AI in an autonomous robotic system vision

A. Kozlowski, R. Mignon[1]

[1]*Section électronique, Haute école de la province de Liège, 4020 Liège (Belgium)*

(Dated: January 25, 2022)

This publication proposes a study of the Nvidia Jetson Nano 2GB development kit to develop a deep learning algorithm capable of detecting competition robots. Using transfer learning on the MobileNet algorithm, robots are detected on live camera feed and their coordinates on images can be retrieved. Finally a study of the Jetson Nano power consumption has been carried out.

INTRODUCTION

Before starting to program an AI to improve our robot, this publication will start with a theoretical aspect to understand what deep learning, machine learning and neural networks are. Then, since we are using the MobileNet algorithm, an explanation will be developed to understand how this algorithm works and its performance. Afterwards, our research will expose the performance of the Jetson Nano compared to other existing technologies. Next, a research on the difference between CPU and GPU has been done in order to determine why GPUs are mainly used for training deep learning algorithms. Following all this, we will describe the different steps to run an AI on the Jetson Nano and we will also detail the results of the consumption of the Jetson Nano in training and running. To finish, we will conclude by the results of accuracy to recognize a robot as well as suggested improvements.

THEORETICAL ASPECTS

Machine Learning and Neural Networks

Machine learning uses mathematical and statistical methods to analyze data and allow computers to "learn" how to make decisions or predictions. There are two types of machine learning :

- Supervised machine learning : depends a set of data generated by human that learn to software how to define data.
- Non supervised machine learning : depends the model acknowledgement from data and the comparison with others data.

These algorithms performances improve gradually as the datasets and training times increase and other models appear.

Artificial Neural Networks (ANN) are a specific type of machine learning process. They are constituted of "neurons" which receive input signals and produce an ouput signal depending of a threshold value. Those neurons can receive the same signal or not and as such, the ouput function is a linear combination of the input depending of the thresholds of each neuron. The algorithm will be trained with a database and the ouput will be evaluated using a loss function (which characterises the performance of the algorithm). By changing the various thresholds, strengthening and weakening certain connections, the algorithm will improve over time.

Deep Learning

Deep learning (DL) is a kind of machine learning based on ANN [1]. It mimics the structure and process of the human brain as it will decompose the tasks into various features performed one after an other. It can recognize representative data from unstructured inputs (in contrast to Machine learning) and products specific actions or decisions. Like machine learning, deep learning can be supervised or unsupervised. The main difference with a basic ANN is the number of layers of processing before the output (see FIG. 1). This allows neural networks to automatically extract features from raw data without additional human input. Neural networks separate data into chunks of data and send them to individual neurons of each layers for processing. Once each features have been completed, the network produces the final output layer and makes a decision.

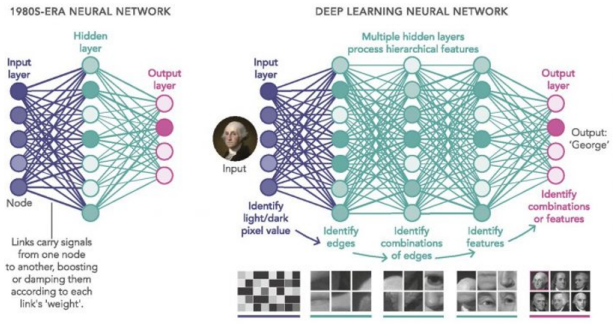

FIG. 1: Comparison between simple ANN and DL [3]

MobileNet Algorithm

This algorithm runs almost like a classical CNN (Convolutive Neuronal Network, several filtering and classifications of pictures). Only the convolution step is different. Indeed, for a CNN, the convolution is 3x3 size but, with MobileNet, the convolution is divided by 2 parts that we call depthwise convolution and pointwise convolution.

The depthwise convolution is 3x3 size and is used to apply a single filter to each input channel (a channel of an image represents a RGB level, so there are 3). Then, for the pointwise convolution, this layer applies a convolution of 1x1 size to combine the outputs of depthwise convolution.

In other terms, a standard convolution will take a picture with its channels and apply its filters then combine the results. Conversely, MobileNet will apply filters to each channel and then will recombine the results (the depth of parallelepipeds on the FIG.2 represents the channels)

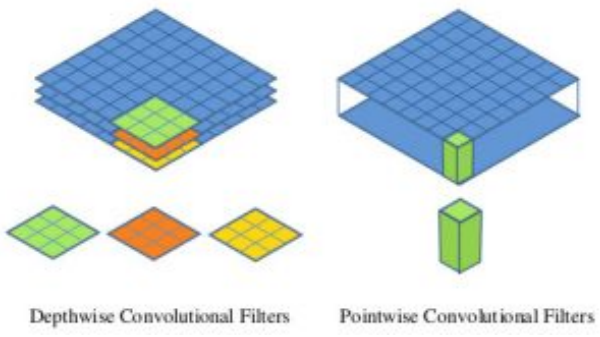

FIG. 2: Depthwise convolution and Pointwise convolution

MobileNet reduces computation times by a factor of 8 to 9 compared to standard convolutions. However, the accuracy is also reduced. In our case, it is more than enough to detect robots.

