EFFICIENT CONVOLUTIONAL NEURAL NETWORK WEIGHT COMPRESSION FOR SPACE DATA CLASSIFICATION ON MULTI-FPGA PLATFORMS

George Pitsis[*]*, Grigorios Tsagkatakis, Christos Kozanitis, Ioannis Kalomoiris*[*]*,*

Aggelos Ioannou, Apostolos Dollas[*]*, Manolis GH Katevenis*[†]*, Panagiotis Tsakalides*[†]

Institute of Computer Science
Foundation for Research and Technology, Hellas
N. Plastira 100, V. Vouton, Heraklion, 70013 Greece

ABSTRACT

Convolutional Neural Networks (CNNs) represent the cutting edge in signal analysis tasks like classification and regression. Realization of such architectures in hardware capable of performing high throughput computations, with minimal energy consumption, is a key enabling factor towards the proliferation of analysis immediately after acquisition. Our driving problem is a satellite-based remote sensing platform in which onboard signal processing and classification tasks must take place, given strict bandwidth and energy limitations. In this work, we exploit the implementation of a CNN on Field Programmable Gate Array (FPGA) platforms and explore different ways to minimize the impact of different hardware restrictions to performance. We compare our results against competing technologies such as Graphics Processing Units (GPU) in terms of throughput, latency and energy consumption. In actual experimental runs we demonstrate competitive latency and throughput of the FPGA platform vs. GPU technology at an order-of-magnitude energy savings, which is especially important for space-borne computing.

Index Terms— Convolutional Neural Networks, Field Programmable Gate Arrays, Hardware for Deep Learning, Remote Sensing, Space Data

1. INTRODUCTION

Deep Learning frameworks like Convolutional Neural Networks (CNNs) are becoming the new gold standard in a diverse set of signal analysis tasks, ranging from image classification [1] to seizure detection in EEG signals [2]. In the domain of remote sensing, CNN based methods have demonstrated exceptional performance [3], surpassing competing methods in numerous situations, from the fusion of observations from multiple platforms [4] to pixel level scene classification [5], and more exotic ones like identifying patterns in urban environments from satellite images [6].

However, despite their potential in term of analysis metrics, existing approaches still rely on the traditional acquire-compress-transmit paradigm, where the bulk of processing happens in dedicated ground-stations with large-scale resources that consist mainly of Graphics Processing Units (GPUs), e.g. [7]. This paradigm is facing significant challenges which stem from the dramatic increase in terms of observation volume at a minimal growth of available bandwidth, especially for miniaturized platforms like CubeSats or unmanned aerial vehicles [8, 9], not to mention the low latency requirements.

The onboard signal processing and analysis mandate the use of hardware platforms capable of high throughput analysis tasks at low latency while adhering to the severe energy limitations that characterize miniaturized platforms [10]. A major challenge towards this objective is that CNN-based inference requires multiplications and aggregations of millions of model parameters, typically encoded as 32-bit floating point numbers. Implementing floating point operations on hardware, however, results to significant size and power requirements, while a simple mapping to fixed point operations, such as the approach employed by the Tensor Processing Unit developed by Google [11] can lead to dramatic performance losses, as demonstrated in our experimental results.

In this work, we explore the realization of CNN-based inference on Field Programmable Gate Array (FPGA) platforms for the automated analysis of deep space observations acquired by satellites. The key novelties of this work include: (i) the exploration of different weight compression mechanisms, which include reductions on the representation of the stored weights through mapping to fixed point representations, as well as the clustering of weights through hierarchical clustering schemes that encode the weight distribution; (ii) the demonstration of how appropriate compression schemes allow for the efficient and scalable realization of the entire inference architecture, both in single- and multi-FPGA systems; (iii) the experimental demonstration that not only are

[*]also at the School of ECE, Technical University of Crete, Chania, Greece

[†]also at the Dept. of Computer Science, University of Crete, Heraklion, Greece

FPGAs an appropriate hardware platform for this task, but their performance in terms of throughput, latency, and energy efficiency surpass the established GPU paradigm.

2. DATA AND NETWORK DESCRIPTION

We focus our study on spectroscopic observations across a wide range of spectral bands with the aim of estimating the acceleration of the universe through a precise estimation of the redshift property associated with individual galaxies. As a case study, we consider simulated measurements which will be acquired by the upcoming ESA Euclid deep space mission, using publicly available specifications of the platform [12, 13]. We formulate the regression problem of estimating the continuous redshift estimation as a supervised multi-class classification task, where the potential range of redshift detectable by the instrument is divided into equally spaced bins [14]. More specifically, our example corresponds to a spectroscopic signal in the range 1.1-2.0 μm, encoded into 1800 spectral bands and associated with a one out of 800 classes corresponding to a particular redshift range z = [1, 1.8).

