Optimization of YOLOv4 and YOLOv4-tiny Models with Customized Configurations using Intel® Distribution of OpenVINO™ Toolkit

White Paper

August 2023

Document number: 787419-1.0

Contents

Tables

Figures

Revision History

Date	Revision	Description
August 2023	1.0	Initial release

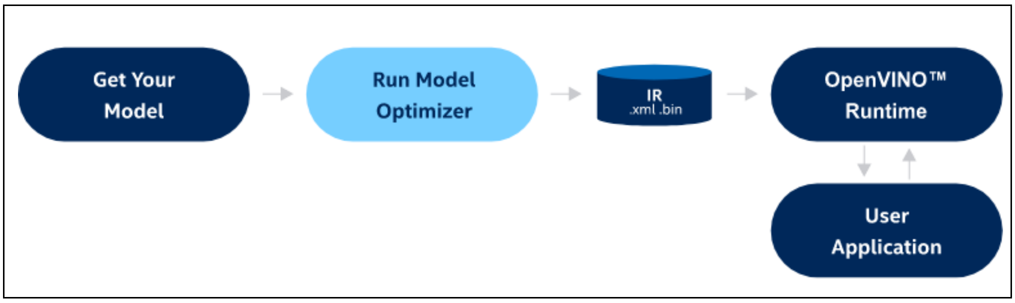
§

1.0 Introduction

This document presents two methods for optimizing the YOLOv4 and YOLOv4-tiny models with customized classes and anchors using Intel® Distribution of OpenVINO™ Toolkit. The object detection use case based on the optimized models is used to verify the corresponding inference results. Furthermore, the decreased inference time of the optimized model is demonstrated with the real-time use case of an ISV.

The Intel® Distribution of OpenVINO™ Toolkit enables developers to quickly optimize and deploy the AI workloads with improved performance across the Intel® platforms from edge to cloud. The following figure illustrates the workflow for optimizing and deploying a pre-trained model using Model Optimizer and OpenVINO™ Runtime API.

Figure 1. Model Optimization and Deployment Workflow



Model Optimizer is a Python-based tool that creates an intermediate representation (IR) of the model to facilitate the optimal execution of inference across the Intel® devices (CPU, GPU, VPU). The OpenVINO™ Runtime API contains the hardware-specific plugins for implementing the inference with the optimized model in IR format. This paper covers the BKMs for optimizing YOLOv4 and YOLOv4-tiny models with customized configurations and demonstrates the decreased inference time of the optimized YOLOv4-tiny model with the license plate detection use case.

1.1 Acronyms

Table 1. Acronyms

Term	Description
BKM	Best Known Method
OpenVINO™	Open Visual Inference & Neural Network Optimization
ISV	Independent Software Vendor
DUT	Device Under Test
16-bit Floating Point	FP16

Term	Description
32-bit Floating Point	FP32
ALPR	Automatic License Plate Recognition

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
YOLOv4 Model Optimization in OpenVINO™ documentation	https://docs.openvino.ai/2022.2/openvino_docs_MO_DG_prepare_model_convert_model_tf_specific_Convert_YOLO_From_Tensorflow.html#converting-a-yolov4-model-to-ir
Object Detection Demo in Open Model Zoo	https://github.com/openvinotoolkit/open_model_zoo/tree/master/demos/object_detection_demo/python
Pre-trained YOLOv4-tiny model for fire detection	https://github.com/ngocdaumai/Fire-Detection-Using-YoloV4/tree/main
Darknet Framework	https://github.com/AlexeyAB/darknet
Tensorflow* Keras YOLOv4 Model	https://github.com/david8862/keras-YOLOv3-model-set

Table 3. Devices Under Test

DUT-1	Model	NUC11TNHv5
	CPU	Intel® Core™ i5-1145G7 x 8
	GPU	Intel® Iris® Xe Graphics
	Memory	16 GB
	OS	Ubuntu 20.04 LTS
	OpenVINO™	2022.2.0
DUT-2	CPU	Intel® Core™ i7-10750H x 12

