Clustered Network Adaptation Methodology for the Resource Constrained Platform

Pratibha Verma[*], Pradip Sasmal[!], Chandrajit Pal[#], Sumohana Channappayya[##], Amit Acharyya[**]
[ee21resch01007[*],ee21pdf12[!]]@iith.ac.in, palchandrajit[#]@gmail.com, [sumohana[##], amit_acharyya[**]]@ee.iith.ac.in
Department of Electrical Engineering
Indian Institute of Technology, Hyderabad, India

***Abstract*—Recently, deep neural networks (DNNs) are widely used for many artificial intelligence (AI) applications, including robotics, self driving vehicles etc. Although DNNs deliver state-of-the-art accuracy on many AI tasks, but are both computationally and memory intensive. This hinders the deployment of DNNs on mobile and IoT edge devices with limited hardware resources and power budgets. Traditional DNN compression is applicable during training to obtain an efficient inference engine. When the inference engine runs on the hardware platform, constrained by battery backup it brings additional challenge in terms of reducing the complexity (like memory requirement, area on hardware etc.). To reduce the memory complexity, we are proposing a new low complex methodology named as "Clustering Algorithm" to eliminate the redundancies present within the filter coefficients (i.e. weights). This algorithm is a three stage pipeline: quantization, coefficient clustering and code-assignment, that work together to reduce the memory storage of neural networks on the cost of low run-time memory requirement. To show the efficacy of the proposed "Clustering Algorithm", we executed it on the network obtained after applying NetAdapt Algorithm, a popular inference engine on pre-trained AlexNet for CIFAR-10 dataset. We observe that our "Clustering Algorithm" reduces the storage required by five layers of already pruned pre-trained AlexNet by 3x (approximately 65%) on FPGA, and 2x (approximately 51%) on CPU. This allows the model to fit into on-chip SRAM cache rather than off-chip DRAM memory.**

***Index Terms*—Deep Learning; Model Compression; Memory Complexity; Convolutional Neural Network; Clustering**

I. Introduction

The deep neural network (DNN) techniques was primarily evolved for computer vision tasks [1], [2] but, have revolutionized a variety of fields like artificial intelligence [3], speech recognition [4] and language processing [5]. The DNNs are very powerful and uses multiple layers and large number of weights to extract the high-level features to classify complex problems. But the large number of weights consumes considerable storage and memory bandwidth. For example, the model sizes of AlexNet trained on CIFAR-10 and VGG-16 trained on ImageNet are 190MB and 500MB [6], respectively. As the network is getting deeper in pursuit of higher accuracy, the number of layers are increasing, therefore number of weights increases, which makes difficult to deploy DNNs on resource constrained mobile and edge platform [7]. However, the deployment of DNNs on such platforms has many advantages such as better privacy, less run-time execution, small network bandwidth etc., but the large storage requirement prevents the neural network to commercially incorporate into mobile apps. Another challenge is to reduce energy consumption. For large neural network implementation, considerable amount of energy is required for both fetching weights from memory and computation for dot products. Since mobile devices are battery operated, so such type of power hungry applications are not recommended to deploy on the fore-mentioned resource constrained platforms. Energy consumption is mostly dominated by memory access. For large networks, the costly DRAM access is required because DNNs do not fit in on-chip storage. For these large networks, the final goal is to reduce the required storage and energy consumption to run the inference, so that DNNs can be deployed on such devices. To serve this purpose many inference engines have been developed like NetAdapt [8], HAT [9] etc. To meet the target hardware resources, NetAdapt and HAT both take direct metrics like latency, power consumption in their optimization loop. This allows these inference engines to achieve reduced memory storage which makes DNNs small enough to implement them on resource constrained platform.

Since the hardware resources are changing dynamically, the question remains in the possibility to reduce the storage memory further on top of already optimized inference engine while preserving the original accuracy. Motivated by this, we propose a novel "Clustering Algorithm". For the inference engine's output network, our algorithm which works in three stages (as shown in Fig.1) (see Section III for explanation), is able to exploit the redundancy present in filter coefficients. Without any significant change in weight values, this algorithm reduces the overall memory size without loss of accuracy.

The main contribution of this paper is to meet the dynamically changing storage requirement on a resource constrained platform, by exploiting the redundancy present in the filter coefficients. We propose a loss-less (in terms of accuracy) and low computationally complex methodology termed as "Clustering Algorithm" on the network obtained after applying inference engine for a pre-trained network in order to reduce memory usage.

To demonstrate the efficacy of the proposed "Clustering Algorithm", we implemented it on the output of NetAdapt for pre-trained AlexNet with CIFAR-10 data-set. We show 51% to 65% network storage reduction in comparison to standalone NetAdapt algorithm.

