# Sensitivity Analysis and Compression Opportunities in DNNs Using Weight Sharing

Etienne Dupuis, David Novo, Ian O'Connor, Alberto Bosio

# Sensitivity Analysis and Compression Opportunities in DNNs Using Weight Sharing

Etienne Dupuis[1], David Novo[2], Ian O'Connor[1], and Alberto Bosio[1]
[1]*Ecole Centrale de Lyon, Institut des Nanotechnologies de Lyon, France*
[2]*LIRMM, Université de Montpellier, CNRS, France*
{etienne.dupuis, ian.oconnor, alberto.bosio}@ec-lyon.fr, and david.novo@lirmm.fr

***Abstract*—Deep artificial Neural Networks (DNNs) are currently one of the most intensively and widely used predictive models in the field of machine learning. However, the computational workload involved in DNNs is typically out of reach for low-power embedded devices. The approximate computing paradigm can be exploited to reduce the DNN complexity. It improves performance and energy-efficiency by relaxing the need for fully accurate operations. There are a large number of implementation options leveraging many approximation techniques (e.g., pruning, quantization, weight-sharing, low-rank factorization, knowledge distillation, etc.). However, to the best of our knowledge, a few or no automated approach exists to explore, select and generate the best approximate version of a given DNN according to design objectives. The goal of this paper is to demonstrate that the design space exploration phase can enable significant network compression without noticeable accuracy loss. We demonstrate this via an example based on weight sharing and show that our direct conversion method can obtain a 4.85x compression rate with 0.14% accuracy loss in ResNet18 and 4.91x compression rate with 0.44% accuracy loss in SqueezeNet without involving retraining steps.**

***Index Terms*—Deep Neural Networks, Approximate Computing, Model Compression, Weight Sharing, Design Space Exploration, Embedded System, Hardware Accelerator**

## I. Introduction

Deep Neural Networks (DNNs) are currently one of the most widely used predictive tools in the field of machine learning. They are used today in various applications such as object recognition, drug discovery and natural language processing, as well as safety-critical applications like autonomous driving.

Recent DNNs are articulated around the use of multiple chained convolution layers, called convolutional neural networks (CNNs), using a dot product between input and filter. They also rely on fully-connected layers composing the end of the network and pooling layers to reduce the dimension of activation maps, and thus, the number of parameters to be trained. More recently, Dropout, Batch-normalization were introduced to permit or to accelerate convergence during training. Trained parameters are used by convolutional and fully connected layers (e.g., the 50-layer ResNet [1] network has around 26 million weight parameters), making these kinds of layers the main memory needing and the de-facto main target of most compression methods [2].

Unfortunately, the computational workload and memory needs of CNNs are often out of reach for low-power embedded devices [3]. Novel computing paradigms and emerging technologies are under investigation to make CNNs available for edge computing. Among them, the Approximate Computing (AxC) paradigm leverages the inherent resilience of CNNs to errors to improve energy efficiency, by relaxing the need for fully accurate operations. CNNs have a high degree of redundancy in terms of their architecture and parameters, this redundancy is not necessary for accurate prediction. This observation has paved the way for several highly recognized approximation techniques [3]. The most popular ones include pruning [4], quantization [5], low-rank factorization [6], knowledge distillation [7] and weight-sharing [4]. In this work, we focus on the weight-sharing technique because of its remarkable model compression results.

The weight-sharing technique identifies clusters of values that will be shared between weights (see Section II for details). In the literature, there are many practical applications of weight-sharing. Existing works can be divided into two groups. The first one involves training steps after or during the weights-sharing, while the second group targets weight-sharing without retraining. Hashed net [8] proposes to randomly group weights into buckets sharing the same value before the training step. A more efficient method is presented in DeepCompression [4], it achieves impressive results due to the use of various levels of redundancy in deep neural networks through pruning, clustering, and Huffman coding. Deep $K$-means [9] uses regularization terms to encourage weights to concentrate at retraining time and proposes interesting data re-shaping techniques optimized for higher data reuse with row-stationary dataflow [10] at inference time. Soft weight-sharing [11] uses a simple (re)training step with a regularizer to efficiently compress the network using pruning and weight-sharing. It is also possible to reduce complexity by encoding both weights and features map, then compute multiplications using a lookup table. Both LookNN [12] and Quantized CNNs [13] are based on this concept and achieve good results at inference time, particularly if energy efficiency is taken into account.

