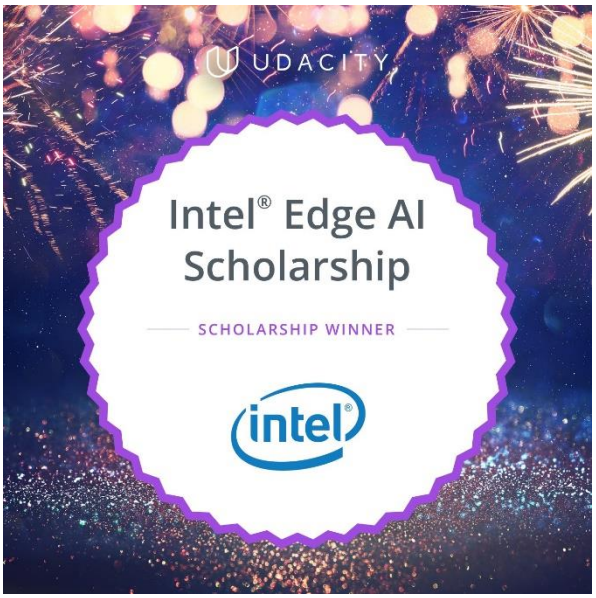


Intel® Edge AI for IoT Developers Nanodegree Program

Project #01 Deploy a People Counter App at the Edge

Name: Marwan Saeed Alsharabbi

Date: 08-May-2020

2020

Instructions

Project Introduction

In this project, you will utilize the Intel® Distribution of the OpenVINO™ Toolkit to build a People Counter app, including performing inference on an input video, extracting and analyzing the output data, then sending that data to a server. The model will be deployed on the edge, such that only data on 1) the number of people in the frame, 2) time those people spent in frame, and 3) the total number of people counted are sent to a MQTT server; inference will be done on the local machine.

You will also create a write-up comparing the performance of the model before and after use of the OpenVINO™ Toolkit, as well as examine potential use cases for your deployed people counter app.

In completing the project, you'll get hands-on experience optimizing models for inference with the OpenVINO™ Toolkit, as well as building skills handling and extracting useful information from the deployed models.

Provided Files

You will be provided with complete files implementing:

1. A MQTT server - receives JSON from your primary code subsequent to inference concerning people counted, duration they spent in frame, and total people counted. This will feed to the UI server.
2. A UI server - displays a video feed as well as statistics received from the MQTT server.
3. A FFmpeg server - receives output image frames including any detected outputs inferred by the model. This is then fed to the UI server.
   While the above files are provided complete so that no additional front-end or web services work is required from you, feel free to adjust these as you see fit. For instance, how can you save on network bandwidth by adding a toggle to turn off the sending and receiving of the image frame, only focusing on the statistics from the lightweight MQTT server?

   Additionally, you will be provided with a video file to test your implementation, although your code should allow for other inputs, such as a single image or webcam stream.

   You will also be provided with starter code for your implementation, split into two files:

1. `inference.py` - Here, you will load the Intermediate Representation of the model, and work with the Inference Engine to actually perform inference on an input.
2. `main.py` - Here, you will:

- Connect to the MQTT server
- Handle the input stream
- Use your work from `inference.py` to perform inference
- Extract the model output, and draw any necessary information on the frame (bounding boxes, semantic masks, etc.)
- Perform analysis on the output to determine the number of people in frame, time spent in frame, and the total number of people counted
- Send statistics to the MQTT server
- Send processed frame to FFmpeg

More detail on the above files will be provided in the upcoming project instructions.

Project Set-Up - Classroom Workspace

We have provided a CPU classroom workspace, set up with the Intel® Distribution of the OpenVINO™ Toolkit, along with all necessary dependencies installed, later in this lesson. No additional set-up is required in the workspace. However, if you would like to work on your local machine, we have also provided additional local set-up instructions below.

