Building OpenVINO™-based Object Detection Demo in C++ using g++ Compiler

Technical Paper

January 2024

Author:
Ramesh Perumal

Document number: 813409 -1.0

Contents

Tables

Figures

Revision History

Date	Revision	Description
January 2024	1.0	Initial release.

1.0 Introduction

The OpenVINO™-based C++ demos in Open Model Zoo are built using CMake that allows developers to build simple to complex software across multiple platforms with a single set of input files. In real business use cases, the OpenVINO™ applications are integrated into the end user's target application and the build steps may vary depending on the build tool in the production setup. However, if the end users are using g++ compiler to build their applications, they do not want to change their build tool. To facilitate this, this paper presents the steps to build the C++ object detection demo using g++ compiler on Ubuntu 20.04. The inference outputs of the resulting executable are demonstrated using YOLOv-3-tiny model to verify the proposed method.

1.1 Acronyms

Table 1. Acronyms

Term	Description
OpenVINO™	Open Visual Inference & Neural Network Optimization
DUT	Device Under Test

1.2 Reference Documents

Log in to the Resource and Documentation Center (rdc.intel.com) to search and download the document numbers listed in the following table. Contact your Intel field representative for access.

Note: Third-party links are provided as a reference only. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether the referenced data is accurate.

Table 2. Reference Document

Document	Document No./Location
OpenVINO™ Toolkit	https://software.seek.intel.com/openvino-toolkit
Install OpenVINO™ Runtime on Linux from an Archive File	https://docs.openvino.ai/2023.0/openvino_docs_install_guides_installing_openvino_from_archive_linux.html
Open Model Zoo	https://github.com/openvinotoolkit/open_model_zoo/tree/master
yolo-v3-tiny-tf	https://docs.openvino.ai/2023.2/omz_models_model_yolo_v3_tiny_tf.html
C++ Object Detection Demo	https://github.com/openvinotoolkit/open_model_zoo/tree/master/demos/object_detection_demo/cpp
Building C++ Demos on Linux using CMake	https://github.com/openvinotoolkit/open_model_zoo/tree/master/demos#build-the-demo-applications-on-linux
CMake	https://cmake.org/
g++ Compiler	https://gcc.gnu.org/

Table 3. Device Under Test

Model	Intel® NUC 11 Pro Kit (NUC11TNHi3)
CPU	Intel® Core™ i3-1115G4 x 4
GPU	Intel® UHD Graphics
Memory	16 GB
OS	Ubuntu 20.04 LTS
OpenVINO™	Version 2023.0.2

§

2.0 Building C++ Object Detection Sample using g++

2.1 Install OpenVINO™ Runtime on Linux from an Archive File

1. Create the `/opt/intel` folder for OpenVINO™

```
sudo mkdir /opt/intel
```

2. Move to the current user's *Downloads* folder and download the OpenVINO™ Runtime archive file, extract the files, rename the extracted folder and move it to the desired path.

```
curl -L
https://storage.openvinotoolkit.org/repositories/openvino/packag
es/2023.0.2/linux/l_openvino_toolkit_ubuntu20_2023.0.2.11065.e66
2b1a3301_x86_64.tgz --output openvino_2023.0.2.tgz
tar -xf openvino_2023.0.2.tgz
sudo mv
l_openvino_toolkit_ubuntu20_2023.0.2.11065.e662b1a3301_x86_64
/opt/intel/openvino_2023.0.2
```

3. Install required system dependencies.

```
cd /opt/intel/openvino_2023.0.2
sudo -E ./install_dependencies/install_openvino_dependencies.sh
```

4. Create a symbolic link to the OpenVINO installation folder for simplicity.

```
cd /opt/intel
sudo ln -s openvino_2023.0.2 openvino_2023
```

5. Configure the environment before you can compile and run OpenVINO™ applications.

```
source /opt/intel/openvino_2023/setupvars.sh
```

2.2 Build Steps

1. Create a new folder demo with the *main.cpp* file and *common* folder containing the dependencies required for running the object detection demo in Open Model Zoo repository.

