INDIAN INSTITUTE OF TECHNOLOGY DELHI

DEPTT./CENTRE Computer Science and Engineering

CERTIFICATE PAGE

REPORT OF THE PROJECT SHORTLISTED UNDER SURA-2018

Project titled: **Realtime Detection using Deep Learning on Embedded Platforms**

Submitted by:

Name of the Student	Centre	Entry no.	Contact	Signature
Mayank Singh Chauhan	CSE	2016CS50394	7838298181	
Arshdeep Singh	CSE	2016CS50625	9582839784	

Name and Signature of Facilitator: Rijurekha Sen

Centre: Computer Science and Engineering

Contact: 9810591052

Date of submission of report in IRD: 22/10/2018

1 Abstract

Classifying and counting vehicles in road traffic has numerous applications in the transportation engineering domain. However, the wide variety of vehicles (two-wheelers, three-wheelers, cars, buses, trucks etc.) plying on roads of developing regions without any lane discipline, makes vehicle classification and counting a hard problem to automate. In this project, we used state of the art Convolutional Neural Network (CNN) based object detection models and test them for multiple vehicle classes using data from KITTI dataset. As robust network connectivity is scarce in developing regions for continuous video transmissions from the road to cloud servers, we also evaluate the latency, energy and hardware cost of embedded implementations of our CNN model based inferences.

2 Introduction

Traffic congestion and air pollution levels are becoming life threatening in developing region cities like Delhi and the National Capital Region (NCR). The local government is being forced to take concrete steps to make public transport better by gradually adding subway and bus infrastructure [1], albeit under budget constraints. Policy decisions like odd-even rules (vehicles with odd and even numbered plates are allowed on alternate days) are being tried to curb the number of private cars [2, 3, 4]. A lot of times such policy decisions are met with angry protests from citizens in news and social media. In absence of data driven empirical analysis of the potential and actual impact of such urban transport policies, debates surrounding the policies often become political rhetoric. Building systems to gather and analyze transport, air quality and similar datasets is therefore necessary, for data driven policy debates.

This project focuses on a particular kind of empirical measurement, namely counting and classification of vehicles and pedestrians from roadside cameras. These numbers can be used in road infrastructure planning, e.g. in construction of signalized intersections, fly-overs, foot-bridges, underpasses, footpaths and bike lanes. Classified counts can also help in evaluating the effect of policies like odd-even, to see if private transport numbers go down during the policy enforcement period as expected. We discuss these and more motivational use cases of automated vehicle counting and classification. We show how some of these use cases can benefit from empirical data.

Non-laned driving in developing regions with high heterogeneity of vehicles and pedestrians, make automated counting and classification a hard problem. State of the art computer vision methods of CNN based object detection, to handle this task. CNN models need annotated datasets from the target domain for supervised learning. Annotated video frames from in-vehicle and roadside cameras are available for western traffic, and have recently been in high demand to train computer vision models for self-driving cars etc.

We identified several differences between the annotated video and image datasets of western traffic and our traffic videos, that might cause the accuracy difference of training models on one dataset and testing on the other. Four-wheelers and motorbikes look similar across countries, but there are many vehicles in Delhi-NCR which look completely different from the western world (e.g. auto-rickshaws, e-rickshaws, cycle-rickshaws, trucks and buses). Secondly, our lack of lane discipline causes higher levels of occlusion, where a large vehicle like a bus is occluded by many smaller vehicles. Thirdly, our roads are not rectangular grid shaped as seen in developed country videos, but have different adhoc intersection designs, creating different views of the captured traffic flows. Finally, since self-driving cars is one of the main application focus in developed countries, many images and videos are captured from the view point of the driver. This view significantly differs from the view of traffic a road-side camera gets.

Vehicle counting and classification can be useful in two kind of applications. The first kind is delay tolerant, where processing can be done at any arbitrary latency after video capture. The computations in this case will affect long term policy like infrastructure planning or help in evaluating policy impact like that of odd-even rule. The second kind of applications require low latency real time processing. Here the computations can be used in catching speed violations based on vehicle class, or illegal use of roadways by heavy vehicles outside their allotted time slots. Low latency is needed to catch the violators and penalize them in real time. In this project, we mainly focus on prospect of real time inferences using trained CNN models, especially using on-road embedded platforms.

