Improving Inference Latency and Energy of Network-on-Chip based Convolutional Neural Networks through Weights Compression

Giuseppe Ascia*, Vincenzo Catania*, John Jose†, Salvatore Monteleone‡, Maurizio Palesi*, and Davide Patti*
* Dept. of Electrical, Electronic and Computer Engineering, University of Catania, first.last@dieei.unict.it
† Dept. of Computer Science and Engineering, Indian Institute of Technology Guwahati, johnjose@iitg.ac.in
‡ CY Advanced Studies & ETIS Lab, CY Cergy Paris Université, ENSEA, CNRS, salvatore.monteleone@u-cergy.fr

***Abstract*—Network-on-Chip (NoC) based Convolutional Neural Network (CNN) accelerators are energy and performance limited by the communication traffic. In fact, to run an inference, the amount of traffic generated both on-chip and off-chip to fetch the parameters of the network, namely, filters and weights, accounts for a large fraction of the energy and latency. This paper presents a technique for compressing the network parameters in such a way to reduce the amount of traffic for fetching the network parameters thus improving the overall performance and energy figures of the accelerator. The lossy nature of the proposed compression technique results in a degradation of the accuracy of the network which we show being, nevertheless, widely justified by the achievable latency and energy consumption improvements. The proposed technique is applied to several widespread CNN models in which the trade-off accuracy vs. inference latency and inference energy is discussed. We show that up to 63% inference latency reduction and 67% inference energy reduction can be achieved with less than 5% top 5 accuracy degradation without the need of retraining the network.**

***Index Terms*—Deep neural network accelerator; weights compression; approximate deep neural network; accuracy/latency/energy trade-off.**

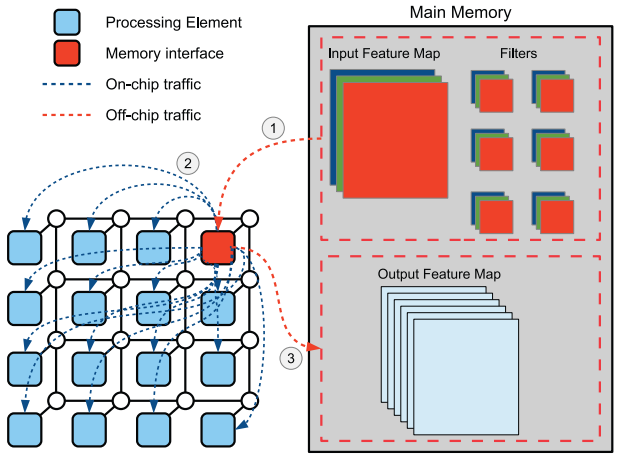


Fig. 1: Traffic generated in a convolutional layer: (1) load filters input feature map, (2) dispatch them to PEs, (3) store the output feature map into main memory.

I. Introduction

Many applications in the realm of natural language processing, visual data processing and speech and audio processing, are nowadays very effectively implemented by means of convolutional neural networks (CNN) [1]. Unfortunately, even the inference task sometimes requires computation capabilities and memory availability that are usually not available in resource constrained devices, including, mobile terminals and IoT edge devices. High computation and/or high memory demanding tasks are usually carried out in the cloud, making internet connectivity the limiting factor [2]. Due to the higher cost in terms of latency and energy which characterizes communication as compared to computation, the current trend is trying to compute as much as possible locally and parsimoniously using the internet. Thus, more and more mobile systems-on-chip (SoCs) are integrating accelerators for making it possible the execution of compute-intensive tasks, traditionally realized in the cloud, into the chip. The ever more use of deep learning based techniques in many domains, is pushing research and industry toward the design and development of high performance, low cost, high energy efficient Artificial Neural Network (ANN) accelerators [3].

The reference architecture of such accelerators is a many-core system in which cores are high parallel arithmetic specialized processing elements (PEs) interconnected by means of a Network-on-Chip (NoC) [4], [3]. The amount of parameters in typical CNN models can be as much as hundreds of megabytes as compared to the kilobytes scale memory usually available into accelerator. Thus, the high data volume i) to fetch network parameters and input feature maps from the main memory, ii) to dispatch them to PEs and, iii) storing back the output feature map into the main memory (Fig. 1), has a tremendous impact on both performance and energy metrics. Thus, reducing the amount of memory space needed to store the network parameters would improve the performance and the energy efficiency of the accelerator.

In this paper, we present a compression technique aimed at reducing the memory occupied by the parameters of the network, and, consequently, the amount of both on-chip and off-chip communication traffic. The proposed compression technique is lossy, in the sense that the compressed parameters when decompressed to be used by the PEs might not be exactly the same as the original ones (*i.e.*, before the compression).

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00017

However, thanks to the *forgiving nature* [5] of NN based applications, and the tunable compression ratio *vs.* approximation level provided by the proposed compression technique, interesting trade-off in the space accuracy *vs.* inference latency *vs.* inference energy consumption can be achieved.

Overall, the proposed compression technique:

1) Allows to reduce the memory footprint of an already designed and trained CNN without the need of retraining allowing the mapping into memory constrained devices (Sec. IV-B).
2) Can be applied on top of model compression approaches, including, parameter pruning and sharing, low-rank factorization, knowledge distillation, *etc.* [6] (Sec. IV-D).
3) Thanks to its tunable compression ratio nature, it allows to play in the multi-objective design space accuracy *vs.* latency *vs.* energy, giving the designer the freedom of selecting the most appropriate Pareto point for his/her current needs (Sec. IV-C).

The rest of the paper is organized as follows. Sec. II reviews recent techniques for compacting and accelerating CNN models. After presenting a motivational example aimed at showing the impact of communication traffic induced for fetching the model parameters during an inference, Sec. III presents the proposed compression technique. It is then applied to several CNN models and assessed in Sec. IV. Finally, Sec. V draws conclusion and outlines future work.