To perform the signal classification, we consider a CNN architecture comprising of three main types of layers, namely convolutional layers, non-linear activation layers, and fully connected layers. In the proposed architecture, three groups of convolutions and activations are utilized, where each convolution employs 8×1 kernels and the non-linear activation function corresponds to Rectified Linear Units (ReLU). The outputs of the third activation function are introduced to a fully-connected layer, which computes the output class through the use of a multi-class Softmax regression. The network was implemented using the Tensorflow library and trained using $400,000$ examples. The performance is quantified in terms of top-1 accuracy (or equivalent classification error), where a 99% classification accuracy was achieved for the testing examples [14]. The overall processing required for an input signal is 61 million floating-point operations.

3. COMPRESSION OPPORTUNITIES

Typical CNN models consist of millions of parameters, whose storage requirements are too large for the capabilities of memory-limited FPGA-based implementations. In FPGAs, the on-chip static memory, which is called Block RAM (BRAM), is immensely fast but its size is too small - just a few MB; on the other hand, external memory (Dynamic RAM - DRAM) has significantly larger size (tens of GB) but limited bandwidth. The CNN under investigation is parameterized by ∼22.7M weights, which require approximately 173 MB of storage using 64-bit double precision floating point variables. It, thus, becomes essential to reduce memory size in order to accelerate the network and achieve high throughput/Watt due to reduced memory traffic [15].

Previous work tries to address this challenge by restricting the weights of the CNN from Hybrid Block Float [16] to binary values [17, 18], however, the generalization capacity of such network is still under investigation. Our work is similar in the spirit to Han *et al.* [19], who demonstrated that a combination of pruning, weight quantization, and Human encoding, in conjunction with re-training of the network, can achieve an x49 reduction of the memory footprint. We demonstrate that additional gains can be achieved by introducing more sophisticated weight clustering schemes, including hierarchical clustering and inverse density normalization of codebooks. Furthermore, unlike [19], we demonstrate significant system performance gains with minimal impact in terms of analysis performance, i.e., without the need for retraining the CNN.

3.1. Weight Pruning

The first technique that we explored was the weight pruning of the fully connected layer, which sets weights under a threshold (called "pruning factor") to zero. The results of Figure 1 demonstrate that there is a trade-off between network accuracy and the pruning factor. Based on these results, we selected a pruning factor of 0.01 (highlighted in Figure 1), as it achieves the highest weight reduction (68.7%) with the least possible loss of network accuracy (0.2% error rate).

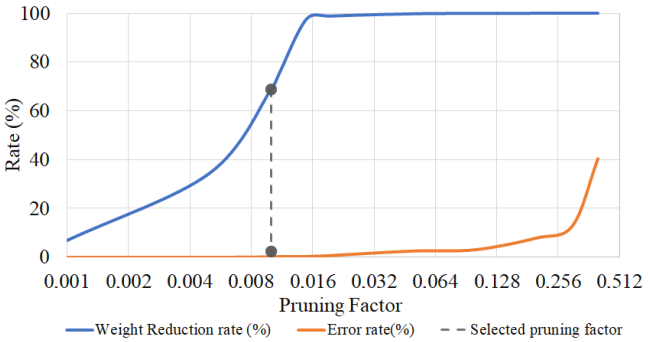

Fig. 1. Impact of weight pruning.

3.2. Using Static and Dynamic Fixed Point Operations

While using floating-point operations guarantees high accuracy results, floating point solutions in hardware are significantly more expensive in terms of resources, e.g., number of Digital Signal Processors. The alternative solution involves the use of fixed-point encoding which in FPGA designs are very efficient [15], provided the resolution and range of data are known beforehand so that the appropriate format is selected. Another option is the use of dynamic fixed-point encoding, where instead of using a global scaling factor, multiple factors can be used depending on the application's needs.

Table 1 reports the compression and associated classification error for both static and dynamic fixed-point formats (for the dynamic cases, several bits are used for the output layer,

while for all cases, the convolutional layers is encoded using 32-bit fixed point). The results demonstrate that although the memory footprint is improved significantly with static fixed-point, there is substantial reduction in accuracy. Dynamic fixed-point encoding improves the error rate vs. static encoding; nevertheless, a substantial error-rate is still present.