Project Set-Up - Local

Below, we've included instructions by operating system if you want to deploy your app on a local machine. You can clone the full repository here.

Linux/Ubuntu

See here in the project repository.

Windows

See here in the project repository.

Mac

See here in the project repository.

Project Instructions

Now, you should be all set up and ready to get started on your People Counter app using the OpenVINO™ Toolkit. Although you are free to dive right into the starter code and work everything out on your own, we've also provided some additional helpful instructions below to help you along the way. First, we'll cover the steps to get your code up and running, and then we'll get into what to do in the write-up portion of the project.

Project Instructions: Code

As noted in the project introduction, there are two primary files in which you'll work to implement the People Counter app - `inference.py` and `main.py`.
The `inference.py` file is the suggested starting point, as you'll need much of the code there to fully implement the `TODO`'s throughout `main.py`.
To start, you'll need to choose a model and utilize the Model Optimizer to begin.

Choosing a Model & The Model Optimizer

You will choose the model you want to work with in this project - the only requirement is that the model is not already one of the pre-converted Intermediate Representations already available from Intel®.

Please provide a link for your reviewer to the original model you choose, as well as the what you entered into the command line to convert your model with the Model Optimizer. You only need to include the converted IR model in your submission, not the original.

While what type of model you use is up to you to decide, you must be able to extract information regarding the number of people in the frame, how long they have been in frame, and the total amount of people counted from your model's output. Models using bounding boxes or semantic masks as the output will likely be helpful in that regard, but feel free to use any model you think might be helpful here.

Note that you may need to do additional processing of the output to handle incorrect detections, such as adjusting confidence threshold or accounting for 1-2 frames where the model fails to see a person already counted and would otherwise double count.

If you are otherwise unable to find a suitable model after attempting and successfully converting at least three other models, you can document in your write-up what the models were, how you converted them, and why they failed, and then utilize any of the Intel® Pre-Trained Models that may perform better.

Inference.py

1-Start by taking a look through the starter code to familiarize yourself with it. In the file, you will implement parts of the `Network` class, which will be used in `main.py` to interact with the loaded, pre-trained model.
You'll actually be able to use code quite similar to what you implemented in the final exercise of the course, but using a new model and different information sent over MQTT.

- **You do not have to exactly use the starter template.** If you want to change up the functions within, feel free to do so.

2- In `load_model()`:

- Set a `self.plugin` variable with `IECore`
- If applicable, add a CPU extension to `self.plugin`
- Load the Intermediate Representation model

- Check whether all layers are supported, or use a CPU extension. If a given layer is not supported, let the user know which layers
- Load the model network into a `self.net_plugin` variable

3- Implement `get_input_shape()` to return the shape of the input layer
4-Implement `exec_net()` by setting a `self.infer_request_handle` variable to an instance of `self.net_plugin.start_async`
5-Implement `get_output()` by handling how results are returned based on whether an `output` is provided to the function.

Main.py

1-Implement `connect_mqtt()` by creating and connecting to the MQTT client
2-In `infer_on_stream()` load the model into a `infer_network` variable
3-Implement handling video or image input, or note to the user that it was unable to use the input
4-Use CV2 to read from the video capture
5-Pre-process the image frame as needed for input into the network model
6-Implement processing of the network output

- Extract any bounding boxes, semantic masks, etc. from the result - make sure to use `prob_threshold` if working with bounding boxes!
- If using bounding boxes, draw the resulting boxes with `cv2.rectangle` on the frame
- Update `current_count` as necessary

7-Back in `infer_on_stream()`, calculate and send relevant information on `count`, `total` and `duration` to the MQTT server
8-Send the frame (now including any relevant output information) to the FFmpeg server
9-Separately, if the user input a single image, write out an output image.
It's been a lot of hard work, but now it's time to check out your app, deployed at the edge!

Running the App

Sourcing the Environment

When opening a new terminal window, in order to source the environment, use the command:

```
source /opt/intel/openvino/bin/setupvars.sh -pyver 3.5
```

Any new terminals you open will again need this command run.