	GPU	Intel® UHD Graphics
	Memory	16 GB
	OS	Ubuntu 20.04 LTS
	OpenVINO™	2022.2.0

§

2.0 Optimization of YOLOv4 and YOLOv4-tiny Models

2.1 Prerequisites

a. Create a Python* virtual environment and upgrade the *pip* version in DUT-1

```
python -m venv ov_venv
source ov_venv/bin/activate
python -m pip install --upgrade pip
```

b. Install OpenVINO™ Development Tools including the model optimization utility OpenVINO™ Runtime, and TensorFlow

```
pip install openvino-dev[tensorflow]==2022.2.0
```

2.2 Optimization of Models with Customized Classes

The YOLOv4 models are trained in the Darknet framework and consist of two files: *.cfg* file with model configurations and *.weights* file with model weights. Usually, the models are pre-trained on the customized dataset by changing the input shape or the number of classes in the configuration file. For these cases, the following steps are used to convert the YOLOv4/YOLOv4-tiny model into IR format.

a. Download the models

i. YOLOv4

```
omz_downloader --name yolo-v4-tf
```

The above command creates the *public/yolo-v4-tf* folder with the model weights (*yolov4.weights*) trained on COCO dataset and the keras-YOLOv3-model-set repository for converting the model into Tensorflow's SavedModel format. Replace *yolov4.weights* and *yolov4.cfg* (in *public/yolo-v4-tf/keras-YOLOv3-model-set/cfg*) with the customized YOLOv4 *weights* and *cfg* files.

ii. YOLOv4-tiny

```
omz_downloader --name yolo-v4-tiny-tf
```

The above command creates the *public/yolo-v4-tiny-tf* folder with the model weights (*yolov4-tiny.weights*) trained on COCO dataset and the keras-YOLOv3-model-set repository for converting the model into Tensorflow's protobuf binary format. Replace *yolov4-tiny.weights* and *yolov4-tiny.cfg* (in *public/yolo-v4-tiny-tf/keras-YOLOv3-model-set/cfg*) with the customized YOLOv4-tiny *weights* and *cfg* files.

b. Convert the models from Darknet framework into IR format with FP32 and FP16 precisions

   i. YOLOv4

```
omz_converter --name yolo-v4-tf
```

   This command achieves the YOLOv4 model conversion in two steps: (1) Converts the model weights from Darknet into TensorFlow's SavedModel format; (2) Converts the model from TensorFlow into IR format with FP32 and FP16 precisions using Model Optimizer.

   ii. YOLOv4-tiny

```
omz_converter --name yolo-v4-tiny-tf
```

   This command achieves the YOLOv4-tiny model conversion in three steps: (1) Converts the model weights from Darknet into Keras (*.h5*) format; (2) Converts the model from Keras into Tensorflow's protobuf binary (*.pb*) format; (3) Converts the model from Tensorflow into IR format with FP32 and FP16 precisions using Model Optimizer.

2.2.1 Verification of Inference Results

For the verification purpose, the pre-trained YOLOv4-tiny model for fire detection is optimized according to Section 2.2, and the detection results are presented using the object detection demo in Open Model Zoo repository.

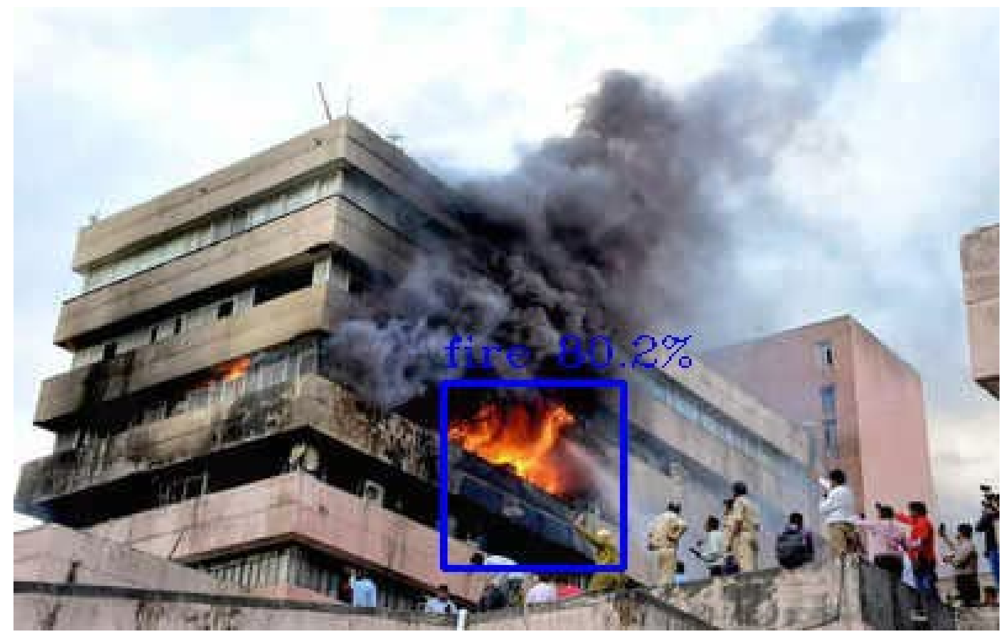
a. Clone the Open Model Zoo repository

```
git clone --recurse-submodules
https://github.com/openvinotoolkit/open_model_zoo.git
```