Table 1. Impact of fixed point encoding

	Format	Compression	Error rate (%)
Static	32 bit	x2	0.8
	24 bit	x2.7	3.8
	16 bit	x4	20.1
Dynamic	12 bit	x5.3	2.8
	8 bit	x8	8.1
	7 bit	x9.1	10.2

3.3. Weight Clustering

Our experimentation with fixed point (both static, and dynamic) of Section 3.2 revealed an important trade-off: the memory footprint can be significantly improved at the cost of prediction accuracy. To work around this trade-off, we explored a hybrid solution which maintains the floating point representation of weights (using double or single precision), but with a much smaller memory footprint. In this case, weights of similar magnitude are grouped together and represented using a small number of centroids obtained through clustering, such that for storing k centroids, only $\log_2 k$ bits are necessary. The impact in terms of accuracy associated with a different number of selected centroids is shown in Figure 2.

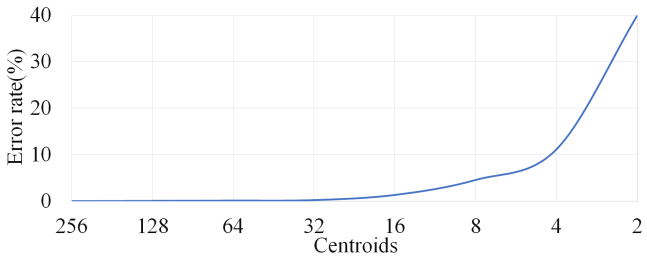

Fig. 2. Impact of single-level clustering of weights

To further increase the compression of weights, two extensions of the single-level clustering were explored, namely, hierarchical clustering and frequency based normalization. For the case of hierarchical clustering, we consider a two-level bottom-up approach such that initially weights are clustered to a larger cluster and subsequently to a smaller one. Furthermore, analysis of weights revealed that they coarsely follow a Gaussian distribution centered around zero. In order to exploit this observation we also impose a normalization, achieved by initializing the codebook of centroids based on the inverse of the frequency of occurrence. This way values around zero are more densely packed as compared to larger values, i.e., there is a small number of centroids representing values around zero as compared to large ones.

Figure 3 shows that for the same number of bits per weight, hierarchical clustering and frequency based optimization lead to significant performance benefits for a given bit budget. Based on these results, we opted for normalized hierarchical clustering of 256 and 16 centroids (i.e. we need only 4 bits per weight), thus achieving **x16** compression at a minimal **0.6%** classification error. The combination of the methods shown in Figure 3 is a major contribution of this work.

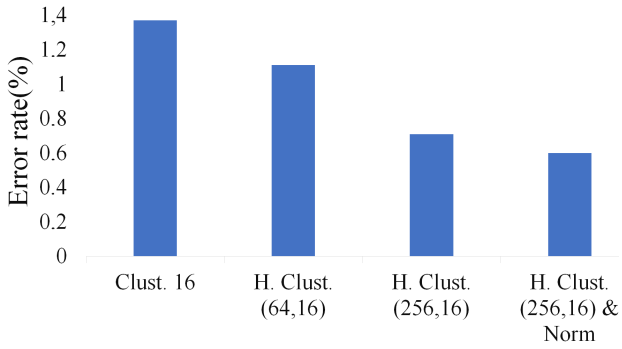

Fig. 3. Impact of different weight clustering schemes. H. Clust= Hierarchical Clustering, Norm= Inverse Density, (n,k)=(level1 centroids, level2 centroids)

4. HARDWARE IMPLEMENTATION

The first hardware architecture approach, based on the initial weights, was deployed in 2 FPGAs. The first FPGA computed the Convolutional Layers, while the second computed the Fully Connected Layer. The main bottleneck of this design was the I/O - 174 MB from main memory for every signal weight's footprint. Subsequently, we developed the optimized architecture using our results for the weight compression (see Section 3.3). We stored in FPGA the codebook (32-bit Floating Point) in the BRAM and use an on-FPGA 256-bit channel to stream the 4-Bit indices of weights. As a result, we end up with a 16x I/O reduction, being able to stream 64 weights in one cycle, thus uncovering massive parallelism at the operation level. Finally, resource optimizations were realized in order to fit the network into an FPGA. Another fully implemented optimization is the creation of a huge pipeline from the input signal loading the network until the final Classification. To achieve pipelining between network layers, shift-type FIFOs were used by extracting the intermediate results from previous layers in such a way that the subsequent layer can start processing the corresponding partial result.