Transfer Learning

The principle of transfer learning is to use an already trained and proven algorithm and modify the last layers to allow it to perform a different but similar task. In our case, we will try to detect robots which were not part of the 90 classes present in the provided MobileNet-V2 algorithm. The algorithm will be fine-tuned by retraining and should be able to detect a new class as it already acquired "skills" on the previous training.

Jetson Nano 2GB specifications

Nvidia Jetson nano is a developer kit that has a built-in GPU composed of 128 cores, a CPU (Quad core) and 2 GB of RAM (LPDDR4). It is designed to train DL computer vision algorithms in real time via USB webcam connection [6].

There are other technologies that do the same but with different latency. This benchmark (in FIG.3) shows the time of latency (in fps) for a similar framework (Tensorflow) using different technologies (Jetson Nano, Raspberry, Intel Compute stick and Google edge TPU dev board).

We can see that for our Model (ModelNet-v2), the jetson nano is faster than the Intel Neural Compute stick 2 but slower than the Google edge TPU dev board. However, there is a significant difference in price (150 euros for Google vs 70 euros for Jetson nano). Regarding the performance/price ratio, Nvidia seems to be the best choice.

Furthermore, Nvidia provides tutorials to learn and practice IA computer vision.

Model	Application	Framework	NVIDIA Jetson Nano	Raspberry Pi 3	Raspberry Pi 3 + Intel Neural Compute Stick 2	Google Edge TPU Dev Board
ResNet-50 (224×224)	Classification	TensorFlow	36 FPS	1.4 FPS	16 FPS	DNR
MobileNet-v2 (300×300)	Classification	TensorFlow	64 FPS	2.5 FPS	30 FPS	130 FPS
SSD ResNet-18 (960×544)	Object Detection	TensorFlow	5 FPS	DNR	DNR	DNR
SSD ResNet-18 (480×272)	Object Detection	TensorFlow	16 FPS	DNR	DNR	DNR
SSD ResNet-18 (300×300)	Object Detection	TensorFlow	18 FPS	DNR	DNR	DNR
SSD Mobilenet-V2 (960×544)	Object Detection	TensorFlow	8 FPS	DNR	1.8 FPS	DNR
SSD Mobilenet-V2 (480×272)	Object Detection	TensorFlow	27 FPS	DNR	7 FPS	DNR

FIG. 3: Models performances on various boards

GPU over CPU

A CPU (central processing unit) works together with a GPU (graphics processing unit) to increase the throughput of data and the number of concurrent calculations within an application. GPUs were originally designed to create images for computer graphics and video game consoles, but since the early 2010's, GPUs can also be used to accelerate calculations involving massive amounts of data.

A CPU can never be fully replaced by a GPU. A GPU complements CPU architecture by allowing repetitive calculations within an application to be run in parallel while the main program continues to run on the CPU. The CPU can be thought of as the taskmaster of the entire system, coordinating a wide range of general-purpose computing tasks, with the GPU performing a narrower range of more specialized tasks. Using the power of

parallel-computing, a GPU can perform more computation in the same amount of time as compared to a CPU.

However, the presence of FPGAs in the electronic field allows to compete with the GPUs by accelerating the neural networking process as FPGAs can also perform parallel-computing. In addition, FPGAs use lower power than GPU. Given that we have done a consumption study, this advantage would be beneficial to develop in the near future for better performance at low consumption.

METHODOLOGY AND TESTING

We followed Nvidia tutorials [4] to understand how to adapt already existing algorithms and projects to create our own detection AI.

We firstly created a database of images containing robots. The database was then formatted to be compatible with the training algorithm. Training was performed for around 30 epochs and the created model was then exported into onnx. The detectnet algorithm was then used to run the previously exported model and we were able to see the performances on real time camera feed. Finally, the database was updated and the model was retrained and the results compared. A python program using detectnet and our model was written to display only the position of detected robots on the image.

Creation of the database

For the database we needed to find or take pictures of various robots that are commonly used during robot contests (such as Eurobot). We proceeded to find pictures of those kind of robot on the internet (mainly using google image).