II. Related Work

In recent years, we have observed an ever more usage of deep learning based applications in many different contexts. Unfortunately, current neural network models require computational and memory capabilities that, in many cases, exceed that of portable devices with limited resources (*e.g.*, memory, CPU, energy, bandwidth). This problem calls for joint solutions from different disciplines, including, machine learning, optimization, computer architectures, data compression, *etc.*. As this paper is devoted on a new compression technique, in this section we review recent works on compressing neural networks.

Based on [6], model compression and acceleration for deep neural networks can be classified into four categories, namely, parameter pruning and sharing, low-rank factorization, transferred/compact convolutional filters, and knowledge distillation. Methods based on parameters pruning and sharing try to remove parameters that are not crucial to the model performance. The common practice is pruning redundant, non-informative weights in a pre-trained CNN model [7], [8]. Thus, pruning schemes typically produce connection pruning in CNNs. Quantization and binarization techniques are usually classified into the parameter pruning and sharing techniques category. Network quantization compresses the original network by reducing the number of bits required to represent weights with a consequent reduction of the memory footprint for storing the network parameters [9]. Another advantage is the possibility of using arithmetic hardware modules working on lower data width, with a consequent improvement in silicon area and energy consumption. Extremizing quantization techniques, binarization allows to represent weights with just a single bit [10], [11]. Methods based on low-rank factorization aim at reducing the convolutional layer by tensor decomposition which is motivated by the fact that there is a significant amount of redundancy in the 4D tensor [12]. As convolution operations contribute to the bulk of most computation in deep CNNs, reducing the convolution layer has a positive impact on the performance of the network. In [13], [14] the authors propose the use of transferred convolutional filters to compress CNN models. The basic idea is exploiting the equivariant group theory for applying transform to layers or filters to compress the whole network model. The last category of compression technique uses knowledge distillations [15], [16] to compress deep and wide networks into shallower ones. The compressed model mimics the function learned by the complex model.

Most of the aforementioned works require the accurate analysis of the original network, its transformation and re-training, either, incremental or from scratch. Conversely, the proposed compression technique is of general applicability and does not require any re-training phase. More important, it can be applied on top of other compression techniques to further improve the compression ratios.

III. Compression Technique

A. Motivational Example

To have an idea of how the memory traffic impacts the latency and energy metrics, we perform a layer level analysis during the execution of a CNN inference. The normalized latency and energy breakdown layer by layer for LeNet-5 is shown in Fig. 2. (Please also refer to Sec. IV for the experimental setup and simulation models.) As it can be observed, main memory access is the main responsible for the inference latency. The on-chip communication system, together with main memory determines the energy consumption of the accelerator.

The same conclusions can be drawn also for the other CNNs that will be considered in the experimental section. Reducing the memory traffic and on-chip communication volume would have a positive impact on both latency and energy metrics. Thus, in the next subsection, we present a new compression technique aimed at attaining such goal.

B. Lossy Compression of CNN Parameters

Several data compression techniques have been proposed in literature. The general idea is that of exploiting spatial redundancy characteristics and statistical properties, usually found into a data set, for the sake of compression. For instance, in vector graphics images, repetitive patterns of pixels with the same RGB components are very frequent and run length encoding provides high compression ratios. In our case, the data set is formed by the model parameters that, unfortunately, do not expose any redundancy or statistical property. In fact, their entropy is so high that makes unsuitable the application of any traditional compression technique. For instance, Fig. 3

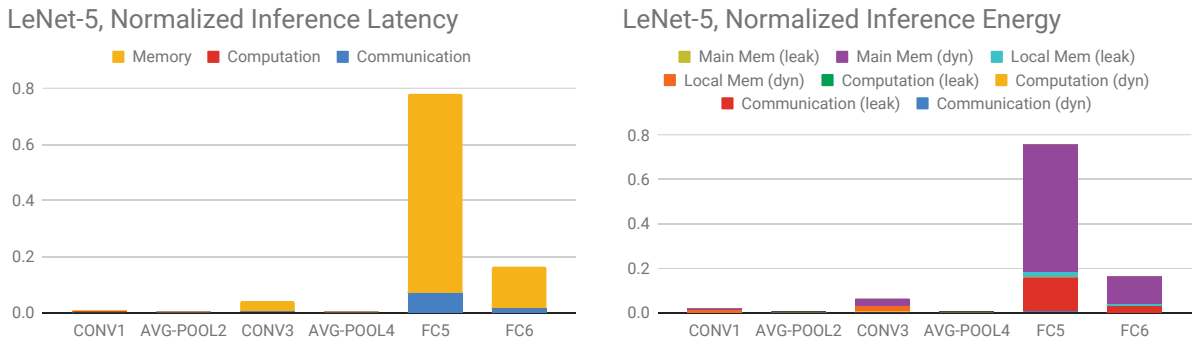


Fig. 2: Normalized latency and energy breakdown layer by layer for LeNet-5.

shows the entropy of the weights of different CNNs as compared to that of random data and that of a text file. As the entropy measures the equiprobability regardless of real unpredictability, the entropy of random data can be considered as an upper bound whereas that of a text file (which is usually characterized by a high level of redundancy) is shown for the sake of comparison. As it can be observed, the entropy of CNN weights is very similar to that random numbers thus showing that, traditional compression techniques would not be effective in compressing such data. Another problem is that compressed information have to be decompressed before being used. In our case, compressed weights and filters must be decompressed on the fly before being delivered to the PEs. Unfortunately, traditional compression techniques rely on the use of software implemented decompression algorithms usually hardly to be implemented in hardware. Based on the above considerations, we designed a new lossy compression technique specifically tailored to compress high entropy data set.