Starting the Servers

Before running `main.py`, you will need to have the MQTT, FFmpeg and UI servers all up and running. If you are running your app on a local device, make sure you have followed the earlier set-up instructions so that all dependencies are installed.

You'll need terminals open for each of these. For each, `cd` into the main directory containing all of your People Counter app files.

Note: You will need to run `npm install` in the `webservice/server` and `webservice/ui` directories if you have not already.

From there:

- For the MQTT server:

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

- For the UI:

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

- For the FFPMEG server:

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

Starting the App Itself

As you may have noticed, there are a number of arguments that can be passed into `main.py` when run from a terminal. While you should make sure to check them out in the code itself, it's important to note you'll also want to add some additional commands to make sure the output image frames are sent to the FFmpeg server.

The arguments for `main.py` can be entered as follows (you may need to specify `python3` on your system):

```
python main.py -i {path_to_input_file} -m {path_to_model} -l
{path_to_cpu_extension} -d {device_type} -pt
{probability_threshold}
```

The arguments for FFmpeg can be entered as follows - note that we'll include the values here that will work optimally with the FFmpeg server and UI instead of placeholders. If you have not updated the FFmpeg `server.conf` file, this will match to what is configured therein.

```
ffmpeg -v warning -f rawvideo -pixel_format bgr24 -
video_size 768x432 -framerate 24 -i -
http://0.0.0.0:3004/fac.ffm
```

To run these two together, *while you have the ffserver, MQTT server, and UI server already running through separate terminals*, you can use a pipe symbol (`|`) to combine them as one. Here is an example, along with possible paths for `main.py` arguments included:

```
python main.py -i resources/Pedestrian_Detect_2_1_1.mp4 -m
your-model.xml -l
/opt/intel/openvino/deployment_tools/inference_engine/lib/in
tel64/libcpu_extension_sse4.so -d CPU -pt 0.6 | ffmpeg -v
warning -f rawvideo -pixel_format bgr24 -video_size 768x432
-framerate 24 -i - http://0.0.0.0:3004/fac.ffm
```

Note: The primary CPU extension file differs in naming between Linux and Mac. On Linux, the name is `libcpu_extension_sse4.so`, while on Mac it is `libcpu_extension.dylib`.

Viewing the App in Your Browser

If you are in the classroom workspace, use the "Open App" button to view the app in the browser, or if working locally, navigate to http://0.0.0.0:3004 in your browser. You should be able to see the video stream with any relevant outputs (bounding boxes, semantic masks, etc.) onto the video. Additionally, you can click the icon in the upper right to expand to show some statistical information; clicking another icon under the existing charts on this new menu will expand the final piece of information.

Project Instructions: Write-up

Once you have the edge app up and running, you're not quite finished! You will also create a write-up to help show your understanding of the OpenVINO™ Toolkit and its impact on performance, as well as articulating the use cases of the great application you deployed at the edge.

1. Explain the process behind converting any custom layers. Explain the potential reasons for handling custom layers in a trained model.
2. Run the pre-trained model without the use of the OpenVINO™ Toolkit. Compare the performance of the model with and without the use of the toolkit (size, speed, CPU overhead). What about differences in network needs and costs of using cloud services as opposed to at the edge?
3. Explain potential use cases of a people counter app, such as in retail applications. This is more than just listing the use cases - explain how they apply to the app, and how they might be useful.
4. Discuss lighting, model accuracy, and camera focal length/image size, and the effects these may have on an end user requirement.
5. **If you were unable to find a suitable model and instead used an existing Intel® Pre-Trained Model**, you should document the three non-IR models you tried first, how you converted them, and why they failed.

