TuboPark - AI

Write something, or press 'space' for AI, '/' for commands...

🔎 Model picking

Using the HAILO AI kit, we'll start by testing the different models directly available from its python library via Tappas (Optimized Execution of Video-Processing Pipelines).

We'll focus on `object detection` models, as they are the best fit for our task.

Model & Supported tasks

YoloV8n:

- Detection
- Instance segmentation
- Pose estimation

Our pick

We have chosen the pre-trained detection model of YOLOv8n because it meets our requirements without adding unnecessary complexity. Our use case is straightforward, and the detection capabilities of YOLOv8n are sufficient for our needs.

🏋️ Training the model

Collecting the Data

To collect the training and validation images for our dataset, we use puppeteer to scrap:

- Google
- Duckduckgo
- Bing

Formatting / Normalizing the Data

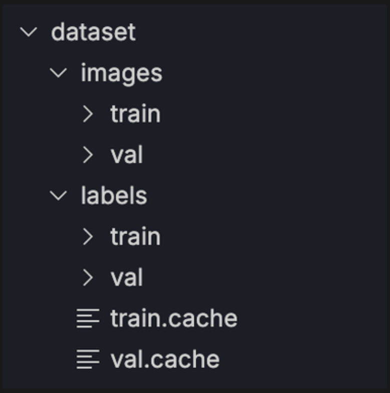
Now that we have some images, we need to format them (resize) and remove duplicate. To do so, we'll be using a custom python script.

Creating a Dataset

To create the dataset, we'll be using Roboflow, which will allow us to create custom labels.
A YOLO label should have the following format:

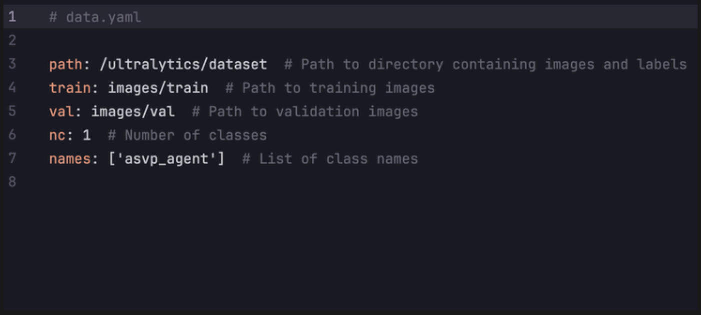
```
asvp_agent 0.80 128 42 320 180 – The first column indicates the
class label ("asvp_agent"). – The second column is the confidence
score for the respective object. – The next four values represent
the bounding box coordinates (x_min, y_min, x_max, y_max).
```

Once done, the dataset structure should look like:



```
dataset
  images
    train
    val
  labels
    train
    val
    train.cache
    val.cache
```

We then create a `data.yaml` file for the dataset mapping:



```
# data.yaml

path: /ultralytics/dataset  # Path to directory containing images and labels
train: images/train  # Path to training images
val: images/val  # Path to validation images
nc: 1  # Number of classes
names: ['asvp_agent']  # List of class names

```

Training

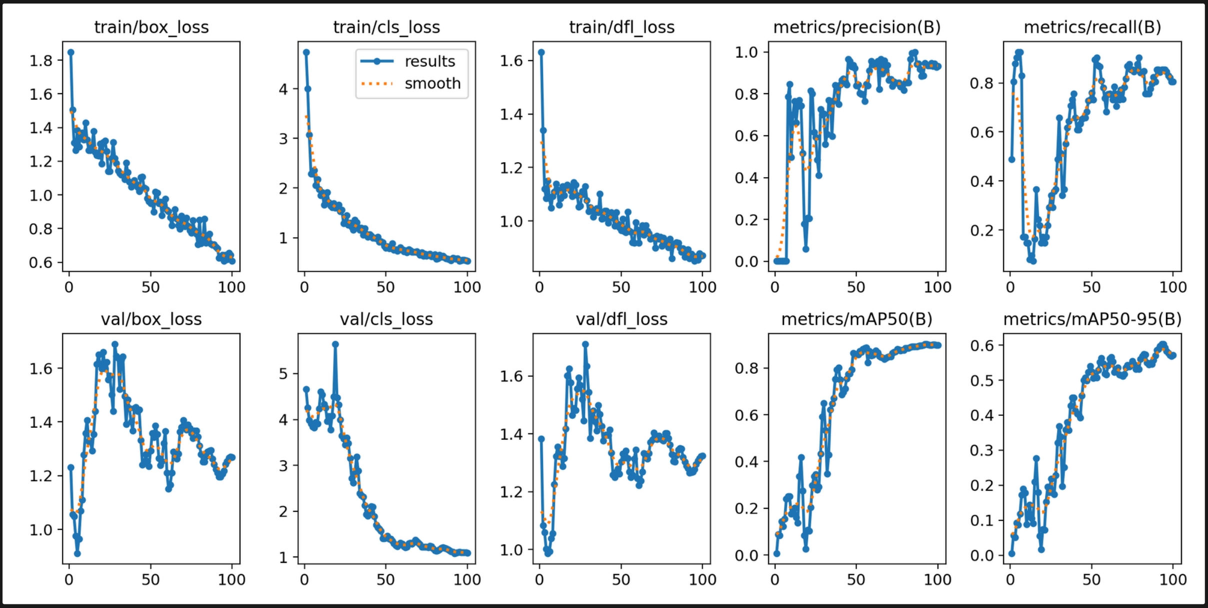
The training will be done using Ultralytics and the YOLO flag `train` :

```
yolo detect train data=data.yaml model=yolov8n.pt
name=asvp_agent_yolov8n epochs=100 imgsz=640
```

