

BUILDING IMAGE CLASSIFIER AND OBJECT DETECTORS USING INTEL[®] NEURAL COMPUTE STICK 2 AND OPENVINO[™] TOOLKIT

LEGAL DISCLAIMER

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at [intel.com](https://www.intel.com).

This sample source code is released under the Intel Sample Source Code License Agreement.

Intel, the Intel logo, and OpenVINO are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

AGENDA

Chapter Outcome: Deploy an object detector model on Intel® Neural Compute Stick 2

Part one

- What is an Object Detector?
- Popular object detection topologies
- OpenVINO™ toolkit support for object detection

Part two

- Deploy pretrained YOLO* v3 model on Intel Neural Compute Stick 2 using the OpenVINO Toolkit

PART 1: UNDERSTANDING OBJECT DETECTION

A MORE CHALLENGING COMPUTER VISION PROBLEM

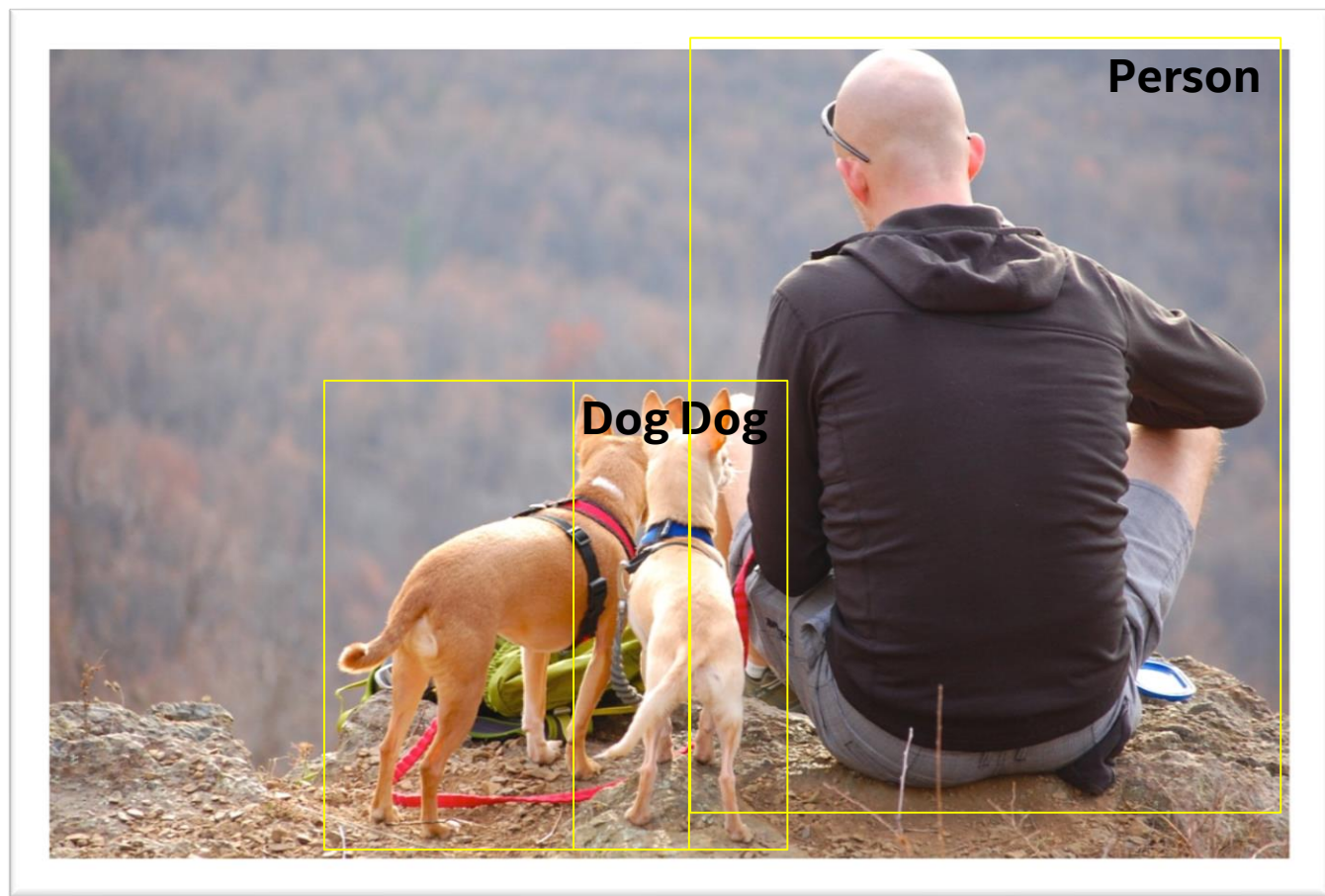
Multiple objects/subjects in a single image