Why is embedded processing interesting to explore in this context? Since broadband network connectivity across different road intersections and highways is not reliable in developing countries, transfer of video frames from the road to cloud servers for running computer vision models on them can become a bottleneck. We therefore evaluate embedded platforms on their ability to run inference tasks i.e. given a pre-trained CNN based object detection model and a video frame, whether the embedded platform can process the frame to give classified counts. We measure the latency incurred and energy drawn per inference task on three off-the-shelf embedded platforms (Nvidia Jetson TX2, Raspberry PI Model 3B and Intel Movidius Neural Compute Stick). Our evaluations in Section 4 show the feasibility of embedded processing and also shows the cost-latency-energy trade-offs of particular hardware-software combinations.

Our detailed work is available at[1] for use by both computer vision and transportation researchers, and potentially also by government organizations working on road traffic measurement and management.

[1] `https://github.com/mayanksingh2298/detection_hardware_characterization`

3 CNN based object detection

The goal of object detection is to recognize instances of predefined set of object classes(car, bicycle, etc.) and describing location of each such object in an image using a bounding box. The first object detector came out in 2001 and was called the Viola Jones Object Detector. Although, it was technically classified as an object detector, it's primary use case was for facial detection. It provided a real time solution and was adopted by many computer vision libraries at the time. The field was substantially accelerated with the advent of Deep Learning. The first Deep Learning object detector model was called the Overfeat Network [13] which used Convolutional Neural Networks (CNNs) along with a sliding window approach. There are currently two methods of constructing object detectors- the single step approach and the two step approach. The two step approach is generally more accurate, whereas the single step approach has been faster and is highly memory efficient. The single step approach classifies and gets the location of the objects in a single step, whereas the two step approach divides this process into two parts. First one being the step where it generates a set of regions in an image with high probability of having an object. The second step then performs detection and classification of the objects by taking these regions as input. These two steps are known as Region Proposal step and the Object Detection step.

Since, the two stage models are relatively slower and have a higher memory and energy consumption than the one stage models, it is infeasible to run such models on embedded devices which have limited set of resources. Hence, we use one stage models such as YOLO and Single Shot Detection. Also, advantage of using YOLO is that not only it provides a realtime detection, but also it's accuracy is only slighty worse than the two stage model such as Faster RCNN. Whereas, SSD has slightly better accuracy than YOLO, but with a slightly slower run time.

In our experiments we use the YOLO object detector. We evaluate YOLO inference latency and energy on the embedded platforms. Additionally, we also evaluate the Mobilenet-SSD object detector, which has been reported to have similar accuracy and latency as YOLO. To reduce the complexity of implementing every CNN from scratch, software frameworks like Tensorflow, Caffe, Pytorch are available, where functions for basic computational units like convolution, RELU, pooling etc. are already implemented. CNNs can be created calling these base functions as required. We use Mobilenet-SSD implementations on Caffe and Tensorflow. Thus we evaluate three CNN object detection software frameworks: YOLO, Tensorflow Mobilenet-SSD and Caffe Mobilenet-SSD. We will refer to these as yolo, tf and caffe respectively in subsequent discussions. We download open source pre-trained models for these three softwares and only run the inference task on the three embedded platforms to measure latency and energy. Figure 1 shows the outputs of detection models on a small set of example images.

Figure 1: Sample images with object detection bounding box and class label outputs

4 Embedded CNN inference

The training and evaluation of the CNN based object detection models were done on a GPU server. For an on-road deployment of such a system, relying on fiber for transferring video from the road to the remote GPU server will be difficult in developing regions, as broadband connectivity across different road intersections might vary. Also, good cellular connectivity will incur recurring costs.

This motivates us to explore the possibility of in-situ computer vision on embedded platforms. If using pre-trained CNN models (trained on GPU servers), state of the art CNN based object detectors can run their inference stage on embedded platforms co-located with the camera infrastructure on the roads, then only the counts per object class can be sent to the remote server for further analysis. The raw video streams need not be transmitted. We evaluate the cost, support for CNN software framework, latency and energy of multiple off-the-shelf embedded platforms in this section, for different CNN based object detection inference tasks.