The objective was to find robots of different size and shapes in various situations. If we increase the number of pictures with different camera angles and backgrounds, our model should be able to find robots in more situations than if we were to use only the same angle or the same background. In that case, changing the background would drastically reduce the performances of the detection which is why we decided to provide those kind of pictures. If we wanted to have a highly effective detection with the same point of view and the same background, we would have modified the database but in that kind of application the use of deep learning might not be justified.

The final database size was composed of 170 images. Increasing the number of pictures and retrain the model should increase the performances but would take a larger amount of training time.

Formatting data and creating directories

In order the use the training algorithm contained in the docker container provided by Nvidia, various files and directories had to be created. In the database directory, 3 other directories and a text file were created. We will explain their structure and their purpose.

The "labels.txt" file must contain the name of the classes in alphabetical order that we want the AI to detect. In our case there is only one class : "robot". While training the algorithm will automatically create a second class which is name "BACKGROUND" by default. In our case we want the algorithm to only find robots and consider everything else on the image as background.

The "JPEGImages" directory will contain all our images in ".jpg" format. Their size can be different but they should be renamed to only contain basic characters and we renamed them "robotX" with X the number of the image for simplicity.

The "Annotations" directory will contain a ".xml" file for each images. They are associated with the images using their names. They contain information about the bounding boxes that we created on each images to specify the position of the robots manually for training. In order to create those files we used a program called "labelimg" provided by the user "tzutalin" on github [2].

Finally the "ImageSets" directory contains another directory called "Main" in which 4 text files can be found. Those files will list which images the algorithm should use for training, validation and testing. A ratio of 80%/10%/10% is recommended and has been used.

Training Algorithm

For training the algorithm, the already provided by Nvidia "train.py" file was used. The mobilenet algorithm was selected as the base model for transfer learning and our training and validation sets were used to fine-tune mobilenet and obtain our model. We reduced the number of workers (limiting the parallel-computing power of the GPU) to avoid crashes and attain a number of 30 epochs which is considered to be enough to get a good detection. This limitation increased stability but increased the training time (8 hours in total). After testing, the database was modified and the algorithm retrained.

Exporting model and testing

In order to export the model, the "onnx_export.py" file was used to convert our ".pth" file into an usable "onnx" model. The detectnet program was then used to run our model and real-time live video streaming was used to check the detection results. After retraining the results were also compared and an improvement has been

made. Finally, a custom detection Python file was written in order to retrieve only data regarding the position of detected robots with a set threshold (80% of certainty to display the value).

Jetson Nano power consumption

We found a study[5] relative to the Jetson Nano power consumption where a shunt resistor was placed in serial connection with the power supply and an oscilloscope was used to probe each end of the resistor. The current through the resistor is calculated using Ohm's law. The product of this current with the input voltage results in power consumption.

In this study, the authors have selected four classification networks : MobileNetV1, MobileNetV2, ResNet18, and ResNet50. All of them were pre-trained for ImageNet classification. As an input, they are used a tensor with a size of 3x224x224 like we did.

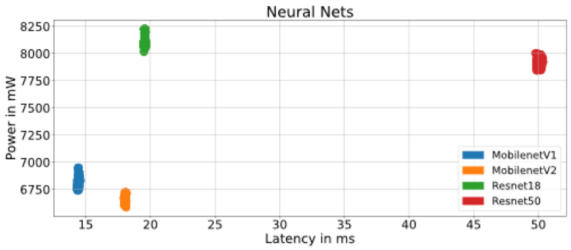

FIG. 4: Power consumption with the Jetson Nano

When comparing the MobileNets, we notice a slightly higher consumption in MobileNetV1. Furthermore, MobileNetV1 is 4ms faster than MobileNetV2. ResNet50 is the slowest, with a latency of 50ms. It is nearly 3 times slower than ResNet18. The power consumption of the two ResNets are similar. ResNet18 consumes a little bit more power than ResNet50. In general, MobileNets are faster and more energy-efficient than ResNets at a comparable accuracy. The fastest inference and most energy-efficient classification can be achieved with MobileNetV1. When power is optimized, MobileNetV2 is 1% better than V1. Smaller ResNets seem to be outdated; not only do they need much more power, but also, the latency is higher compared to MobileNets.

RESULTS AND DISCUSSION

Detection

Detection of robots was achieved with our custom model as can be seen on FIG. 5. Moreover, the live camera feed detection can be observed and the position of the detected robots was extracted.

The performances are good for a first approach with deep-learning detection algorithms but can still be greatly improved. Humans or other objects are sometimes mistaken for robots but robots are nearly always detected. Increasing the threshold value for detection minimises the number of wrong objects detected but sometimes causes robots not to be detected.

In order to improve the performances of the model, a larger robot database (over 1000 images in various backgrounds and point of view) should be used. As such, several hours of training are required because the number of images in the sets are larger and at least 30 epochs should be done. An important note for training is that too much epochs should be avoided as the over-fitting issue could arise (inducing the model to only be able to detect robots present in the database).

