# Udacity Project 1 - Deploy a People Counter App at the Edge

Marcos Augusto Bellezi

## 0 - Project's Github Home

The project can be cloned from it's Github Home:

https://github.com/mbellezi/nd131-openvino-fundamentals-project-starter

## 1 - Choosing the Model

I'v used the Tensorflow Model Zoo to find a model that could do the object detection with bounding boxes. On that it's fast enough for real time detection on the edge could be the SSD Mobilenet V2 COCO.

Downloaded from:

```
wget http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz
```

Converting the model to IR:

```
tar -xzvf ssd_mobilenet_v2_coco_2018_03_29.tar.gz
cd ssd_mobilenet_v2_coco_2018_03_29
$OV/deployment_tools/model_optimizer/mo_tf.py --input_model frozen_inference_graph.pb --transformations_config $OV/deployment_t
```

The converted model was saved to the model folder. The preprocessing layer was removed. The scaling layer was kept.

The model work well for the most part of the detection. Only one of the persons has some detection problems but reducing the probability threshold solved the problem.

### 1.1 - Model Original Inputs and Outputs

The documentation of the inputs and outputs of the model can be found here:

`https://aihub.cloud.google.com/u/0/p/products%2F8c6878ba-d32d-411d-bac2-2f884b748c4f` [1]

**Inputs (from [1])**

A three-channel image of variable size - the model does **NOT** support batching. The input tensor is a `tf.float32` tensor with shape `[1, height, width, 3]` with values in `[0.0, 1.0]`.

**Outputs (from [1])**

The output dictionary contains:

- `detection_boxes`: a `tf.float32` tensor of shape `[N, 4]` containing bounding box coordinates in the following order: `[ymin, xmin, ymax, xmax]`.
- `detection_class_entities`: a `tf.string` tensor of shape `[N]` containing detection class names as Freebase MIDs.
- `detection_class_names`: a `tf.string` tensor of shape `[N]` containing human-readable detection class names.
- `detection_class_labels`: a `tf.int64` tensor of shape `[N]` with class indices.
- `detection_scores`: a `tf.float32` tensor of shape `[N]` containing detection scores.

### 1.2 - Model Converted Inputs and Outputs

The model original input layer was dropped during conversion. The converted model has the the following input and outputs:

**Inputs**

Shape: [1, 3, 300, 300], 3 color layers with 300×300 image dimension. The model was converted with the option o invert the RGB order.

**Outputs**

Shape: [1, 1, 100, 7], one hundred detection classes and for each class is used the following data:

- Index 1: class ID
- Index 2: detection score
- indexes 3-6: xmin, ymin, xmax, ymax

## 2 - Details about the implementation of the Bounding Box detection

Two classes were created: `BoundingBox` and `BoundingBoxTracker`. The `BoundingBox` is used to encapsulate all the information about a bounding box, as it's ID, location, duration, visibility etc.

The `BoundingBoxTracker` maintains a dictionary of the existing bounding boxes and controls it's visibility and the updating of the bounding boxes after each new frame. The central point of the bounding boxes are used to compare proximity and correlates a new bounding box with a existing one. The algorithm is simple but work well even with multiple objects. It could be improved taking in account the bounding boxes sizes and velocity vectors for a better accuracy. A new bounding box is considered valid after some frames of continuous detection and leaves after some frames of continuous absence (this is configurable).

## 3 - Running the App

Here are the instructions to run the detection app with the provided sample detection video:

Go to the project folder and start the servers:

- For the MQTT server:

```
cd webservice/server/node-server
node ./server.js &
cd ../../..
```

- For the UI:

```
cd webservice/ui
npm run dev &
cd ../..
```

- For the FFPMEG server:

```
sudo ffserver -f ./ffmpeg/server.conf
```

Running the app using the sample video (the `-–preview` option will open an openCV window for previewing, if there is not a X server do not use this option):

```
python3 main.py --preview -m model/frozen_inference_graph.xml -p 0.30  --input Pedestrian_Detect_2_1_1.mp4 | ~/ffmpeg/ffmpeg -v
```

