Overview
It is becoming increasingly easy to recognize objects in just one step with algorithms like YOLO (“You Only Look Once”), pre-trained models from Open Model Zoo, and highly efficient architectures like MobileNETV2 for object detection on mobile devices.

This document will demonstrate how to get started with an object detection program using MobileNetV2, OpenVINO™ Toolkit, and the COCO Dataset in the Kaggle environment.

Getting Started
First, you need to install prerequisites for OpenVINO™ Toolkit and set up a virtual environment for TensorFlow, a machine learning platform that generates models of objects to recognize:

```
#install pre-reqs for OpenVINO (should be patched in later releases)
!add-apt-repository -y ppa:deadsnakes/ppa
!apt-get -qq update
!apt-get -qq install -y libpython3.7-dev
# setup the environment -- we ignored the error in dependencies solving in this demo as a workaround.
#%pip install --upgrade pip
%pip install -q openvino-dev==2022.1.0 &> error_log.txt
```

Note: You may need to restart the kernel to use updated packages.
Then you need to set up the OpenVINO™ Toolkit inference engine, which uses models to perform actual recognition.

```
From openvino.runtime import Core
import collections
import os
import sys
import time

import cv2
import numpy as np
```
# Getting Started

**Directory where model will be downloaded**

```python
base_model_dir = "model"
```

**Model name as named in Open Model Zoo**

```python
model_name = "ssdlite_mobilenet_v2"
```

The download command is executed as follows:

```bash
$download_command
```

**Output path for the conversion**

```python
converted_model_path = f"model/public/{model_name}/{precision}/{model_name}.xml"
```

If the converted model path does not exist, the conversion command is executed as follows:

```bash
$convert_command
```

**Loading the model**

You need only a few lines of code to run the model. You have to create an Inference Engine, which will read the network architecture and model weights from the .bin and .xml files to load onto the desired device.

*Note: OpenVINO® Toolkit can decide which hardware will offer the best performance by using AUTO. While GPU may be the best hardware in many cases, it has a longer startup time.*
# initialize inference engine
ie_core = Core()

# read the network and corresponding weights from file
model = ie_core.read_model(model=converted_model_path)

# load the model on the CPU (you can choose manually CPU, GPU, MYRIAD etc.)
# or let the engine choose best available device (AUTO)
compiled_model = ie_core.compile_model(model=model, device_name="AUTO")

# get input and output names of nodes
input_layer = compiled_model.input(0)
output_layer = compiled_model.output(0)

# get input size
height, width = list(input_layer.shape)[1:3]

With the SSDLite MobileNetV2, you have one input and one output.

input_layer.any_name, output_layer.any_name

**Processing the results**

First, you list all available classes and create colors. The post-process step transforms boxes with normalized coordinates into boxes with pixel coordinates. Then non-maximum suppression is used to reject overlapping detections and those below the probability threshold (0.5). The model places boxes and labels around objects that are recognized.

# https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/
classes = [
    "background", "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train",
    "truck", "boat", "traffic light", "fire hydrant", "street sign", "stop sign",
    "parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
    "bear", "zebra", "giraffe", "hat", "backpack", "umbrella", "shoe", "eye glasses",
    "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite",
    "baseball bat", "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle",
    "plate", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", "apple",
    "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake", "chair",
    "couch", "potted plant", "bed", "mirror", "dining table", "window", "desk", "toilet",
    "door", "tv", "laptop", "mouse", "remote", "keyboard", "cell phone", "microwave", "oven",
    "teddy bear", "hair drier", "toothbrush", "hair brush"
]

# colors for above classes (Rainbow Color Map)
colors = cv2.applyColorMap(
    src=np.arange(0, 255, 255 / len(classes), dtype=np.float32).astype(np.uint8),
    colormap=cv2.COLORMAP_RAINBOW
).squeeze()
def process_results(frame, results, thresh=0.6):
    # size of the original frame
    h, w = frame.shape[:2]
    # results is a tensor [1, 1, 100, 7]
    results = results.squeeze()
    boxes = []
    labels = []
    scores = []
    for _, label, score, xmin, ymin, xmax, ymax in results:
        # create a box with pixels coordinates from the box with normalized coordinates [0,1]
        boxes.append(
            tuple(map(int, (xmin * w, ymin * h, (xmax - xmin) * w, (ymax - ymin) * h))))
        labels.append(int(label))
        scores.append(float(score))