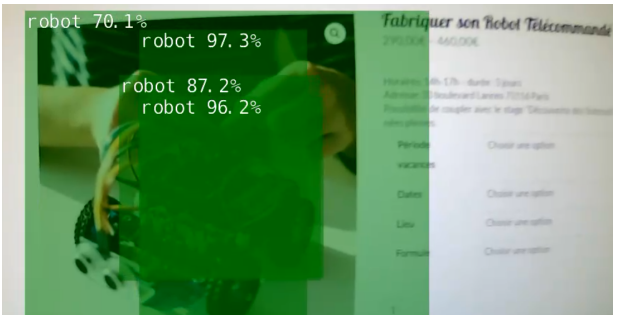

FIG. 5: Robot detection with live camera

Power consumption

Our experiment with a shunt resistor allowed us to obtain those plots FIG. 6 and FIG. 7

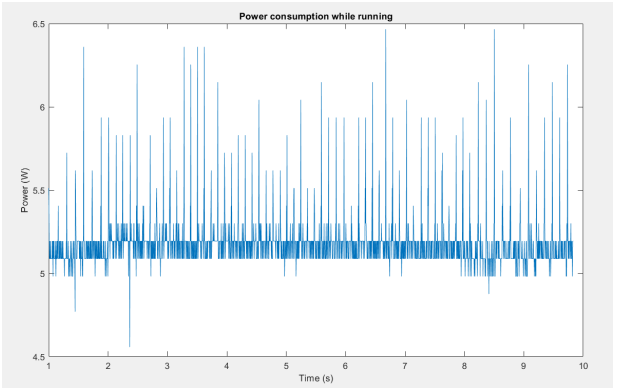

FIG. 6: Power consumptions while running the Jetson Nano

Using these plots, we have calculated the running mean value and the training mean value that are 5,18 W for the running mean value and 3.4716 W for the training mean value. It is important to notice that the training

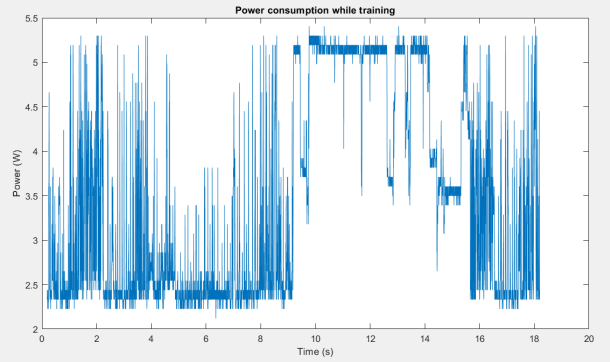

FIG. 7: Power consumptions while training the Jetson Nano

value can change and sometimes go to 6W. The algorithm indicated a CPU latency of 32ms.

CONCLUSIONS

We have used the Jetson Nano AI kit developed by Nvidia and we have followed their tutorial to run an algorithm of objects detection. Then, we have learned to create our own database of robot images to fine-tune a DL algorithm called MobileNet. After training, we exported the model and achieved robot detection. We performed a power consumption study while the Jetson Nano training the model and when it was running it. Due to its high power consumption it may not be the best solution to interface with a competition robot. Finally, we wrote a Python program that sends directions when a robot is detected to avoid collision.

ACKNOWLEDGMENTS

We would like to thank Professors Christophe BROSE, Sylvain GUICHAUX and Gilles SCHEEN for their continuous support and advice during the realisation of this project. We are thankful for this opportunity, the resources and material they provided. We could not have achieved those results without them.

[1] *Deep Learning* Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016)
[2] https://github.com/tzutalin/labelImg
[3] https://www.newworldai.com/what-is-deep-learning-nature-of-machine-learning-and-beauty-of-deep-neural-network
[4] https://github.com/dusty-nv/jetson-inference
[5] *Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano.* Stephan Holly, Alexander Wendt, Martin Lechner
[6] https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[7] https://www.intel.fr/content/www/fr/fr/products/docs/processors/movidius-vpu/myriad-x-product-brief.html
[8] *Fpgas and embedded vision applications.* Vision systems design
[9] *Fpga vs gpu vs cpu hardware options for ai applications.* Avnet
[10] *Cpu vs gpu.* Omnisci
[11] https://www.le-coin-du-digital.com/index.php/2018/07/11/deep-learning-vs-machine-learning Julie Lorenzini
[12] *Focus : MobileNet, une reconnaissance d'images temps réel et embarquée surpuissante.* Lambert R.
[13] *Accelerating DNNs with Xilinx Alveo Accelerator Cards.* Xilinx