**People Counter App Writeup**

This writeup is for the first project in the **Intel’s Edge AI for IoT Developers** nanodegree program which is building a people counter app which works on the edge.

Firstly, I have used the Udacity workspace for this project assignment. Now, let us get on with the discussion.

***Model Used***

For this purpose, I have used a model from the tensorflow model zoo. Specifically, I have used the SSD mobilenet v2 which is trained on the coco dataset. I can detect upto 80 objects and the inference time is also very low. We will just be using it to detect a **person** for this project though. The main reasons for choosing this model are:

* Fast Inference, suitable for edge applications
* Smaller Size, which will be even lesser with model optimization using the Intel open vino toolkit.
* Acceptable accuracy

So, there was no requirement for adding any custom layers to this model as it perfectly fit the required specifications for conversion into an IR (Intermediate Representation) and it work perfectly in the application too.

***Comparison of pre-trained and IR performance***

In this section, we will be comparing the performance of the models before and after going through the model optimizer. So, this is the platforms we will be using for testing:

1. For the normal pre-trained model, we will be testing it on a Windows PC. I have setup a virtual environment with all the necessary libraries and typed in the starter code to test the model on the same video used in the project workspace.
2. The IR will be tested on the Udacity Workspace.

We will be comparing the following parameters to assess the performance in both scenarios:

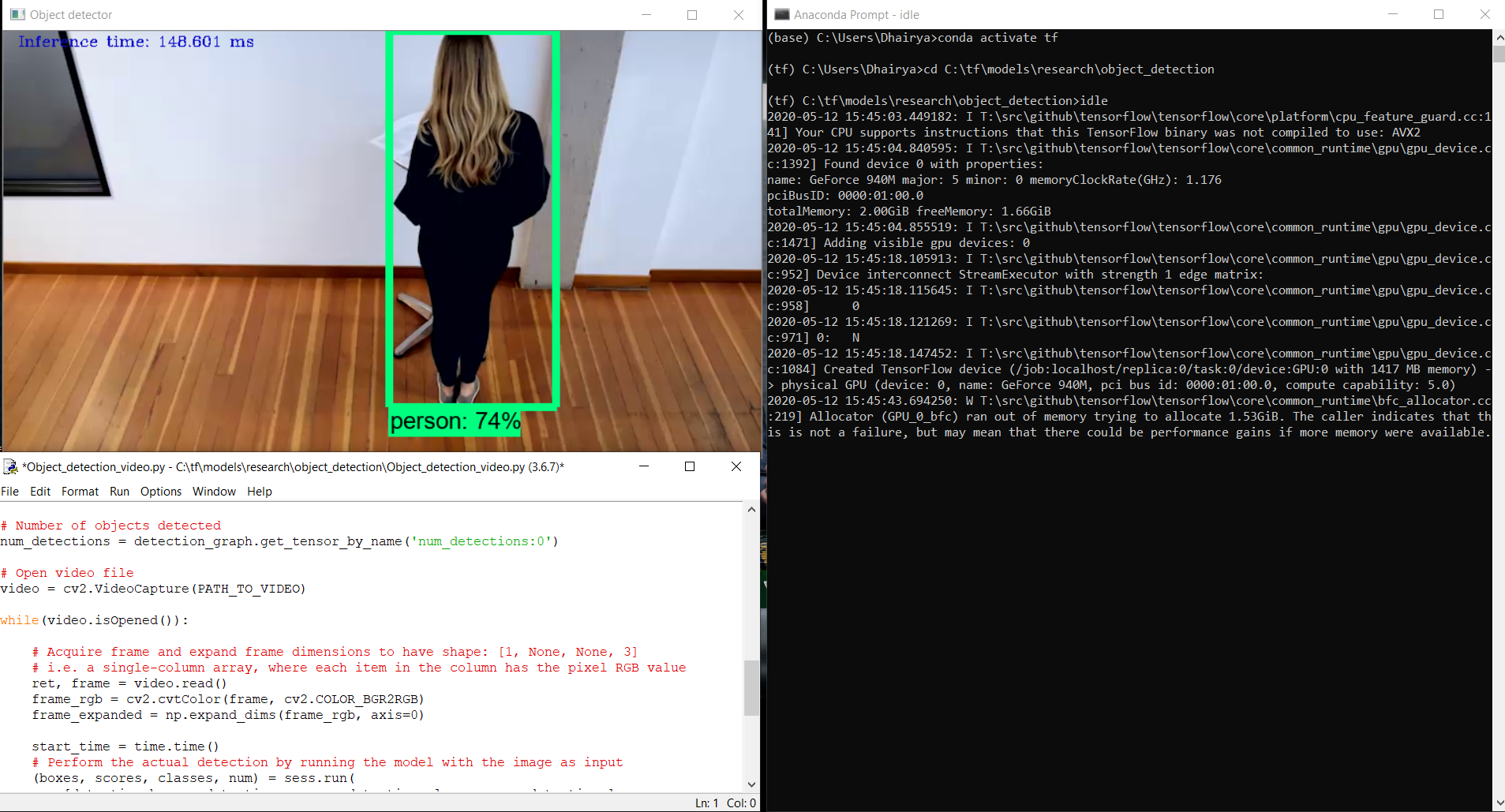
* Accuracy of detection
* Inference Time
* Size of Model

