Distributing deep neural networks for maximising computing capabilities and power efficiency in swarm

Victor Gacoin, Anthony Kolar, Chengfang Ren, Régis Guinvarc'H

Distributing deep neural networks for maximising computing capabilities and power efficiency in swarm

Victor Gacoin*, Anthony Kolar†, Chengfang Ren* and Regis Guinvarc'h*
*SONDRA, CentraleSupélec, France
Email: victor.gacoin@gmail.com, chengfang.ren@centralesupelec.fr, regis.guinvarch@centralesupelec.fr
†GeePs, Group of electrical engineering - Paris, CNRS,
CentraleSupélec, Univ. Paris-Sud, Univ. Paris-Saclay, Sorbonne Univ., France
Email: anthony.kolar@centralesupelec.fr

***Abstract*—Deploying neural networks models over embedded devices have an increased interest and many works is ongoing on that topic. Energy consumption, model sizes and inference time are critical issues as explained in the literature. In the context of IoT and edge computing, tradeoff have been studied in order to get a low cost but rapid answer, robust to connection issue exploiting early exiting or distributing deep neural networks. Those approaches exploits the cloud as an endpoint, balancing the load with respect to different computing capabilities. In this paper, we propose to extend those approaches to networks of embedded devices such as a swarm of drones, where every device has the same computing capabilities (in terms of energy and speed). Computing load may be balanced among the whole swarm in order to maximise either the lifetime of specific devices or lifetime of the whole swarm. We develop criteria to best cut and distribute those networks, validate them through power measurement and express the different tradeoffs we have to address.**

I. Introduction

As popularity of deep neural networks has increased over the past few years, such as computing capabilities, they have become much more bigger and energy consuming. It appears though that deploying neural networks models over embedded devices has an increased interest too and research on that topic is ongoing, especially regarding inference. Energy consumption, model sizes and inference time are important issues as explained in [1]. Common approach is to speed up inference optimising the structure of the network, reducing size models by various techniques such as SplitNet [2] which divides semantically parameters sets allowing at the same time parallelisation ; or by compressing, pruning and quantization as described in [3] with their "Deep Compression" framework. Even hardware architecture are developed to optimise some of those techniques (for instance efficient inference engine [4] specifically adapted to inference of compressed neural networks). Tradeoffs are also addressed to study, given specific use case, in what extent a design can be modified in order to keep an acceptable accuracy [5].

In the context of IoT and edge computing, tradeoffs have been studied in order to get a low cost but rapid answer, robust to connection issue, exploiting early exiting [6] or balancing the load over the Cloud, the Edge and end devices [7] [8]. Those approaches exploits the cloud as an endpoint, sharing the workload with respect to different computing capabilities and network latencies. More precisely, minimizing latencies is used as a principal criteria to define their distributed topologies.

We propose to extend in a context of embedded devices networks such as swarm of drones, where every device has the same computing capabilities (in terms of energy and speed). Computing load may be balanced among the whole swarm in order to maximise either the lifetime of specific devices (ones who carry sensors for instance) or lifetime of the whole swarm.

In this paper we try to adapt existing neural network design to a specific use case: distributing inference stage on several homogeneous embedded devices in an energy and computing efficient way. In the first part, splitting principles and criteria are formally introduced. We then apply latter methodologies to Resnet20 in second part, comparing different topologies and their accuracy and studying their energy footprint. Finally results are compiled with respect to our specific criteria and discuss these results.

II. Splitting principles

A. Linear graph

The basic idea is to split a given neural network into several parts. A neural network is usually implemented as a graph made of vertices (layers) and edges (tensors). A naive approach consists in identifying linear subgraphs, split them and extract them, interfacing inputs and outputs properly. Literally, given a linear graph $G(V, E)$, V being its set of vertices and E its set of edges, vertices can be listed $v_1, v_2, ..., v_n$ such that the edges are defined by $e_i = \{v_i, v_{i+1}\},\ i = 1, 2, ..., n-1$. We define a set of subgraphs $(G_i(V_i, E_i))_{1 \leq i < I}$ as an ordered partition of its sets V et E with $\forall i \in [1, I],\ V_i = (v_j)_{k_{i-1} \leq j < k_i}$ with $k_0 = 1,\ k_I = n$ and $\forall i \in [1, I],\ k_{i-1} < k_i$