2. Delete the files *query_wrapper.cpp* and *query_wrapper.h* in *common/cpp/monitors/src* as these two files are required only on Windows OS.

3. Use `pkg-config --libs openvino` and `pkg-config --libs opencv4` to get the library path and libraries to be linked for OpenVINO™ and OpenCV.

4. Compile the object detection sample *main.cpp* using g++.

```
g++ main.cpp ./common/cpp/utils/src/*.cpp
./common/cpp/models/src/*.cpp ./common/cpp/monitors/src/*
./common/cpp/pipelines/src/*.cpp -I/usr/include/opencv4 -
I/opt/intel/openvino_2023/runtime/include -
I/home/<user>/demo/common/cpp/models/include -
I/home/<user>/demo/common/cpp/monitors/include -
I/home/<user>/demo/common/cpp/pipelines/include -
I/home/<user>/demo/common/cpp/utils/include -
L/opt/intel/openvino_2023.0.1/runtime/lib/intel64/ -
lopenvino_onnx_frontend -lopenvino_paddle_frontend -
lopenvino_pytorch_frontend -lopenvino_tensorflow_frontend -
lopenvino_tensorflow_lite_frontend -lopenvino_c -lopenvino -
L/usr/lib/x86_64-linux-gnu/ -lopencv_stitching -lopencv_aruco -
lopencv_bgsegm -lopencv_bioinspired -lopencv_ccalib -
lopencv_dnn_objdetect -lopencv_dnn_superres -lopencv_dpm -
lopencv_highgui -lopencv_face -lopencv_freetype -lopencv_fuzzy -
lopencv_hdf -lopencv_hfs -lopencv_img_hash -
lopencv_line_descriptor -lopencv_quality -lopencv_reg -
lopencv_rgbd -lopencv_saliency -lopencv_shape -lopencv_stereo -
lopencv_structured_light -lopencv_phase_unwrapping -
lopencv_superres -lopencv_optflow -lopencv_surface_matching -
lopencv_tracking -lopencv_datasets -lopencv_text -lopencv_dnn -
lopencv_plot -lopencv_ml -lopencv_videostab -lopencv_videoio -
lopencv_viz -lopencv_ximgproc -lopencv_video -lopencv_xobjdetect
-lopencv_objdetect -lopencv_calib3d -lopencv_imgcodecs -
lopencv_features2d -lopencv_flann -lopencv_xphoto -lopencv_photo
-lopencv_imgproc -lopencv_core -lgflags -o main.o
```

The resulting executable *main.o* is saved in the current folder.

§

3.0 Verification of Inference Results

1. Create a Python* virtual environment and install the OpenVINO™ Development Tools.

```
python -m venv ov_venv
source ov_venv/bin/activate
python -m pip install --upgrade pip
pip install openvino-dev[tensorflow]==2023.0.2
```

2. Download and optimize the YOLO-v3-tiny model.

```
omz_downloader --name yolo-v3-tiny-tf
omz_converter --name yolo-v3-tiny-tf
```

3. Download the COCO dataset label file *coco_80cl.txt* .

4. Activate the OpenVINO™ environment and run the compiled object detection executable with the optimized YOLO-v3-tiny model on the test input image.