The `-–cam` option can be used instead of `-–input` to use a live video from the webcam (in my case MacBook webcam)

```
python3 main.py --preview -m model/frozen_inference_graph.xml -p 0.40  --cam | ~/ffmpeg/ffmpeg -v warning -f rawvideo -pixel_fo
```

# 4 - Project Write-up

## 4.1 - Converting custom layers

For this project it's was not used any custom layer. If the model been converted has layers that were not directly supported by OpenVino, this layers are considered custom layers. The list of supported layers depend on the original model's framework.

For example, if the model is a **Cafee** model, one have two options to support the layer:

1. **Register the layer as an extension with the Model Optimizer:**

   In this case you can provide a small piece of Python code that allows the Model Optimizer to generates a valid internal representation of the layer. In this case you do not need Caffe to run de model.

   The first step is register a custom layer as a Model Optimizer Extension to provide a class `PythonProposalOp` that must return a series of attributes to the Model Optimizer and calculates the output shape for the layer.

   Than, you have to register rules to pass the extension layer properties from the Caffe model to the Intermediate Representation. This is done creating a `FrontExtractorOp` class. The full example code can be viewed at:

   https://docs.openvinotoolkit.org/2019_R3/_docs_MO_DG_prepare_model_customize_model_optimizer_Extending_M

2. **Register the layer as a custom layer and use Cafee to calculate the layer:**

   This is a deprecated option and you must run Cafee to do the calculations. This is not recommended and the procedure can be found here:

   https://docs.openvinotoolkit.org/2019_R3/_docs_MO_DG_prepare_model_customize_model_optimizer_Legacy_Mod

If the model is a **Tensorflow** model, the options are:

1. **Register the layer as an extension with the Model Optimizer:**

   This is the same procedure discussed in the Cafee session.

2. **register unknown subgraph operations as a series of known subgraph operations**

   If the Model Optimizer can not recognize a  subgraph of operations in your model but you know this operations exists inside the Model Optimizer you can express the unknown operation as a series of known operations for the optimizer. The details of this can be found here:

   https://docs.openvinotoolkit.org/2019_R3/_docs_MO_DG_prepare_model_customize_model_optimizer_Subgraph

3. **Offload the layer calculation to Tensorflow**

In this case the layer calculations runs on Tensorflow. This can be used with limitations on development to test a complex model and can run on GPU. The details are here:

https://docs.openvinotoolkit.org/2019_R3/_docs_MO_DG_prepare_model_customize_model_optimizer_Offloadin

All this approach is useful when a new Neural Network architecture is created and some operations are not yet supported or recognized by the Model Optimizer

## 4.2 - Running the Pre-Trained Model without OpenVino

As the original model is a TensorFlow one, it was used one Google Colab example of object detection to infer it's speed. The model chosen was the same model and one image from this project sample video was also uploaded. The Colab notebook can be found here:

Google Colaboratory

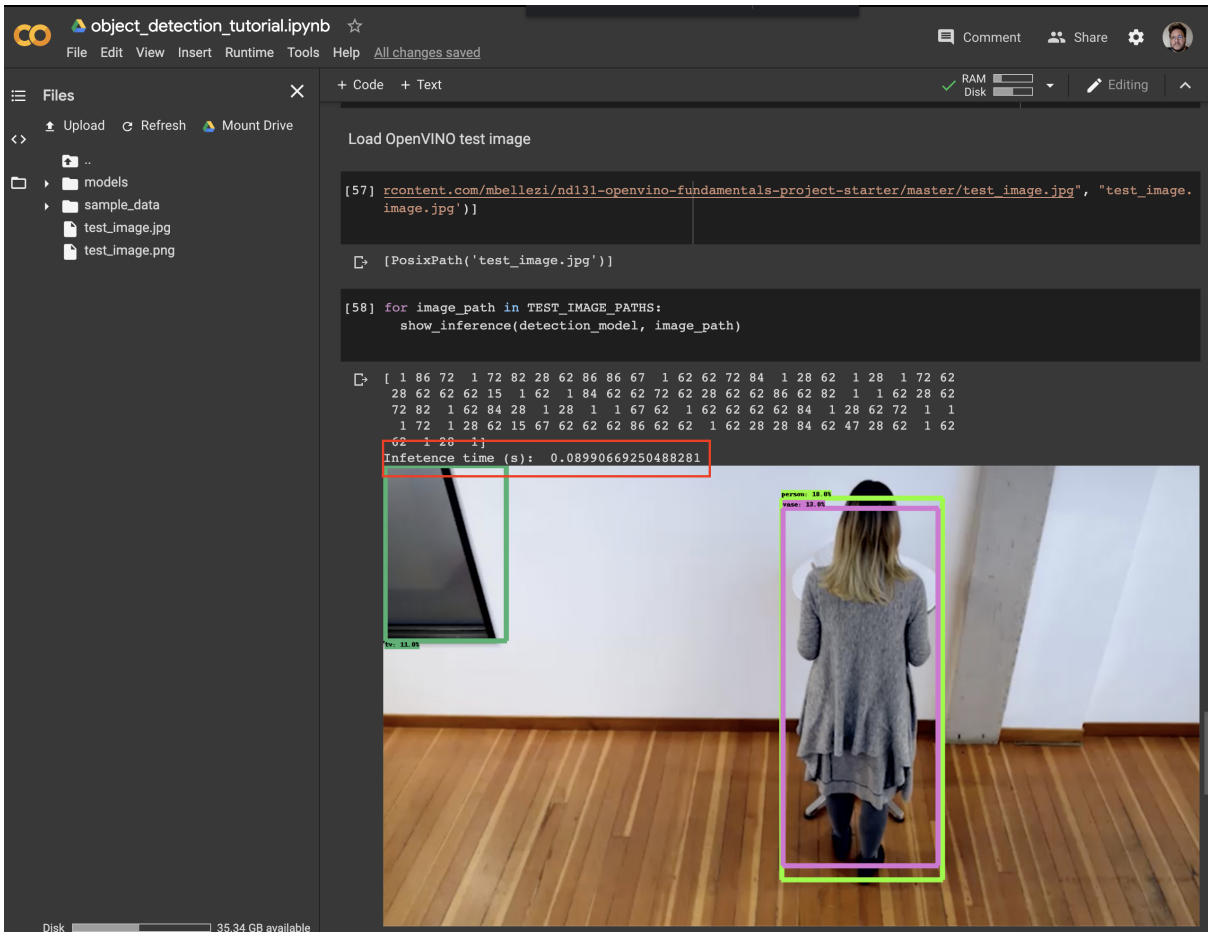
https://colab.research.google.com/drive/1IL7LP9WRrLroDjYo8ScoeAaA0RH9AbB5?usp=sharing

The inference time was measure using Google Colab CPU and TensorFlow and the time was as bellow:

**Comparison TensorFlow vs OpenVino**

| Framework | Inference time (lower is better) |
|---|---|
| OpenVino | 89 ms |
| TensorFlow | 11 ms |

Below is the example inference captured on Google Colab



As can be seen, the TensorFlow in this case is 8 times slower then OpenVino. If you add the latency to the datacenter (30ms), the upload of the image to the datacenter (around 100ms on a 40Mbps network) you would get a total of 219 ms, and you also need to add the transfer time to the frontend app. In this case, using OpenVino, one could have a near real time monitoring.

If you use the Google Cloud, for example, to to the detection, you also have to spin a virtual machine (around US$ 30 / month) but if your application streams the video full time you will have a additional cost of US$ 0,02/GB. For a 1Mbps feed working 24×7 gives you 324GB / month and plus U$ 6,48 / month.

### 4.3 - Potential Use Cases for the People Counter

This people counter project could be an interesting asset for a wide range of stores or services.