Man with dogs image source: <https://pixabay.com/en/man-dogs-hiking-edge-cliff-1181873/>

OBJECT DETECTION

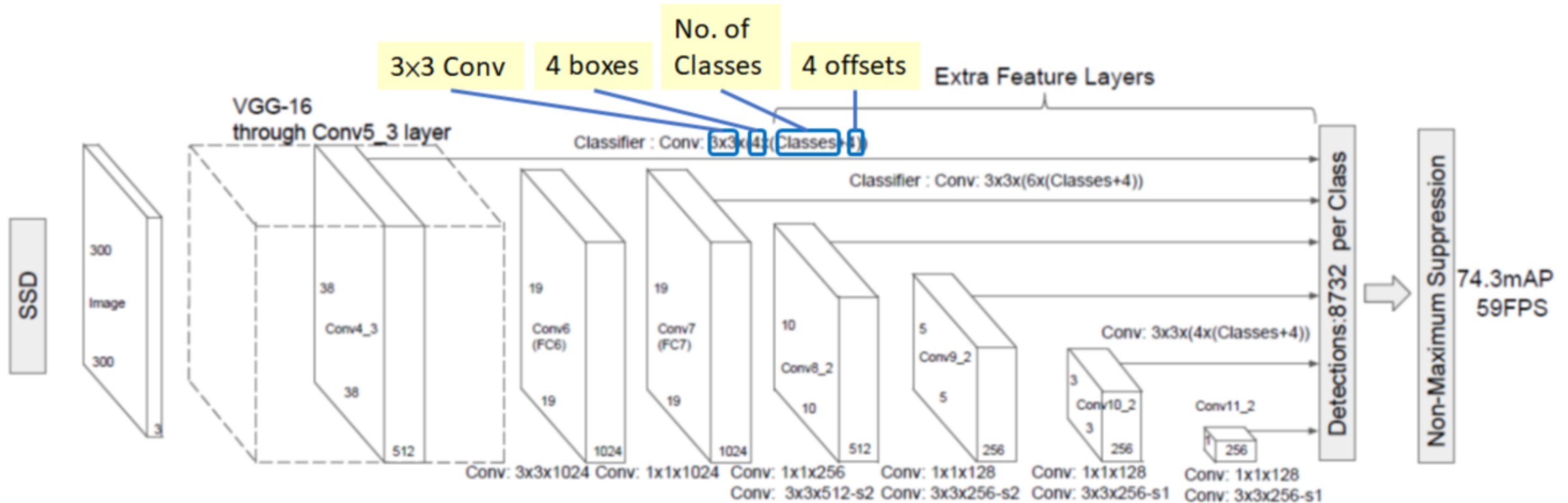
Identify different objects in an image, and locate them



Man with dogs image source: <https://pixabay.com/en/man-dogs-hiking-edge-cliff-1181873/>

