RVC3 - Model deployment

This document describes how to export your model to a blob format supported on RVC3. RVC3 supports FP16 (similar as on RVC2 devices) and INT8 (new) blobs. FP16 performance is expected to be slightly worse compared to RVC2, while INT8 should be faster at a small accuracy cost. In *General model deployment* steps 1 and 3 are related to FP16 blob compilation, while optional step 2 can be used to produce INT8 blob.

General model deployment

Steps

The following steps describe how to produce a blob for RVC3. Keep in mind that we only describe good practices and what usually works best.

Step 1 (generate IR):

Generate IR (XML and BIN) as before - preferably use –data_type FP32 instead of FP16. Note that some models might have troubles with latest IRv11 introduced in OV 22.1, so you might have to use `-use_legacy_frontend` flag if you are using OV 22.1

Step 2 (optional - quantize the model):

If you want to use INT8 inference on RVC3 you need to quantize the model.

You can refer to official documentation of POT on the link below to define metrics, but for quick tests you can try default quantization. For this, you will need some images. For example, download coco128.zip and extract it somewhere on your computer: https://ultralytics.com/assets/coco128.zip. For best quantization and lowest accuracy drop, we suggest you use images from your training or validation set.

To install POT (Post-training Optimization Tool):

```
# (inside Python environment)
python -m pip install --upgrade pip
pip install openvino-dev==2022.1
```

Next, define a pot-config.json file. As a template you can use this https://pastebin.com/S62gUtuP, but make sure to specify a proper path to XML, BIN, and dataset.

Call:
`pot -c pot-config.json -d`

This will create a directory with the results, where your quantized XML and BIN will be stored. Move those to RVC3. You can use `scp` to copy the files from your computer to RVC3. For example:
`scp path/to/model.xml <rvc3-ip>:/arbitrary-path/model.xml`
`scp path/to/model.bin <rvc3-ip>:/arbitrary-path/model.bin`
(Note: arbitrary-path must exist on RVC3)

Step 3 (Compilation - on RVC3)

Move to the directory with XML and BIN.
Then initialize the environment:
`source /opt/openvino/setupvars.sh`

Compile the model:
`/opt/openvino/tools/compile_tool/compile_tool -d VPUX.3400 -m model.xml -ip U8  -ov_api_1_0`

Some models might not compile with the latest version of OpenVINO API. This is why sometimes `-ov_api_1_0` is necessary. Feel free to try and export without it.

You can also create a file config and specify it when compiling: `-c config`. There it is possible to set `VPUX_COMPILER_TYPE [MCM,MLIR],PERFORMANCE_HINT [THROUGHPUT,LATENCY]`…
Note:

- Contrary to RVC2, `VPUX_INFERENCE_SHAVES` flag exists but is ignored and should be set at inference time.
- MLIR compiler type is used by default, and is the recommended compiler to use.

You can use benchmark app to test if the model is successfully compiled:
`/data/build_samples/aarch64/benchmark_app -d VPUX -m model.blob -t 10`

Notes

- Due to tools being in development, this document will likely change.
- Due to the development state of the tools, an error could occur in either of the stages above. Feel free to reach out on Discuss (preferred) or Discord for help.

- More complex models compiled with MLIR might be slow (slower than on MX). We are actively working on that. It is recommended to use standard activation functions (ReLU) over newer ones (SiLU), as they are more optimized on the RVC3.
- There is a known limitation where opset8 is not supported on RVC3. Model optimizers can produce XML which uses Opset8 for operations like MaxPool or Softmax. While Luxonis will optimize that and potentially solve that through Blobconverter, no optimizations at this moment exist. If you get an XML with Opset8, check OpenVINO operations for equivalent operation of lower Opset, and manually modify the XML. Luxonis will be happy to help you out with this process through Discuss.
- Other limitations will be exposed here.

Specific models - additional steps

1. YoloV6n

YoloV6 has two releases. If you use the latest version, please look for Release 2 (R2) instructions. If you are using an older version, check Release 1 (R1). Export process is different only in the first step.

Step 1 (Generating IR) - YoloV6n Release 1 (deprecated):

- *Upload trained weights .pt to tools.luxonis.com, choose YoloV6, write down your input shape, and export. Downloaded .zip will contain an .onnx model. Extract this in directory you want.*
- *Generate IR with OV 22.1:*
  ```
  mo --input_model bestyolov6-simplified.onnx --output_dir FP16
  --model_name yolov6n --data_type FP16 --reverse_input_channel
  --scale 255 --output
  output1_yolov6,output2_yolov6,output3_yolov6
  ```
- ***IMPORTANT!** Look for the produced XML. You will need to rename the output layers in .xml produced in step 1 (they get automatically renamed due to 22.1 OV). Look for layers with `type="Result"`. There will be at least three. First could for example have an attribute `name="259/sink_port_0"`. Rename it to `name="output1_yolov6"` Now search for `name**s**="259"` and rename it to `names="output1_yolov6"`. Do the same for the remaining two Result layers.*

Step 1 (Generating IR) - YoloV6n Release 2 and Release 3:

- Upload trained weights .pt to tools.luxonis.com, choose YoloV6, write down your input shape, and export. Downloaded .zip will contain .bin and .xml files. Extract this in the directory you want.

Step 2 (optional - quantize the model):

- Follow the steps discussed in the previous section. In JSON, `"target_device"` must be set to `"VPU"`.

Step 3 (Compilation - on RVC3)

- **IMPORTANT!** Move xml and bin to RVC3.
- Follow the steps discussed in the previous section.

2. Lightweight OpenPose

Step 1 (Generating IR):

- Navigate to the root directory of GitHub project
- Use checkpoints from a trained model and first convert them to ONNX using:
  `PYTHONPATH=./ python3 scripts/convert_to_onnx.py --checkpoint-path checkpoint_iter_370000.pth`.
  This will generate a `human-pose-estimation.onnx` model.
- Activate the OpenVINO environment and call:
  `mo --input_model human-pose-estimation.onnx --input data --mean_values data[128.0,128.0,128.0] --scale_values data[256] --output stage_1_output_0_pafs,stage_1_output_1_heatmaps --data_type FP16 --output_dir FP16`.
  This will generate .xml and .bin files in FP16 directory.

Step 2 (optional - quantize the model):

- Follow the steps discussed in the previous section. In JSON, `"target_device"` must be set to `"ANY"`.

Step 3 (Compilation - on RVC3)

- **IMPORTANT!** Move xml and bin to RVC3.
- Follow the steps discussed in the previous section.