Let $W = \{w_1, w_2, \ldots, w_n\}$ be the succession of model parameters to be compressed. Let us partition the succession W in sub-successions such that each sub-succession is monotonic. That is, $W = \{M_1, M_2, \ldots, M_m\}$ where M_i is a monotonic sub-succession $M_i = \{w_{f_i}, w_{f_i+1}, \ldots, w_{l_i}\}$ where f_i and l_i are the indexes of the first and last element of the monotonic sub-succession M_i, respectively. For each monotonic sub-succession M_i, we calculate the linear regression using the least squares criterion on points (j, w_{f_i+j}) with $j = 0, 1, \ldots, l_i - f_i$. Thus, for each M_i we have the two coefficients of the line which minimize the mean squared error between the points in M_i and the points of the line. Let m_i and q_i be the coefficient of such line for the points of the monotonic sub-succession M_i. Thus, instead of storing into memory the original $w_i, i = 1, 2, \ldots, n$ model parameters, we can store, for each monotonic sub-succession, three parameters: the two coefficients of the line and the length of the sub-succession. A pictorial description of the compression technique is shown in Fig. 4. As it can be observed, the original 18 model parameters are clusterized into 6 clusters M_i each of them represented by the line computed by means of the least squares method applied to the parameters into M_i.

It should be pointed out that, the effectiveness of the proposed compression technique depends on the amount of monotonic sub-successions found into the succession of model parameters. The worst case is shown in Fig. 5(a) in which model parameters are pair by pair inversely monotonic. Here, the compression ratio is 1 as m will be equal to $n/2$. To increase the compression ratio, we relax the strict sense monotonic definition by introducing a tolerance threshold representing the maximum difference among two subsequent elements of the succession within which the monotonic criterion can be relaxed. Formally, we say that a succession $\{w_1, w_2, \ldots, w_n\}$ is monotonic decreasing in the weak sense with tolerance threshold δ if:

$$w_i > w_{i+1} \vee |w_i - w_{i+1}| \leq \delta \ \forall \ i = 1, 2, \ldots, n-1. \quad (1)$$

Thus, by replacing the strict sense monotonic definition with the weak sense monotonic definition for the construction of the monotonic sub-successions M_i, the average size of the clusters increases with a consequent increase of the compression ratio. For instance, considering the worst case scenario shown in Fig. 5(a), the use of the weak sense monotonic definition results to a single cluster as shown in Fig. 5(b).

C. Decompression Unit

After compression, the original model parameters are replaced with compressed model parameters in the form of pairs $\langle m_i, q_i \rangle$. Compressed model parameters have to be decompressed before being used by PEs. The decompression of a pair $\langle m_i, q_i \rangle$ involves the computation of the linear expression $m_i x + q_i$ when x is made to vary from 0 to $|M_i| - 1$. The decompressed model parameters will be, of course, an approximation of the actual model parameters. In contrast to compression which is performed off-line, decompression is critical as it must be performed on the fly and must not impact latency. The linear expression computation involves multiplication operation which is expensive. Fortunately, the decompressed model parameters need to be generated in sequence allowing to avoid multiplication. That is, for a

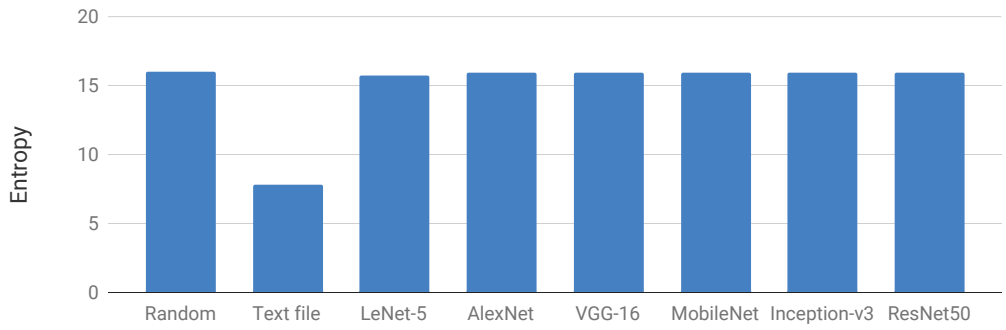


Fig. 3: Entropy of random data, text file, and weights of different CNNs.

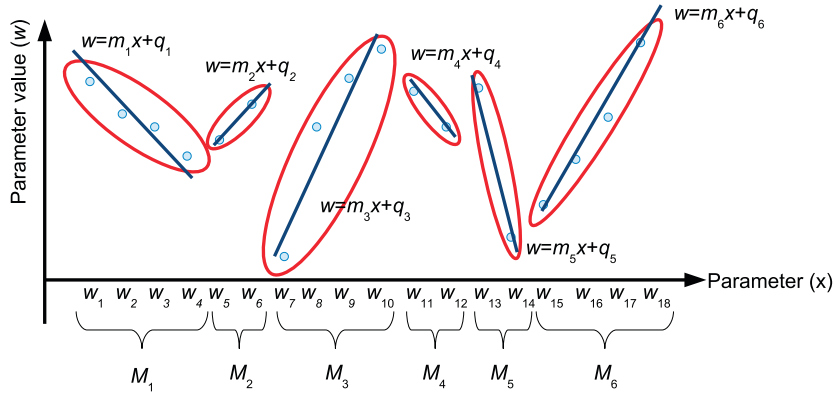


Fig. 4: Pictorial description of the compression technique.