Let's define a cost function C which quantifies the cost of a task with respect to a specific criterion. The latter can be an energy based cost function or a time based cost function for

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

instance. A task is performed by a vertex v_i of the graph. It takes as an input a data block b and produces the input of the next vertex which is b'. Then $b' = f(b)$ with f a deterministic function which could be linear or non-linear. Therefore, the cost of calculation of f by vertex i is $C(f_i(b))$. Let's now define transfer of data block b from vertex i to vertex j, $T_{i\to j}(b)$.

As described in fog computing, if $C(T_{k\to k+1}(b)) + C(f_1 \circ ... \circ f_k(b)) < C(f_{k+1} \circ ... \circ f_n(b))$, then it's worth transferring block b from vertex i to vertex j since this one is more efficient. Furthermore, a split is said balanced if $C(T_{k\to k+1}(b)) + C(f_1 \circ ... \circ f_k(b)) \approx C(f_{k+1} \circ ... \circ f_n(b))$.

Another advantage of such dividing is that it allows piping calculations. Imagine a stream of inputs on a sensor device, like a camera and the process has to be performed locally. Rate of processing, in other word the time taken by the device to process one frame, is a limitation. It is possible to reduce the charge of the sensor, delegating computations to other devices which are not, or less, busy.

B. Layer splitting

A graph can be split either horizontally or vertically. A vertical cut consists in dividing a linear graph into two linear subgraphs as defined before. It is equivalent to removing an edge. A horizontal cut is trickier, it aims at duplicating a linear sub-graph dividing its number of parameters, like if the graph was unzipped. One of these sub-graphs is called a branch as depicted on Fig 1. Assume a linear graph is made of convolutional nodes that consist of $2 * n$ convolutional filters. This node can be divided into 2 convolutional nodes made of n filters. Then the input edge (tensor) is duplicated in order to feed each sub-graph and output tensors have to be merged (concatenated) to form input of node k.

Therefore there are 2 main free parameters. The layer(s) where the network is eventually merged back k and the number of split n. Experiments are done to study the effect of k on the global accuracy for a given n.

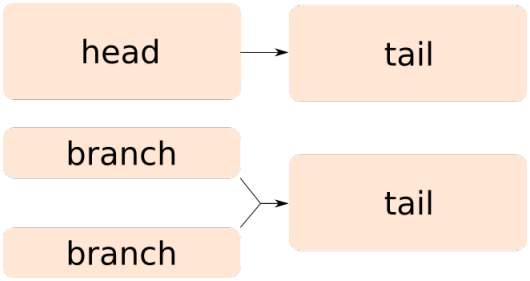

Fig. 1. Horizontal 2-split

While vertical split does not change the neural network topology, a horizontal split does. A network can be vertically split after training of the baseline model. However horizontal split remove some dependencies and parameters.

C. Performance criteria

When splitting a network vertically, forming a head and a tail, the goal is to maximise its accuracy while minimising the cost derived from its structure. Then, considering S as the layers set, let's define the performance as, $\forall k \in S$:

$$P_k = \frac{\overline{acc}_k}{\overline{C}(T_{k\to k+1}) + \overline{C}(f_1 \circ ... \circ f_k) + \overline{C}(f_{k+1} \circ ... \circ f_n)}$$

with:

- $\overline{acc}_k$ the average accuracy of the split model. Calculated from experimental results.
- $\overline{C}(T_{k\to k+1})$ the transfert cost of output tensor of node k to node $k+1$. It's based on the assumption that latency is proportional to the size of the tensor.
- $\overline{C}(f_1 \circ ... \circ f_k) + \overline{C}(f_{k+1} \circ ... \circ f_n)$ the average computation cost of the model, which is the sum of the average computation cost of the submodels. Calculated from experimental results.