POPULAR OBJECT DETECTION TOPOLOGIES

Single Shot MultiBox Detector (SSD) – Network Architecture



<https://arxiv.org/pdf/1512.02325.pdf>

SINGLE SHOT MULTIBOX DETECTOR—SSD

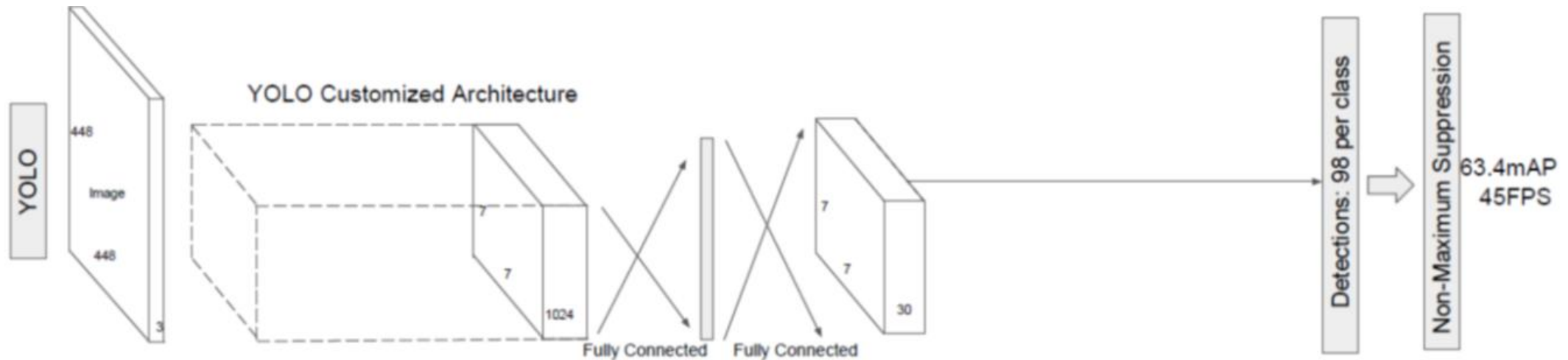
- Builds on VGG-16 as the “base network”.
- Replaces fully connected layers in the VGG-16 topology with auxiliary feature layers that extract features at multiple scales.
- Uses a default set of bounding box shapes at various aspect ratios and compares these to the bounding box in the ground truth image to identify localization.
- Therefore, each added feature layer produces either (a) a prediction score for a category, or (b) shape offset relative to default box coordinates. This implies that each forward pass classifies as well as localizes the object.

<https://arxiv.org/pdf/1512.02325.pdf>

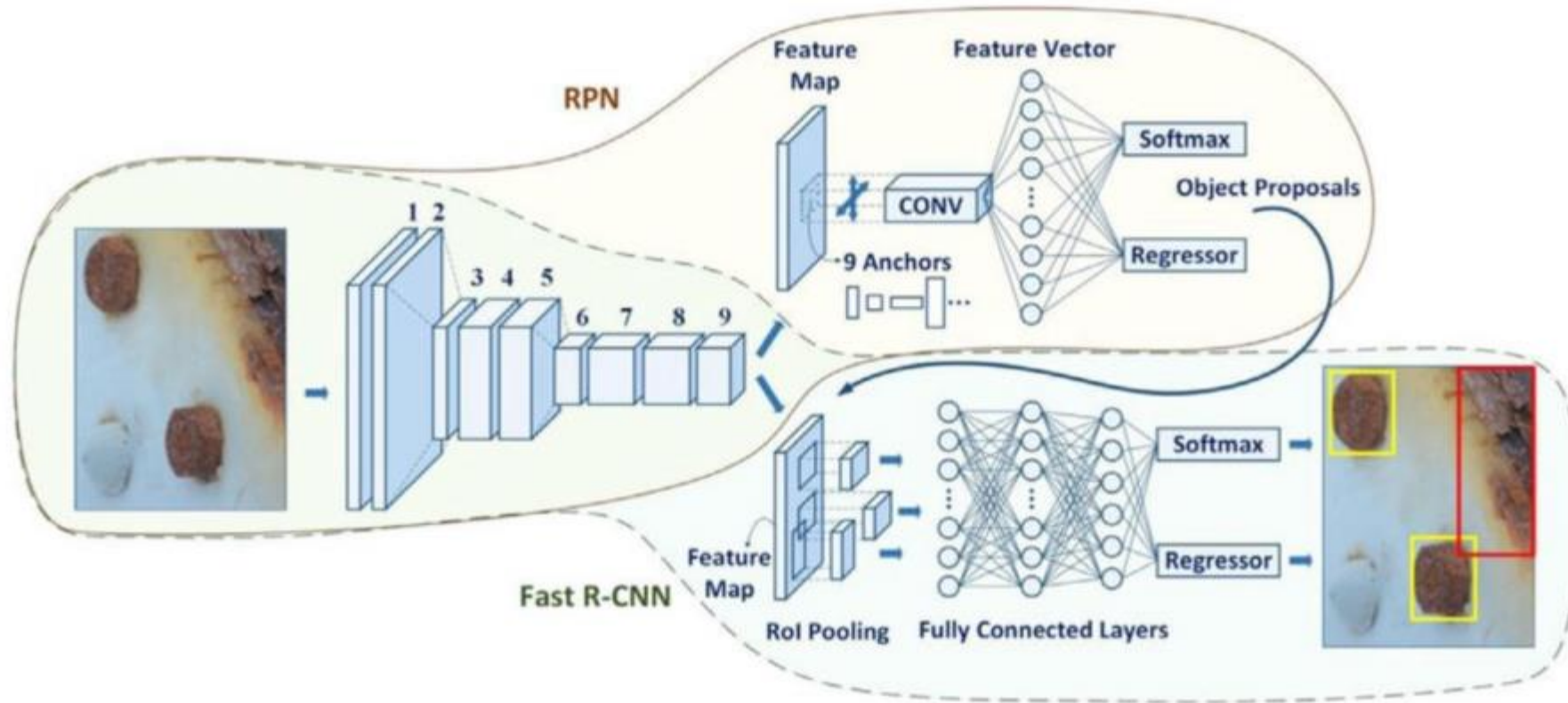
YOU ONLY LOOK ONCE (YOLO)