4.1 Embedded Platforms

Table 1 lists the embedded platforms used in our evaluations. The first platform, NVIDIA Jetson TX2 is the most powerful embedded platform available in the market with impressive CPU and GPU support, and significant memory size. Its cost however is 10-24 times that of the other two platforms evaluated. The second platform, Raspberry PI has powerful CPU cores and moderate memory size, at a very affordable price. The third platform, Intel Movidius Neural Compute Stick is a USB stick offered by Intel, which has specialized hardware called the Vision Processing Unit (VPU). Different computations necessary in vision tasks, like convolution operations in CNN, are implemented in hardware in this VPU. This hardware accelerator stick can be plugged into the USB port of a Raspberry PI, to create an embedded platform with boosted computer vision performance. Both Jetson TX2 and Raspberry PI run Linux based operating system like Ubuntu, on which different CNN software frameworks can be installed to run the object detection inferences given a trained model. We will refer to these three embedded platforms as jetson, raspi and movi respectively in subsequent discussion.

We also evaluated two Android smartphones from Motorola and Samsung, but their cost-latency trade-offs in running the inference tasks were not comparable to these three platforms. Smartphones are more generic platforms targeted towards personal use, where cost increases due to the presence of different sensors, radios, display and also to support rich application software. The additional hardware/software are not necessary for dedicated tasks like on road traffic monitoring, hence embedded platforms with less features as listed in Table 1 are more suitable. We therefore omit the evaluation numbers of smartphones from this discussion.

Platform	Cost (INR)	Processor	RAM
NVIDIA Jetson TX2	70K	ARM Cortex-A57 (quad-core CPU) @ 2GHz + NVIDIA Denver2 (dual-core CPU) @ 2GHz 256-core Pascal GPU @ 1300MHz	8 GB
Raspberry PI 3B	2.7K	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU	1 GB
Raspberry PI 3B + Intel Movidius Neural Compute Stick	7.8K	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU Intel Movidius Vision Processing Unit	1 GB

Table 1: Embedded Hardware Platforms

4.2 Inference

Our project majorly focused on testing how good the state of the art convolution networks work on embedded devices. Can they yield reliable results in real-time? Or is it that their latency is so high that they cannot be put to use unless better hardware and software is developed? To answer such questions, we had to install the necessary frameworks(TensorFlow and Caffe) in all our embedded devices, which was itself a very daunting task. After which we tried to benchmark the models on different systems. For a proper analysis, just benchmarking is not enough, we need to find the time taken to run each layer of the neural network and see where the bottleneck lies. These values can be used to improve both the hardware of these devices in future. As a measure of accuracy, we calculated the mean Average Precision score on the KITTI dataset. In the end we completed our documentation by looking at the energy consumption of each of these devices during inference.

4.2.1 Installation

Embedded devices like Raspberry Pi and NVIDIA Jetson TX2 are small computers in themselves. We need to install the proper system and application softwares to make them good for use. So we installed Ubuntu MATE 16.04.2 (Xenial) on the Pi while TX2 came loaded with NVIDIA's Linux for Tegra distribution based on Ubuntu. Then we built OpenCV 3.4.1 on each of them from the sources, TensorFlow $>$ 1.4 and BVLC Caffe on each of the two devices. To run a neural network on the Intel's Neural Compute Stick you need to compile your model into a form supported by the stick. Such tools are available in SDK which can be found on Intel's Github repository [2]. It also has tools to profile the model when run using the Neural Compute Stick. There is no need to install the whole

[2] `https://github.com/movidius/ncsdk`

SDK on the embedded device because we only need to do inference on it. For this reason Intel ships a subset of their SDK which they call NCSAPI and it takes much lesser memory. The concept of Movidius stick is relatively new, so their SDK do not support all the popular state of the art models yet. To quote an example, we weren't able to compile the Tensorflow implementation of Mobilenet SSD for the stick.

4.2.2 Benchmarking

We tried to convey an overall idea about the latency of the model on different systems by finding the time taken to read the image, infer the image and visualize the bounding box predictions on the image. The results are documented in the table below. [3]