Although retraining the model after compression allows accuracy loss recovery, there are many reasons to develop design methods that can largely benefit from clustering without requiring any further retraining. For example, retraining may requires access to the full training data that cannot always possible due to the size of the entire dataset, but also for legal and privacy compliance reasons. Furthermore, it is a long and computation-intensive step, and thus, it is costly and can delay the CNN time-to-market.

This is where conversion algorithms offer an alternative by working directly on trained models. An example of this approach can be found in vector quantization [14], where a clustering approach based on $K$-means is applied to fully connected layers of a trained network.

Despite the promising results, all existing approaches suffer from important limitations: requiring (re)training [8] [4] [9] [11] [12], [13], or target only fully-connected layers network [14]. Our goal is to compress a deep neural network targeting embedded device constraints. We choose to address this problem using a systematic exploration to convert a trained model, without requiring retraining, and targeting both fully-connected and convolutional layers to leverage redundancies where they exist.

In the proposed method, model weights are clustered using the $K$-means algorithm with a layer-specific number of clusters. To achieve compression of fully-connected and convolutional layers of a CNN, we perform a layer-specific sensitivity analysis of the trained network. Then, a hierarchical exploration of the design space is exploited to get the final result. The main contributions of this paper can be summarized as follows:

1) We propose a method to analyze the sensitivity of network layers and identifies the *good* range of clusters;
2) We propose a design space exploration framework guided by the sensitivity analysis to efficiently compress the overall CNN.

The rest of the paper is organized as follows: Section II introduces the concepts and advantages of the weight-sharing method. Section III describes the proposed method for the layer sensitivity analysis and the design space exploration, while section IV presents the results obtained. Finally, section V concludes the paper.

## II. Weight-Sharing Approximation

As seen in the previous section, deep neural network approximation can be achieved using different approaches. Among them, the weight-sharing technique identifies clusters of weights that will share a common value. Usually, clusters are identified by using the $K$-means [15] algorithm. According to the literature, the $K$-means based weight-sharing method achieved outstanding performance compared to other methods by using a scalar pool of values instead of data-representation based values [9].

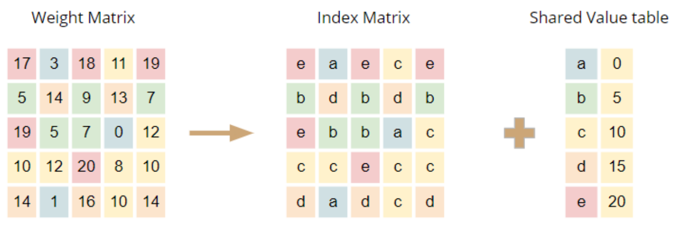

Fig. II shows an example to clarify the benefits of weight-sharing. The first matrix corresponds to a $5 \times 5$ convolutional kernel (filter) whose values were computed during training. The matrix contains $N = 25$ values in the range from 0 to 20, which can be represented using 5bits, resulting in a total $Size = N * B = 25 * 5 = 125bits$. In this example we identified 5 clusters, namely ‘a’, ‘b’, ‘c’, ‘d’ and ‘e’, replacing the 25 original values as shown in the second matrix.

Instead of storing each weight value, only the indexes of the corresponding shared values are required and thus stored in the memory. Accordingly, the size of the weight matrix can be reduced from $B$ bits for each value to $log_2(K)$ with $K$ being the number of different shared values. The size of the network then becomes $N * log_2(K) + K * B$ instead of $N * B$. The main objective of $K$-means clustering is to reduce $K$ while still having a good representation of the original data.

Coming back to the example of Fig. II, instead of storing 25 values, we store only the indexes of the clusters. 5 clusters indexes can be represented using 3bits. The index is finally used to access shared values in the table (as shown in the figure) to retrieve the weight value. Accordingly, with 5 clusters ($K = 5$), we obtain $IndexMatrixSize = N * log_2(K) = 75bits$ and $ValueTableSize = K * B = 25bits$ resulting in a total $Size = 100bits$. We can thus save 25 bits with respect to the original kernel matrix, achieving a $1.25\times$ compression ratio (CR).