Once the training is completed, we'll do a validation run using unprocessed images, to assess the quality of your trained model:

```
yolo detect val data=data.yaml
model=runs/detect/train/weights/best.pt source=images/google
```

Here are some perf results of our first training run:



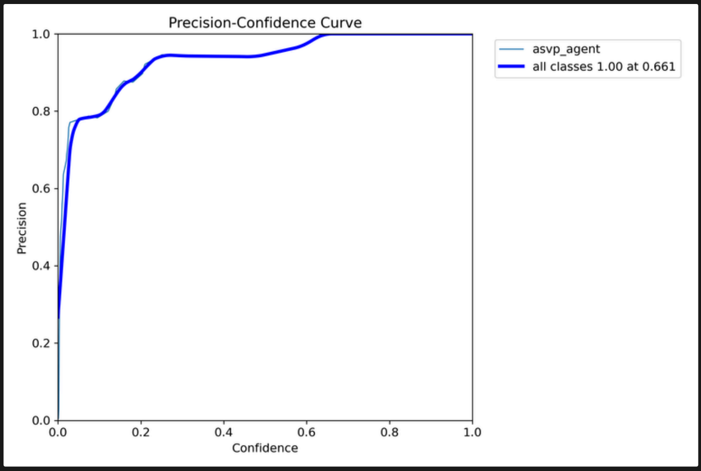
Training Metrics

- **train/box_loss**: Bounding box regression loss during training.
- **train/cls_loss**: Classification loss during training.
- **train/dfl_loss**: Distribution Focal Loss during training.
- **metrics/precision(B)**: Precision metric during training.
- **metrics/recall(B)**: Recall metric during training.

Validation Metrics

- **val/box_loss**: Bounding box regression loss during validation.
- **val/clr_loss**: Classification loss during validation.
- **val/dfl_loss**: Distribution Focal Loss during validation.
- **metrics/mAP50(B)**: Mean Average Precision at 50% IoU during validation.
- **metrics/mAP50-95(B)**: Mean Average Precision at IoU thresholds from 50% to 95% during validation.

The `P curve` graph & a sampled batch of some `val` images from our training dataset:



A **Precision-Confidence Curve** shows how the precision of a model varies with different confidence thresholds. It helps in selecting an optimal confidence threshold to balance precision and recall according to the specific requirements of the task.

- **Precision**: The ratio of true positive predictions to the total positive predictions.
- **Confidence Score**: The model's certainty about its prediction.

📦 Formating for HAILO

Now that our model is trained, validated and tested, it's time to format it, enabling us to load it onto the HAILO AI kit on the raspi.

To do so, we'll start by running the following:

```
yolo export model=runs/detect/train2/weights/best.pt imgsz=640
format=onnx opset=11
```

It'll format the trained model to `ONNX` .

ONNX is an open-source format designed to represent machine learning models. It enables models to be transferred between different frameworks, ensuring interoperability.

Use the Hailo Model Zoo command (this can take up to 30 minutes):

```
hailomz compile yolov8s --ckpt=<trained-model-name>.onnx --hw-arch
hailo8l --calib-path barcode-detect/test/images/ --classes 2 --
performance
```

- **--ckpt**: Specifies the trained model file in ONNX format.
- **--hw-arch**: Targets the Hailo8L hardware architecture.
- **--calib-path**: Provides the path to calibration images.
- **--classes**: Indicates the number of classes (2 in this case).
- **--performance**: Optimizes the model for performance.

Calibration images are a set of representative images used during the process of quantizing a neural network model. Quantization is the process of converting a model's weights and activations from higher precision (e.g., 32-bit floating point) to lower precision (e.g., 8-bit integer) to reduce the model's size and improve inference speed, especially on edge devices.

🔩 Implement in API