Network Architecture

- Applies a single neural network to the entire image
- The image is divided into smaller regions and the network makes predictions for bounding boxes and classification labels
- The bounding boxes are weighted by predicted probabilities
- Read more about YOLO and test some samples:
<https://pjreddie.com/darknet/yolo/>



FASTER R-CNN—NETWORK ARCHITECTURE



FASTER R-CNN

The Regional Proposal Network is a deep, fully convolutional network that first generates regional proposals using a selective search

- Takes an input image of any size and outputs a set of rectangular object proposals

These regional proposals are fed to a Fast R-CNN

- The RoI (Region of Interest) Pooling phase accepts a set of feature maps and the regional proposals
- The pooled area is fed through a convolutional neural network (CNN) and two fully connected layers to identify the final bounding box and class labels

SUPPORTED OBJECT DETECTION TOPOLOGIES

SUPPORTED IMAGE CLASSIFICATION TOPOLOGIES

Most common object detection topologies are supported through the OpenVINO™ Toolkit.

- SSD MobileNet
- SSD Lite MobileNet
- SSD tResNe50
- Faster R-CNN Inception V2/ResNet*
- Faster R-CNN NasNet COCO
- Mask R-CNN Inception ResNet V2 COCO
- Mask R-CNN ResNet 50 COCO
- Faster R-CNN Inception ResNet V2 Open Images
- Faster R-CNN ResNet 101 AVA v2.1

For a complete list of all supported topologies, refer to the [OpenVINO™ Toolkit documentation](#)

PART 2: DEPLOYING AN OBJECT DETECTION MODEL ON INTEL[®] NEURAL COMPUTE STICK 2 USING THE OPENVINO[™] TOOLKIT

WORKFLOW—DEPLOY OBJECT DETECTOR USING OPENVINO™ TOOLKIT

Step 1: Freeze the TensorFlow* model

- Obtain Darknet framework configuration, YOLO* v3 weights, and COCO labels, and freeze them into a frozen graph format (.pb file)

Step 2: Create intermediate representation (IR) files through the Model Optimizer

- Using the frozen graph from Step 2, generate the .bin and .xml files.
- Alternative: Use the frozen graph from the Model Downloader

Step 3: Test the model with the IR files

- Use the Inference Engine in the target environment via OpenVINO™ toolkit sample applications
- **We will focus on deployment to the Intel® Neural Compute Stick 2 (Intel® NCS2)**

Step 4: Integrate the Inference Engine in your application

- Deploy the model in the target environment (CPU, processor graphics, Intel NCS2, field-programmable gate array (FPGA))
- We will cover this in Chapter 5 when we learn how to deploy custom models

STEP 1: FREEZE THE TENSORFLOW* MODEL

In this chapter, we will use a pretrained YOLO* v3 model to demonstrate how an object detector can be deployed

For custom models, refer to Chapter 5.

YOLO v3 model is trained using the Darknet framework. Generating a frozen graph for this model requires:

- Downloading the YOLO v3 framework configuration from the YOLO git
- Obtaining the classification labels (COCO labels) against which the model is trained
- Pretrained weights for the YOLO v3 model
- Run a converter using the above inputs to create a protobuf file

STEP 1: FREEZE THE TENSORFLOW* MODEL

Instructions

(a) Download the YOLO* v3 framework configuration

```
mkdir yolov3  
cd yolov3  
git clone https://github.com/mystic123/tensorflow-yolo-v3.git  
cd tensorflow-yolo-v3
```

- This provides some tools, like the `convert_weights_pb.py`, that builds a Darknet model
- We will also use this file for creating a frozen graph in a later step

(b) Download the YOLO v3 weights

```
wget https://pjreddie.com/media/files/yolov3.weights
```

STEP 1: FREEZE THE TENSORFLOW* MODEL

Instructions (continued)

(c) Download the COCO labels file

```
wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
```