However, choosing the scope of the weight-sharing method is not obvious. To compress a network, one could try to find shared values in the whole network or different shared values for each layer or even reduce further the scope to the channel or kernel level. In our previous work [16], we found that working at the level of the layer is enough to achieve a good compression ratio without inducing substantial accuracy loss. Accordingly, in this work, we apply weight-sharing at the layer level.

## III. Proposed Method

This section presents the proposed method for approximating models using weight-sharing. We need to find out the suitable number of clusters $K$ to represent the convolution and fully connected layers weights. Indeed, minimizing $K$ maximizes the compression rate and maximizing $K$ minimizes induced accuracy loss. Solving this trade-off is not a simple task, it involves a complete design space exploration and computation of the accuracy loss by scoring the network (i.e., measuring accuracy of the network using testing data set) many times. In this work, we propose a hierarchical method aiming to reduce the complexity of such exploration. The proposed method is based on a layer sensitivity analysis to evaluate the weight-sharing opportunity for each layer. The next subsections detail the main steps.

### A. Layer Sensitivity Analysis

Each layer of a deep neural network has a specific role during the inference (i.e., when the trained CNN is used to infer/predict the input samples), it is reasonable to assume that different layers may have a different sensitivity w.r.t weight-sharing. Some layers can be drastically approximated with a very small number of clusters without noticeable accuracy loss, while others are more sensitive and require a larger number of clusters. Identifying and quantifying the sensitivity of a particular layer can be achieved by measuring the impact of clustering that layer on the whole network accuracy. For each layer, we are varying the number of clusters and then we study the variation of the accuracy loss over the compression ratio.

Finding a suitable number of clusters $K$ for each layer can be done by following the procedure illustrated in Algorithm 1. For now, we are only using the resulting accuracy loss to prune the solution space. The appropriate level of clustering for each layer is then the one that has the smallest impact on the network accuracy loss.

**Algorithm 1:** Layer sensitivity identification algorithm featuring a layer-wise design space exploration

```
Input: $network$(Trained Network), $minCluster$ (integer), $maxCluster$ (integer)
Output: $layerK$ (number of cluster for each layer)
$layerK \leftarrow list()$;
for $layer$ in $network$ do
    $bestAccLoss \leftarrow None$;
    for $K \leftarrow minCluster$ to $maxCluster$ do
        $tempNet \leftarrow$K$-means(layer, K)$;
        $accLoss \leftarrow score(tempNet, valDataset)$;
        if $accLoss < bestAccLoss$ then
            $bestAccLoss \leftarrow accLoss$;
            $bestK \leftarrow K$;
        end
    end
    $layerK[layer] \leftarrow bestK$;
end
```

This algorithm involves a lot of scoring steps (i.e., network accuracy evaluation). This could be avoided by stopping the analysis after reaching a user-defined accuracy loss threshold. Defining a range of clusters has the benefit of avoiding falling in local minima. At the cost of having to carefully choose the cluster number range to obtain satisfying results.

The result of the sensitivity analysis is the range of clusters number that can be used for each layer. That range will further be exploited by the proposed design space exploration as detailed in the next subsection.

### *B. Design Space Exploration*

Fig. 1 sketches the overall flow of the proposed design space exploration framework. First of all, we start from a trained network (i.e., the baseline model in the figure). We use trained models from the open-source framework ONNX [17] model zoo as input. Allowing our framework to be fully compatible with any model that can be converted in ONNX format. And so, any available framework can be used for training. We are using MXNET [18] for network scoring. Within the framework, all the layers of the network are processed. The

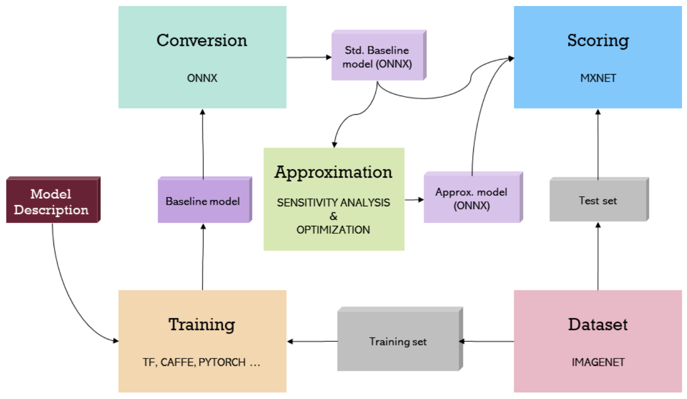

Fig. 1. Overall flow of the proposed method

result is an approximate network characterized by its accuracy loss and its compression rate as compared to the baseline.