Minimum Viable *Project*

While following the project instructions and meeting every rubric requirement is all that is required to pass the project, you should also see that as your MVP - minimum viable *project*. To make your submission stand out from others, there are a number of ways you can build even further on your People Counter app:

1. Add an alarm or notification when the app detects above a certain number of people on video, or people are on camera longer than a certain length of time.
2. Try out different models than the People Counter, including a model you have trained. Note that this may require some alterations to what information is passed through MQTT and what would need to be displayed by the UI.
3. Deploy to an IoT device (outside the classroom workspace or your personal computer), such as a Raspberry Pi with Intel® Neural Compute Stick.
4. Add a recognition aspect to your app to be able to tell if a previously counted person returns to the frame. The recognition model could also be processed on a second piece of hardware.
5. Add a toggle to the UI to shut off the camera feed and show stats only (as well as to toggle the camera feed back on). Show how this affects performance (including network effects) and power.

These are just a few quick ideas to get you started on further building out your app past the main requirements! Remember, none of these are required.

If you come up with an additional feature you really like, let us know on Twitter @udacity and @IntelSoftware!

Requirements

Hardware

- 6th to 10th generation Intel® Core™ processor with Iris® Pro graphics or Intel® HD Graphics.
- OR use of Intel® Neural Compute Stick 2 (NCS2)
- OR Udacity classroom workspace for the related course

Software

- Intel® Distribution of OpenVINO™ toolkit 2019 R3 release
- Node v6.17.1
- Npm v3.10.10
- CMake
- MQTT Mosca server

People Counter App at the Edge

In this application, app counts the number of people in the current frame, the duration that a person is in the frame (time elapsed between entering and exiting a frame) and the total count of people. Furthermore, app send values to Mosca server namely, total counted people, current count and the duration of person. This project is a part of Intel Edge AI for IoT Developers Nanodegree program by Udacity.

Select the Model ssd_mobilenet_v2_coco

For this project, I used SSD MobileNet V2 COCO model and converted it to an Intermediate Representation for use with the Model Optimizer. Utilizing the Inference Engine, I used the model to perform inference on an input video, and extract useful data concerning the count of people in frame and how long they stay in frame. I sent this information as well as the output frame over MQTT, in order to view it from a separate UI server over a network.

Download model:

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

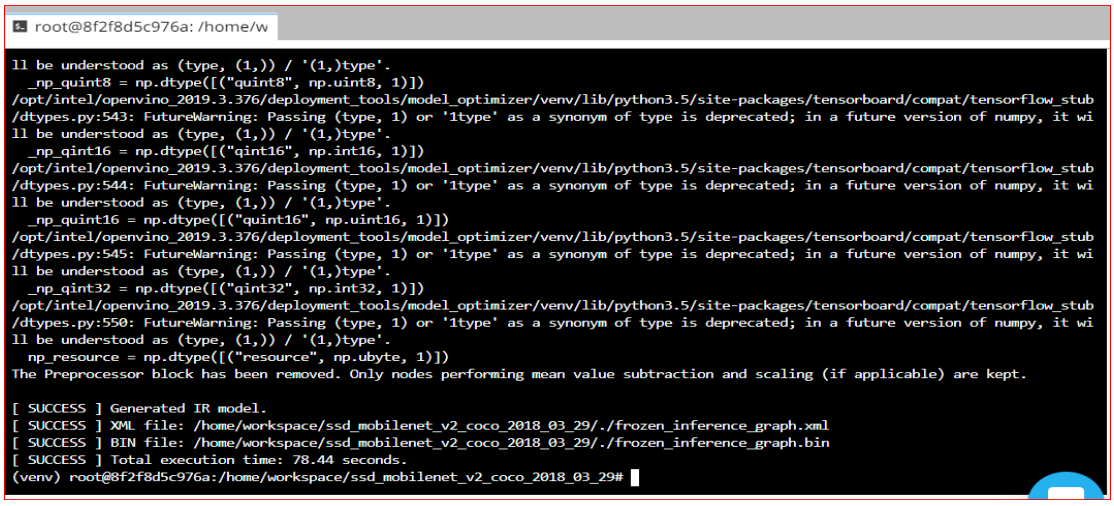
Extracting the tar-gz file :

```
tar -xzvf ssd_mobilenet_v2_coco_2018_03_29.tar.gz
```

