Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application

Chapter · January 2021

DOI: 10.1007/978-3-030-68017-6_51

5 authors, including:

Rodrigo Tufino
Universidad Politécnica Salesiana (UPS)
3 PUBLICATIONS 1 CITATION
SEE PROFILE

Paulina Morillo
Universidad Politécnica Salesiana (UPS)
12 PUBLICATIONS 13 CITATIONS
SEE PROFILE

Diego Vallejo
Universitat Politècnica de València
17 PUBLICATIONS 15 CITATIONS
SEE PROFILE

Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application

Sebastián Valladares[1], Mayerly Toscano[1], Rodrigo Tufiño[2], Paulina Morillo[2], and Diego Vallejo-Huanga[2,3](✉)

[1] Department of Computer Science, Universidad Politécnica Salesiana, Quito, Ecuador
{pvalladares,mtoscanor}@est.ups.edu.ec

[2] IDEIAGEOCA Research Group, Universidad Politécnica Salesiana, Quito, Ecuador
{rtufino, pmorillo,dvallejoh}@ups.edu.ec

[3] Department of Physics and Mathematics, Universidad de las Américas, Quito, Ecuador
diego.vallejo.huanga@udla.edu.ec

Abstract. The use of applications with Machine Learning (ML) is increasingly frequent and their daily use demands more capacity for processing information in real-time. Many devices deploy their infrastructure in Cloud Computing environments, where latency and network partitions are the main challenges they must face. However, in environments such as Dew Computing, these problems are mitigated since the information processing is carried out directly on the data source, so an Internet connection is not necessary. This article evaluates the performance of the Nvidia Jetson Nano platform, under a Dew Computing approach, with an application that uses ML to solve a problem of identification and counting of land vehicles, in real-time, in the city of Quito, Ecuador. Two types of experiments were conducted, the first experiment aimed to evaluate the use of processing resources (CPU and GPU), device temperature, power consumption, and the amount of RAM used in the ML task. The second experiment sought to evaluate the effectiveness of the platform combined with OpenDataCam.

Keywords: Vehicle detection · Dew computing · OpenDataCam · Darknet · YOLO · CUDA · GPU

1 Introduction

The drastic increase in computing capacities, and the improvement in the performance of telecommunications networks, have allowed the consolidation of paradigms that allow offering computing services remotely. Currently, there are three environments in which data is processed and distributed remotely [1]: Cloud Computing, with great features and computing power; Fog Computing, with less computing power but with processing closer to the data source and Dew Computing, where the information is processed directly in the end devices.

The main objective of Dew Computing is to take full advantage of the potential of local computers and cloud services [2], minimizing the impact generated by the loss of internet connection, ensuring security and privacy of the processed data [3]. The energy consumption in the execution of processes in integrated systems is very important since the power supply of these is small. In edge devices, resources are limited so large processing cannot be performed [1].

On the other hand, the use of applications with Machine Learning (ML) is increasingly frequent and their daily use demands more capacity for processing information in real-time. In 2019, the Nvidia company launches a device called Jetson Nano focused on the resolution of ML tasks, which allows fast and powerful deployment, under a Dew Computing approach. Jetson Nano has low power consumption at an affordable price, and is compatible with many AI frameworks, making it easy for developers to integrate their models into the product [4].

Currently, there are several scientific researches that uses methods for the detection and counting of objects with ML. In the work of Vaidya and Paunwala [5], the Jetson Nano platform is used for the recognition of traffic signs by means of Convolutional Neural Networks (CNN), to improve response times and execute processes in-situ. Gotthans et al. [6], with deep CNN, detect fires through video. The processing is done in Jetson Nano, resulting in a balance between precision and efficiency.

You Only Look Once (YOLO) [7], is an open-source system aimed at detecting objects in real-time, which uses a CNN to detect objects in images. For its operation, the CNN divides the image into regions, predicting identification frames and probabilities for each region, allowing a low detection error for new inputs. Lin and Sun in [8] use YOLO for vehicle detection. A limitation of this work is the detection of vehicles with low lighting in the video, especially at night. Asha and Narasimhadhan in [9], propose a method of counting vehicles with YOLO, which, in addition to tracking the object, determines its future trajectory.

This article presents an analysis of the performance of the Nvidia Jetson Nano platform and the effectiveness of performing a real-time ML application with YOLO for the recognition and counting of ground vehicles.