**Corresponding author

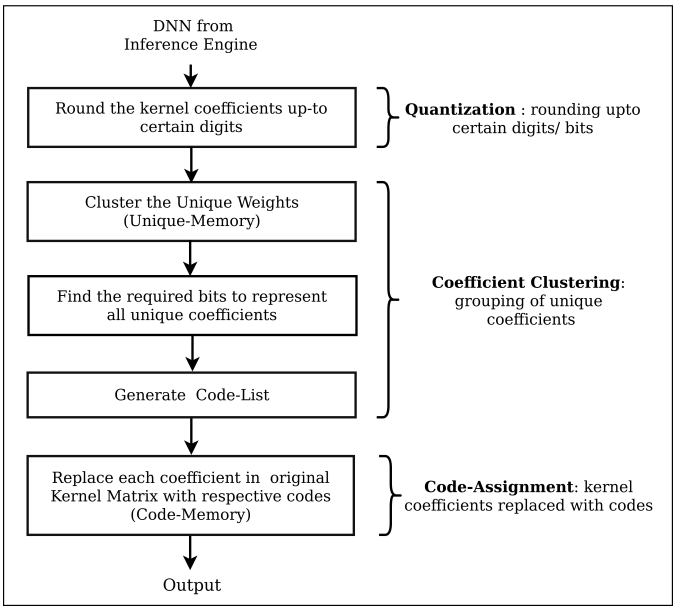


Fig. 1: The flow-diagram of proposed "Clustering Algorithm". This algorithm consists of three stage pipeline: the quantization , coefficient clustering and the code-assignment.

II. Related Work

Various studies are present on simplification of deep neural networks for real time execution on hardware platforms. Typically, neural networks are over-parameterized, so there is significant redundancy present in DNN models [10]. This leads to a wastage of both computation and memory usage. For redundancy removal, in [11], a fixed point implementation activations with 8-bit integer (vs 32-bit floating point) is proposed. The authors in [12] introduced a quantization of neural network by using L_2 error minimization technique and showed better accuracy on CIFAR-10 and MNIST data-sets. HashedNets introduced model size reduction methodology by using hash function which randomly groups connection weights [13]. Those connections which are in same hash bucket, shares a single parameter value. Here weights are not learned through training, they are pre-determined by the hash function. This does not capture the nature of images. In [14], the authors discussed the compressed deep convnets which uses vector quantization. This resulted the loss of 1% accuracy. Both of these methods focused on fully connected layer only and ignored the convolutional layers.

For the comprehensive survey readers can refer to [15]. For the reduction of network complexity and over-fitting, the network pruning method has been used. The most simplification related works are pruning based methods [16]–[18], which aim to remove individual weights from deep neural networks. Instead of individual weights, the entire filter can also be removed [19], [20]. But the disadvantage of these methods is to manually choosing the compression rate for every layer. Sparsifying regularizers are used in MorphNet [21] to determine the layer-wise compression rate automatically. In ADC [22], reinforcement learning is used to learn a policy for choosing the compression rates. But all the above methods are guided by only indirect metrics and thus may lead to sub-optimal performance.

The energy-aware pruning [23] methodology incorporates the estimated energy numbers into pruning algorithm and uses an energy model [24]. This requires the estimation of direct metrics for every target platform, which needs detailed platform knowledge including its hardware architecture [25] along with the network to array mapping which is used in the tool-chain. The other approach to simplify the DNNs is directly designing the efficient network architectures. It is observed that traditional compression algorithms like [6], majorly applicable during training phase in order to obtain an efficient inference engine. But, when there is a question of application of DNN on resource constrained platform, further optimization is required in terms of complexity and memory storage. This raises the need of hardware-software holistic approach.

The recently proposed inference engine, NetAdapt [8] algorithm incorporates the direct metrics into its optimization loop. It takes one input from user (pre-trained DNN) and second input from hardware (i.e. direct metric) and then it optimizes the DNN according to available hardware budget so that the size of incoming DNN can fit for existing hardware resources to give maximum accuracy. In a similar way, HAT [9] also includes target hardware parameters in its optimization loop to achieve better accuracy.

III. Methodology

The proposed methodology comprises of further optimization of inference engine's output network. We proposed a "Clustering Algorithm", which will allow user to eliminate the redundancies present in filter weights without hampering network accuracy. The three stages of the proposed "Clustering Algorithm" are explained below.