Change path to `cd ssd_mobilenet_v2_coco_2018_03_29`

Convert the model to an Intermediate Representation with the following arguments:

```
python /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --
input_model frozen_inference_graph.pb --
tensorflow_object_detection_api_pipeline_config pipeline.config --
reverse_input_channels --tensorflow_use_custom_operations_config
/opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support.json
```



Run the application

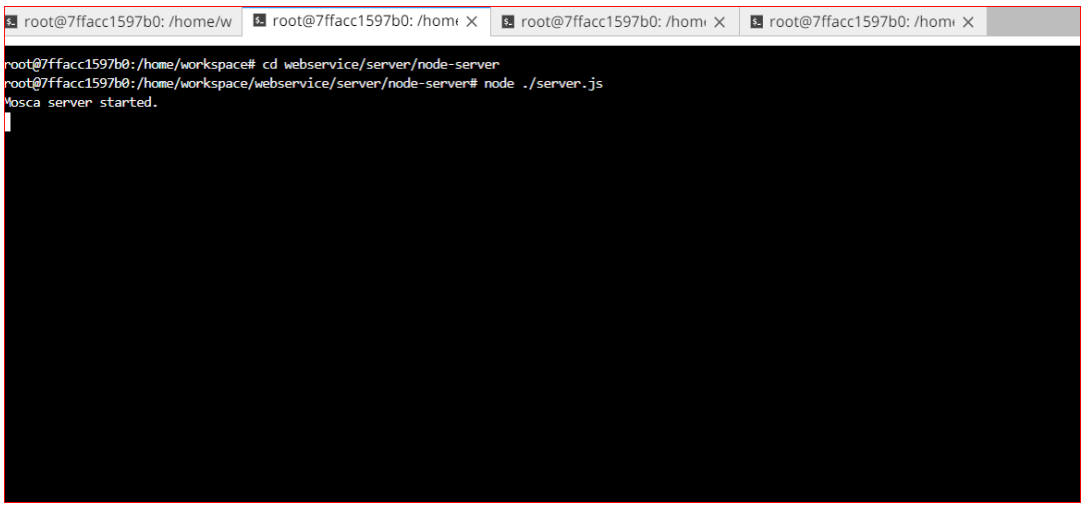
You can use the button to source the environment in the initial terminal, although any new terminals using the toolkit will need the command as detailed in Step 4 below.

From the main directory:

Step 1 - Start the Mosca server

cd webservice/server/node-server

node ./server.js



You should see the following message, if successful:

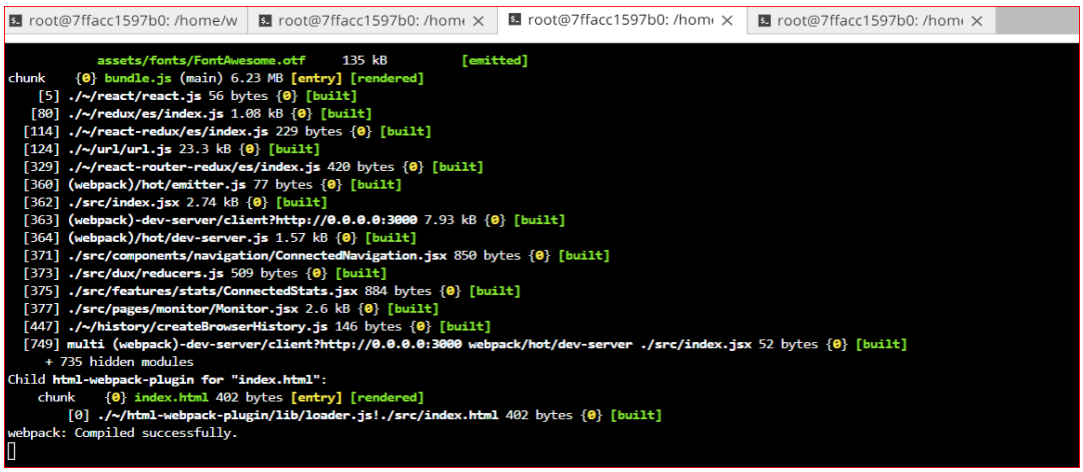
```
Mosca server started.
```

Step 2 - Start the GUI

Open new terminal and run below commands.

