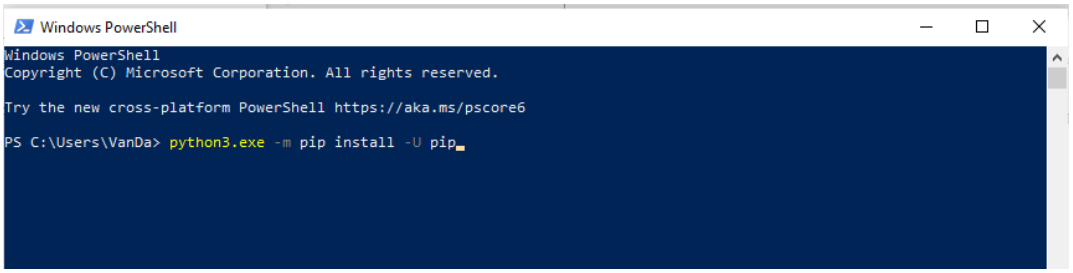


6. Upgrade Pip

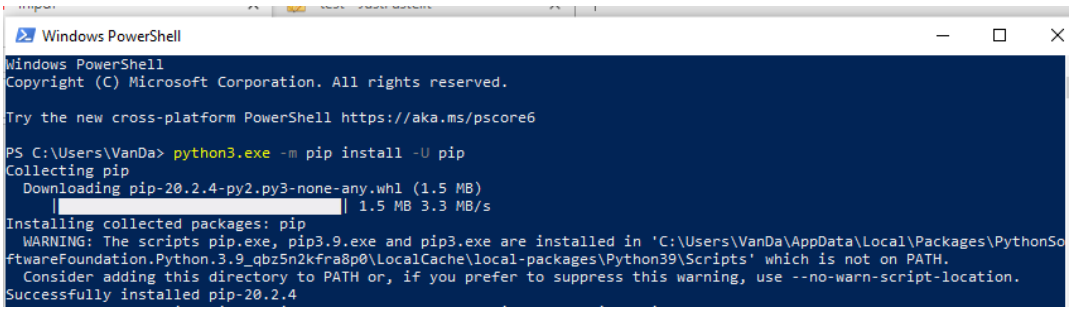
   Latest pip version is required in order to correctly install PyPI requirements.

   To upgrade pip, type the following command in the powershell

```
python3.exe -m pip install -U pip
```



   After successful install, you should see pip installed in the latest version



7. Run the DepthAI demo code

   See *Verify installation* for details

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

1.13 Minimal working code sample

1.13.1 Demo

1.13.2 Source code

```
import cv2
import depthai

device = depthai.Device('', False)

p = device.create_pipeline(config={
    "streams": ["metaout", "previewout"],
    "ai": {
        "blob_file": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-ssd.blob.
↪sh14cmx14NCE1",
        "blob_file_config": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-
↪ssd.json",
        'shaves' : 14,
        'cmx_slices' : 14,
        'NN_engines' : 1,
    }
})

if p is None:
    raise RuntimeError("Error initializing pipelne")

detections = []

while True:
    nnet_packets, data_packets = p.get_available_nnet_and_data_packets()

    for nnet_packet in nnet_packets:
        detections = list(nnet_packet.getDetectedObjects())

    for packet in data_packets:
        if packet.stream_name == 'previewout':
            data = packet.getData()
```

(continues on next page)

(continued from previous page)

```
            data0 = data[0, :, :]
            data1 = data[1, :, :]
            data2 = data[2, :, :]
            frame = cv2.merge([data0, data1, data2])

            img_h = frame.shape[0]
            img_w = frame.shape[1]

            for detection in detections:
                pt1 = int(detection.x_min * img_w), int(detection.y_min * img_h)
                pt2 = int(detection.x_max * img_w), int(detection.y_max * img_h)

                cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)

            cv2.imshow('previewout', frame)

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

del p
del device
```

1.13.3 Explanation

The code is divided into three phases: **initialization**, **processing results** and **deinitialization**.