The developed framework applies the clustering layer by layer, allowing to solve trade-offs locally, and thus, reduce the design space complexity. We call this method hierarchical exploration because in our study layers are arranged by rank, i.e., meaning the order in the network from input to output. We used a greedy algorithm to choose the best number of clusters based on the accuracy loss.

For each solution, we thus need to score the approximate network on the testing set to quantify the accuracy loss. We then use a greedy approach to choose the best number of clusters for each layer, aimed at minimizing the accuracy loss. The identification of the best number of clusters is done accordingly to the sensitivity analysis step outcomes. This will reduce the overall number of evaluation since the number of clusters have been already identified and here refined when taking account the overall CNN. Each identified solution (i.e., each compressed CNN) is finally evaluated by computing the compression ratio and by the accuracy loss induced by the clustering w.r.t. the baseline model.

## IV. Experiments

In this section, we will first describe the framework and the experimental setup used to evaluate our compression method on existing CNNs and the results we obtained.

### *A. Experimental Setup*

The experiments target large image classifier CNNs trained and tested using the LSVRC2012 [19], a popular dataset having a testing set of 50k images (50 images per class). The chosen CNNs are ResNet [1] for its near state-of-the-art classification accuracy and SqueezeNet [20] for its implementation-oriented approach that includes a relatively small number of parameters. We took the trained model from the ONNX model zoo and fed directly to our weigh-clustering framework as input. The following subsection describes results on ResNet and Squeezeneet. Both are designed for the ImageNet dataset and resize the image to 224x224 and have a 1000 class output. ResNet-18 reaches a top-1 accuracy of nearly 69.69%

while SqueezeNet reaches 55.99%. We are working on a full precision model (i.e., network weights are stored using 32 bit floating point representation) without using reduced quantization or fixed-point representation.

All the experiments were executed on a server equipped with a single NVIDIA TESLA V100 GPU and the latest CUDA version of MXNET [18] for the network scoring task. We are using the latest version of ResNet18 and SqueezeNetV1.1 from one model zoo as an input network. Scoring one network on the 50 000 pictures Imagenet testing data set take respectively 80s and 50s. From this data, it is clear that the scoring of the model to compute accuracy is an intensive task and it corresponds to the bottleneck of the proposed framework. This is why the sensitivity analysis plays an important role because it already provides a reduced set of clusters to be further refined (i.e., modified). In this sense, we can save execution time because we can reduce the overall number of scoring tasks.

The following subsection presents the results obtained using the proposed approach on both ResNet and SqueezeNet.

## B. Identifying a Layer Sensitivity

For a given input network we are looking to measure the resilience to the approximation of each layer. To achieve this task, we are monitoring the accuracy loss and the compression rate induced by weight sharing when the number of clusters varies.

Compression rate (CR) is calculated for each layer using the following formula:

$$\begin{aligned} CR &= \frac{baselineModelMemory}{approximatedModelMemory} \\ &= \frac{W * B}{W * ceil(log_2(K)) + K * B} \end{aligned} \tag{1}$$

Having $W$ the number of weights of the layer, $K$ the number of clusters of the approximated layer and $B$ the number of bits used to represent a single value. We have a direct impact of $K$ variation on the compression rate.