cd webservice/ui

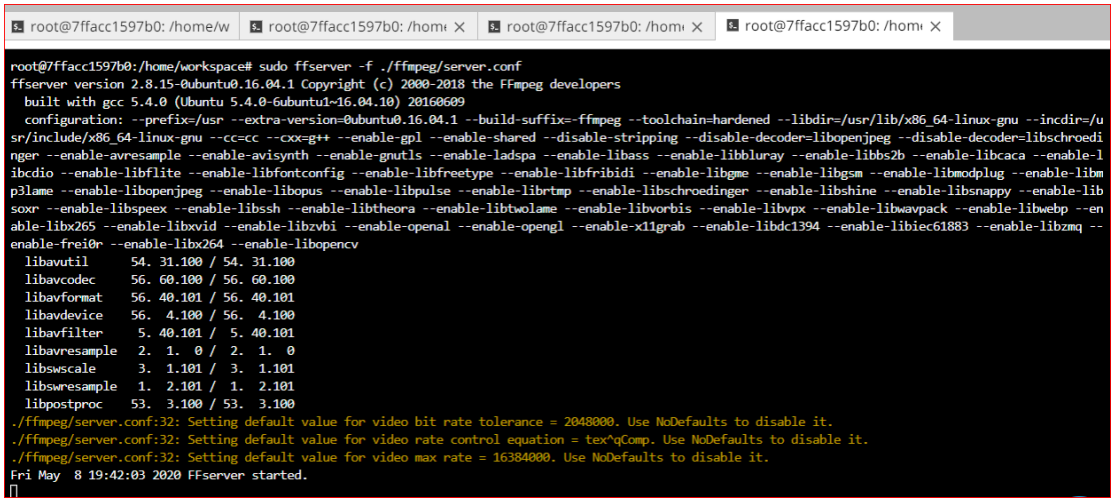
npm run dev



Step 3 - FFmpeg Server

Open new terminal and run the below commands.

sudo ffserver -f ./ffmpeg/server.conf



Step 4 - Run the code

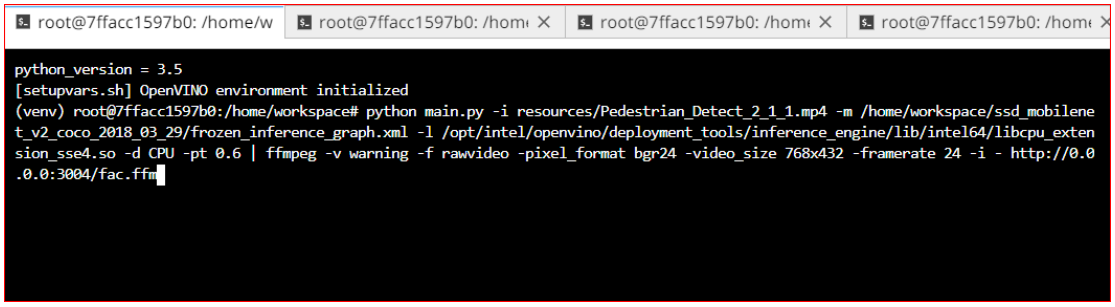
Open a new terminal to run the code.