4.1. Porting to QFDB FPGAs

The target board is the Quad-FPGA Daughter-Board (QFDB) designed as part of the EU-ExaNeSt project in order to offer both high compute density and high flexibility. It contains four Xilinx Zynq Ultrascale+ FPGAs (model: ZCU9), 64 GB

of DRAM (16GB per FPGA) and SSD storage. These FPGAs integrate four ARMv8 Cortex-A53 (PS), connected to the Programmable Logic (PL) through a low latency interface. The PL features 274KLUTs, 2520 DSP cores, 4 MBytes of BRAM and a DDR controller. An all-to-all topology is used for FPGA interconnection which is accomplished with the use of multiple high-speed serial transceivers, reaching 16.3Gbps.

The tremendous resource savings due to parameter compression create a new opportunity for parallelism: Batching. Instead of computing the results for a single data signal, two data signals can be processed in parallel by each device of the QFDB. By creating 4 Instances of the Accelerator we end up in Batch 8. Table 2 presents the improvement in the performance of the network after the use of compressed weights. More specifically, while using the same bandwidth from the DDR we end up with a **25x** speedup vs. the initial approach. The optimized design reaches a peak performance of 416 GFLOPS/s when its pipeline is full.

Table 2. Performance comparison between designs

Design	Throughput (Signals/s)	Performance (GFLOPS/s)	Bandwidth (GB/s)
Initial	173	10.5	9.23
Optimized	4334	265	9.23

5. COMPARATIVE EVALUATION

We compared our QFDB design against an NVidia Quadro P1000 GPU, which is of slightly better technology generation vs. the FPGAs of the QFDB. In both setups we took measurements from predictions on 10K signals and explored the impact in terms of throughput, latency and energy requirements for a diverse set of batch sizes in Figures 4 and 5.

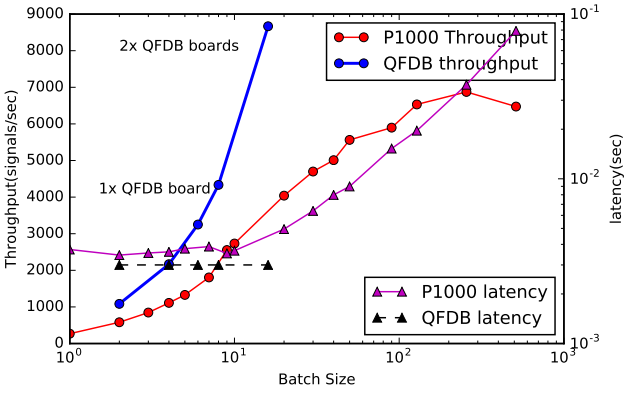

Fig. 4. Throughput and latency of QFDB and P1000 for different batch sizes, using one and two QFDB boards.

Figure 4 demonstrates that for the GPU, the throughput increases exponentially at the cost of latency as the batch size increases. Thus, the maximum throughput of the QFDB, which is achieved for a batch size of 8, outperforms the throughput of the P1000 GPU as long as the latter operates with a batch size smaller than 30. On the other hand, the latency of each batch increases by up to two orders of magnitude for large batch sizes, compared to the QFDB where it remains independent from the batch size. The high energy efficiency of the QFDB board allows the deployment of two QFDB boards to work in parallel in order to outperform the GPU throughput (see rightmost datapoint of Figure 4). According to Figure 5, the P1000 GPU achieves its lowest energy consumption at the highest batch size, as it takes the shortest time to complete the execution of 10,000 predictions. Yet, this requires $\times 5$ more energy vs. the QFDB. These results demonstrate that the use of two QFDBs in parallel achieve $\times \mathbf{1.23}$ better throughput vs. the P1000 GPU, while they consume in aggregate $\times \mathbf{5}$ less energy and they have more than an order of magnitude lower latency.

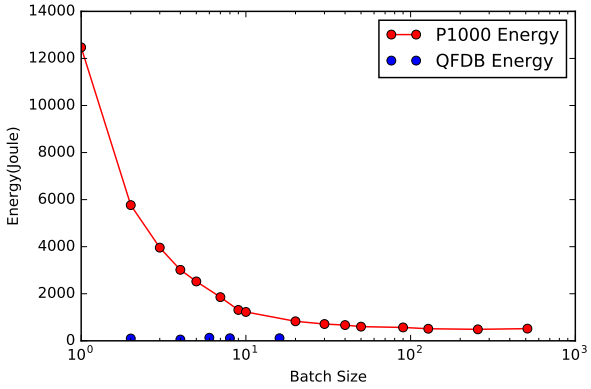

Fig. 5. Energy efficiency comparisons. A single QFDB takes a maximum of 109 Joules, while the P1000 requires 517 Joules (larger batch sizes where it's most efficient). For two QFDB boards running in parallel the energy is maintained at 109 Joules since the effective throughput doubles.