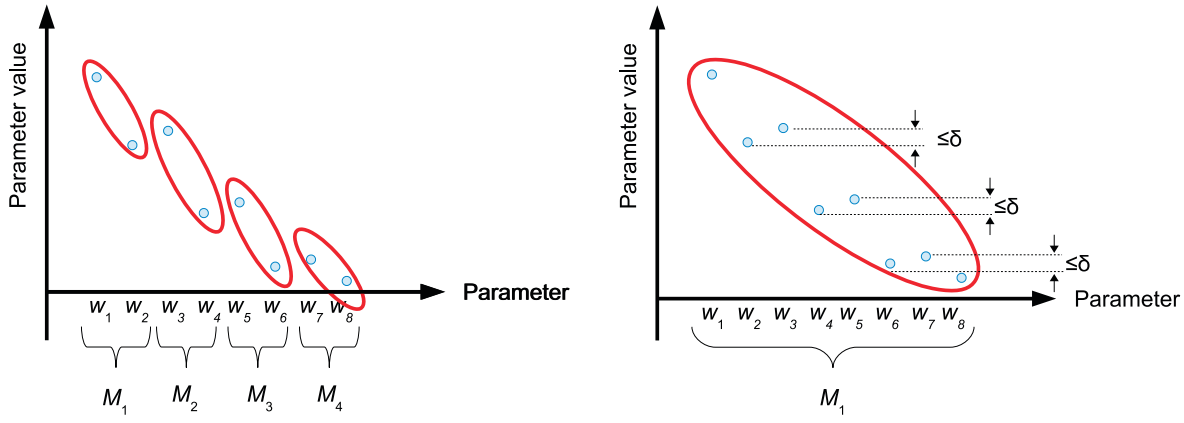


Fig. 5: Compression using the strict sense monotonic criterion (a) and the weak sense monotonic criterion with tolerance threshold δ.

given compressed model parameter $\langle m_i, q_i \rangle$, the uncompressed model parameters are computed as follows:

$$\tilde{w}_1 = q_i$$
$$\tilde{w}_j = \tilde{w}_{j-1} + m_i, \ j = 2, 3, \ldots, |M_i| \quad (2)$$

thus, no multiplication but just accumulation operations are needed for decompression.

The high level description of the decompression unit is shown in Fig. 6. Inputs are the compressed model parameters pair $\langle m_i, q_i \rangle$ and the number of approximated model parameters $|M_i|$ to be generated. The control unit is a simple two state FSM which allows to generate the approximated model parameters. In the *Init* state, the first approximated model parameter $\tilde{w}_1 = q_i$ in generated, whereas in *Run* state, the subsequent approximated model parameters $\tilde{w}_j$ are computed as in Eq. (2).

IV. Experiments

A. Experimental Setup and Evaluation Flow

The reference architecture used in the experiments is a NoC based CNN accelerator inspired to the Simba chiplet [4], in which the decompression unit presented in subsection III-C is embedded in each PE, as shown in Fig. 7. The NoC is a mesh with 16 nodes where the nodes in the corners host the memory interfaces whereas the rest are PEs. Links are 64-bit wide and the operating clock frequency is set to 1 GHz. Each PE includes 8 KB of local memory and 8 parallel lanes of vector multiply-accumulate (MAC) units. Each vector MAC performs an 8-way dot product and accumulates the partial sum into the accumulation buffer every cycle. The experimental platform is a simulated parameterized NoC-based Deep Neural Network that allows to assess different architectural configurations in terms of performance and energy [17]. The RTL models of the PE and router have been synthesized with Synopsis Design Compiler and mapped on a 45 nm CMOS LVT library from Nangate [18]. The links have been modelled with HSPICE and the parasitics extraction from layout has been made using Cadence Virtuoso. The power figures collected by the circuit level analysis have been used to back-annotate the cycle-accurate NoC simulator [19]. For memory, both local and main memory, we used CACTI [20] to estimate the energy consumption (both leakage and dynamic) and timing information.

We use the evaluation flow shown in Fig. 8 for assessing the performance, energy, and accuracy figures when the proposed compression technique is applied on different CNN models. The network model is first trained and then assessed on a test dataset to compute the top 5 accuracy of the original CNN model. The block *Layer Selection* selects the layer to be compressed (see below for details). The parameters of the selected layer are extracted and compressed for a user provided δ value. The compressed parameter are then decompressed to replace the original parameters obtaining the approximated network model, which is tested on the same test dataset obtaining the top 5 accuracy of the approximated CNN model.

TABLE I: Fraction of the parameters accounted by layers selected for compression.

Network Model	no. params x1000	Layer name	Type	Fraction
LeNet-5	62	dense_1	FC	80%
AlexNet	24,000	dense_2	FC	70%
VGG-16	138,000	dense_1	FC	77%
MobileNet	4,250	conv_preds	CONV	19%
Inception-v3	23,850	pred	CONV	9%
ResNet50	25,640	fc1000	FC	8%

The compressed model parameters replace the original model parameters to obtain the compressed network model. Finally, both the original and the compressed network models are simulated using the appropriate configuration of the simulation platform.

Regarding the *Layer Selection* block, the layer with the largest number of parameters and more in depth located is selected. In fact, in a preliminary analysis we found that, network accuracy decreases as soon as we compress layers close to the inputs and as soon as we increase the number of compressed layers. For this reason, in this work we decided to limit the compression to only one layer. Fig. 9 shows the sensitivity analysis for LeNet-5 and AlexNet. The sensitivity of a layer measures the impact on the accuracy when the weights of the layer are perturbed. As it can be observed, the sensitivity of the layers close to the input of the network is higher than that of the deepest layers. Thus, it justifies the layer selection policy of selecting the deepest and the larger (in terms of number of parameters) layer.

B. Compression Assessment

Before evaluating the impact on the accuracy of the network due to the approximation of the model parameters induced by the lossy nature of the proposed compression technique, let us first analyze the compression ratio achievable for different CNN models. We consider six representative network models, namely, LeNet-5, AlexNet, VGG-16, MobileNet, Inception-v3, and ResNet which cover a wide spectrum in terms of complexity both in the number of layers and number of parameters.