Initialization is done here, as it's initializing the device and making sure that the pipeline is created. Please note that `sh14cmx14NCE1` in a blob file definition means that this blob was compiled to use 14 MyriadX SHAVEs, 14 MyriadX CMXes and 1 MyriadX Neural Compute Engine. These params needs to be provided in `ai` configuration, using fields `shaves`, `cmx_slices` and `NN_engines` respectively

```
device = depthai.Device('', False)

p = device.create_pipeline(config={
    "streams": ["metaout", "previewout"],
    "ai": {
        "blob_file": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-ssd.blob.
↪sh14cmx14NCE1",
        "blob_file_config": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-
↪ssd.json",
        'shaves' : 14,
        'cmx_slices' : 14,
        'NN_engines' : 1,
    }
})

if p is None:
    raise RuntimeError("Error initializing pipelne")
```

Deinitialization is basically only two lines of code, and whereas it's not necessary to include it, it's definitely recommended

```
del p
del device
```

Now, the results processing consists of two phases - parsing nnet results and displaying the frames.

Parsing neural network results

Below, you'll se the part that's parsing the results from neural network

```
detections = []

while True:
    nnet_packets, data_packets = p.get_available_nnet_and_data_packets()

    for nnet_packet in nnet_packets:
        detections = list(nnet_packet.getDetectedObjects())
```

Neural network configuration we specified earlier, in `blob_file_config` field, allows DepthAI to prepare results in a correct format and remove incorrect entries (e.g. those with confidence below threshold).

Each object in this array is a *`Detection`* instance, which we can easily use later in the code

Displaying the frames

```
for packet in data_packets:
    if packet.stream_name == 'previewout':
        data = packet.getData()
        data0 = data[0, :, :]
        data1 = data[1, :, :]
        data2 = data[2, :, :]
        frame = cv2.merge([data0, data1, data2])

        img_h = frame.shape[0]
        img_w = frame.shape[1]

        for detection in detections:
            pt1 = int(detection.x_min * img_w), int(detection.y_min * img_h)
            pt2 = int(detection.x_max * img_w), int(detection.y_max * img_h)

            cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)

        cv2.imshow('previewout', frame)

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

This stage is also divided into three phases - preparing the frame, augumenting the frame and adding control signals

Preparing the frame basically means that we're transforming the frame to OpenCV-usable form.

First, we need to assure we're operating on packet from `previewout` stream, so it's a frame from 4K color camera.

Next, we get the data from the packet and transform it from `CHW` (Channel, Height, Width) form used by DepthAI to `HWC` (Height, Width, Channel) that is used by OpenCV.

```
for packet in data_packets:
    if packet.stream_name == 'previewout':
        data = packet.getData()  # e.x. shape (3, 300, 300)
        data0 = data[0, :, :]
        data1 = data[1, :, :]
        data2 = data[2, :, :]
        frame = cv2.merge([data0, data1, data2])  # e.x. shape (300, 300, 3)
```

Augumenting the frame means any process that changes what is being displayed. In this example, I'm adding red rectangles around detected items. You can also add here text displays, latency info - basically whatever your business logic requires.

Since the position of the bounding boxes are returned from neural network as floats in range `(0, 1)`, which specify position of the point relative to it's width/height, we need to transform it into the actual point on the image (which you can see as we're doing e.x. `int(detection.x_min * img_w)`).

Next, using `cv2.rectangle`, we're printing the actual rectangle on the `frame`. Finally, when the frame is ready, we display it using `cv2.imshow` function.

```
img_h = frame.shape[0]
img_w = frame.shape[1]

for detection in detections:
    pt1 = int(detection.x_min * img_w), int(detection.y_min * img_h)
    pt2 = int(detection.x_max * img_w), int(detection.y_max * img_h)

    cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)

cv2.imshow('previewout', frame)
```

Adding control signals is the last part, where you can add interactivity to the displayed image. We're adding just one command - to terminate the program - when you press the `q` button.

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

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

1.14 Color camera selfie maker

This sample requires TK library to run (for opening file dialog)

It also requires face detection model, see *this tutorial* to see how to compile one

1.14.1 Demo

Capturing process

Captured image

1.14.2 Source code

```
import cv2
import depthai

device = depthai.Device('', False)

pipeline = device.create_pipeline(config={
    'streams': ['previewout', 'metaout'],
    'ai': {
        "blob_file": "/path/to/face-detection-retail-0004.blob",
        "blob_file_config": "/path/to/face-detection-retail-0004.json",
    }
})

if pipeline is None:
    raise RuntimeError('Pipeline creation failed!')