This logic is still valid if the head is horizontally split, considering the cost of the head as a composed cost of its branches.

Considering this performance function, the objective is to find k that:

- maximises P ie $\arg\max_k P_k$
- balances the split

III. Application to Resnet

A. Models generation

In order to demonstrate our methodology, Resnet20 [9] has been chosen as a baseline model for its relatively small size (approximately 280k parmaters) and its specific design. Other well-known convolutional neural networks would have been chosen, the approach would be similar as well.
Resnet is made of 3 groups of 3 convolutional blocks. For clarity each block will be considered as a single complex layer. Therefore this new representation is made of 16 layers. Fig 2 shows output size of those layers which is directly proportional to the number of convolutional filters of each block and to input size. Then it actually shows how much computation each layer requires. One can see first layers are the most computation intensive.

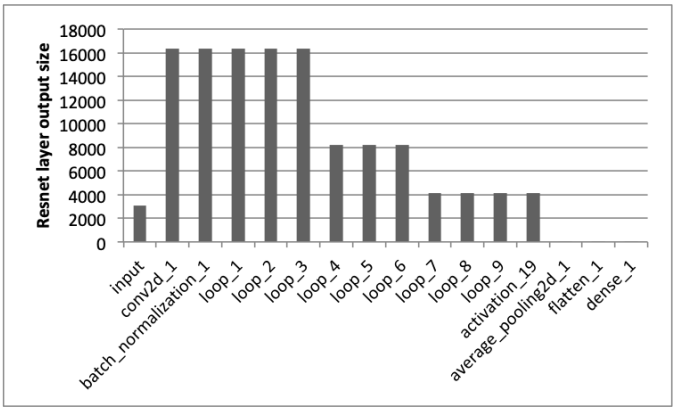

Fig. 2. Resnet layer output size

As mentionned before, horizontal split has an impact on the overall accuracy. Models must be trained given a specific topology. Table I show the evolution of accuracy according to different models. Models are trained on CIFAR10 dataset [10] made of 32x32 colour images belonging to 10 different categories. The training is performed over 50000 images and 10000 are used for testing.

TABLE I
SPLIT RESNET20 NETWORKS STATISTICS

	k	acc	loss
1-split		0.915	0.4267
2-split	6	0.9099	0.4491
	9	0.9083	0.4471
	12	0.8886	0.4728
4-split	6	0.9132	0.4438
	9	0.9061	0.4560
	12	0.8578	0.5198

While 1-split model corresponds to the baseline Resnet20, 6 different networks have been built from baseline: a 2-split and 4-split architecture with a vertical cut at $k \in \{6, 9, 12\}$. Values of k coincide to the last layer of each main block. They have been chosen to be consistent with the way Resnet20 is designed. Models are built with TensorFlow [11] in python, using a custom version available on the online repository. As a first observation of Table I, accuracy decreases with k which seems logical since there are fewer dependencies between convolutional filters and parameters due to splits. However the accuracy drop remains below 1% for $k \in \{6, 9\}$, and is higher for $k = 12$. This table merely gives hints to address the tradeoff between accuracy drop and number of splits but it remains dependant of the specific use case.

B. Power consumption

Now that the performance drop has been determined for different splits, let's consider the overall energy cost, for a device first and then for a swarm of devices. Note that consumption depends on the implementation. The purpose of the study is not to optimise the inference time on a given platform but rather to have an idea of the differences between the different networks architectures.

For this purpose an assembling similar to [12] has been used. A $100m\Omega$ shunt (R) is used to monitor real time current drawn by the Raspberry Pi 3 Model B.