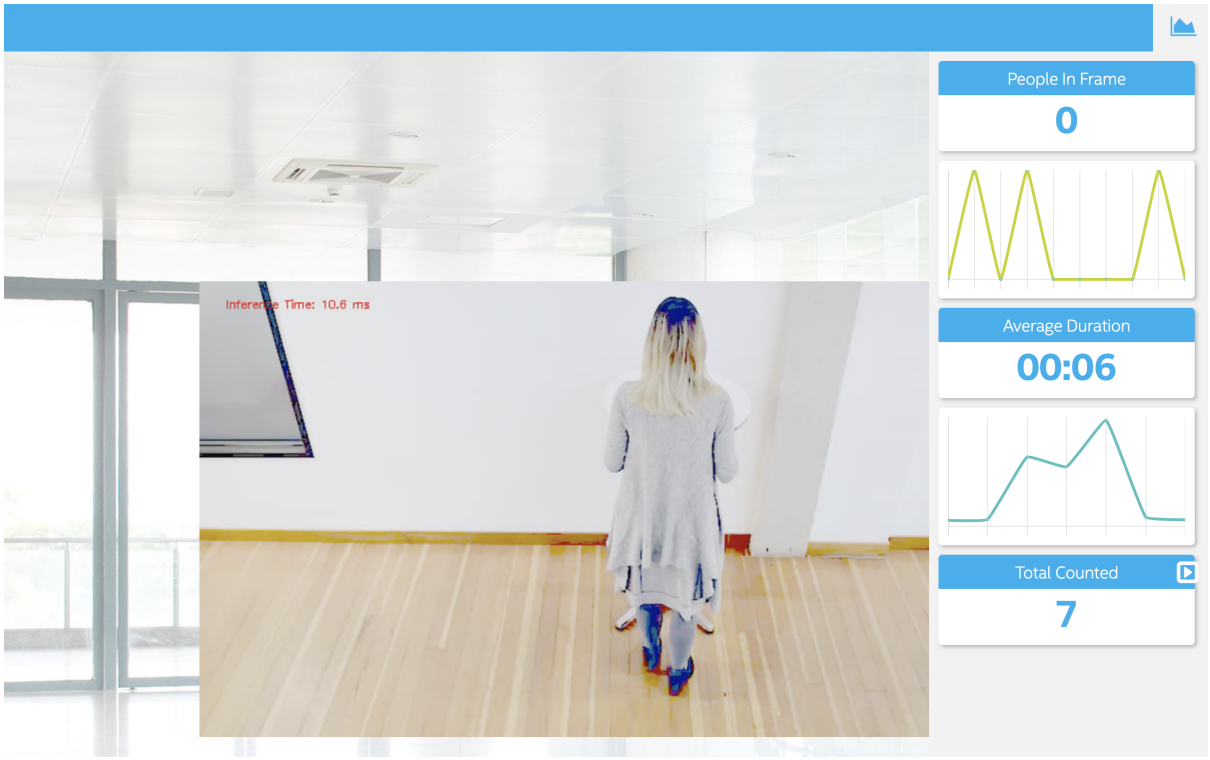
In a store, for example, the number of customers and the average time could be used as indicators for the Marketing and sales teams. If coupled with sales, it would give the conversion rate of people inside the store. The number of customers can also be used to do A/B tests of marketing campaigns inside (using the sales) and outside (marketing announcements for attracting new customers). If you deploy multiple detectors, you start to have a more fine grained vision by department driving the sales and marketing teams. This detection could be easily implemented using Raspberry Pi coupled with the Intel Neural Stick 2 USB and communication data with a server inside the store.

For services, it can be used, for example, do dynamically calculate queues size, mean time of service, and set alarms for adjusting  the number of employees based on the serving time. Could also be used to create seasonal data and help doing long time planning for number od employees required during a season.