detections = []
face_frame = None

while True:
    nnet_packets, data_packets = pipeline.get_available_nnet_and_data_packets()

    for nnet_packet in nnet_packets:
        detections = list(nnet_packet.getDetectedObjects())

    for packet in data_packets:
        if packet.stream_name == 'previewout':
            data = packet.getData()
            data0 = data[0, :, :]
            data1 = data[1, :, :]
            data2 = data[2, :, :]
            frame = cv2.merge([data0, data1, data2])

            img_h = frame.shape[0]
            img_w = frame.shape[1]

            for detection in detections:
                left = int(detection.x_min * img_w)
                top = int(detection.y_min * img_h)
                right = int(detection.x_max * img_w)
                bottom = int(detection.y_max * img_h)

                face_frame = frame[top:bottom, left:right]
                if face_frame.size == 0:
                    continue
                cv2.imshow('face', face_frame)

    key = cv2.waitKey(1)
    if key == ord('q'):
        break
    if key == ord(' ') and face_frame is not None:
        from tkinter import Tk, messagebox
        from tkinter.filedialog import asksaveasfilename
        Tk().withdraw()
        filepath = asksaveasfilename(defaultextension=".png", filetypes=(("Image files
↪", "*.png"),("All Files", "*.*")))
```

(continues on next page)

(continued from previous page)

```
        cv2.imwrite(filepath, face_frame)
        messagebox.showinfo("Success", "Image saved successfully!")
        Tk().destroy()

del pipeline
del device
```

1.14.3 Explanation

> **Warning: New to the DepthAI?**
>
> DepthAI basics are explained in *Minimal working code sample* and *Hello World* tutorial.

Our network returns bounding boxes of the faces it detects (we have them stored in `detections` array). So in this sample, we have to do two main things: **crop the frame** to contain only the face and **save it** to the location specified by user.

Performing the crop

Cropping the frame requires us to modify the *Minimal working code sample*, so that we don't produce two points for rectangle, but instead we need all four points: two of them that determine start of the crop (`top` starts Y-axis crop and `left` starts X-axis crop), and another two as the end of the crop (`bottom` ends Y-axis crop and `right` ends X-axis crop)

```
left = int(detection.x_min * img_w)
top = int(detection.y_min * img_h)
right = int(detection.x_max * img_w)
bottom = int(detection.y_max * img_h)
```

Now, since our frame is in `HWC` format (Height, Width, Channels), we first crop the Y-axis (being height) and then the X-axis (being width). So the cropping code looks like this:

```
face_frame = frame[top:bottom, left:right]
```

Now, there's one additional thing to do. Since sometimes the network may produce such bounding box, what when cropped will produce an empty frame, we have to secure ourselves from this scenario, as `cv2.imshow` will throw an error if invoked with empty frame.

```
if face_frame.size == 0:
    continue
cv2.imshow('face', face_frame)
```

Storing the frame

To save the image we'll use just a single line of code, invoking `cv2.imwrite`. Rest of the code, utilizing `tkinter` package, is optional and can be removed if you don't require user interaction to save the frame.

In this sample, we use `tkinter` for two dialog boxes:

- To obtain destination filepath (stored as `filepath`) that allows us to invoke `cv2.imwrite` as it requires path as it's first argument
- To confirm that the file was saved successfully

```
key = cv2.waitKey(1)
if key == ord('q'):
    break
if key == ord(' ') and face_frame is not None:
    from tkinter import Tk, messagebox
    from tkinter.filedialog import asksaveasfilename
    Tk().withdraw()  # do not open root TK window
    filepath = asksaveasfilename(defaultextension=".png", filetypes=(("Image files",
↪"*.png"),("All Files", "*.*")))
    cv2.imwrite(filepath, face_frame)  # save the image to user-specified path
    messagebox.showinfo("Success", "Image saved successfully!")  # show confirmation␣
↪dialog
    Tk().destroy()  # destroy confirmation dialog
```

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

1.15 Mono cameras selfie maker

This sample requires TK library to run (for opening file dialog)

It also requires face detection model, see *this tutorial* to see how to compile one

Stereo camera pair is required to run this example, it can either be *BW1097 - RaspberryPi Compute Module*, BW1098OBC - USB3 with Onboard Cameras or any custom setup using *DepthAI Mono Camera*

1.15.1 Demo

Capturing process

Captured image

1.15.2 Source code