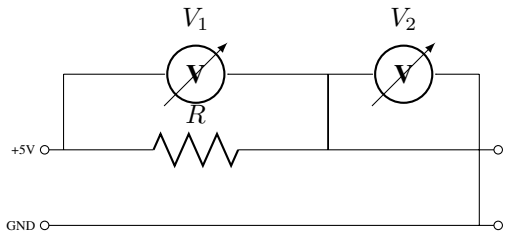

Fig. 3. Derivation of USB alimentation for power measurement

USB oscilloscope Digilent digital discovery 2 is used to measure V_1 and V_2. V_1 is acquired on the first channel, V_2 on the second with a precision of 14 bits. The power P is computed on the fly on the third channel as $V_1 * 10 * V_2$ derived from:

$$P = \frac{V_1 * V_2}{R}$$

CPU consumption is measured in 5 states:

- idle: nothing is running but necessary services: *dbus, dhcpd, dphys-swapfiles, fake-hwclock, networking, ssh, udev*
- downloading: simulated with the tool *speedtest-cli*
- uploading: idem
- loading: loading network parameter into memory takes some time and is easily identifiable while measuring the computation
- computing: corresponds to an approximate cpu load of 370%

Tensions are picked when states are stabilised, averaging over 512 ms to minimise side effects due to non relevant activity variations. Peak values are shown on Fig 4 as well, even if only mean values will be considered to get numerical values.

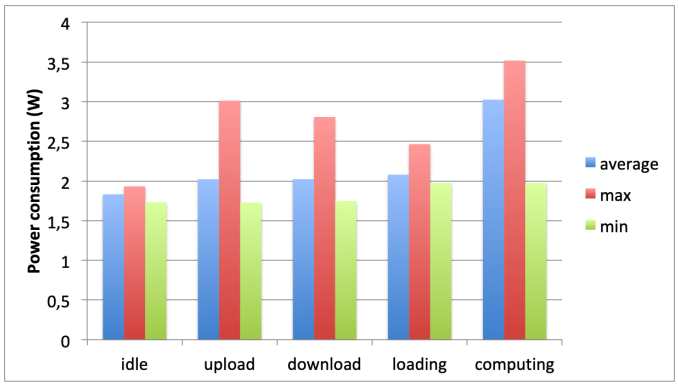

Fig. 4. Power consumption of different states (in Watt)

Since in normal mode it is difficult to segregate the state of the Raspberry Pi 3 Model B, we will define P_{idle} as the average of power consumption of idle, uploading, downloading and loading states shown on Fig 4. P_c corresponds to the power consumption of computing mode. Then, it gives $P_{idle} = 1.99W$ and $P_c = 3.03W$.

Consumption due to data transfer over wifi can be ignored since it does not increase significantly the overall power consumption. Then the split is balanced if $C(f_1 \circ ... \circ f_k(b)) \approx C(f_{k+1} \circ ... \circ f_n(b))$ and we can reduce the performance to

$$\forall k \in S,\ P_k = \frac{\overline{acc}_k}{\overline{C}(f_1 \circ ... \circ f_k) + \overline{C}(f_{k+1} \circ ... \circ f_n)}$$

Furthermore, consumption cost is directly proportional to execution time, therefore to inference time of submodels.

C. Performances of models

In the previous section power consumption costs have been determined. The overall performance of a model can therefore

be computed given the formula. We retained only 2-split model in this part to demonstrate the approach. The head corresponds to the subgraph $G_1(V_1, E_1)$ with $V_1 = (v_j)_{1 \leq j < k}$ and the tail to the subgraph $G_2(V_2, E_2)$ with $V_2 = (v_j)_{k \leq j < n}$ (head and tail are the result of a simple vertical cut). A branch is the subgraph G_1 "unzipped" i.e. where each convolutional node has half the number of filters than original node (result of horizontal cut on the head).

For a given energy E_a, assuming a homogenous consumption of the device, its lifetime in computing mode is $t_c = \frac{E_a}{P_c}$ and in idle mode $t_{idle} = \frac{E_a}{P_{idle}}$

TABLE II
INFERENCE TIME IN SECONDS ON RASPBERRY PI 3 MODEL B OF A 2-SPLIT RESNET20

k	6	9	12
head	0.05410	0.0669	0.08826
tail	0.05142	0.04052	0.00462
branch	0.02971	0.03455	0.05809

The total inference time of a distributed network is $t_{head} + t_{tail}$ for a simple vertical split (resp. $t_{branch} + t_{tail}$ for a horizontal split) and the total equivalent consumption is given by $P_c * (t_{head} + t_{tail})$ (resp. $P_c * (2 * t_{branch} + t_{tail})$, both branches are computed in parallel by 2 different devices).