b. Verify the detection results of the optimized models in IR format using the object detection demo

```
python3
open_model_zoo/demos/object_detection_demo/python/object_d
etection_demo.py -d CPU -i test_input.jpg -m public/yolo-
v4-tiny-tf/FP16/yolo-v4-tiny-tf.xml -at yolov4 -r --labels
class_names.txt -o pred_out.jpg --no_show -nstreams 1 -
nireq 1
```

Figure 2. Detection Result of the Optimized YOLOv4-tiny Model



The detection result in Figure 2clearly illustrates that the optimized YOLOv4-tiny model can detect fire in the input image.

2.3 Optimization of Models with Customized Classes and Anchors

The values of anchors in the YOLOv4/YOLOv4-tiny model configuration file depend on the aspect ratio of the objects in the training dataset. Moreover, they determine the position of bounding box in the detection results. The default anchors of the two models trained on the COCO dataset are presented in Table 4.

Table 4. Default Anchors in YOLOv4 and YOLOv4-tiny Models

Model	Anchors
YOLOv4	12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
YOLOv4-tiny	10,14, 23,27, 37,58, 81,82, 135,169, 344,319

If the aspect ratio of the objects in the use case varies drastically from those in the COCO dataset, then the default anchors may not predict the accurate bounding boxes in the detection results. In these cases, the anchors are computed from the user's dataset and the method discussed in Section 2.2 may not give accurate detection results. So, the following steps are proposed to accommodate the changes in anchors while post-processing the inference results. The optimization steps are slightly modified from those in the OpenVINO™ documentation to get the accurate detection results with the customized inference script.

a. Clone the keras-YOLOv3-model-set repository

```
git clone https://github.com/david8862/keras-YOLOv3-model-
set
```

b. Convert the model to the TensorFlow's SavedModel format

```
python3 keras-YOLOv3-model-
set/tools/model_converter/convert.py  yolov4-tiny-
custom.cfg yolov4-tiny-custom.weights yolov4-tiny-tf2 --
yolo4_reorder
```

The execution of the above command creates the text file *yolov4-tiny-tf2_anchors.txt* containing the customized anchors and the protobuf model and variables are saved in the folder *yolov4-tiny-tf2*. To convert the YOLOv4 model, simply replace the model *weights* and *cfg* file.

c. Convert the model from Tensorflow's SavedModel into IR formats with FP32 and FP16 precisions using Model Optimizer

```
mo --saved_model_dir yolov4-tiny-tf2 --output_dir yolov4-
tiny-FP32 --input_shape [1,416,416,3] --model_name yolov4-
tiny --input=image_input

mo --saved_model_dir yolov4-tiny-tf2 --output_dir yolov4-
tiny-FP16 --compress_to_fp16 --input_shape [1,416,416,3] -
-model_name yolov4-tiny --input=image_input
```

If the model input shape differs from the default value, set the value of *input_shape* parameter to the target input shape of the model.