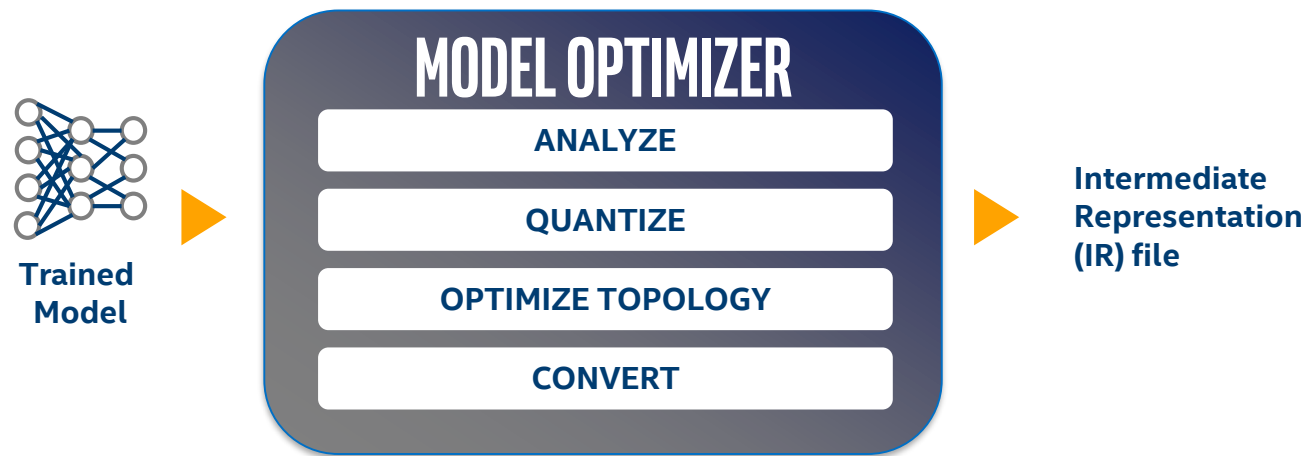
- This gives a list of 80 labels that the model is trained with (<https://github.com/pjreddie/darknet/blob/master/data/coco.names>)

(d) Freeze the graph

```
python3 convert_weights_pb.py --class_names coco.names --data_format NHWC --weights_file yolov3.weights
```

STEP 2: CREATE IR THROUGH THE MODEL OPTIMIZER

Now that we have a frozen graph, the next step is to use the Model Optimizer of the OpenVINO™ toolkit to generate the intermediate representation files



STEP 2: CREATE IR THROUGH THE MODEL OPTIMIZER

Instructions

```
python3 <MO_INSTALL_DIR>/mo_tf.py --input_model frozen_darknet_yolov3_model.pb --batch 1 --  
tensorflow_use_custom_operations_config  
/opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/yolo_v3.json -o FP16 --  
data_type FP16
```

- `--input_model`: Frozen graph from Step 2
- `--batch`: Batch size
- `--tensorflow_use_custom_operations_config`: Allows TensorFlow* custom extensions for non-standard layers (learn more about this in Chapter 5)
- `-o`: Output path where the .bin and .xml files are saved
- `--data_type`: Required precision for the hardware you choose to deploy on. Since we will be deploying on the Intel® Neural Compute Stick 2, we select FP16.

STEP 3: TEST THE IR FILES WITH SAMPLE APPS

The OpenVINO™ toolkit installation comes with many sample applications that integrate the Inference Engine capabilities for demonstration

- Build these samples using the OpenVINO toolkit installation instructions
- To test the IR created in Steps 1 – 3, we will use the `object_detection_demo_yolov3_async` demo

Instructions

- The python demos will be found in the `<OPENVINO_INSTALL_DIR>/deployment_tools/open_model_zoo/demos/python_demos` directory
- Download a sample video from <https://github.com/intel-iot-devkit/sample-videos>
- Run the following command to execute the sample using a sample video and infer on the

```
<OPENVINO_INSTALL_DIR>/deployment_tools/open_model_zoo/demos/python_demos/object_detection_demo_yolov3_async$python3 object_detection_demo_yolov3_async.py -i <VIDEO_PATH>/person-bicycle-car-detection.mp4 -m <PATH_to_FP16_model_xml_file> -d MYRIAD
```

STEP 3: TEST THE IR FILES WITH SAMPLE APPS

Output:



Demonstrates:

- A bounding box around objects detected
- The label of the object detected (Example: Person is 0, car is 672. To understand this, check the coco.names labels file we downloaded in Step 1. The labels are 0 indexed)
- % accuracy
- Time taken to run the inference

In Chapter 5, we will show you how to use a custom topology with the OpenVINO™ toolkit