2 Materials and Methods

To analyze the performance of the Nvidia Jetson Nano platform, an ML algorithm was executed with a task of detecting ground vehicles: cars, buses, trucks and motorcycles, in real time, with a camera. Jetson Nano is a compact size device (69 mm $\times$ 45 mm) with Maxwell 128-core GPU, Quad-core ARM A57 CPU, 4 USB 3.0 ports; for storage, it has a MicroSD card slot, connections are via Gigabit Ethernet and M.2 Key E and a 2.0 Micro-B port to connect the power source [10]. The Jetson Nano platform does not have an embedded camera, but in this work, the Nvidia Jetson Nano HD AI model IMX219-77 was used to capture video in real time. The OS used was GNU/Linux Ubuntu 18.04.

For the identification and counting of vehicles, the OpenDataCam (ODC) [11] software was used, an open-source tool to quantify and visualize objects in real-time, through a web interface. In our work, we setup this tool with three features. The first one is an ML application with Darknet [12] that allows the identification of objects from an image, the second is a web application programmed in Node JS that allows user interaction with the software, and the last one is a MongoDB database to save the information. Darknet is compiled with the OpenCV libraries, for the acquisition of the video, CUDA for the computation on GPU, and YOLO [7] as neural network model. Figure 1 shows how the components of the application are related.

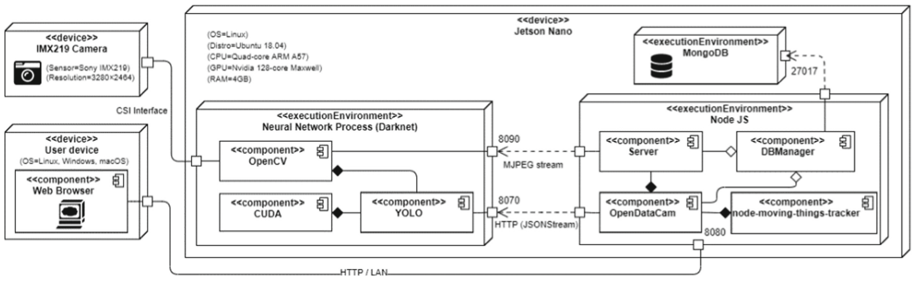

Fig. 1. Vehicle detection and counting system component diagram.

The process starts with the *Server* component, through which the user views the video in real-time, establishes the area of interest to identify the objects, and observes the counted and vehicle tracking. Besides, the configuration parameters are read, the web application is published and the connection to the database is initialized. The YOLO component is also initialized, configuring the parameters that the Darknet requires to identify the vehicles and the communication ports with the other components. Port 8090 is used to obtain the Video Stream and port 8070 is used to deliver the result of the object identification through a Json Stream.

On the other hand, the *OpenDataCam* component takes care of the logic for counting vehicles. In each video frame, the program sends a request to port 8070 to obtain the identified objects. This information is sent to the *node-moving-things-tracker* component [13], which contains an algorithm that allows persistent tracking of multiple moving objects from the object detection inputs provided by YOLO. This tracking is done frame by frame within the area of interest. When a vehicle has been identified, it is counted as a single object, and the record is stored in MongoDB. Also, its position and tracking information is sent to the web application.

The performance evaluation of Jetson Nano was carried out in two stages, the first one analyzes the performance of the Nvidia Jetson Nano platform with the Jetson Stats tool [14], using the metrics: CPU (%), the average use of its four cores in a given time; GPU (%), GPU usage over a period of time; Temperature (°C), the temperature at the end of each test; RAM (GB), amount of RAM memory used in each test and; Energy (mW), the total amount of energy consumed by the system in each test. The second stage determines the effectiveness of OpenDataCam, by calculating the Relative Error, based on the data obtained in three environments with changes in light with respect to the time of day and different amounts of traffic. For the performance analysis of vehicle recognition, i.e. detection or not of a land vehicle, ML metrics were used: Accuracy (Acc), Precision (Pr), Recall (Re), and F1 Measure (F1).

3 Experiments and Discussions

To make the system easier to manage and operate, a screen sharing tool was configured through Virtual Network Computing (VNC). Additionally, a static IP address (IP: 192.168.1.3, Mask: 255.255.255.0, Gateway: 192.168.1.1) was configured to establish a point-to-point connection, via a network cable, and avoid interruptions when using the remote display. The display monitors, with Jetson Stats, the temperature, CPU, GPU, RAM, and power usage that the Jetson Nano platform consumes while the algorithm is running. In turn, through the browser, it is possible to access the Node JS server for the execution of ODC.