2.3.1 Verification of Inference Results

The Python script *yolo_v4_ov_inference.py* is developed by updating the existing script *yolo.py* in keras-YOLOv3-model-set repository with the following changes:

1. Comment out the lines 12, 14-17, 20-22, 24-25, 33, 68

2. Insert the following line in the import statements to use the OpenVINO™ Runtime API

```
from openvino.runtime import Core
```

3. Replace the *_generate_model(self)* function in *YOLO_np* class with the following to load and compile the optimized YOLOv4/YOLOv4-tiny model using OpenVINO™ Runtime

```
def _generate_model(self):
   weights_path = os.path.expanduser(self.weights_path)
   core = Core()
   # Read the optimized model
   model = core.read_model(weights_path)
   # Load the model on the target device
   compiled_model = core.compile_model(model, 'CPU')
   return compiled_model
```

4. Replace the *predict(self, image_data, image_shape)* function in *YOLO_np* class with the following to post process the inference results using

```
def predict(self, image_data, image_shape):
   num_anchors = len(self.anchors)

   output_blob = []
   for i in range(len(self.yolo_model.outputs)):
      output_blob.append(self.yolo_model.output(i))

   output = self.yolo_model([image_data])
   predictions = []
   for i in range(len(output_blob)):
      predictions.append(output[output_blob[i]])

   out_boxes, out_classes, out_scores =
   yolo3_postprocess_np(predictions, image_shape,
   self.anchors, len(self.class_names),
   self.model_input_shape, max_boxes=100,
   confidence=self.score, iou_threshold=self.iou,
   elim_grid_sense=self.elim_grid_sense)
   return out_boxes, out_classes, out_scores
```

5. Insert the following line in *detect_img(yolo)* function to save the detection results

```
r_image.save('prediction_out.jpg')
```

6. Verify the detection results with the inference script

   a. Copy the customized inference script *yolo_v4_ov_inference.py* to the folder *keras-YOLOv3-model-set*.

   b. Install the dependencies required for running the inference script

```
pip install keras_applications imgaug
```

   c. Run the inference script

```
python3 yolo_v4_ov_inference.py --weights_path <path to
yolov4-tiny.xml> --classes_path <path to class_name.txt> -
-image --anchors_path <path to yolov4-tiny-
tf2_anchors.txt>
```

§

3.0 Performance Improvement in Real-Time Object Detection Use Case

3.1 Use Case

Automatic License Plate Recognition (ALPR) is commonly used in security barrier use cases to monitor the entry of vehicles into/out of the target workspace. It uses the deep learning models for vehicle detection, license plate detection, and optical character recognition for recognizing the license plate number. An ISV would like to increase the inference speed of license plate detection using OpenVINO™ toolkit. However, the license plate detection is based on the YOLOv4-tiny model with customized anchors and classes, where the anchors are computed from their dataset. The subsequent section demonstrates the improved detection accuracy and inference speed using the method presented in Section 2.3.

3.2 Performance Improvements

First, the pre-trained YOLOv4-tiny model provided by the ISV is optimized according to Section 2.2. The corresponding detection result in Figure 3 shows the inaccurate prediction of bounding box (in blue) for the target license plate in the given image. On the other hand, the YOLOv4-tiny model optimized by the steps in Section 2.3 enables accurate prediction of bounding box (in red) which is aligned closely to the license plate as shown in Figure 4. Thus, the proposed method in Section 2.3 significantly improves the detection accuracy as the post-processing steps in the inference script are customized based on the anchors.