Fig 2 shows the accuracy loss related to the compression rate for each cluster number ($K$). We have seen previously that we have a direct impact of cluster number on compression rate but this is not true for the accuracy loss, which largely depends on the clustering results representing data. For a different number of clusters, the distance between clusters centroid and original data representation varies drastically. For the example the figure, we are varying the number of clusters used in the $K$-means algorithm from 20 to 60 with a step of 2. The gap in compression rate value between the two groups of points is explained by $K$ crossing a threshold allowing the index value ($ceil(log_2(K))$) to be represented with one bit less. For example, moving from 256 to 255 different clusters allow lowering the index bit from 9 to 8 bit, which has a large impact on the compression rate, because this index bit count is multiplied by the number of weight of the layer. We can observe some clusterized version of the layer that has better

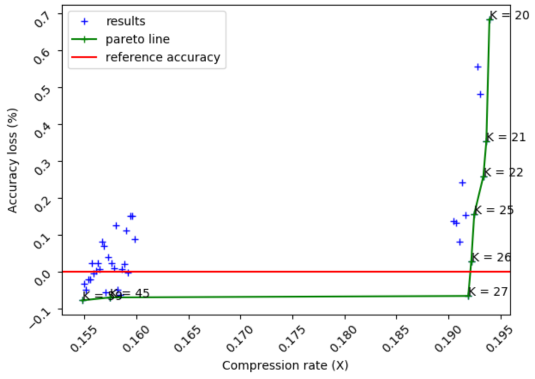

Fig. 2. Sensitivity Analysis: Accuracy loss and compression rate of different version of the Layer 28 of SqueezeNet, the range of value for K is from 20 to 60

accuracy than the baseline layer (negative accuracy loss, like $K = 45$). These unexpected improvements of the accuracy might be due to the fact that training step fall into a local-minima.

The Pareto front represents optimal points. From this set of pareto optimal $K$, applying the Greedy discrimination algorithm results in selecting the one with the lowest accuracy loss. In the example of Fig. 2, the best solution is the one with 48 clusters, which also lead to achieve a better accuracy compared to the baseline model.

Fig. 3 shows the number of clusters achieving the best accuracy for each layer of the network and the corresponding compression rate of the layer. Layers are ordered by size (number of weight). We can see 3 layers with higher compression opportunity (more resilience to high clustering) for ResNet and 4 for SqueezeNet. From the results depicted in Fig. 3, we set the range of the number of clusters from 20 to 60 (corresponding to the minim and maximum plotted in the histograms). This range is used as input for our hierarchical design space exploration framework.

## C. Hierarchical Design Space Exploration

The results of the proposed framework are presented in Table I. Each row of the table corresponds to an experiment. For each experiment, we detail the Network topology, the Cluster Range and the type of experiment (column “Keep layer Bool”). Two types of experiments were carried out:

1) The results of the sensitivity analysis are not considered during the exploration (ID 1 to 4);
2) The results of the sensitivity analysis are considered during the exploration (ID 5 to 8).

For each experiment, we considered the Accuracy loss of the AxC model w.r.t. the top-1 of the baseline model. Finally, we reported the memory footprint in MegaBytes for the Baseline and AxC model. The Compression Ration (CR) is the last column The Cluster Range set 20 to 60 is determined from the sensitivity analysis. The second one 40 to 80 is used to