Framework	Architecture	System	Read	Infer	Visualize
Caffe	SSD-Mobilenet	NCS+Computer	17	89	0.009
Caffe	SSD-Mobilenet	Jetson	22.4	127.8	0.6
Caffe	SSD-Mobilenet	Raspberry Pi 3 CPU	90.3	550.6	2.3
Caffe	SSD-Mobilenet	Raspberry Pi 3 Movidius	54.7	125.9	1.3
Caffe	SSD-Mobilenet	GPU Computer	12.2	24.2	0.1
Caffe	SSD-Mobilenet	Moto G4 Plus	197	1154	7.7
Caffe	SSD-Mobilenet	Samsung S7 Edge	9.56	200	0.33
Tensorflow	SSD-Mobilenet	Moto G4 Plus	83	1390	2.1
Tensorflow	SSD-Mobilenet	Samsung S7 edge	2	753	2.1
Tensorflow	SSD-Mobilenet	Jetson	18	219	0
Tensorflow	SSD-Mobilenet	Raspberry Pi 3 CPU	60	799	0
Tensorflow	SSD-Mobilenet	GPU computer	12.4	62.5	0
Caffe	Tiny-YoloV2	NCS+Computer	17	174	0
Caffe	Tiny-YoloV2	Jetson	28	123	1
Caffe	Tiny-YoloV2	Raspberry Pi 3 CPU	99	637	4
Caffe	Tiny-YoloV2	Raspberry Pi 3 Movidius	64	215	0.22
Caffe	Tiny-YoloV2	GPU Computer	11	16	0
Caffe	Tiny-YoloV2	Moto G4 Plus	130	1364	0
Caffe	Tiny-YoloV2	Samsung S7 Edge	27	232	0

[3] `time measured in milliseconds`

One thing is pretty clear that GPU+Computer takes the least time. Then comes the Jetson TX2. Talking about the raspberry pi device, when we use the Intel Movidius stick, it shows tremendous improvement over CPU. As we already mentioned NCS doesn't support the Tensorflow implementation of SSD-Mobilenet yet, hence no benchmarking has been done for it.

4.2.3 Profiling

We did a layer by layer profile of the neural network to find bottlenecks. The results are mentioned in the reports which can be found at our Github repository [4].

4.2.4 mAP scores

It was observed that all the three models were better at detecting vehicles than detecting pedestrians. This shows that good fine tuning is required before they can actually be put to use.

Framework	Architecture	Person class	Vehicle class
Caffe	SSD-Mobilenet	1.64%	22.56%
Tensorflow	SSD-Mobilenet	1.75%	26.92%
Caffe	Tiny-YoloV2	2.41%	7.22%

4.2.5 Energy values

Figure 2 shows the average current drawn with standard deviation as error bars on the left y-axis. Inference latency is shown on the right y-axis. The x-axis denotes different object detection software and embedded hardware combinations. The Movidius stick does not support running Tensorflow models, so the combination tf-raspi-movi is missing from the x-axis.

For a given CNN software (yolo, caffe or tf), there is a trend across the hardware platforms. Jetson uses high energy at low latency, raspi uses low energy with high latency and raspi-movi strikes an optimal balance using low energy comparable to raspi and incurring low latency comparable to jetson. For a given hardware platform (jetson, raspi or movi), there is again a trend among the CNN softwares. Yolo and caffe incur similar latency, while

[4] `https://github.com/mayanksingh2298/detection_hardware_characterization/tree/master/profiling/caffe/reports`

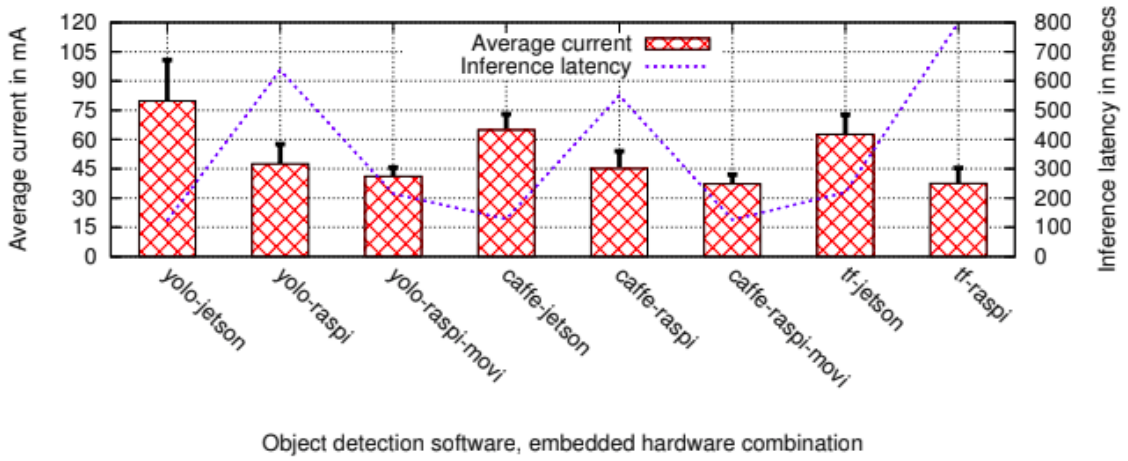

Figure 2: Inference latency and energy of object detection DNN on embedded platforms