Setup the environment You must configure the environment to use the Intel® Distribution of OpenVINO™ toolkit one time per session by running the following command:

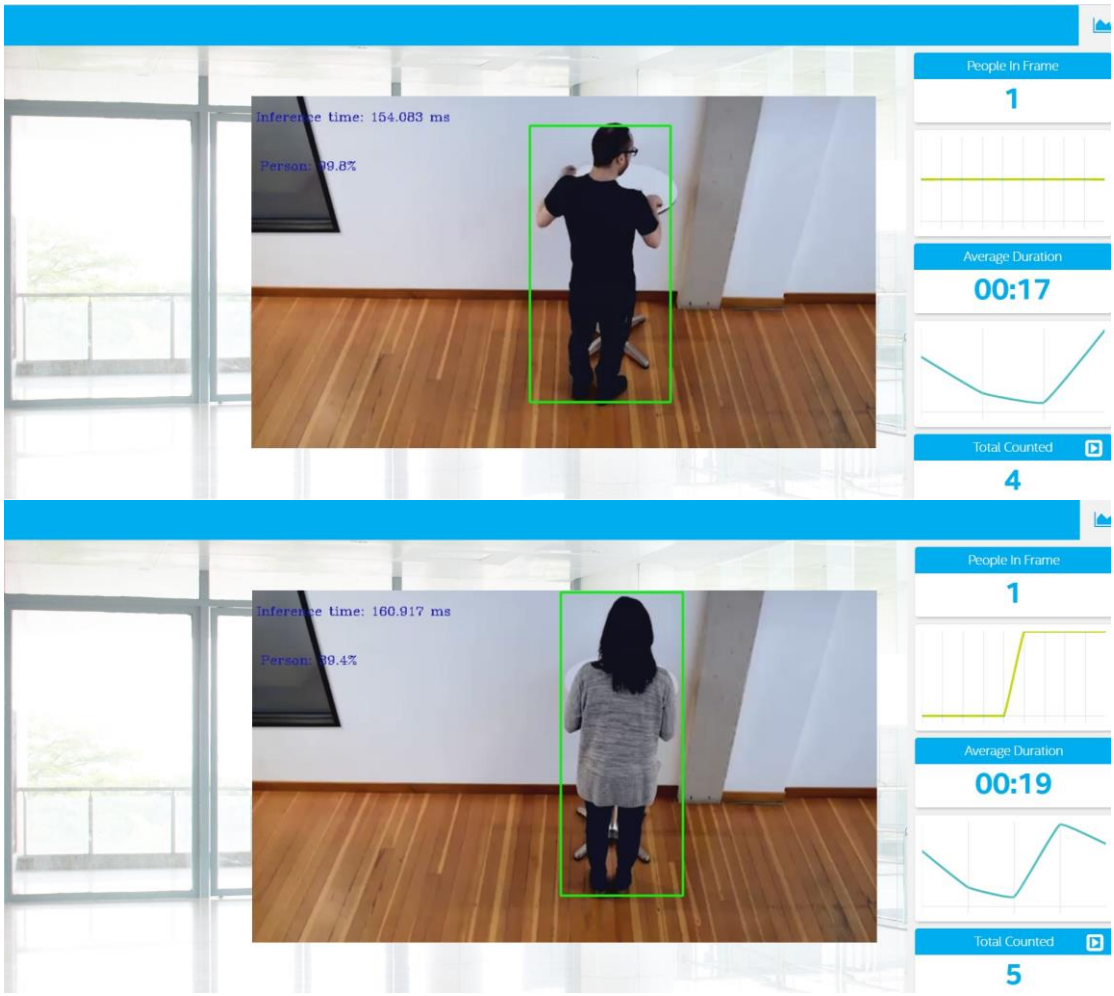
```
source /opt/intel/openvino/bin/setupvars.sh -pyver 3.5
```

Running on the CPU

```
python main.py -i resources/Pedestrian_Detect_2_1_1.mp4 -m /home/workspace/ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.xml -l /opt/intel/openvino/deployment_tools/inference_engine/lib/intel64/libcpu_extension_sse4.so -d CPU -pt 0.3 | ffmpeg -v warning -f rawvideo -pixel_format bgr24 -video_size 768x432 -framerate 24 -i - http://0.0.0.0:3004/fac.ffm
```