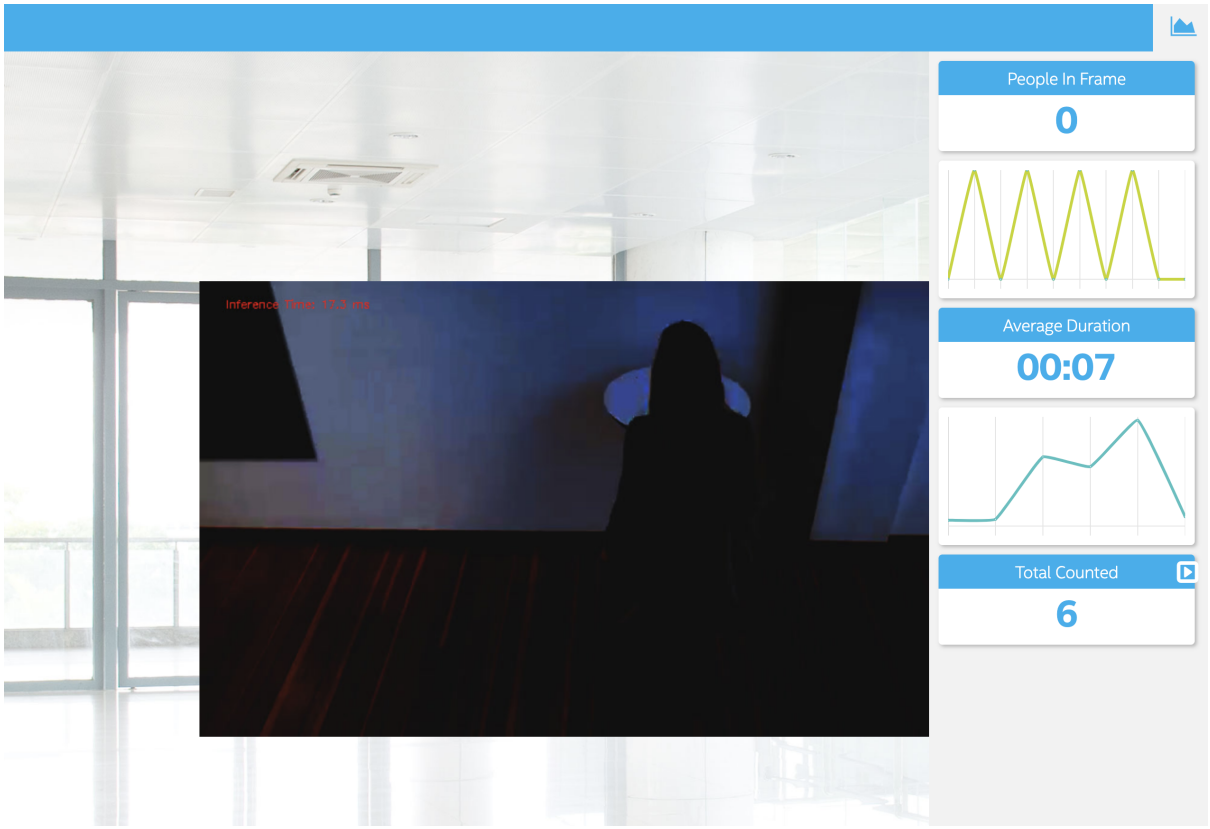
### 4.4 - Discuss lighting, model accuracy, and camera focal length/image size

For the model work properly, the conditions of the image used for inference are very important. The model was trained using a set of parameters as illumination,  distance from the camera, camera angle etc. If the inference is done using images with very different conditions one can expect a lower accuracy on the detection. As example, for this project was created a new param called `--gamma` to test different gamma corrections. As can be seen below, if the bright and contrast of the image is much different than the usual, the accuracy drops (the woman is not identified):

**gamma: 0.20**




**gamma: 8.00**




As the image is resized to a 300×300 image for inference, if the object in the image is too small, this also will lead to a lower accuracy because it may not be enough pixels for a detection. This could be the case with cameras with a very wide angle where objects become too small. The camera angle is also important and should match the angle of the images used for training of the Neural Network for an optimal accuracy.