    # apply non-maximum suppression to get rid of many overlapping entities
    # see https://paperswithcode.com/method/non-maximum-suppression
    # this algorithm returns indices of objects to keep
    indices = cv2.dnn.NMSBoxes(
        bboxes=boxes, scores=scores, score_threshold=thresh, nms_threshold=0.6)

    # if there are no boxes
    if len(indices) == 0:
        return []

    # filter detected objects
    return [(labels[idx], scores[idx], boxes[idx]) for idx in indices.flatten()]

def draw_boxes(frame, boxes):
    for label, score, box in boxes:
        # choose color for the label
        color = tuple(map(int, colors[label]))
        # draw box
        x2 = box[0] + box[2]
        y2 = box[1] + box[3]
        cv2.rectangle(img=frame, pt1=box[:2], pt2=(x2, y2), color=color, thickness=3)
        # draw label name inside the box
        cv2.putText(
            img=frame,
            text=f"{classes[label]} (score:.2f)",
            org=(box[0] + 10, box[1] + 30),
            fontFace=cv2.FONT_HERSHEY_COMPLEX,
            fontScale=frame.shape[1] / 1000,
            color=color,
            thickness=1,
            lineType=cv2.LINE_AA,
        )

    return frame
The actual detection of objects is defined in this function:

```python
def run_object_detection(frame):
    input_img = cv2.resize(src=frame, dsize=(width, height), interpolation=cv2.INTER_AREA)

    # create batch of images (size = 1)
    input_img = input_img[np.newaxis, ...]

    # measure processing time
    start_time = time.time()
    # get results
    results = compiled_model([input_img])[output_layer]
    stop_time = time.time()

    # get boxes from network results
    boxes = process_results(frame=frame, results=results)

    # draw boxes on a frame
    frame = draw_boxes(frame=frame, boxes=boxes)

    infer_time = round((stop_time-start_time)*1000,2)
    print("Inference Time: " + str(infer_time) + "ms")
```

The Results
You can copy and paste these code snippets to run object detection with different images.

```python
import matplotlib.pyplot as plt

dog = cv2.imread('..input/cat-and-dog/test_set/test_set/dogs/dog.4045.jpg')
run_object_detection(dog)
dog = cv2.cvtColor(dog, cv2.COLOR_BGR2RGB)
plt.imshow(dog)
plt.show()

cat = cv2.imread('..input/cat-and-dog/test_set/test_set/cats/cat.4024.jpg')
run_object_detection(cat)
cat = cv2.cvtColor(cat, cv2.COLOR_BGR2RGB)
plt.imshow(cat)
plt.show()
```
```python
cow = cv2.imread('../input/cowphotos/500px-Cow_female_black_white.jpeg')
run_object_detection(cow)
cow = cv2.cvtColor(cow, cv2.COLOR_BGR2RGB)
plt.imshow(cow)
plt.show()
```
Inference Time: 18.05ms

```python
car = cv2.imread('../input/toycars/1040x585-2021-1027-best-toy-car-f8442f.jpg')
run_object_detection(car)
car = cv2.cvtColor(car, cv2.COLOR_BGR2RGB)
plt.imshow(car)
plt.show()
```

Inference Time: 18.01ms

```python
cars = cv2.imread('../input/traffics/20150331__sjtraffic1.jpg')
run_object_detection(cars)
car = cv2.cvtColor(cars, cv2.COLOR_BGR2RGB)
plt.imshow(cars)
plt.show()
```
Success! OpenVINO™ Toolkit was able to detect objects with great accuracy! To learn more about object detection and recognition, read this blog and check out the Intel® AI Dev Team Adventures for more foundational techniques and hands-on training.

< Back to Home

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software, or service activation.

© Intel Corporation, Intel, the Intel logo, OpenVINO, the OpenVINO logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.