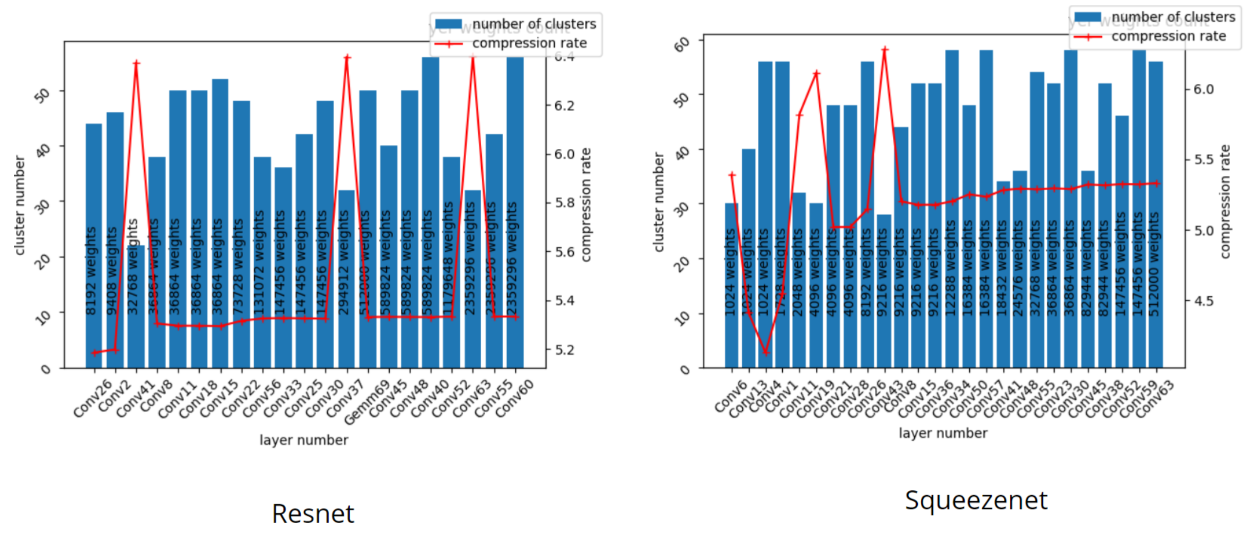

Fig. 3. Network sensitivity to clustering, layer-wise analysis on ResNet and SqueezeNet, layer are ordered by number of weights

variate the results of the sensitivity analysis and explore other solutions.

The results have shown that better results in terms of memory compression are obtained by using the range 20 to 60 thus confirming the efficiency of the sensitivity analysis. For these cases, higher CR leads to lower accuracy loss that was expected.

Another interesting result is the comparison between the type of experiments confirming once again the importance of the sensitivity analysis. For ID 5 we got 5.80% against ID 1 of 5.71%. However, CR is not the only important point. Even if we got a higher CR, the accuracy loss is lower (thus better). Again for ID 4, we got an accuracy loss of 0.54% while for ID 1 we got 0.73%.

To conclude, the best results either in terms of CR or accuracy loss are obtained when cluster range came from results of sensitivity analysis corresponding to IDs 1, 3, 5 and 7. Concerning the type of experiments, we got for one case (ID 5) the best result compared to ID 1. On the other hand, for experiment ID 7, obtained accuracy and compression rate are slightly lower than ID 3 meaning that keeping layer weights clustered during the next layers analysis is efficient in this case. We plan in our future works to check other directions for the design space exploration. More details are given in the next section.

Analyzing the resulting approximated networks we discover that storing values represent less than 1% of the memory occupation after clustering. As shown in Fig 4, the value table (i.e., weight values stored in the look-up table) is negligible compared to the index table representing the memory of each layer for a SqueezeNet after optimization using our hierarchical exploration method.

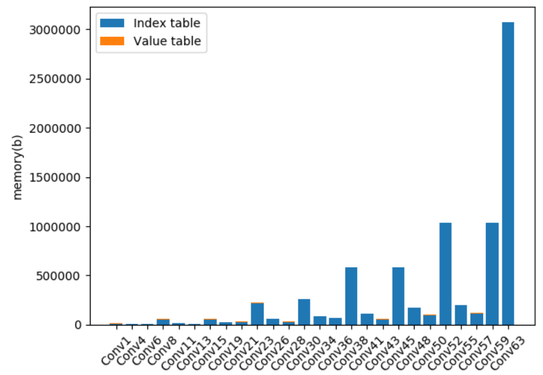

Fig. 4. Layer compression analysis for SqueezeNet

## V. Conclusion and Future Work

This paper presented our work towards a design space exploration method for compressing trained deep neural networks. We have shown that our method can sucessfully compress a deep neural network with very small accuracy loss. Reaching around 5x compression with less than 1% top-1 accuracy loss. The most important advantage is the fact that our approach is a one-shot conversion, and thus, we can avoid the prohibitive cost and constraints tied to network retraining.