```
import cv2
import depthai

device = depthai.Device('', False)

pipeline = device.create_pipeline(config={
    'streams': ['left', 'right', 'metaout'],
    'ai': {
        "blob_file": "/path/to/face-detection-retail-0004.blob",
        "blob_file_config": "/path/to/face-detection-retail-0004.json",
    },
    'camera': {'mono': {'resolution_h': 720, 'fps': 30}},
})

if pipeline is None:
    raise RuntimeError('Pipeline creation failed!')

detections = []
face_frame_left = None
face_frame_right = None

while True:
    nnet_packets, data_packets = pipeline.get_available_nnet_and_data_packets()

    for nnet_packet in nnet_packets:
        detections = list(nnet_packet.getDetectedObjects())

    for packet in data_packets:
        if packet.stream_name == 'left' or packet.stream_name == 'right':
            frame = packet.getData()

            img_h = frame.shape[0]
```

(continues on next page)

(continued from previous page)

```
            img_w = frame.shape[1]

            for detection in detections:
                left = int(detection.x_min * img_w)
                top = int(detection.y_min * img_h)
                right = int(detection.x_max * img_w)
                bottom = int(detection.y_max * img_h)

                face_frame = frame[top:bottom, left:right]
                if face_frame.size == 0:
                    continue
                cv2.imshow(f'face-{packet.stream_name}', face_frame)
                if packet.stream_name == 'left':
                    face_frame_left = face_frame
                else:
                    face_frame_right = face_frame

    key = cv2.waitKey(1)
    if key == ord('q'):
        break
    if key == ord(' ') and face_frame_left is not None and face_frame_right is not␣
↪None:
        from tkinter import Tk, messagebox
        from tkinter.filedialog import asksaveasfilename
        Tk().withdraw()
        filename = asksaveasfilename(defaultextension=".png", filetypes=(("Image files
↪", "*.png"),("All Files", "*.*")))
        joined_frame = cv2.hconcat([face_frame_left, face_frame_right])
        cv2.imwrite(filename, joined_frame)
        messagebox.showinfo("Success", "Image saved successfully!")
        Tk().destroy()

del pipeline
del device
```

1.15.3 Explanation

Warning: New to the DepthAI?

DepthAI basics are explained in *Minimal working code sample* and *Hello World* tutorial.

Our network returns bounding boxes of the faces it detects (we have them stored in `detections` array). So in this sample, we have to do two main things: **crop the frame** to contain only the face and **save it** to the location specified by user.

Performing the crop

Cropping the frame requires us to modify the *Minimal working code sample*, so that we don't produce two points for rectangle, but instead we need all four points: two of them that determine start of the crop (`top` starts Y-axis crop and `left` starts X-axis crop), and another two as the end of the crop (`bottom` ends Y-axis crop and `right` ends X-axis crop)

```
left = int(detection.x_min * img_w)
top = int(detection.y_min * img_h)
right = int(detection.x_max * img_w)
bottom = int(detection.y_max * img_h)
```

Now, since our frame is in `HWC` format (Height, Width, Channels), we first crop the Y-axis (being height) and then the X-axis (being width). So the cropping code looks like this:

```
face_frame = frame[top:bottom, left:right]
```

Now, there's one additional thing to do. Since sometimes the network may produce such bounding box, what when cropped will produce an empty frame, we have to secure ourselves from this scenario, as `cv2.imshow` will throw an error if invoked with empty frame.