So, let us get started with the comparison.

1. **The pre-trained model**

So, for the pre trained model I just downloaded the tar.gz file from the tensorflow model zoo GitHub page. Then, I tested it using a simple object detection script using tensorflow-gpu python library. I ran it on the video provided on the project workspace. Here are the results:

* It achieved as average accuracy of around 70% looking at the detection accuracy of all the 6 people. It had problems detecting the person 2 in the video which was solved partially by reducing the confidence threshold.
* The Inference time of the model is 150ms on average during all times.
* The size of the *frozen\_inference\_graph.pb* (the model) is **66.4 megabytes.**



The python file created to test this has been added to the GitHub repo for the project submission too.

1. **The IR version of the Model**

In this part, we will discuss how we created the IR of the mobilnet model using the model optimizer provided in the openvino toolkit, loaded this IR into the Inference Engine and then tested it for our app. All the files required are added to the submission repo.

Firstly, we will get the 2 files required for the conversion of a tensorflow model into an IR, which are the **.pb** file and **pipeline.config** file. Once, we got this, we will use the command given below to convert this into an IR, which consists of 2 files, a **.bin file and a .xml file.**

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

So, after successful running this will produce the required files in the directory that you are currently in. I have created a *models* folder for this and it consists of the required .bin and .xml files.

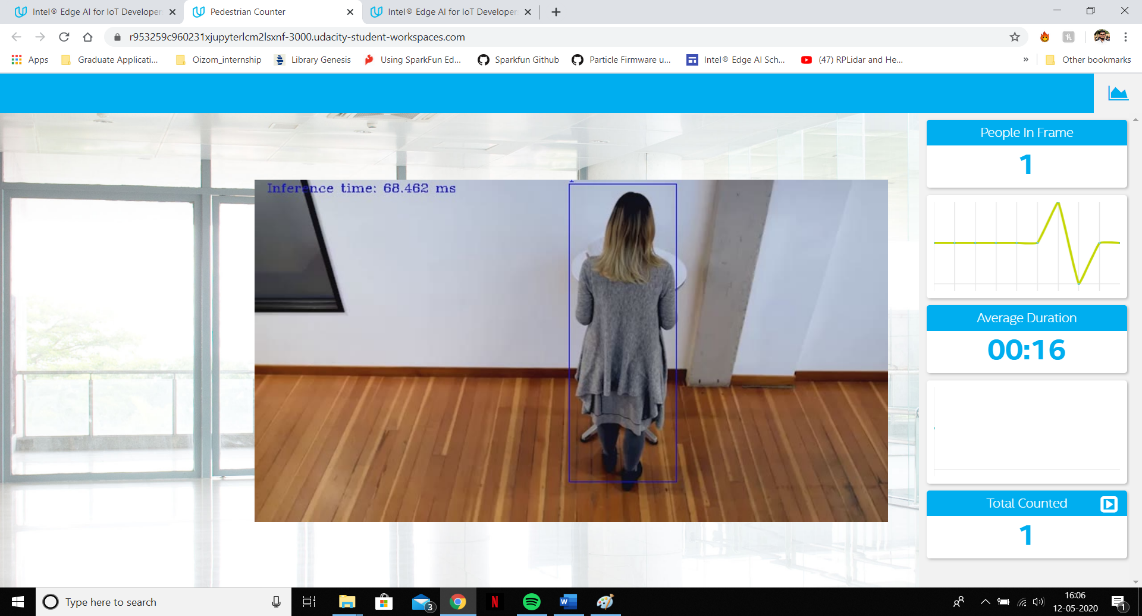
Now, we simply load this into the **Inference Engine** using a simple python code which is named **feed\_model.py** in the git repo. Run the following command to do so:

python feed\_network.py -m /home/workspace/models/🡨**Your Model Name🡪**

Now, we fill in all the to do tasks in both **main.py and inference.py** and then we are all ready to go. After the codes are finished, go on and follow the guide to open up 4 different terminals and start all the servers. Once that is done, just run the following code in the 4th terminal:

python main.py -i resources/Pedestrian\_Detect\_2\_1\_1.mp4 -m **<< Model Path>>** -l /opt/intel/openvino/deployment\_tools/inference\_engine/lib/intel64/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

Change **<<model path>>** with the path of your .xml file of the Intermediate Representation. If everything is right, the code with run without any errors and you can see your application running using the **Open App** button in the guide. It will look something like this:



Now, the performance stats improve a lot compared to the case before:

* **Average accuracy: 78-80%**
* **Inference Time** reduces to almost half staying around 70ms on average.
* **The model size** also reduces greatly. The main .xml file is just 109 Kilobytes and the size of the .bin file is 60 megabytes.

So, overall, we can conclude that the model optimizer truly increases the performance of any given model and reduces the inference time greatly.

***Potential use-cases for the People counter app***

There are several use-cases for this kind of application. Some of them are:

* Using this application in departmental stores can help them assess the behaviour of their customers towards particular sub sections in their store. For example, how often do people visit a particular section, how much time do they spend in that section, etc. This data greatly helps in the behavioural analysis of the customers, helps in determining what type of products are purchased more and hence beneficial for the store.
* This app would be very useful in hospitals and clinics to keep a track on how many patients visit in a day, average time of each session the doctor takes, etc. This data can be used to increase the efficiency of operation of that particular hospital using this data. With some additional features like face recognition we can also keep a track on the admitted patients.
* Attendance Systems can be automated using this application as well. We would just require to add a person profile for each student so that app can easily recognize if that particular person is present in the class or not and automatically fill the attendance online, for example on a mqtt server like the one we used in our current project.

So, these were some applications I thought could use this app.

***Assess Effects on End User Needs***

We will discuss some of parameters which affect the performance of the deployed model. These parameters change according the environment in which the model is deployed so we should choose the model according the use-case.

* **Lighting:** Lighting has a great effect on both the detection rate and the accuracy of the model. When there is negligible lighting, the captured pictures are pitch dark and just not enough for our model to detect anything relevant, as the model analyses the image by just rgb pixel values, which change greatly with lighting. So, to reduce this effect, we can add some software pre-processing like increasing the contrast and lighting in the image and increasing the sharpness too if the captured image is out of focus or too dull.
* **Model Accuracy:** This is the most important and the primary parameter when it comes to model selection for a particular use-case. For example, if we want to just detect the presence of someone in a room, the accuracy is not that important but if we want to detect the presence of a particular entity like a person or a cat in the room, the accuracy states the likeliness that the detected thing is what. This is a very important parameter as every model has its limits. So, we need to choose a model with maximum efficiency but also keep in mind other factors like inference time. For instance, large models have great accuracy but the prediction time is much more, which is not ideal for real time detection applications. So, these trade off should be carefully considered while choosing the model.

Hence, this concludes my project writeup. I would just like to thank Udacity and Intel for providing me this opportunity.