Result by Model `ssd_mobilenet_v2_coco_2018_03_29`



This model is better than others and gives expected results with 0.3 probability threshold

Report

Explaining Custom Layers

Custom layers are layers that are not included into a list of known layers. If your topology contains any layers that are not in the list of known layers, the Model Optimizer classifies them as custom. These custom layers need to be specifically registered following the steps corresponding to each framework after which the Model Optimizer will take it into consideration while producing the Intermediate Representation. Intel openVINO already has extensions for custom layers.

Comparing Model Performance

To compare models before and after conversion to Intermediate Representations I've first run the inference with the downloaded TensorFlow models, then I've converted the models to an IR file and run the inference on the same input.

The difference between model accuracy pre- and post-conversion was almost the same, indeed converting a model to an IR files has minimum effects on loss of accuracy.

Converting the model to an IR has helped reducing the size of the model of about 2MB.

Inference time of model post-conversion was almost twice as fast when compared to inference time pre-conversion.

The performance between the pre-conversion and post-conversion model is also due to a difference between cloud and edge computing, respectively. Indeed:

Cloud computing:

Cloud Computing is more suitable for projects which deal with big data and where latency is not a major concern. The processing power is very high. Moreover, the Cloud can be accessed from anywhere on multiple devices, making ideal for projects that need to run and used on a variety of platforms. Because Network communications are expensive (bandwidth, power consumption, etc.) and sometimes not possible (like in remote locations or during natural disasters), cloud computing can be very cost expensive and not always adapt for user needs.

Edge computing:

Edge Computing is usually ideal for operations with extreme latency concerns, for example when applications need immediate data analytics or must calculate results with a minimum amount of latency. When there is not enough or a working network bandwidth to send the data to the cloud, edge computing is a better option than cloud computing. However, Edge Computing requires a robust security and advanced authentication methods to avoid external attacks.

Assess Model Use Cases

Some of the potential use cases of the *people counter app* are:

- **Track activity in retail or departmental store**: understand how many customers per day and in which areas of the stores customers spend most of the time. It could help to reconfigure the store to avoid crowded areas and make better use of the space.
- **Monitor factory / building work spaces**: similar to the above use case, it would allow understanding where in work space people gather the most and how much time to they spend. Moreover, it could also be used for security reasons, monitoring the sensitivity areas of the building.

Assess Effects on End User Needs

Lighting, model accuracy, and camera focal length/image size have different effects on a deployed edge model. The potential effects of each of these are as follows person may not be detected in dark place because of weak lighting and if model accuracy is low, app may not detect person properly in the frame.

Model Research

All models taken from TensorFlow Object Detection Model Zoo and Open model zoo, respectively TF detection model zoo, OpenVINO public models

ssd_mobilenet_v2_coco

Download model:

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

Extracting the tar.gz file: `tar -xzvf ssd_mobilenet_v2_coco_2018_03_29.tar.gz`
Change path to `cd ssd_mobilenet_v2_coco_2018_03_29`
Convert the model to an Intermediate Representation with the following arguments:

```
python /opt/intel/openvino/deployment_tools/model_optimizer/mo.py --
input_model frozen_inference_graph.pb --
tensorflow_object_detection_api_pipeline_config pipeline.config --
reverse_input_channels --tensorflow_use_custom_operations_config
/opt/intel/openvino/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support.json
```

This model is better than others and gives expected results with 0.3 probability threshold

I hope to be home to the project requirements despite the valuable information that we learned from the lessons and also the project, but we do not know the exact correct result

Help resources Forums :https://knowledge.udacity.com
https://github.com/intel-iot-devkit/people-counter-python/blob/master/main.py

I wish success to all.

Marwan Saeed Alsharabbi