```
if face_frame.size == 0:
    continue
cv2.imshow('face', face_frame)
```

Later on, as we're having two cameras operating same time, we're assigning the shown frame to either left or right face frame variable, which will help us later during image saving

```
if packet.stream_name == 'left':
    face_frame_left = face_frame
else:
    face_frame_right = face_frame
```

Storing the frame

To save the image we'll need to do two things:

- Merge the face frames from both left and right cameras into one frame
- Save the prepared frame to the disk

Thankfully, OpenCV has it all sorted out, so for each point we'll use just a single line of code, invoking `cv2.hconcat` for frames merging and `cv2.imwrite` to store the image

Rest of the code, utilizing `tkinter` package, is optional and can be removed if you don't require user interaction to save the frame.

In this sample, we use `tkinter` for two dialog boxes:

- To obtain destination filepath (stored as `filepath`) that allows us to invoke `cv2.imwrite` as it requires path as it's first argument
- To confirm that the file was saved successfully

```
key = cv2.waitKey(1)
if key == ord('q'):
    break
if key == ord(' ') and face_frame_left is not None and face_frame_right is not None:
```

(continues on next page)

(continued from previous page)

```
    from tkinter import Tk, messagebox
    from tkinter.filedialog import asksaveasfilename
    Tk().withdraw()
    filename = asksaveasfilename(defaultextension=".png", filetypes=(("Image files",
↪"*.png"),("All Files", "*.*")))
    joined_frame = cv2.hconcat([face_frame_left, face_frame_right])
    cv2.imwrite(filename, joined_frame)
    messagebox.showinfo("Success", "Image saved successfully!")
    Tk().destroy()
```

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

1.16 Object tracker

1.16.1 Demo

1.16.2 Source code

```
import cv2
import depthai

device = depthai.Device('', False)

p = device.create_pipeline(config={
    "streams": ["previewout", "object_tracker"],
    "ai": {
        #blob compiled for maximum 12 shaves
        "blob_file": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-ssd.blob.
↪sh12cmx12NCE1",
        "blob_file_config": "/path/to/depthai/resources/nn/mobilenet-ssd/mobilenet-
↪ssd.json",
        "shaves" : 12,
        "cmx_slices" : 12,
        "NN_engines" : 1,
    },
    'ot': {
        'max_tracklets': 20,
        'confidence_threshold': 0.5,
    },
})

if p is None:
    raise RuntimeError("Error initializing pipelne")

labels = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car
↪", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person",
↪"pottedplant", "sheep", "sofa", "train", "tvmonitor"]
tracklets = None

while True:
    for packet in p.get_available_data_packets():
        if packet.stream_name == 'object_tracker':
            tracklets = packet.getObjectTracker()
```

(continues on next page)

(continued from previous page)

```
        elif packet.stream_name == 'previewout':
            data = packet.getData()
            data0 = data[0, :, :]
            data1 = data[1, :, :]
            data2 = data[2, :, :]
            frame = cv2.merge([data0, data1, data2])

            traklets_nr = tracklets.getNrTracklets() if tracklets is not None else 0

            for i in range(traklets_nr):
                tracklet = tracklets.getTracklet(i)
                left = tracklet.getLeftCoord()
                top = tracklet.getTopCoord()
                right = tracklet.getRightCoord()
                bottom = tracklet.getBottomCoord()
                tracklet_label = labels[tracklet.getLabel()]

                cv2.rectangle(frame, (left, top), (right, bottom), (255, 0, 0))

                middle_pt = (int(left + (right - left) / 2), int(top + (bottom - top)␣
↪/ 2))
                cv2.circle(frame, middle_pt, 0, (255, 0, 0), -1)
                cv2.putText(frame, f"ID {tracklet.getId()}", middle_pt, cv2.FONT_
↪HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

                cv2.putText(frame, tracklet_label, (left, bottom - 40), cv2.FONT_
↪HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
                cv2.putText(frame, tracklet.getStatus(), (left, bottom - 20), cv2.
↪FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

            cv2.imshow('previewout', frame)

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

del p
del device
```

1.16.3 Explanation

> **Warning: New to the DepthAI?**
>
> DepthAI basics are explained in *Minimal working code sample* and *Hello World* tutorial.

DepthAI is capable of doing object tracking from the device itself, so you don't have to write your own custom code for it.

First, we change the method for retrieving the data packets from the pipeline, as we're not using `metaout` stream

```
for packet in p.get_available_data_packets():
```

Next up, if the packet is from stream `object_tracker` we use a special method available only in packets from this stream to obtain tracklets object - this method will throw an error if used on another stream