Figure 3. Detection Result with Inaccurate Prediction of Bounding Box

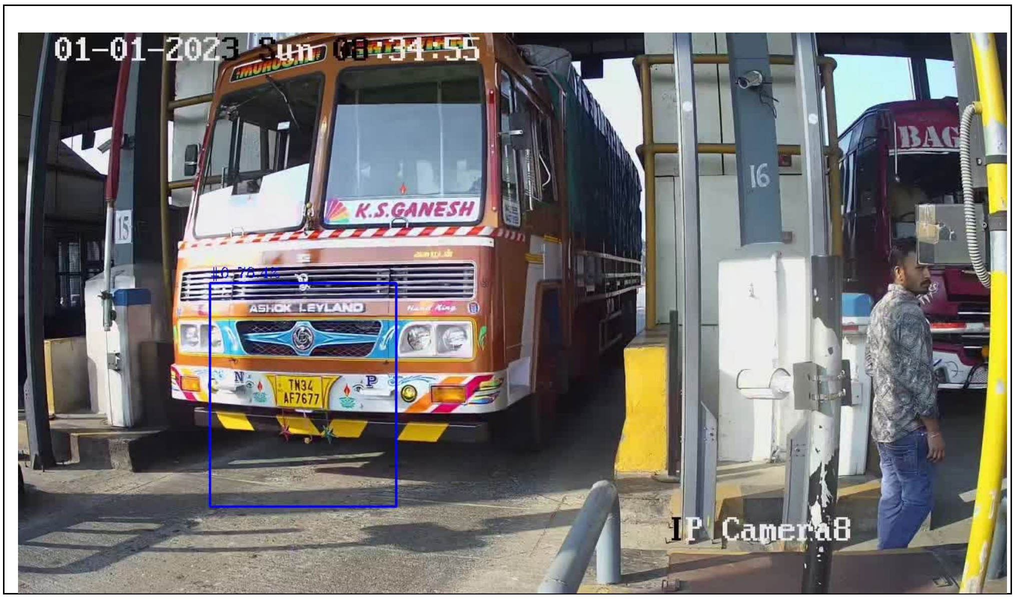
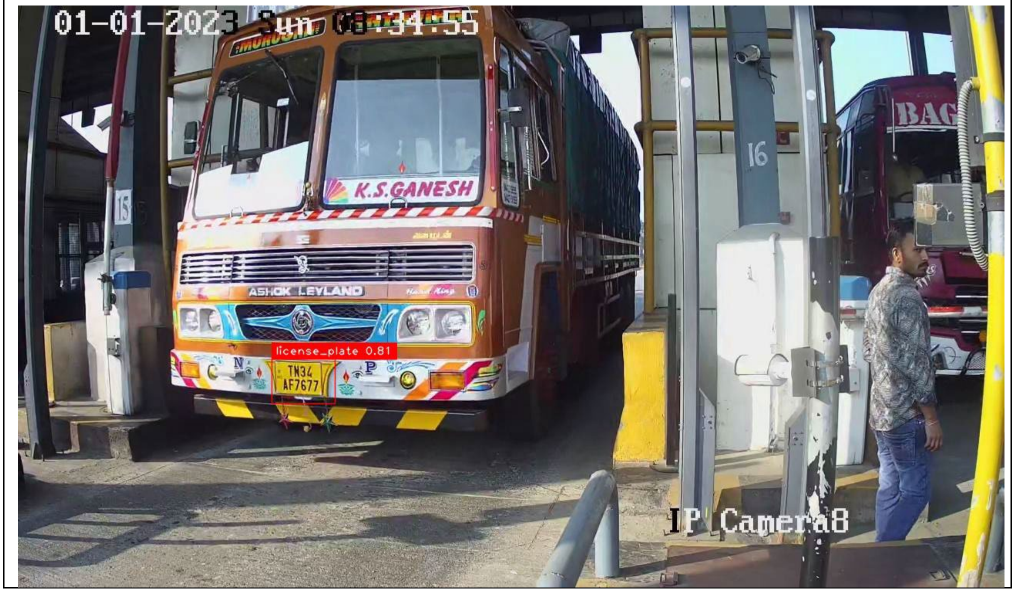


Figure 4. Detection Result with Accurate Prediction of Bounding Box



Moreover, the ISV reported that the inference time per image is decreased by **98%** from 680ms (using Darknet framework) to **16ms** with the optimized YOLOv4-tiny model on DUT-2 (Intel® Core™ i7-10750H CPU).

§

4.0 Conclusion

This white paper presents the BKMs for optimizing YOLOv4 and YOLOv4-tiny models with customized classes and anchors using OpenVINO™ toolkit. This also covers the steps to verify the inference results of the optimized models. The resulting performance improvement is demonstrated through the YOLOv4-tiny-based license plate detection in the real-time ALPR use case of an ISV. The results reveal that the proposed method improves the detection accuracy while decreasing the inference time by **98%**.

§