Tab. II reports the compression efficiency for different tolerance thresholds. For each network model, column δ is the tolerance threshold used for compression. It is reported as percentage of the max amplitude of the model parameters. That is, $\delta = x\%$ means that the δ used in Eq. (1) is $x \times [\max(W) - \min(W)]/100$ where W is the set of model parameters. As only one layer is compressed (see IV-A), the table reports two compression ratio columns. *CR* is the compression ratio referred to the compressed layer, whereas *Weighted CR* is the overall compression ratio weighted among all the model parameters. Column *Mem fp reduction* reports the percentage of memory footprint reduction due to compression. Finally, column *MSE* reports the mean squared error between the original model parameters and the approximated model parameters. Tab. I reports, for each network model,

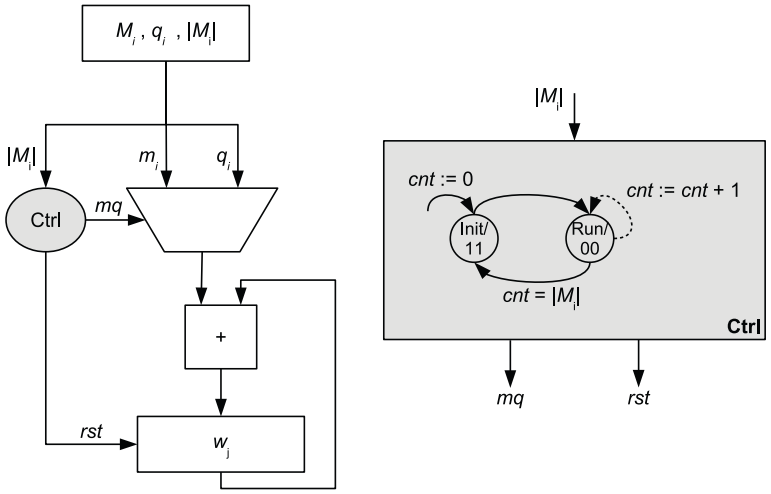


Fig. 6: High level description of the decompression unit: datapath on the left, and FSM of the control uniting on the right.

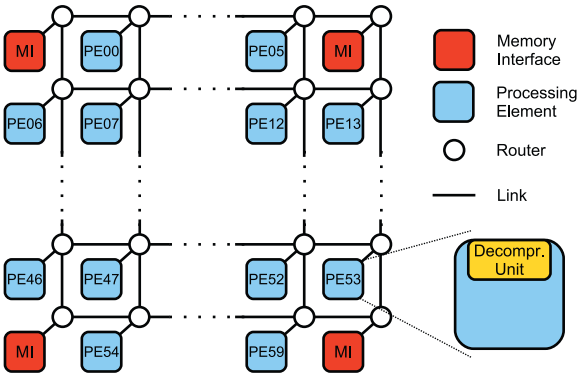


Fig. 7: Reference architecture of the NoC based CNN accelerator.

the total number of parameters, the name and the type of the compressed layer, and the fraction of parameters of that layer as respect to the total parameters.

For LeNet-5, we compress the first fully connected layer, which accounts for the 80% of the total model parameters. Over 2x of CR is obtained starting from $\delta = 15\%$ with a MSE in the order of 10^{-4} which corresponds to the 0.03% of the variation range of the model parameters for that layer. For AlexNet we compress the second fully connected layer which accounts for the 70% of the total model parameters. The CR is almost the same of that found for LeNet-5, but the MSE is in the order of 10^{-6} even for a δ of 20%. For VGG-16, we compress the first fully connected layer which accounts for the 77% of the total model parameters. Here we limit the δ variation to 8% as, even if the MSE is low, the top 5 accuracy (see next subsection) rapidly fall for higher δ values. For the same δ the CR is higher than what we found for LeNet-5 and AlexNet. Up to almost 5x memory footprint reduction is obtained for a δ of 8%. For MobileNet

TABLE II: Compression efficiency for different network models and different tolerance thresholds.

Network Model	δ	CR	Weighted CR	Mem fp reduction	MSE
LeNet-5	0%	1.21	1.17	14%	5.90e-5
	5%	1.38	1.30	24%	8.80e-5
	10%	1.74	1.58	39%	1.38e-4
	15%	2.50	2.17	57%	2.01e-4
	20%	4.02	3.36	74%	2.55e-4
AlexNet	0%	1.21	1.15	12%	9.23e-7
	5%	1.51	1.35	24%	1.69e-6
	10%	2.38	1.97	41%	3.04e-6
	15%	4.77	3.63	55%	4.25e-6
	20%	11.44	8.28	64%	4.96e-6
VGG-16	0%	1.21	1.16	13%	3.63e-8
	2%	1.43	1.32	22%	5.62e-8
	4%	1.94	1.70	36%	8.97e-8
	6%	3.04	2.51	50%	1.25e-7
	8%	5.28	4.18	61%	1.57e-7
MobileNet	0%	1.21	1.05	4%	1.40e-5
	2%	1.42	1.10	7%	2.06e-5
	4%	1.87	1.21	11%	3.20e-5
	6%	2.74	1.42	15%	4.49e-5
	8%	4.31	1.80	19%	5.59e-5
Inception-v3	0%	1.22	1.02	2%	4.16e-6
	5%	1.65	1.06	3%	7.97e-6
	10%	2.82	1.16	5%	1.37e-5
	15%	5.46	1.38	7%	1.83e-5
	20%	11.42	1.89	8%	2.12e-5
ResNet50	0%	1.21	1.02	2%	4.40e-6
	2%	1.76	1.06	4%	8.03e-6
	4%	3.31	1.18	6%	1.33e-5
	6%	6.57	1.45	7%	1.71e-5
	8%	12.79	1.94	8%	1.95e-5