We intend to continue our work on approximating deep neural networks by first exploring other design space parameters for the weight sharing compression. More in detail, our approximation method includes several parameters that can be explored to produce different solutions:

1) **The order of exploration:** Layers can be treated in different orders, the more intuitive method is to process layers in a topological order (i.e., from the input to the output). Alternatively, they can also be sorted by ascending or descending weights count, aiming to address first the layer with the highest compression opportunity and then recover accuracy loss with the others.
2) **The candidate number of clusters:** Defined as a range

TABLE I
COMPRESSING RESNET AND SQUEEZENET ON IMAGENET

| ID | Network | #Cluster Range [min, max, step] | Keep layer Bool | Baseline Model top-1 (%) | AxC Model top-1 (%) | Accuracy loss top-1 (%) | Baseline Model memory (Mb) | AxC Model memory (Mb) | CR |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ResNet18 | [20, 60, 1] | True | 69.69 | 68.95 | 0.73 | 356.4 | 62.4 | 5.71 |
| 2 | ResNet18 | [40, 80, 1] | True | 69.69 | 69.55 | 0.14 | 356.4 | 73.5 | 4.85 |
| 3 | SqueezeNet | [20, 60, 1] | True | 55.99 | 54.69 | 1.30 | 37.6 | 6.9 | 5.44 |
| 4 | SqueezeNet | [40, 80, 1] | True | 55.99 | 55.52 | 0.44 | 37.6 | 7.7 | 4.91 |
| 5 | ResNet18 | [20, 60, 1] | False | 69.69 | 69.15 | 0.54 | 356.4 | 61.5 | 5.80 |
| 6 | ResNet18 | [40, 80, 1] | False | 69.69 | 69.13 | 0.56 | 356.4 | 69.8 | 5.11 |
| 7 | SqueezeNet | [20, 60, 1] | False | 55.99 | 53.58 | 2.41 | 37.6 | 7.1 | 5.30 |
| 8 | SqueezeNet | [40, 80, 1] | False | 55.99 | 54.74 | 1.25 | 37.6 | 8.0 | 4.70 |

of values, described by min, max, and step. It can be either the same for all layers of a network or specific and depends on layer size. It can even be a logarithmic scale, considering that for a sufficiently high number of clusters, a slight change in cluster count has a negligible impact on the accuracy loss of the network.

We then address performance and hardware metrics. By using a different algorithm for choosing parameters, because we rely on a greedy algorithm taking only the accuracy loss into account. Target metrics are firstly deep neural network-specific metrics like accuracy loss and compression rate, secondly performance metrics like throughput rate and latency and thirdly hardware-oriented metrics like energy consumption and surface used in case of FPGA implementation. We also intend to add parameters like $K$-means initialization and layer-specific cluster range. Allowing the design space to be enlarged and explore potentially better parameter combinations.

Finally, we will extend the framework to support different approximation strategies such as pruning, quantization, low-rank factorization, knowledge distillation, etc.. As the use of a combination of orthogonal approximation methods allows more levels of redundancy inherent to neural networks to be leveraged.

## VI. ACKNOWLEDGEMENT

This work has been funded by the French National Research Agency (ANR) through the AdequatedDL research project (ANR-18-CE23-0012).

## REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[2] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, p. 292, 03 2019.

[3] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, Dec 2017.

[4] W. J. D. Song Han, Huizi Mao, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv*, 2016.

[5] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson, "UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks," *arXiv*, Apr. 2018.

[6] A. Acharya, R. Goel, A. Metallinou, and I. Dhillon, "Online Embedding Compression for Text Classification using Low Rank Matrix Factorization," *arXiv*, Nov. 2018.

[7] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv*, Mar. 2015.

[8] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *CoRR*, vol. abs/1504.04788, 2015.

[9] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, "Deep k-Means: Re-Training and Parameter Sharing with Harder Cluster Assignments for Compressing Deep Convolutions," *arXiv*, June 2018.

[10] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, June 2016.

[11] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *ArXiv*, vol. abs/1702.04008, 2017.

[12] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural Network with No Multiplication," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 1779–1784, 2017.

[13] Y. W. Q. H. Jiaxiang Wu, Cong Leng and J. Cheng, "Quantized convolutional neural networks for mobile devices," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv*, Dec. 2014.

[15] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *JSTOR: Applied Statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[16] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the automatic exploration of weight sharing for deep neural network compression," *to appear in Proceedings of DATE2020*, 2020.

[17] J. Bai, F. Lu, K. Zhang, *et al.*, "Onnx: Open neural network exchange." https://github.com/onnx/onnx, 2019.

[18] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[20] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$1mb model size," *CoRR*, vol. abs/1602.07360, 2016.