A. Quantization

The quantization process is an approach to eliminate the redundancy within the kernel (filter) coefficients. In this first stage, we are converting the kernel matrices to their linear forms and the floating point values are rounded up-to certain digits. Note that, since we are not using approximation of the filter coefficients to the nearest integer values, the accuracy of the network will remain unaltered.

B. Coefficient Clustering

The second stage is clustering of weight values. The whole process is performed in three steps. The step-1 is to create the "Unique Memory" by grouping of similar numbers obtained after quantization. Basically, the weights which contain same information are identified and are stored in a memory. Here, weight values are stored in ascending order. Next, in the step-2, total number of bits are computed to represent all unique weights, which depends on the number of unique weights. If N is number of unique weights, then we require $n = \frac{\log N}{\log 2}$

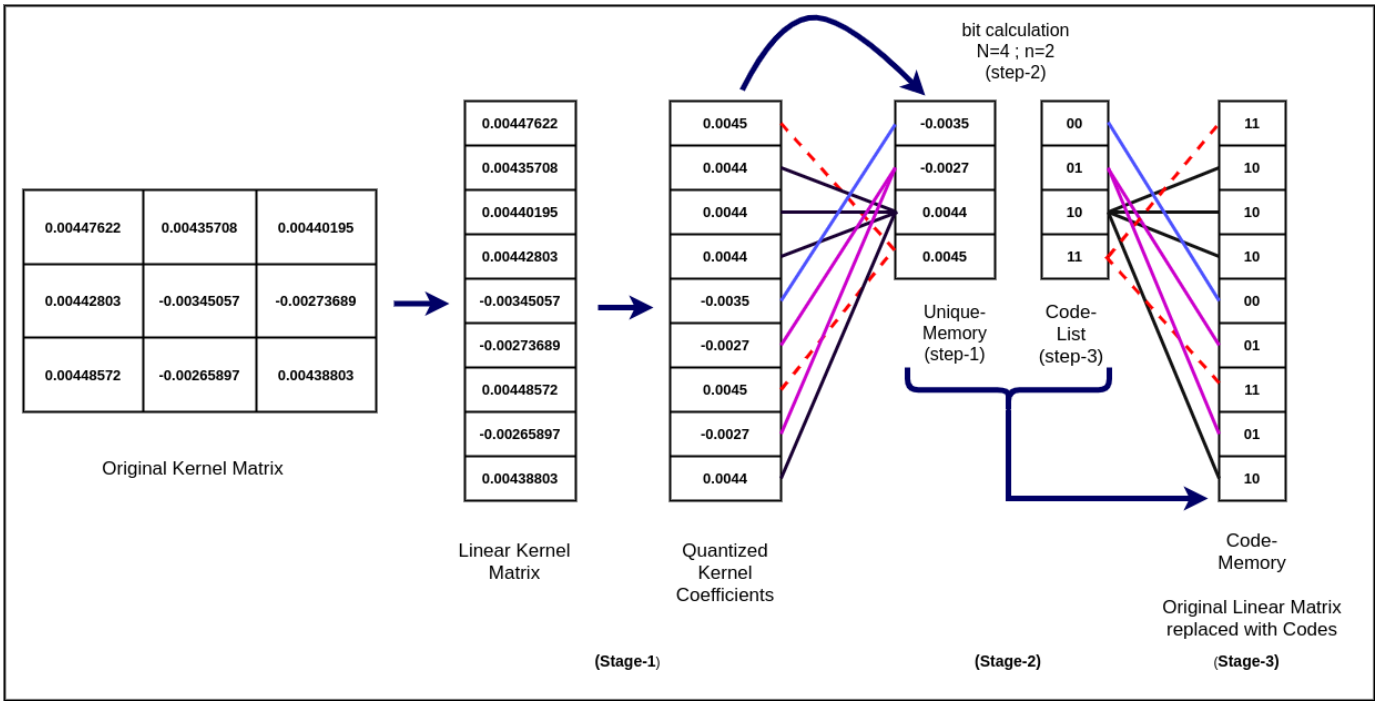


Fig. 2: The working flow of proposed "Clustering Algorithm". This shows all three pipeline stages works together to reduce the storage requirement of neural networks.

bits to represent them. Finally, in the step-3, the specific code is allocated to each unique values. These are unsigned codes and are generated in equal number as the number of unique weights. This list of codes is named as "Code-List".

C. Code-Assignment

The codes in "Code-List" will be treated as an address for unique value at the same location in "Unique-Memory" i.e., each unique value is assigned with specific code. This will be treated as dictionary for future reconstruction. Now these address values are replaced with the unique values in original kernel and this is named as "Code-Memory". This will preserve the filter shape for reconstruction. As a result, all the values in original kernel can be represented via less number of bits compared to 32 bits. Finally, the memory storage is tested on FPGA and CPU to verify the reduced memory requirement.