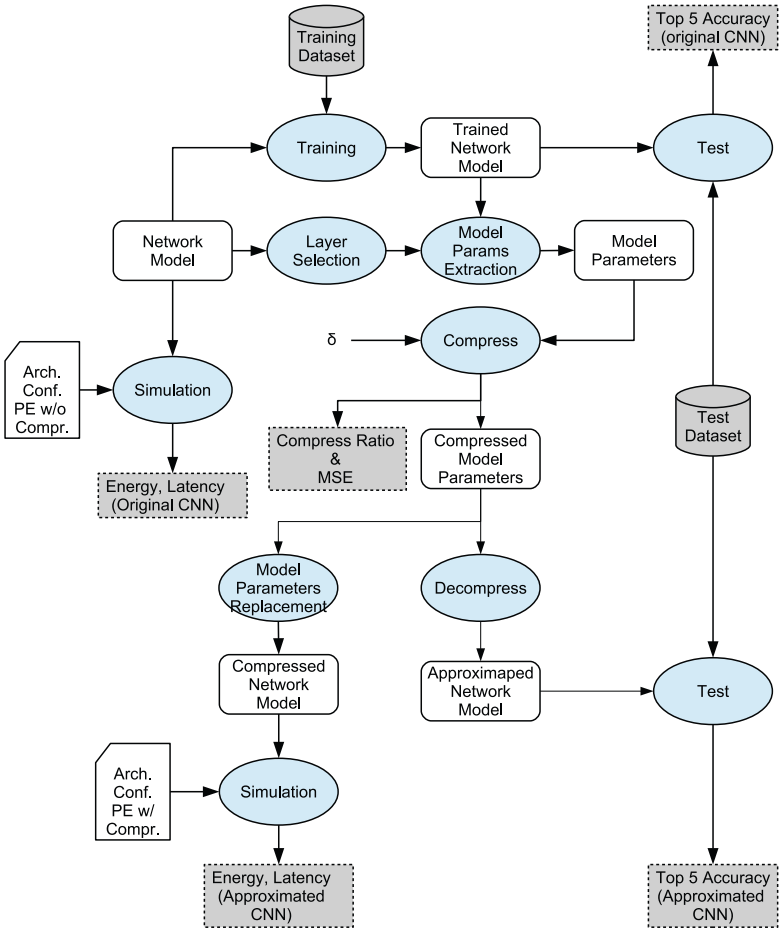


Fig. 8: Block diagram of the evaluation flow.

we compress the last convolutional layer which accounts for only 19% of the total parameters. For this reason, even if the compression ratio for this layer is as high as 4.3x for a δ of 8%, the weighted compression rate is as low as 1.8x. It should be pointed out that, MobileNet can be considered as an already compressed network model in which parameters are more uniformly distributed among the layers than the other considered network models. Similar results are found for Inception-v2 and ResNet50. As the analysis is limited to the compression of a single layer, and due to the fact that such layer accounts for less than 10% of the total parameters, the higher weighted compression rate is less than 2x even if the layer compression rate is greater than 10x.

C. *Accuracy* vs. *Latency* vs. *Energy*

We have seen how the proposed compression technique allows high compression ratios and low approximation levels of the model parameters in terms of MSE. The approximation of the model parameters impacts the accuracy of the network. On the other side, the reduced memory footprint reduces both the communication and memory traffic, with a consequent improvement of latency and energy figures. In this subsection, we assess the trade-off accuracy *vs.* inference latency *vs.* inference energy.

Fig. 10 shows, for each network model, the accuracy *vs.* inference latency and the accuracy *vs.* inference energy for different δ values. Inference latency and inference energy are normalized as respect to the inference latency and inference energy of the original network model, respectively. In each graph, the first data series refers to the original network model whereas the other series are for the compressed (*i.e.*, approximated) network models. The latter, are referenced with suffix x-δ, where x stands for *approximated* and δ is the considered δ value. Accuracy is reported on the left y-axes, whereas latency and energy are reported on the right y-axes. Both latency and energy are breakdown into their sub-components. Latency is breakdown into three sub-components, namely, memory, communication, and computation. Energy is breakdown into six sub-components, namely, communication, computation, local memory, main memory, and, for each of them, both dynamic (dyn) and leakage (leak) components are reported.

LeNet-5 results are shown in Figs. 10a and 10b. Please

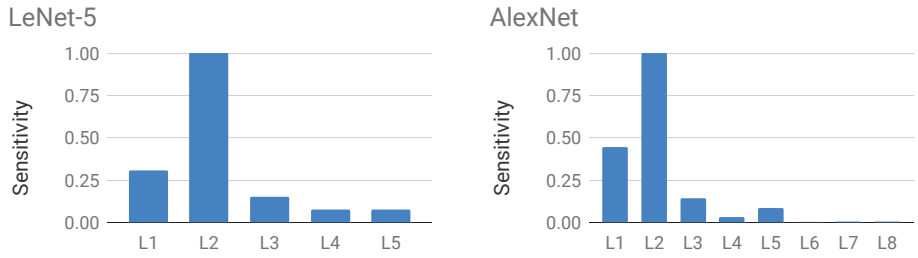


Fig. 9: Normalized sensitivity degree of each layer for LeNet-5 and AlexNet.

notice that, since this is a relatively simple network where accuracy is computed on the classification of 10 different classes of hand-written digits [21], a top5 accuracy would make no sense, and thus a top1 accuracy has been used instead. As in the previous Table II, five different δ values ranging from 0% to 20% are considered. Test accuracy degrades by just 1.7% for a δ of 15% with a corresponding latency reduction close to 50%. If an accuracy degradation of 7% is tolerated, up to 60% of latency reduction can be obtained. Energy saving ranges from 13% to 60% and it is mainly due to communication and main memory energy reduction. Similar savings are observed for AlexNet in Figs. 10c and 10d. With an accuracy degradation of less than 2% up to 58% latency reduction and 54% energy saving can be obtained with a δ of 20%. For VGG-16, Figs. 10e and 10f, it is interesting to notice that almost 50% of latency reduction and 40% energy reduction can be obtained with a negligible impact on the accuracy which is less than 1% for delta values up to 5%. For MobileNet, Inception-v3, and ResNet50, the energy and latency saving as less evident of what found for the rest of network models. The main reason is related to the fact that the analysis is carried out by compressing a single layer of the network. Thus, as MobileNet, Inception-v3, and ResNet50 are very deep, the selected layer for compression does not account for a significant fraction of the total parameters making the compression less effective.

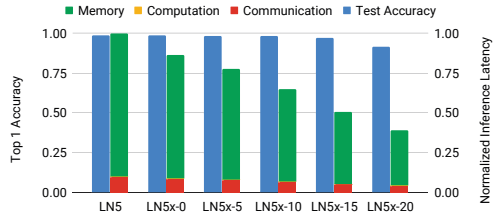
Compressing multiple layers with the appropriate δ value would improve the results. The definition of a technique for the selection of the set of layers to be compressed and the appropriate compression level to be used for each of them with the aim of maximizing the compression ratio under accuracy constraints is left as future work.