```
if packet.stream_name == 'object_tracker':
    tracklets = packet.getObjectTracker()
```

Next up, we're obtaining all tracklet info for further processing

```
tracklet = tracklets.getTracklet(i)
left = tracklet.getLeftCoord()
top = tracklet.getTopCoord()
right = tracklet.getRightCoord()
bottom = tracklet.getBottomCoord()
tracklet_label = labels[tracklet.getLabel()]
```

And rest of the processing is only for visual representation of the tracking objects, which you can skip if you want to use object tracker in different way

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

1.17 Helper - DepthAI as a class

If you want to integrate the DepthAI into your project, this example might be useful for you as it splits the "**how** to get the results", from "**what** to do with them".

This example splits the pipeline initialization from the actual usage.

It's useful if you want to make more subclasses of it (e.x. *MonoDepthAI* that will contain config for mono cameras) or run the DepthAI code in a subprocess (e.x. `Process(target=DepthAI().run).start()`)

1.17.1 Code

```
from pathlib import Path

import cv2
import depthai


class DepthAI:
    def __init__(self):
        self.device = depthai.Device('', False)

        self.p = self.device.create_pipeline(config={
            "streams": ["metaout", "previewout"],
            "ai": {
                "blob_file": "/path/to/model.blob",
                "blob_file_config": "/path/to/config.json"
            }
        })

        self.detections = []

    def run(self):
        while True:
            nnet_packets, data_packets = self.p.get_available_nnet_and_data_packets()

            for nnet_packet in nnet_packets:
```

(continues on next page)

(continued from previous page)

```
                self.detections = list(nnet_packet.getDetectedObjects())

            for packet in data_packets:
                if packet.stream_name == 'previewout':
                    data = packet.getData()
                    if data is None:
                        continue
                    data0 = data[0, :, :]
                    data1 = data[1, :, :]
                    data2 = data[2, :, :]
                    frame = cv2.merge([data0, data1, data2])

                    img_h = frame.shape[0]
                    img_w = frame.shape[1]

                    for detection in self.detections:
                        pt1 = int(detection.x_min * img_w), int(detection.y_min * img_
↪h)
                        pt2 = int(detection.x_max * img_w), int(detection.y_max * img_
↪h)

                        cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)

                    cv2.imshow('previewout', frame)

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

        del self.p
        del self.device


DepthAI().run()
```

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

1.18 Helper - DepthAI as a generator

If you want to integrate the DepthAI into your project, this example might be useful for you as it splits the "**how** to get the results", from "**what** to do with them".

This example uses `yield` keyword to send the results to for loop, which called the method.

It's useful if you want to process the received frames further in your custom code

1.18.1 Code

```
import cv2
import depthai


class DepthAI:
    def __init__(self):
        self.device = depthai.Device('', False)

        self.p = self.device.create_pipeline(config={
            "streams": ["metaout", "previewout"],
            "ai": {
                "blob_file": "/path/to/model.blob",
                "blob_file_config": "/path/to/config.json"
            }
        })

        self.detections = []

    def run(self):
        while True:
            nnet_packets, data_packets = self.p.get_available_nnet_and_data_packets()

            for nnet_packet in nnet_packets:
                self.detections = list(nnet_packet.getDetectedObjects())

            for packet in data_packets:
                if packet.stream_name == 'previewout':
                    data = packet.getData()
                    data0 = data[0, :, :]
                    data1 = data[1, :, :]
                    data2 = data[2, :, :]
                    frame = cv2.merge([data0, data1, data2])

                    img_h = frame.shape[0]
                    img_w = frame.shape[1]

                    results = []
                    for detection in self.detections:
                        pt1 = int(detection.x_min * img_w), int(detection.y_min * img_
↪h)
                        pt2 = int(detection.x_max * img_w), int(detection.y_max * img_
↪h)
                        results.append((pt1, pt2))
                    yield frame, results

    def __del__(self):
        del self.p
        del self.device


d = DepthAI()

for frame, results in d.run():
    for pt1, pt2 in results:
        cv2.rectangle(frame, pt1, pt2, (0, 0, 255), 2)
```

(continues on next page)

(continued from previous page)

```
    cv2.imshow('previewout', frame)

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

del d
```

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

INDEX

E

F

I

L

N

O

S

T

X

Y