The complete working of quantization, coefficient clustering and code-assignment is explained in Fig. 2.

IV. Results and Discussion

To demonstrate the efficacy of our proposed "Clustering Algorithm" in reducing the memory requirement, we tested it on the network obtained after applying NetAdapt on pre-trained AlexNet. CIFAR-10 dataset is used for training purpose. As already mentioned that the NetAdapt is an inference engine which reduces the size of AlexNet to deploy it on resource constrained platform with equivalent accuracy compared with the vanilla counterpart. On top of that we applied our "Clustering Algorithm" with its three stages to remove the redundancies present in kernel weights to achieve better memory reduction, for the five convolutional layer in AlexNet.

We have quantized the filter coefficients up to four digits. To verify the memory usage on FPGA, we converted the original filter coefficients into integer values by multiplying each coefficient with 10^4. And then converted the integer values into binary format, since our "Code-List" and "Code-Memory" are in binary form. Similarly, after creating the "Unique-Memory" all the signed numbers are converted into binary numbers.

TABLE I: Code-length to represent unique values

Layer in AlexNet	Number of Unique Values (N)	Code length for Code-Memory (Number of bits (n))
conv-1	1729	11
conv-2	1806	11
conv-3	1267	11
conv-4	1070	11
conv-5	885	10

Next, we verified the memory requirement of each file in terms of Block RAMs (BRAMs) and the comparison result is shown in Fig. 3a. This figure shows the "Clustering Algorithm" verification on FPGA on top of NetAdapt Algorithm.

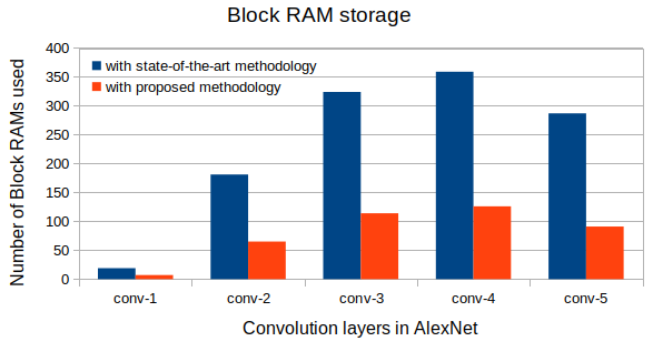
Table. I shows the number of unique values for each convolutional layers of AlexNet and the code-length of each

TABLE II: The compression Statistics for AlexNet with NetAdapt and "Clustering Algorithm".

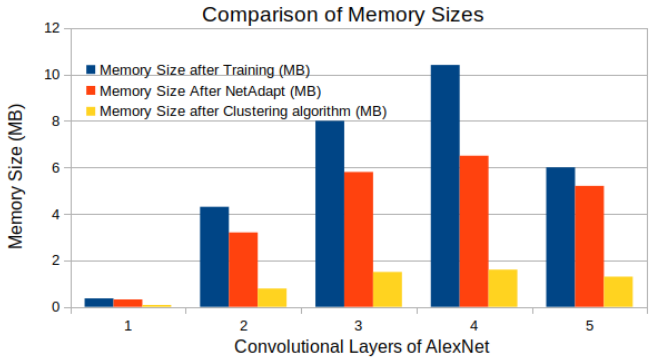
Convolutional Layers of AlexNet	Memory size after training (MB)(P)	Memory size after NetAdapt Algorithm (MB)(Q)	% reduction (P+Q)	Memory size after Clustering Algorithm (MB)(R)	% reduction (P+Q+R)
conv-1	0.3644	0.3187	12.54%	0.1987	37.65%
conv-2	4.3	3.2	25.58 %	1.641	48.7%
conv-3	8.0	5.8	27.5%	2.832	51.17%
conv-4	10.4	6.5	37.5 %	3.129	51.17%
conv-5	6	5.2	13.33%	2.425	53.37%
Total	29.064	21.019	27.68%	10.225	51.35%

code i.e. number of bits required to represent that unique code. We can observe that the number of bits decreasing from 32 to maximum 11 value, which is almost the half value.

That is why we are getting significant reduction in storage memory. The reconstruction of weight matrix for another applications is possible with the help of "Code-List", "Unique-Memory and "Code-Memory". And it is verified that accuracy is not diminishing after reconstruction of matrix thus we like to coin this as a loss-less compression in terms of accuracy.



(a)



(b)

Fig. 3: The implementation results of "Clustering Algorithm" on FPGA and CPU. (a) The "Clustering Algorithm" verification on FPGA (b) The Comparison of reduction in AlexNet storage memory.

