Embedded Real-time Object Detection And Classification Using The Raspberry Pi And Movidius Neural Compute Stick

A Udacity Capstone Proposal

By

Marko Stefanovic

Table Of Contents

Domain Background

In the last couple of years huge advances have been made in classification and detection of objects in images. The main reason for this step is that the application of neural networks became feasible in the field of computer vision. In 2015 the artificial neural network "ResNet" scored on the largest image database "ImageNet" higher than any algorithm before - and surpassed at the same time the visual recognition power of a human. Quickly, a plethora of image applications followed like the google cloud vision API, a service which developers can use for their own applications to analyze the content of any images.

Although there might be limitless possibilities for new computer vision applications with this new technology, there are still some unsolved challenges left. Not every system or application can afford to have a large backend server with lots of computing power at its side like the google cloud vision service has. One can think about autonomous cars which have to recognize obstacles within milliseconds and don't have the time to wait for the response of a server. Or you might want to have a simple camera-based product, which the customer expects to just work out-of-the-box without having to connect it to the internet, subscribing to some cloud service or connect to some side-by computer.

To make a neural networks run on cheap, low powered embedded systems in real time is still an unsolved problem. There have been efforts to make CNNs run on the Raspberry Pi but they didn't cross the real time barrier [PW2014]. Chip manufacturers are working hard to bring to the market specialized hardware which is able to process CNNs in real time. One of such a chip is the Intel Myriad processor which could help systems to process live video images from a camera in real time using a CNN. This chip is made easily accessible for developers through the Movidius Neural Compute Stick which is a small USB device which can run certain CNNs supposedly much faster than on a CPU.

My personal motivation comes from my job: I work for a company which develops smart sensors for the automotive and consumer industry. Right now I work in a project where a mono-camera tracks people and objects they might carry around. I hope that besides finishing the Udacity course I will also be able to apply the results of this project to my project at work.

Problem Statement

In the scope of this project it should be analyzed, if its possible to run a CNN for object detection and classification on the Raspberry Pi 3 in real time which means faster than 50 ms (> 20 FPS). The usage of the Movidius NCS is necessary because former efforts to run such a CNN solely on the RP3 have failed (1 FPS, [BNET2017]). It can be assumed that its simply not possible. The solution with the Raspberry Pi 3 in connection with a Movidius NCS is still not trivial because even one of the smallest public CNNs for such a task - Tiny Yolo - still needs around 200-250ms to run on the stick [TYNCS2017]. Hence the main work for this project will be to use the original Tiny Yolo model as a baseline and derive from it smaller and faster CNNs which fulfill the real time requirements while having an acceptable detection and classification accuracy.

One possible application for which this model could be used later is a real time people detector and tracker which might also be powerful enough to track objects which are carried around by people.

Datasets And Inputs

For this project only openly accessible datasets and inputs will be used. For training, validation and testing the COCO dataset will be used since COCO has a large amount of images with people in it. In addition there is also a reference Tiny Yolo model which was trained on this dataset. The dataset also includes a few object classes which people might carry around like suitcases and bags and is therefore very suitable for our problem domain. It would be interesting to see how many additional classes our CNN can support while fulfilling the real time requirements.

The 2017 version of the COCO dataset will be used since it already contains a train/valid/test set which consists of approximately 120K/5K720K images. The training and validation sets are already labelled with 80 classes of which most of them are irrelevant for us. We will use a subset of those classes based on how well they fit into our problem domain and how many of those can be supported by our reduced model. Here is a possible selection of classes:

Class name	Person	Bike	Backpack	Handbag	Suitcase	Dog	Cell Phone	Car
# instances	66808	3401	5756	7133	2507	4562	5017	12786

There is an imbalance of classes where persons and cars are overrepresented and bike and suitcases slightly underrepresented. We will try to combat this imbalance by either over- and undersampling the training data or by creating synthetic samples (f.e. horizontal flip of images).

The images in the COCO dataset have already different sizes. Furthermore, the darknet framework has an option to randomly rescale the images during training. We will explore this option to see if it has any benefit.