The experiments carried out in the first stage were evaluated in three different scenarios in the city of Quito, Ecuador, considering three different time slots, between 09:00–10:00 (Morning, Mo), 13:00–14:00 (Noon, No), and 16:00–17:00 (Afternoon, Af), with different lighting levels and different levels of two-way traffic. Test 1 was carried out at Av. De los Conquistadores, with the system located at a height of 4 m, to have a focus on the larger vehicles. Test 2 was held at Av. Francisco de Orellana, and Test 3 at Av. 6 de Diciembre. These last two scenarios were taken from a perspective parallel to the vehicles, at a height of 1.45 m. with the purpose of evaluating the effectiveness of the system by having obstruction of the objects to be detected. The results of the three tests are summarized in Table 1.

Table 1. Nvidia Jetson Nano platform performance results measured with the Jetson Stats tool.

Test	Test 1			Test 2			Test 3		
Time slots	Mo	No	Af	Mo	No	Af	Mo	No	Af
CPU (%)	37.88	47.25	40.25	36.75	43.38	38.38	40.25	44.38	42.25
GPU (%)	99	99	99	99	99	99	99	99	99
Temperature (°C)	61.50	67.50	69.86	55.88	70.00	61.50	66.36	71.30	69.86
RAM (GB)	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1
Power (mW)	6297	6525	6871	7114	7780	7505	7467	7647	7449
Mo: Morning; No: Noon; Af: Afternoon									

Regarding the performance of the Jetson Nano, when executing the ML task, the use of the GPU remained constant at 99%, during all the tests, due to the fact that the calculation for the vehicle identification is performed on this component. RAM memory consumption remained at 2.1 GB, approximately 50% of its capacity. Energy consumption is directly related to vehicular flow. Thus, in Test 2, in the time slot with the highest traffic flow (Noon, with 431 vehicles), the highest peak of power consumption was obtained, which was 7780 mW. In most of the experiments, the relationship between CPU usage and temperature device variation was directly proportional.

On the other hand, the experiments carried out in the second stage, which aim to determine the effectiveness of the vehicle counting task, based on the manual counting of the vehicles (Real - Ground Truth) versus the counting and recognition of the System in the three environments with traffic, are shown in Table 2. In this case, it is observed that the system does not detect smaller vehicles, such as motorcycles, in contrast to larger vehicles such as trucks, where the system tends to double the count. This inconvenience could be related to factors such as the position of the camera (height and perspective), the incidence of light (time slot), size of the vehicle, and vehicular traffic. For example, in Test 1 where the camera position is higher, 262 out of 304 vehicles were detected, of which no motorcycle-type vehicle was detected, but most other vehicle types were correctly detected. In Tests 2 and 3, in which the camera was lower and the vehicular traffic was higher, the detection of motorcycles slightly improved, but the error in the detection of truck-type vehicles increased considerably.

Table 2. Summary of vehicle identification and counting results, and their relative error.

		Number of Vehicle												Total
Hour		**Mo**				**No**				**Af**				
Type of Vehicle		🚗	🚌	🚚	🏍	🚗	🚌	🚚	🏍	🚗	🚌	🚚	🏍	
Test 1	Real	107	1	2	7	75	2	3	8	87	2	1	9	304
	System	95	0	2	0	74	2	3	0	83	2	1	0	262
	Error (%)	11.2	100	0	100	1.3	0	0	100	4.6	0	0	100	13.82
Test 2	Real	231	2	8	44	357	8	12	54	366	6	8	34	1130
	System	228	2	21	5	277	3	19	4	183	0	35	6	783
	Error (%)	1.3	0	162.5	88.6	22.4	62.5	58.3	92.6	50	100	337.5	82.3	30.71
Test 3	Real	300	5	6	37	297	8	12	54	288	5	1	46	1059
	System	215	6	12	2	230	9	13	11	212	9	4	7	730
	Error (%)	28.3	20	100	94.5	22.6	12.5	8.3	79.6	26.4	80	300	84.8	31.07

As can be seen in Fig. 2(A), the perspective of the camera located at a higher height in Test 1, facilitates the count, avoiding the occlusion of the objects. However, the small size of motorcycles makes their detection difficult, Fig. 2(B). In Tests 2 and 3, where the perspective of the camera was parallel to the vehicles, the detection percentage was lower, due to the occlusion generated in smaller vehicles concerning others.