The reduction is 3x i.e. from 1170 MB (obtained after NetAdapt) to 403 MB (obtained after applying "Clustering Algorithm") which is approximately 65% reduction, calculated on FPGA. On Desktop (CPU) the reduction is upto 2x i.e., from 21.02 MB to 10.23 MB which is approximately 51%.

The Table. II shows the percentage of memory reduction in five convolutional layers of AlexNet after training, after NetAdapt implementation on trained AlexNet and finally after implementation of "Clustering Algorithm".

Therefore, it can be observed that "Clustering Algorithm" is able to achieve approximately 51% reduction in memory storage on NetAdapt.

Fig. 3b shows the graphical comparison of reduction in AlexNet storage memory requirement after training the model with CIFAR-10 data-set then after NetAdapt implementation on trained AlexNet and finally our "Clustering Algorithm" implementation on top of NetAdapt algorithm.

V. Conclusions

In the present work, we have proposed "Clustering Algorithm" that has ability to eliminate the redundancy of an inference engine in a lossless manner and coming up with a memory efficient approach which saves memory in the range of 51% to 65% . Our method operates by quantizing floating point values of weights followed by clustering unique values and finally replacing the unique value address in original kernel via code-assignment. We performed our experiment on the network obtained after applying NetAdapt on pre-trained AlexNet, which reduces the weight storage 2x to 3x times without loss of accuracy. This leads to less memory storage requirement for deployment of deep neural networks on resource constrained mobile devices. This drastic storage reduction leads to fit the neural network within on-chip SRAM cache which makes deep neural networks energy efficient to run on resource constrained platforms or on edge devices.

VI. Acknowledgement

This work is partially supported by the grant received from Ministry of Electronics and Information Technology (MEITY), Government of India.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[3] Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." arXiv preprint arXiv:1410.5401 (2014).

[4] Graves, Alex, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." Neural networks 18.5-6 (2005): 602-610.

[5] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." Journal of machine learning research 12.ARTICLE (2011): 2493-2537.

[6] Han, Song, 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).

[7] Panwar, Madhuri, et al. "M2DA: a low-complex design methodology for convolutional neural network exploiting data symmetry and redundancy." Circuits, Systems, and Signal Processing 40.3 (2021): 1542-1567.

[8] Yang, Tien-Ju, et al. "Netadapt: Platform-aware neural network adaptation for mobile applications." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[9] Wang, Hanrui, et al. "Hat: Hardware-aware transformers for efficient natural language processing." arXiv preprint arXiv:2005.14187 (2020).

[10] Denil, Misha, et al. "Predicting parameters in deep learning." Advances in neural information processing systems 26 (2013).

[11] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[12] Anwar, Sajid, Kyuyeon Hwang, and Wonyong Sung. "Fixed point optimization of deep convolutional neural networks for object recognition." 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015.

[13] Chen, Wenlin, et al. "Compressing neural networks with the hashing trick." International conference on machine learning. PMLR, 2015.

[14] Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." arXiv preprint arXiv:1412.6115 (2014).

[15] Sze, Vivienne, et al. "Efficient processing of deep neural networks: A tutorial and survey. arXiv 2017." arXiv preprint arXiv:1703.09039 (2017).

[16] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems 28 (2015).

[17] LeCun, Yann, John Denker, and Sara Solla. "Optimal brain damage." Advances in neural information processing systems 2 (1989).

[18] Molchanov, Pavlo, et al. "Pruning convolutional neural networks for resource efficient inference." arXiv preprint arXiv:1611.06440 (2016).

[19] Hu, Hengyuan, et al. "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures." arXiv preprint arXiv:1607.03250 (2016).

[20] Srinivas, Suraj, and R. Venkatesh Babu. "Data-free parameter pruning for deep neural networks." arXiv preprint arXiv:1507.06149 (2015).

[21] Gordon, Ariel, et al. "Morphnet: Fast & simple resource-constrained structure learning of deep networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[22] He, Yihui, and Song Han. "Adc: Automated deep compression and acceleration with reinforcement learning." arXiv preprint arXiv:1802.03494 2 (2018).

[23] T. Yang, Y. Chen and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 6071-6079, doi: 10.1109/CVPR.2017.643.

[24] T. -J. Yang, Y. -H. Chen, J. Emer and V. Sze, "A method to estimate the energy consumption of deep neural networks," 2017 51st Asilomar Conference on Signals, Systems, and Computers, 2017, pp. 1916-1920, doi: 10.1109/ACSSC.2017.8335698.

[25] Y. -H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.