Values shown on Table II give the inference time of networks and sub-networks for input made of a single image (batch of size 1) averaged over 1000 inferences. The reference time is the one of baseline model and is t_{base} =0.097s. For $k = 6$ one can see that inference times are balanced between head and tail. Then, a device handling head inference and another handling tail inference will consume about the same energy.

Imagine two scenarii:

- 2 devices sharing the workload
- 2 devices not sharing the workload, one taking over when the first is dead (no more energy available)

Which scenario optimises the number of inference N_{inf} ?

- first scenario: lifetime is $t_c = \frac{E_a}{P_c}$ and the rate is limited by the longest calculation $t_{max} = \max(t_{head}, t_{branch})$ (we discard time lost at the first inference considering devices are in computing permanent mode). Then,

$$N_{inf} = \frac{t_c}{t_{max}} \; i.e. \; \frac{N_{inf}}{E_a} = \frac{1}{P_c * t_{max}} = 6.1$$

- second scenario: device 1 is computing during t_c time, then device 2 which was in idle state takes over for a duration of $t'_c = \frac{1}{P_c} * (E_a - t_c * P_{idle})$. Then,

$$N'_{inf} = \frac{t_c + t'_c}{t_{base}} \; i.e. \; \frac{N_{inf}}{E_a} = \frac{1}{P_c * t_{base}} * (2 - \frac{P_{idle}}{P_c}) = 4.5$$

Therefore more inferences can be done per unit of energy using first scenario.

It's worth noting that a criterion on t_{max} can be determined from the second scenario:

$$N'_{inf} < N_{inf} \; i.e. \; t_{max} < \frac{t_c * t_{base}}{t_c + t'_c}$$

The same logic can be applied with 3 devices noticing than t_{branch} and t_{tail} are about the same for $k = 9$. We get 8.3 inferences per unit of energy with 3 devices sharing the workload against 4.9 with a relay of 3 devices.

TABLE III
PERFORMANCE OF DIFFERENT SCENARII

	base	balanced 1-split	balanced 2-split
P	3.09	2.85	2.74
N	3.4	6.1	8.3

Table III sums up the different results. It shows that the baseline model gets the best performance in terms of energy. That's consistent since inference time of submodels are poorer than baseline as shown in Table II. However (much) more inferences can be done with balanced submodels shared among 2 (resp. 3) devices. There is therefore a tradeoff between the number of inference N and the performance of the model P.

IV. CONCLUSION

This paper aimed at balancing load of a deep neural network inference among several embedded devices in order to maximise their lifetime. First, we propose a methodology to cut properly complex models into submodels in order to balance computation across devices. Then we have shown that for a given initial energy, more inferences can be performed in a realistic scenario by the group of devices sharing their ressources.

However, it only focuses on stacking inferences, in other words keeping devices busy in computing mode. But different scenarii could be imagined in order to increase the lifetime of the distributed system, fixing inference rate for instance, allowing devices to come back to an idle state between 2 inferences.

REFERENCES

[1] S. Ge, Z. Luo, S. Zhao, X. Jin, and X. Zhang, "Compressing deep neural networks for efficient visual inference," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, July 2017, pp. 667–672.

[2] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1866–1874. [Online]. Available: http://proceedings.mlr.press/v70/kim17b.html

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

[4] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 243–254.

[5] S. Shankar, D. Robertson, Y. Ioannou, A. Criminisi, and R. Cipolla, "Refining architectures of deep convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2212–2220.

[6] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Dec 2016, pp. 2464–2469.

[7] ——, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 328–339.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017, pp. 615–629. [Online]. Available: http://doi.acm.org/10.1145/3037697.3037698

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," vol. 1, 01 2009.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[12] F. Kaup, P. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," *39th Annual IEEE Conference on Local Computer Networks*, pp. 236–243, 2014.