Results on the COCO test set (test-dev 2017) - which is not labelled - could be sent to their evaluation server to get statistics about the quality of our model and to compare it to more powerful models. But this will be considered optional, since the additional effort might be too high and it might be easier to just use 20K images from the training set for testing.

The input videos serve two purposes: firstly they will be used for run-time measurements on the Raspberry Pi and Movidius combination to see if our real time requirements are fulfilled. Secondly they will give us a hint if tracking would be possible for a possibly later application. This will more correspond to a "visual sanity check" that our model would still work in reality and not be part of a rigorous testing routine. For this reason the input videos won't be labelled - also because there won't be enough time for labelling, implementation of a tracking algorithm and a test to evaluate the tracking performance.

The videos will contain sequences of people passing by a camera in different scenarios. We prefer whole video sequences for the input because in our problem domain we generally use cameras which must be able to track an-d classify objects in image sequences and not just in one separate image as in the dataset.

The first video shows people in a shopping mall passing by a camera. Many of them carry bags and some of them suitcases. Furthermore you have people of different sizes and walking speeds. There are also many cases where there is truncation at the image border for example when people pass the camera very close or are occluded by other people. From the second video we will use sequences which contain people walking or standing with suitcases.

1. https://www.youtube.com/watch?v=o1Ye9f641Rg
2. https://www.youtube.com/watch?v=giRtcChNS2E

Solution Statement

The solution will be a modified version of the Tiny Yolo model. It will have a reduced size and run much faster than the original model. The model will be trained on a subset of the COCO database and must contain the person class and possibly other classes. The model will be trained on a desktop computer with a GeForce 770 GPU and will be able to run on a Raspberry Pi 3 with a connected Movidius Neural Compute Stick.

Benchmark Model

The benchmark model for this project will be the Tiny Yolo model trained on the COCO database. The run-time performance of the TinyYolo net will be measured beforehand on a Raspberry Pi 3 in combination with the Movidius Neural Compute Stick. All subsequently created models will have to run faster than this baseline. The detection and classification rate will also be measured on the Tiny Yolo net and taken as the baseline for quality comparisons.

Evaluation Metrics

Run-time: The time for data transmission and inference will be measured by running the input images on the aforementioned hardware. The average run-time over all images will be the run-time metric (separately for transmission and inference). To reach real time performance we will aim for 75ms total time (sum of transmission and inference).

Mean Average Precision: The mAP will be the metric to measure the quality of the detection and classification of objects. It will be the deciding metric to evaluate the expected drop in quality when the model is made smaller and faster.

Project Design

Languages

Python 3 is the common language for the different frameworks we will be using. Python will also be used to implement our custom toolchain and the different metrics.

Environments

We will be using two develop environments: a **host system** where we will train and compile our models and a **target system** where we will determine the run-time behaviour of our models.

Our **host system** will run Ubuntu in a virtual box on a windows 10 system. Ubuntu 16.04 is the recommended OS for compilation of models for the Movidius and can also be used to train our custom Tiny Yolo variations.

Our **target system** will be our "embedded" environment and will consist of a Raspberry Pi 3 in combination with a Movidius NCS. The required OS is Stretch.

Frameworks

Darknet: Tiny yolo requires the darknet framework for training. On the darknet homepage is a tutorial which shows how training is done with the COCO database.

Caffe/Tensorflow: The Movidius NCS requires one of those frameworks for inference. Beforehand the darknet model must be converted to one of these formats. There are already python scripts online available to perform these conversions. The openly available Modidius SDK provides an API for compiling and running models.

Baseline

Our first baseline metrics will be the mean average precision of the Tiny Yolo model trained and tested on the COCO database. We will have to write a script to calculate it from the test results. Our second metric is the run-time which we will calculate by running Tiny Yolo on the Movidius NCS.