6. CONCLUSIONS

In this work, we systematically examine the impact of FPGA hardware constraints on the classification performance of CNN, and approaches to mitigate the identified challenges. By considering sophisticated weight compression schemes via clustering, we dramatically reduce the size of the CNN (from 174 to 11 MB). This reduction allows the realization of the network in a single FPGA, and thus leads to the parallel on-chip execution of inference. Experimental results demonstrate that the proposed solution achieves extremely competitive performance vs. typical GPU platforms in terms of throughput, latency, and energy consumption.

7. ACKNOWLEDGEMENTS

This work was carried out with support from the EuroExa (Grant Agreement 754337) and the DEDALE (Grant Agreement 665044) projects, funded by the European Union Horizon 2020 Research and Innovation Programme. The authors would like to thank Radamanthys Stivaktakis, Theodoros Zois, and Antonis Nikitakis for their contribution.

8. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] U Rajendra Acharya, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Hojjat Adeli, "Deep convolutional neural network for the automated detection and diagnosis of seizure using eeg signals," *Computers in biology and medicine*, vol. 100, pp. 270–278, 2018.

[3] Liangpei Zhang, Lefei Zhang, and Bo Du, "Deep learning for remote sensing data: A technical tutorial on the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 4, no. 2, pp. 22–40, 2016.

[4] Giuseppe Scarpa, Massimiliano Gargiulo, Antonio Mazza, and Raffaele Gaetano, "A cnn-based fusion method for feature extraction from sentinel data," *Remote Sensing*, vol. 10, no. 2, pp. 236, 2018.

[5] Amina Ben Hamida, Alexandre Benoit, Patrick Lambert, and Chokri Ben Amar, "3-d deep learning approach for remote sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, 2018.

[6] Adrian Albert, Jasleen Kaur, and Marta C Gonzalez, "Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1357–1366.

[7] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore, "Google earth engine: Planetary-scale geospatial analysis for everyone," *Remote Sensing of Environment*, vol. 202, pp. 18–27, 2017.

[8] Jacob Manning, David Langerman, Barath Ramesh, Evan Gretok, Christopher Wilson, Alan George, James MacKinnon, and Gary Crum, "Machine-learning space applications on smallsat platforms with tensorflow," in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2018.

[9] Steve Greenland, Murray Ireland, Chisato Kobayashi, Peter Mendham, Mark Post, and David White, "Development of a minaturised forwards looking imager using deep learning for responsive operations," in *4S Symposium 2018*, 2018.

[10] G. Lentaris, I. Stratakos, I. Stamoulias, K. Maragos, D. Soudris, M. Lourakis, X. Zabulis, and D. Gonzalez-Arjona, "Project hipnos: Case study of high performance avionics for active debris removal in space," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017, pp. 350–355.

[11] Norman P Jouppi, Cliff Young, Nishant Patil, and David Patterson, "A domain-specific architecture for deep neural networks," *Communications of the ACM*, vol. 61, no. 9, pp. 50–59, 2018.

[12] Giuseppe D Racca, René Laureijs, Luca Stagnaro, Jean-Christophe Salvignol, José Lorenzo Alvarez, Gonzalo Saavedra Criado, Luis Gaspar Venancio, Alex Short, Paolo Strada, Tobias Bönke, et al., "The euclid mission design," in *Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave*. International Society for Optics and Photonics, 2016, vol. 9904, p. 99040O.

[13] Racca G. Mellier, Y. and R. Laureijs, "Unveiling the dark universe with the euclid space mission," in *42nd COSPAR Scientific Assembly*, 2018, vol. 42 of *COSPAR Meeting*.

[14] Radamanthys Stivaktakis, Grigorios Tsagkatakis, Bruno Moraes, Filipe Abdalla, Jean-Luc Starck, and Panagiotis Tsakalides, "Convolutional neural networks for spectroscopic redshift estimation on euclid data," *arXiv preprint arXiv:1809.09622*, 2018.

[15] K. Kara, D. Alistarh, G. Alonso, O. Mutlu, and C. Zhang, "Fpga-accelerated dense linear machine learning: A precision-convergence trade-off," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 160–167.

[16] Mario Drumond, Tao Lin, Martin Jaggi, and Babak Falsafi, "End-to-end DNN training with block floating point arithmetic," *CoRR*, vol. abs/1804.01526, 2018.

[17] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[19] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.