D. Applying Compression on Quantized Network Models

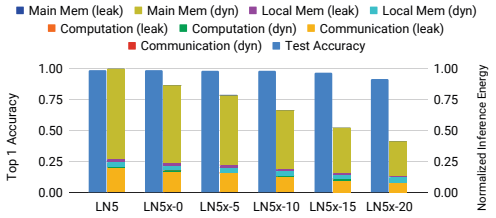
A significant feature of the proposed compression technique is that of being agnostic with respect to the considered network model. Consequently, it can be applied on-top of an already compressed network, allowing a further compression factor. Some of the most used compression techniques are based on *data quantization*, in which shorter fixed-point representation of weights allows to significantly reduce memory footprint and computation resources [22]. To demonstrate the effectiveness of the proposed technique, we will apply it on top of a TensorFlow Lite quantization, introduced by Google as part of the DeepMind project [23]. TensorFlow Lite consists of a set of tools aimed at running low-latency and small-binary size network models on embedded and IoT devices. This tool is well known for achieving good gains in terms of size and performance with minimal impact on accuracy, making it a perfect candidate for applying the proposed compression on top of it.

Starting from a pre-trained model, TensorFlow Lite quantization uses a tool (TensorFlow Lite converter) that loads the HDF5 model of the network *i.e.*, weights and structure, and compressing it according to the chosen optimization model. In particular, TensorFlow Lite supports a reduction of precision of the weights from full floating point to half-precision (16 bits), or even 8-bits integers. In this work, in particular, we will refer to the hybrid 8-bit integer representation, in which floating point values of tensors are approximated according to the formula $real\_value = (int8\_value - zero\_point) \times scale$. This is a quite aggressive setup, useful to stress the applicability of the proposed approach. For further details about the considered quantization scheme, please refer to [24].

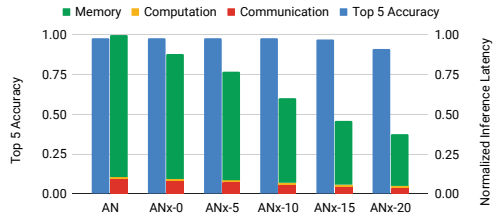
Table III shows the compression efficiency and accuracy obtained when the above quantization technique (QT) is applied, along with the results obtained when the proposed compression technique applied on top the quantized network. The LeNet-5, AlexNet, and VGG-16 networks are analysed to represent small, medium, and large network quantization use cases. Specifically, for each network, the values on the two leftmost columns are referred to the TensorFLow Lite quantized network (QT), while the two rightmost columns (Weighted CR, Top-5 Accuracy) report the same value obtained when applying the proposed compression with different δ values. As can be observed, LeNet-5 accuracy remains almost unaltered when small δ values are applied, degrading when δ values around 20% are considered. Similar results are obtained for AlexNet and VGG-16, where mid-level δ values still introduce compression with a reasonable accuracy loss. With regard to compression ratios, all the three networks, although being already compressed by the quantization process, still show an opportunity for a further compression. In conclusion, these results confirm that the compression operated by the proposed approach acts on a different/orthogonal aspect of the information loss, that is, the monotonic trend of



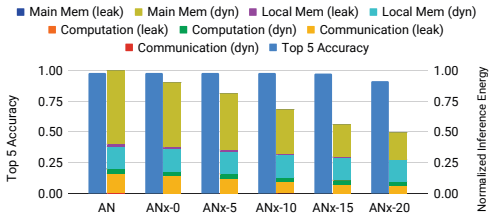
(a) LeNet-5 accuracy *vs.* latency.



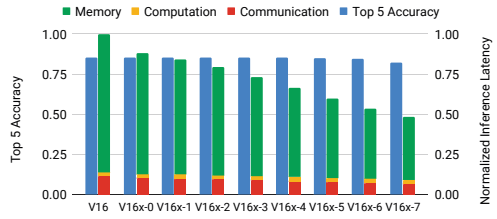
(b) LeNet-5 accuracy *vs.* energy.



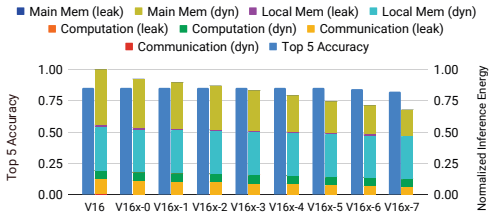
(c) AlexNet accuracy *vs.* latency.



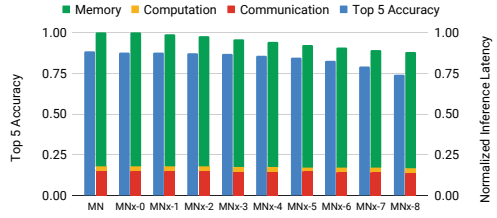
(d) AlexNet accuracy *vs.* energy.



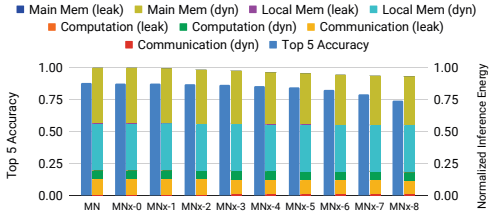
(e) VGG-16 accuracy *vs.* latency.



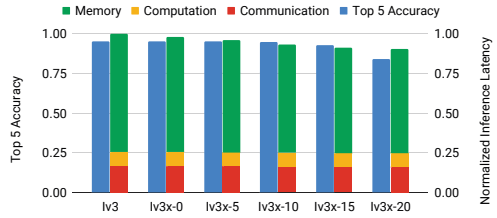
(f) VGG-16 accuracy *vs.* energy.