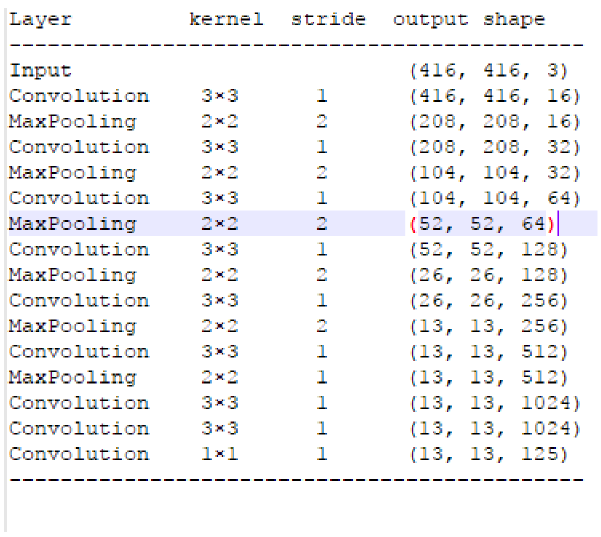
Optimization

This will be the biggest part of our project. It will consist of several iterations of:

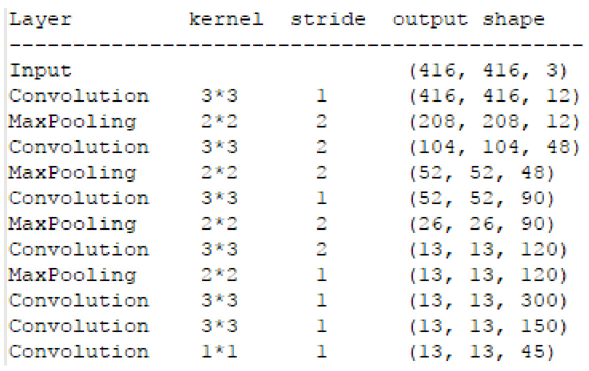
1. Create new model by changing architecture of net
2. On host system: Train on COCO database and determine quality metric
3. On target system: run new model and determine run-time metric
4. Compare with former results

There are a lot of parameters which can be tweaked to optimize our models:

- input resolution
- number of layers
- kernel sizes and strides
- number format



```
Layer          kernel  stride  output shape
---------------------------------------------
Input                          (416, 416, 3)
Convolution    3×3      1      (416, 416, 16)
MaxPooling     2×2      2      (208, 208, 16)
Convolution    3×3      1      (208, 208, 32)
MaxPooling     2×2      2      (104, 104, 32)
Convolution    3×3      1      (104, 104, 64)
MaxPooling     2×2      2      (52, 52, 64)
Convolution    3×3      1      (52, 52, 128)
MaxPooling     2×2      2      (26, 26, 128)
Convolution    3×3      1      (26, 26, 256)
MaxPooling     2×2      2      (13, 13, 256)
Convolution    3×3      1      (13, 13, 512)
MaxPooling     2×2      1      (13, 13, 512)
Convolution    3×3      1      (13, 13, 1024)
Convolution    3×3      1      (13, 13, 1024)
Convolution    1×1      1      (13, 13, 125)
---------------------------------------------
```



```
Layer          kernel  stride  output shape
---------------------------------------------
Input                          (416, 416, 3)
Convolution    3*3      1      (416, 416, 12)
MaxPooling     2*2      2      (208, 208, 12)
Convolution    3*3      2      (104, 104, 48)
MaxPooling     2*2      2      (52, 52, 48)
Convolution    3*3      1      (52, 52, 90)
MaxPooling     2*2      2      (26, 26, 90)
Convolution    3*3      2      (13, 13, 120)
MaxPooling     2*2      1      (13, 13, 120)
Convolution    3*3      1      (13, 13, 300)
Convolution    3*3      1      (13, 13, 150)
Convolution    1*1      1      (13, 13, 45)
```

The two model architectures should demonstrate how a new model during an iteration step might look like. The first model is the original Tiny Yolo net and the second a possible optimized model.

Bibliography

PW2014: Pete Warden, How to optimize Raspberry Pi code using its GPU, 2014, https://petewarden.com/2014/08/07/how-to-optimize-raspberry-pi-code-using-its-gpu/

BNET2017: , the object detection of YOLO V2 can be run on Raspberry PI 3? , , https://groups.google.com/a/dt42.io/d/msg/berrynet/dPk_mnnLIIg/QwbVSfpNDwAJ

TYNCS2017: , Tiny YOLO on NCS, , https://ncsforum.movidius.com/discussion/218/tiny-yolo-on-ncs