Finally, regarding the performance of the model used in object recognition, i.e., vehicle recognition regardless of type, Vehicle (V) or Non-Vehicle (NV), shown in Table 3. The results indicate high percentages of sensitivity in all experiments. The highest precision and accuracy were obtained in the first test (with 86%, respectively), where vehicular traffic was lower and the camera position was higher.

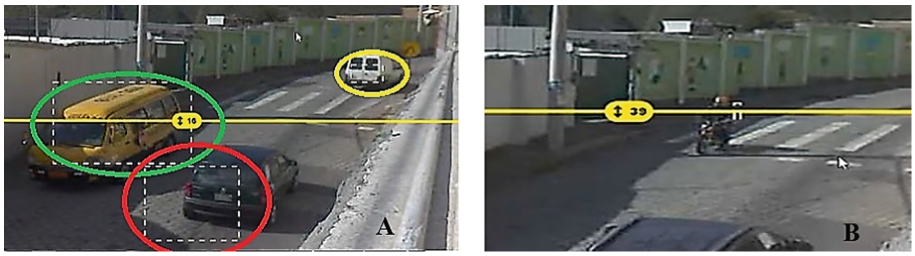

Fig. 2. (A) Detection of three vehicles, two cars and one bus for Test 1 in Av. De los Conquistadores. (B) Failure to detect motorcycles from a height of 4 m in Test 1.

Table 3. Performance evaluation of the model used in the ML task for Vehicle (V) and Non-Vehicle (NV) recognition.

			Real class		ML evaluation metrics			
			V	NV	Acc	Pr	Re	F1
Predicted class	Test 1	V	262	0	0.86	0.86	1	0.93
		NV	42	0				
	Test 2	V	689	47	0.61	0.64	0.94	0.76
		NV	394	0				
	Test 3	V	698	16	0.66	0.67	0.98	0.79
		NV	345	0				

4 Conclusions and Future Work

This paper presents the performance evaluation of the Nvidia Jetson Nano platform on a real-time ML application. According to the results, it can be concluded that Dew Computing solutions support applications that require great processing such as video image recognition. However, the use of GPUs can cause the temperature increase in the devices. On the other hand, the use of OpenDataCam for the counting and identification of vehicles turns out to be effective, but it is necessary to take into account factors such as the position of the camera, lighting, size and speed of the vehicle and vehicular traffic. As future work, it is proposed to improve the configuration of Machine Learning algorithms to increase the precision of the results.

Acknowledgments. This work was supported by IDEIAGEOCA Research Group of Universidad Politécnica Salesiana in Quito, Ecuador.

References

1. Kotlar, M., Bojic, D., Punt, M., Milutinovic, V.: A survey of deep neural networks: deployment location and underlying hardware. In: Symposium Neural Networks Application NEUREL 2018, pp. 1–6 (2018)

2. Wang, Y.: Definition and categorization of dew computing. Open J. Cloud Comput. **3**(1) (2016)
3. Wang, Y., Karolj, S., Rindos, A., Gusev, M., Yang, S., Pan, Y.: Dew computing and transition of internet computing paradigms. ZTE Commun. **15**, 133–136 (2017)
4. Cass, S.: Nvidia makes it easy to embed AI: the Jetson nano packs a lot of machine-learning power into DIY projects-[Hands on]. IEEE Spectr. **57**(7), 14–16 (2020)
5. Vaidya, B., Paunwala, C.: Traffic Sign Recognition Using Color and Spatial Transformer Network on GPU Embedded Development Board (2019)
6. Gotthans, J., Gotthans, T., Marsalek, R.: Deep convolutional neural network for fire detection. In International Conference Radioelektronika, pp. 0–5 (2020)
7. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Conference Computer Vision and Pattern Recognition, vol. 2016-Decem, pp. 779–788 (2016)
8. Lin, J., Sun, M.: A YOLO-based traffic counting system. In: Conference on Technologies and Application of Artificial Intelligence, pp. 82–85 (2018)
9. Asha, C., Narasimhadhan, A.: Vehicle counting for traffic management system using YOLO and correlation filter. In: IEEE International Conference on Electronics Computing and Communication Technologies, pp. 1–6 (2018)
10. Nvidia Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit
11. Groß, B., Kreutzer, M., Durand, T.: OpenDataCam: An open source tool to quantify the world. Move Lab. https://www.move-lab.com/project/opendatacam/
12. Redmon, J.: Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/
13. Durand, T.: node-moving-things-tracker. GitHub. https://github.com/opendatacam/node-moving-things-tracker
14. Bonghi, R.: Jetson Stats. https://pypi.org/project/jetson-stats/