caffe incurs slightly less energy than yolo. Tf uses similar energy as caffe, but incurs higher latency than both yolo and caffe, especially on raspi.

Given these trends, raspi-movi strikes a good balance of energy and latency in terms of hardware platform choice, at a moderate cost of 7.8K INR. This is significantly cheaper than the 70K INR jetson, while the jetson incurs similar latency and higher energy. In terms of software, both yolo and caffe Mobilenet-SSD are comparable in terms of latency and energy, while tf has higher latency especially on raspi and is currently not supported on raspi-movi.

While we evaluate and compare the different hardware platforms and software frameworks on the basis of latency and energy, depending on the application scenarios, one or both of these performance metrics might not be crucial. E.g. for an on road deployment, power is generally available from the lamp-posts or the traffic signals where the embedded computing units are deployed. For short term pilot deployments, not interfering with the road infrastructure and using battery supported units make sense, and energy efficiency will be important only in such scenarios. Similarly, if the goal of counting and classifying vehicles is to plan infrastructure or evaluate a transport policy like odd-even, low latency is not a necessity. Processing can be done at a slower rate, as no real time decisions will be taken based on the computer vision outputs. On the other hand, if the outputs are needed for real-time traffic rule violation detections and giving challans, then low latency becomes important. Thus the cost-latency-energy trade-offs of hardware-software combinations should always be considered in conjunction with the envisioned application requirements. Thus more important than the actual latency-energy values in our evaluation is the take-away, that state of the art CNN based object detectors can run on embedded platforms, thereby

making in-situ processing of video frames feasible without depending on broadband connectivity.

5 Potential Applications

We had discussed many applications of classified vehicle counts. Here we give a small example of how these applications can actually benefit from our system output. Table 4 shows the percentages of different object classes in our dataset. The numbers show a high dependency on private vehicles like cars and two-wheelers. This is the primary reason of increased traffic congestion and one of the potential factors in air quality degradation in Delhi-NCR. These values should be monitored when policies like odd-even are enforced to reduce congestion and air pollution, to check if public to private vehicle ratios improve, and whether absolute numbers of public vehicles like buses increase and cars come down.

Object class	Percentage on	Percentage on and
Bus	3.03151	2.77197
Car	52.3472	61.6214
Auto-rickshaw	10.3994	8.42592
Two-wheeler	26.5118	18.3368
Truck	3.66009	4.47696
Pedestrian	1.5993	2.23298
Cycle/E-rickshaw	2.45067	2.13398

Table 4: Percentages of different vehicle classes in our dataset

The absolute counts of different vehicles is useful also to estimate the Passenger Count Unit (PCU) [5], by multiplying the number of vehicles with the number of passengers each can carry. The PCU numbers will make infrastructure (signalized intersections, fly-overs, underpasses etc.) planning data-driven. Number of pedestrians can help in further planning of infrastructure like foot-bridges and side-walks. We will collaborate closely with the urban transport authorities to share our models for more extensive analysis across roads.

References

[1] "Relief for palam and cantt as delhi metro's magenta line opens this month," 2019. https://www.hindustantimes.com/delhi-news/relief-for-palam-and-cantt-as-delhi-metro-s-\magenta-line-opens-this-month/story-v3VnrbECxlmKjf1L469dbL.html.

[2] T. T. Energy and R. Institute, "Measures to control air pollution in urban centres of india: Policy and institutional framework," in *TERI Policy Brief*, 2018.

[3] "Paris implements odd-even rule to tackle smog," 2019. https://thewire.in/world/paris-odd-even-smog.

[4] "Congestion charge," 2019. http://www.politics.co.uk/reference/congestion-charge.

[5] "Passenger count unit (pcu)," 2019. https://en.wikipedia.org/wiki/Passenger_car_equivalent.