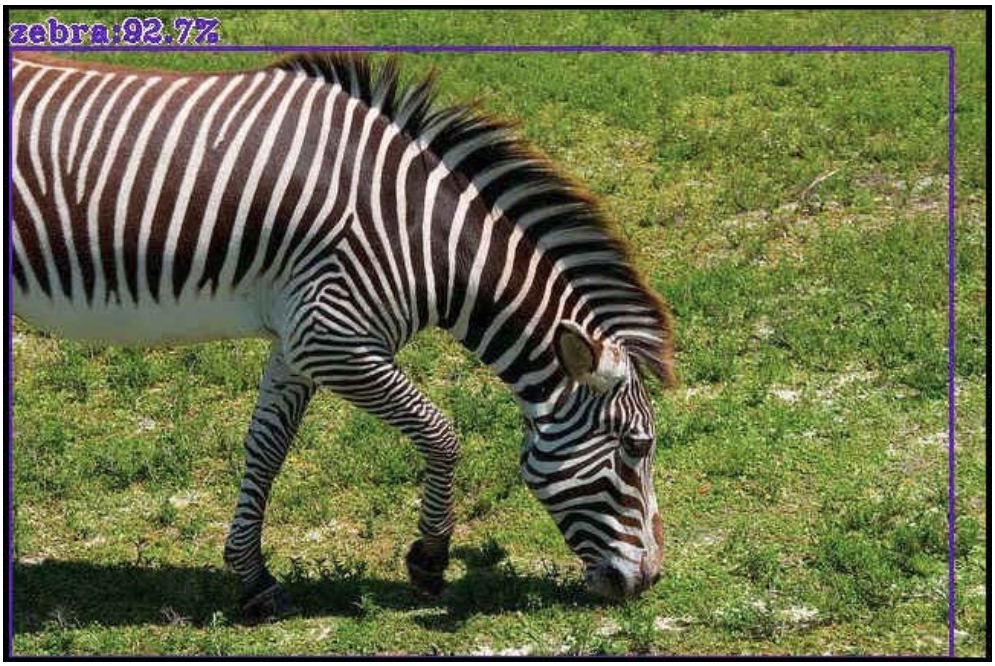
```
source /opt/intel/openvino_2023/setupvars.sh

./main.o -i <test_input.jpg> -m <yolo-v3-tiny-tf.xml> -at yolo -
r -o detection_result.jpg -labels coco_80cl.txt
```

Figure 1. Detection Results of the Compiled Executable with YOLO-v3-tiny Model

A. Command-Line Output

```
[ INFO ] OpenVINO
[ INFO ]        version: 2023.0.1
[ INFO ]        build: 2023.0.1-11005-fa1c41994f3-releases/2023/0
[ INFO ] Reading model /home/ramesh/big_basket/yolo_V3_tiny/public/yolo-v3-tiny-tf/FP16/yol
o-v3-tiny-tf.xml
[ INFO ] Model name: yolo-v3-tiny-tf
[ INFO ]        Inputs:
[ INFO ]                image_input, f32, [1,416,416,3], [N,H,W,C]
[ INFO ]        Outputs:
[ INFO ]                conv2d_12/Conv2D/YoloRegion, f32, [1,255,26,26], [...]
[ INFO ]                conv2d_9/Conv2D/YoloRegion, f32, [1,255,13,13], [...]
[ INFO ] The model /home/ramesh/big_basket/yolo_V3_tiny/public/yolo-v3-tiny-tf/FP16/yolo-v3
-tiny-tf.xml is loaded to CPU
[ INFO ]        Device: CPU
[ INFO ]                Number of streams: 1
[ INFO ]                Number of threads: 2
[ INFO ]        Number of inference requests: 2
[ DEBUG ]  ------------------- Frame # 0--------------------
[ DEBUG ]  Class ID  | Confidence | XMIN | YMIN | XMAX | YMAX
[ DEBUG ]  zebra     | 0.926795   | 0    | 25   | 618  | 425
OpenCV: FFMPEG: tag 0x47504a4d/'MJPG' is not supported with codec id 7 and format 'image2 /
 image2 sequence'
[ INFO ] Metrics report:
[ INFO ]        Latency: 27.7 ms
[ INFO ]        FPS: 36.1
[ INFO ]        Decoding:       2.1 ms
[ INFO ]        Preprocessing:  0.4 ms
[ INFO ]        Inference:      27.0 ms
[ INFO ]        Postprocessing: 0.1 ms
[ INFO ]        Rendering:      0.3 ms
```

B. Detection Output



§

4.0 Conclusion

This technical paper covers the steps for building a C++ object detection sample using g++ compiler. This facilitates a smooth integration of OpenVINO™ C++ sample into the target application and enables the users to keep using their existing g++ compiler in their production setup. Furthermore, the compiled executable is verified by demonstrating the object detection results with the optimized YOLO-v3-tiny model.

§