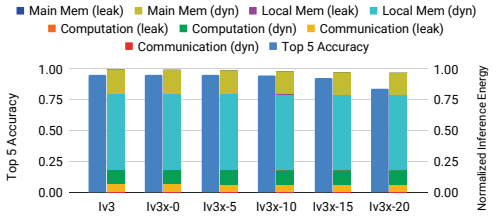
(g) MobileNet accuracy *vs.* latency.



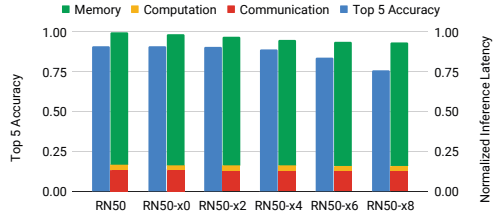
(h) MobileNet accuracy *vs.* energy.



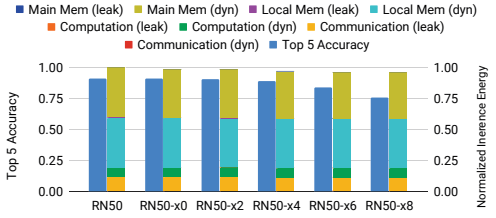
(i) Inception-v3 accuracy *vs.* latency.



(j) Inception-v3 accuracy *vs.* energy.



(k) ResNet50 accuracy *vs.* latency.



(l) ResNet50 accuracy *vs.* energy.

Fig. 10: Accuracy *vs.* inference latency *vs.* inference energy for the original network model and compressed/approximated network models for different δ values.

TABLE III: Compression efficiency for different network models and different tolerance thresholds.

Network Model	Quantization Technique (QT) Weighted CR	Top-5 Accuracy	QT + Proposed Technique δ	Weighted CR	Top-5 Accuracy
LeNet-5	2.41	0.9867	0%	2.62	0.9871
			5%	2.76	0.9864
			10%	3.00	0.9788
			15%	3.31	0.9603
			20%	3.68	0.8747
AlexNet	2.10	0.9794	0%	2.24	0.9794
			5%	2.38	0.9794
			10%	2.66	0.9794
			15%	2.95	0.9735
			20%	3.15	0.9029
VGG-16	2.26	0.8560	0%	1.21	0.8559
			5%	2.35	0.8528
			7%	3.88	0.8327
			8%	5.47	0.7526
			10%	10.27	0.1699

serialized representation of the weights, which is independent from quantization or any other transformation operated on the bits representing each weight.

V. Conclusions

In this work, we introduced a technique for compressing neural network parameters in such a way to reduce the memory and communication traffic. The proposed technique has been applied to several widespread CNN models to investigate the trade-off accuracy *vs.* inference latency *vs.* inference energy consumption. We show that up to 63% inference latency reduction and 67% inference energy reduction can be achieved with less than 5% top 5 accuracy degradation without the need of retraining the network.

In our analysis we have applied the proposed compression technique to a single layer of a CNN (the deepest and that with the highest number of parameters). Compressing more than a single layer with the appropriate δ value would improve the results. Thus, future work will be devoted on defining a technique aimed at selecting the set of layers to be compressed and, for each of them, the appropriate compression level to be used according to the most profitable energy/latency/accuracy trade-off.

VI. Acknowledgement

This work has been supported by the following institutions/grants: (i) the Italian Ministry of Economic Development (MISE) within the research program "UE-PON Imprese e Competitività 2014-2020 Contratto di sviluppo M9 (CDS 000448)" - CUP: C32F18000100008; (ii) the Institute of Advanced Studies of the CY Cergy Paris Université (formerly Université de Cergy-Pontoise) under the Paris Seine Initiative for Excellence ("Investissements d'Avenir" ANR-16-IDEX-0008); (iii) the DIEEI at University of Catania within the research program "Piano per la Ricerca 2016/2018".

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

[1] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Survey*, vol. 51, no. 5, pp. 92:1–92:36, Sep. 2018.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[3] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, June 2019.

[4] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, "A 0.11 pj/op, 0.32-128 tops, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm," in *Symposium on VLSI Circuits*, June 2019, pp. C300–C301.

[5] S. Venkataramani, K. Roy, and A. Raghunathan, "Approximate computing," in *International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems*, Jan 2016, pp. 3–4.

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, 2015, pp. 1135–1143.

[8] K. Ullrich, E. Meeds, and M. Welling, "Soft Weight-Sharing for Neural Network Compression," *arXiv e-prints*, Feb 2017.

[9] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," *CoRR*, vol. abs/1612.01543, 2016.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[11] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[12] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *CoRR*, vol. abs/1404.0736, 2014.

[13] S. Zhai, Y. Cheng, W. Lu, and Z. M. Zhang, "Doubly convolutional neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 1090–1098.

[14] T. S. Cohen and M. Welling, "Group equivariant convolutional networks," *CoRR*, vol. abs/1602.07576, 2016.

[15] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, "Face model compression by distilling knowledge from neurons," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 3560–3566.

[16] A. K. Balan, V. Rathod, K. Murphy, and M. Welling, "Bayesian dark knowledge," *CoRR*, vol. abs/1506.04416, 2015.

[17] G. Ascia, V. Catania, J. Jose, S. Monteleone, M. Palesi, and D. Patti, "Analyzing networks-on-chip based deep neural networks," in *International Symposium on Networks-on-Chip*, oct 2019.

[18] I. NanGate, "NanGate 45nm open cell library," 2008. [Online]. Available: http://www.nangate.com

[19] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Cycle-accurate network on chip simulation with noxim," *ACM Transactions on Modeling and Computer Simulation*, vol. 27, no. 1, pp. 4:1–4:25, Nov. 2016.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, Dec 2007, pp. 3–14.

[21] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, 2014, pp. 269–284.

[23] G. DeepMind, "Tensorflow lite." [Online]. Available: https://www.tensorflow.org/lite/

[24] Google, "Post training quantization." [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization