Survey on Optimization Techniques to Reduce Memory Footprints and Inference Time

Anisha Aswani
Electrical and Computer Engineering
Colorado Satte University
Fort Collins, CO
anisha.aswani@colostate.edu

***Abstract*—Deep neural networks (DNNs) has been on a tremendous rise in the field of visual recognition tasks. As they have high computation and storage size, it is difficult to deploy these networks on embedded systems. Recent techniques on compressing these networks have been studied in this survey. These techniques include quantization, binarization, trained ternary quantization, pruning, and hardware accelerator designs. The techniques described have shown to successfully compress the neural network model all the while keeping the accuracy loss to the minimum. These techniques have been thoroughly surveyed with their pros and cons. They have been experimented and evaluated on previously available datasets like CIFAR-10 and ImageNet. Finally, the paper is concluded and other challenges are discussed.**

Keywords—Deep neural network, Quantization, Binarization, Pruning, and Model Compression

I. INTRODUCTION

Deep neural networks (DNNs) has been used in a lot of applications of visual detection and have been known have impressive accuracy in these applications. DNNs have millions of parameters that require very high computational capability. For DNNs to perform adequately, the high computation is required in the hardware of the model. In their work, Krizhevsky et al. [1] built a DNN model with 60 million parameters having five convolutional layers and three fully-connected layers. A model this big take days to train even on a high GPU. As such, it is very time consuming to train such models to get reasonable accuracy. Most of these architectures rely on the fully-connected layers [2]. This makes the number of parameters in the model to be in millions [2].

Due to DNNs computational cost and high storage requirement, it makes it difficult to deploy them on resource-constrained environments of embedded systems and mobile devices. To make use of DNNs on such devise, it is imperative to find ways to reduce the storage and computational of the original network. Modelling such an efficient neural network would have significant impacts as recent years have witnessed a growth in virtual reality and smart wearable devices. Cell-phones and FPGA hold only megabytes of resources, and finding an efficient way to compact DNNs would be of great significance.

To obtain the compact model of neural networks, machine learning solutions have been found to be very impactful. Along with this, optimization, hardware acceleration, data compression, and hardware design also contribute to compact the network. In this survey paper, recent works on network compression and hardware acceleration of deep neural networks is reviewed.

The different approaches have been classified into parameter pruning and sharing, quantization and binarization. Parameter pruning is based on methods to identify redundant parameters and removing such parameter without affecting output accuracy. Low-rank factorization is based on methods using matrix decomposition to find the important and useful parameters of the network. Parameter pruning and sharing use methods like vector quantization and have sparse constraints which makes them achieve the compact network in several complex steps. Low-rank factorization is shown to be easily implemented on GPU environment.

II. TECHNIQUES

A. Trained Ternary Quantization.

Zhu et al. *[3]* looks into the problem of deploying DNNs on resource-constrained devices. The authors propose Trained Ternary Quantization (TTQ) to reduce precision of weights to ternary values in networks. The paper highlights the quantization method which can learn both ternary values and assignments. Instead of using the traditional {-1,0,+1} or using the mean of absolute value E, {-E,0,+E}, the authors propose to quantize weights to $\{-W_l^n,0,+W_l^p\}$, which uses two full-precision scaling coefficients W_l^p, W_l^n for each layer l. W_l^p and W_l^n are positive and negative weights having different absolute values which have trainable parameters. In the method proposed, full-precision weights are normalized to the range of [-1, +1] by

dividing each weight by the maximum weight. Next, the intermediate full-resolution weights are quantized to {-1, 0, +1} using threshold. This threshold is the same for all the layers which reduces the search space. Finally, trained quantization is implemented by back propagating two gradients. At inference time, only ternary weights are used and the full-resolution weights are discarded.

CIFAR-10 [4, 4] and ImageNet [5, 5] datasets have been used for the implementation of the proposed method. The network is tested on TensorFlow [6] and Caffe [7, 7] frameworks. The proposed model successfully compresses weight and speeds up inference which is achieved by reducing bit precision and introducing sparsity. The results on CIFAR-10 show that the accuracy obtained by ternary quantization outperforms full-precision models of ResNet-32,44,56 by 0.04%, 0.16%, 0.36% respectively. For ImageNet, the ternary quantized model outperforms full-precision AlexNet model by 1.6%. About 16x model compression is achieved by converting most parameters to 2-bit values. Switching the weights from binary to ternary network bring sparsity into the weight. This gives the option to skip computation on zero weights which leads to higher energy efficiency. Due to the introduction of sparsity in weights, the accuracy was observed to be reduced. Using threshold instead of sparsity produced better results as it allows sparsity if each layer to be different. Thus, the proposed model achieves the same accuracy as a full-precision model on both CIFAR-10 and ImageNet as well as reduces parameter size by 16x.

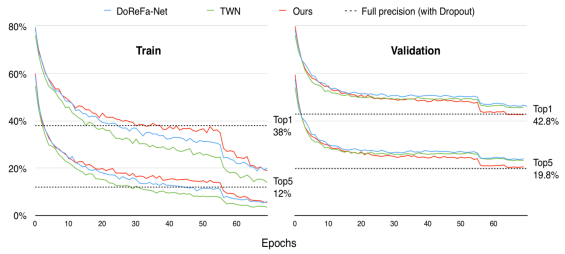


Figure 1: Training and validation accuracy of AlexNet on ImageNet

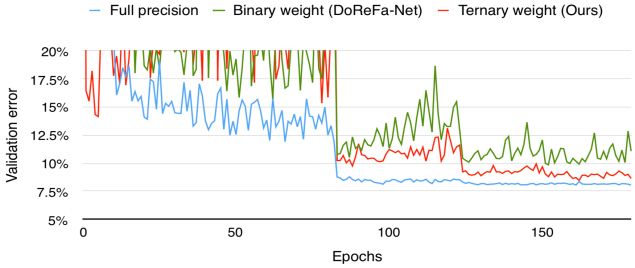


Figure 2: ResNet on CIFAR-10 with different weight precision

B. *Vector Quantization*

Deep convolutional neural networks (CNN) [1] have been known to demonstrate highly efficient results for image classification and object detection. However, deep CNN usually involves large number of layers with a million parameters. This prohibits the usage of deep CNNs on resource-constrained environment such as cell phones or embedded systems. Ging et al. [8] propose an information theoretical vector quantization method for compressing the large number of parameters used in the neural network. The authors explore two methods for compressing the parameters in densely connected layer: (1) Matrix factorization method; and (2) vector quantization. The use of matrix factorization speeds up CNN and also compresses parameters in linear models [9]. For vector quantization, the author has described different methods such as binarization [10], scalar quantization using *k* means, product quantization [11], and residual quantization [12]. Scalar quantization using *k* means catches the redundancy for each neuron. Some local redundancy structure is explored by product quantization. Residual quantization explores the redundancies of global structure between weight vectors.

The above methods were implemented on ILSVRC2012 benchmark image classification dataset. The results show that *k* means method gives a good compression rate without decreasing performance. Product quantization gives the best result of compression rate with minimal decrease in performance as there are important sub-vector local structures in the weight matrices. Thus, the paper presents a method of vector quantization to compress deep CNNs in storage-constrained embedded systems.

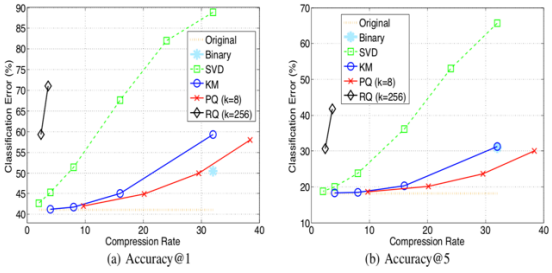


Figure 3: Comparison of different compression methods on ILSVRC dataset

C. *Quantized Convolutional Neural Networks*

For computer vision tasks, convolutional neural networks (CNNs) [13] have demonstrated impressive performance. However, CNNs is hard to deploy on mobile devices due to high computation complexity. Wu

et al. [14] proposes an efficient framework, quantized CNN, that speeds up computation as well as reduces storage and memory overhead of CNN models. This method focuses on quantizing both the filters in convolutional layers and weighting matrices in fully-connected layers. This aims at minimizing the error estimation of each layer. The minimization is done when the parameters are quantized, which preserves the model performance. An effective training scheme is implemented so as to reduce the accumulative error while quantizing multiple layers. In previous works on quantization-based approach, the focus is mainly on fully connected layers [15] [8] which leads to no acceleration of computation during the test phase. Contrary to this, the [14] approach offers simultaneous acceleration and compression of convolution and fully-connected layers. This drastically reduces the run-time memory consumption.

The proposed method is evaluated on two benchmarks, MNIST [16] and ILSVRC-12 [5]. The framework achieves around 4x to 6x speed-up and 15x to 20x compression [14] while keeping the loss in accuracy within one percentage. This model can thereafter be used on mobile devices where it classifies images within one second. The authors successfully proposed a method to enable efficient test-phase computation using quantization of network parameters. The results accomplish extraordinary speed-up and compression rates, with negligible loss in classification accuracy.

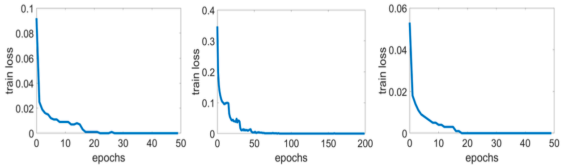
D. Binarization

Deep Neural Networks (DNNs) have been known to have state-of-the-art performance in speech recognition, object recognition, and image classification [17]. DNNs have large parameters with large number of network weights. This leads to training time inefficiency and storage space inefficiency. This paper [18] proposes an algorithm, proximal Newton algorithm [19], with diagonal Hessian approximation which minimizes the loss corresponding to the binary weights. Instead of direct approximation of weights, the method takes into consideration the effect of binarization on the loss that occurs during binarization. This loss is optimized using the proximal Newton algorithm [19] with a diagonal Hessian. Focusing on the proximal step in the proximal algorithm, a closed-formed solution is found for this step. Previous work on weight binarization did not take into consideration the effect to the loss while working on finding the closest binary approximation [20] [21]. This paper takes the loss into consideration during binarization itself [18]. The weight in each layer $\boldsymbol{w}_l$ is binarized as

$$\widehat{\boldsymbol{w}_l} = \alpha_l \boldsymbol{b}_l \qquad (2)$$

Where $\alpha_l > 0$ and $\mathbf{b}_l$ is binary.

The proposed scheme is implemented on feedforward networks and recurrent neural networks. On both the networks, the results show that the proposed scheme, Loss-Aware Binarized network, outperforms BinaryConnect and Binary-Weight-Network. Since this paper focuses on the loss during binarization, it performs better than the existing binarization algorithm and is more robust for wide and deep networks.



(a) MNIST. (b) CIFAR-10. (c) SVHN.

Figure 4: Classification of error convergence on LAB

E. Tensorizing Neural Networks

In an attempt to reduce the computational resources of deep neural networks, Novikov et al. propose a method that focuses on reducing the number of parameters while preserving the accuracy [22]. Fully-connected layers of neural networks usually require large amount of memory, which makes it difficult to deploy neural networks on low-end devices. In this paper, the dense weight matrices of the fully-connected layers are converted to Tensor Train (TT) [23] format to reduce the number of parameters in the neural network. According to the method proposed, the layer that has been most frequently used i.e. the fully-connected layer, is identified. This layer consists of linear transformation, defined by large dense matrix, which transform a high-dimensional input signal to high-dimensional output signal. The Tensor-Train [23] format is used to represent the dense weight matrix of the fully-connected layer with fewer parameters. Stochastic gradient decent is usually used to train neural networks. In this, the gradient is usually computed using back-propagation [24]. A gradient of loss-function is computed with respect to all the parameters of the network. In the proposed method, the gradient of loss-function is computed in a reverse manner, where the loss-function gradient is first computed for the output of the last layer and it proceeds sequentially through the layers. The input of the next layer makes use of the gradient computed earlier with respect to the parameters.

The proposed method is implemented on CIFAR-10 [1] and ImageNet. In CIFAR-10, the convolutional part is substituted by the TT-layer. The compression rate, as compared to the baseline, while using the TensorNet is found to be 1.24. When this algorithm is implemented in ImageNet [5], the first fully-connected layer is converted

to the TT-layer. Compressing the largest layer of the network makes the compression factor go to 7.4. Thus, the paper proposes a method to reduce the number of parameters which in turn preserves the expressive power of the layer. The networks with TT-layers match the performance of their full weights networks but require a smaller number of parameters and reduce the network by a factor of 7 [22].

F. *Using Weights and Connections for Efficient Neural Networks*

Han Song et. Al [25] propose a method for decreasing the storage and computation required by neural networks without effecting the accuracy. The authors implement this by using a technique of learning only important connections. A three-step method is used to prune redundant connections. First, the system is trained to realize which connections are significant. This is done by pruning network connections in a way that does not affect the accuracy. Next, the unimportant connections are pruned. A threshold for the weight is set and connections having weight below the threshold is removed. Finally, the network is retrained to fine tune the weights of the remaining connections so that they compensate for the connections that have been removed.

The network pruning was implemented in Caffe [7].The experiments were carried out on Nvidia TitanX and GTX980 GPUs. The paper shows the trade-off between the accuracy and the number of parameters. With a greater number of parameters pruned, the accuracy was found to be less. From the experiments it was observed that iterative pruning gave the best result with close to no accuracy loss. Thus, the successfully presented an algorithm to reduce memory capacity and band-width requirements, making it easier to deploy on embedded systems.

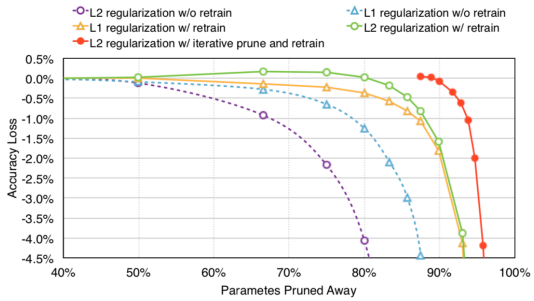


Figure 5: Trade-off curve for parameter reduction and loss in top-5 accuracy

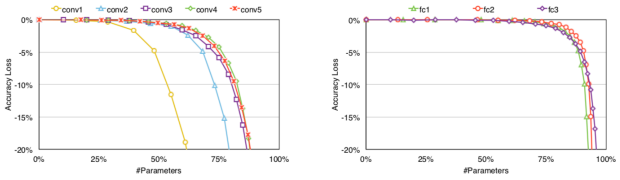


Figure 6: Pruning Sensitivity for CONV layer (left) and FC layer (right) of AlexNet

G. *Bitwise Neural Networks*

Kim et al. [26] propose Bitwise Neural Network (BNN) suitable for resource-constrained environments such as embedded and mobile devices. The method proposes a process for using binary values for weight parameters, bias terms, input, and intermediate hidden layer output signals for the neural network. Every input node, output node, and weight are represented by single bit. The floating or fixed-point arithmetic is replaced with bitwise operations. The network is designed to have more training schemes like weight compression and noisy backpropagation. All variables used are bipolar binaries making the network a complete bitwise network. In the feedforward, instead of using multiplication, addition, and non-linear activation on floating or fixed-point variables, only XNOR or bitwise operations are used. A two-stage approach is used for training: First, the network is trained with a weight compression technique that converts the real-valued model into a BNN; Second the compressed weight is used to initialize BNN parameters, and a noisy propagation is done based on tentative bitwise parameters to train the actual BNN. A sign function is used as the activation function instead of a sigmoid such that the input to the following layer is bipolar binary.

The proposed algorithm is implemented on the hand-written digit recognition task on the MNIST dataset [16]. Since the floating or fixed-point arithmetic is replaced with bitwise operation, the BMM is suitable for resource-constrained environments. BNN implements less spatial complexity, less memory bandwidth, and less power consumption in hardware [26]. The results obtained in this paper shows that bitwise operation perform efficiently with less performance loss. Such an algorithm is computationally efficient and thus can be deployed on embedded systems.

H. *Accelerator for Compressed Deep Neural Network*

Deep Neural Networks (DNNs) have been known to be computationally and memory intensive due to the millions of connections which makes them hard to deploy on embedded systems. This paper proposes an accelerator for sparse and weight sharing neural networks [27]. The

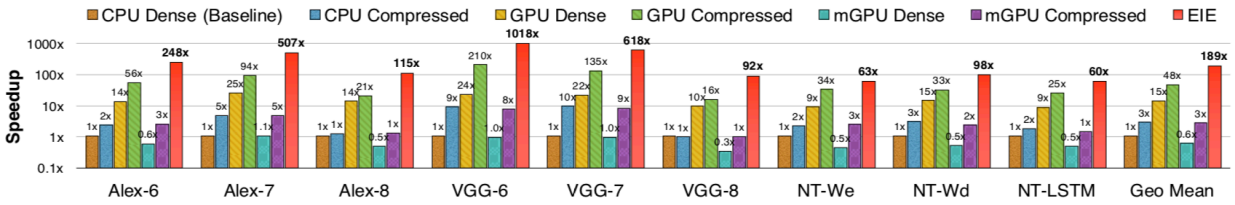


Figure 7: Speedups of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model

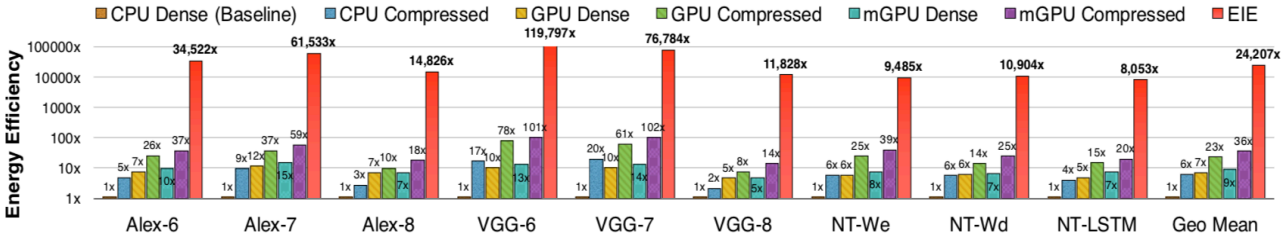


Figure 8: Energy efficiency of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model

computation is done from the hardware but the weights are fetched from DRAM which required more power. The authors propose EIE, an efficient inference engine. This accelerator performs customized sparse matrix vector multiplication and handles weight sharing without any loss in efficiency. This method operates on compressed networks which enables the large neural networks models to fit in on-chip SRAM. The authors proposed method takes into consideration the dynamic sparsity of activation that contributes to save computation.

The proposed method is implemented on different models such as CNN for object detection and LSTM for natural language processing and image captioning. The working of this accelerator on different hardware such as CPU, GPU, FPGA, and other ASIC accelerators is observed. It is observed that the implementation of this accelerator results in 120x better than external DRAM in energy savings [27] as shown in Fig 7. And Fig 8. The number of parameters is pruned by 10x and use of weight sharing makes the model smaller for it to be used in on-chip SRAM instead of DRAM. Thus, the authors successfully propose an accelerator, EIE, that implements sparsity in activations and weights, and uses weight sharing and quantization that reduces the energy need to compute a typical fully connected layer by 3400x compared with GPU [27].

I. Learning Structured Sparsity in Deep Neural Network.

Wen et al. [28] propose a *Structured Sparsity Learning* (SSL) that regularizes the structures of DNNs and creates a compact structure from a bigger DNN. Deploying DNNs on resource-constrained environment has always been memory-intensive and computation-intensive. The proposed method uses Lasso regularization during the training to construct a compressed structure of deep CNNs. SSL implements generic regularization to adaptively adjust the structure of filters, channels, filter shapes of each layer, and structure of depth beyond the layers in DNN. SSL is mainly implemented on convolutional layers of the DNNs. Using SSL, a filter-wise, channel-wise, shape-wise, and depth-wise structure is formulated. For filters and channel, all the unimportant filters and channel are removed. Next the arbitrary shapes of the filter are learned, followed by regularization of the layer depth.

The proposed method is experimented on three databases: MNIST, CIFAR-10, and ImageNet. The experiments show that for CIFAR-10 model that accuracy improves from 91.25% to 92.60% while reducing the number of layers from 20 to 18. Thus, the method proposed can learn to create compact structures of DNNs without any accuracy loss. This speeds up computation for DNN evaluation on CPU and GPU. Classification accuracy is also observed while implementing SSL. Overall, SSL converts a large-scale DNN to a more compact structure to be deployed on embedded and mobile devices.

J. Pruning Filters for Efficient ConvNets

Pruning of weights based on magnitudes reduces a significant number of parameters from the fully connected layers of the network [29]. Given a threshold,

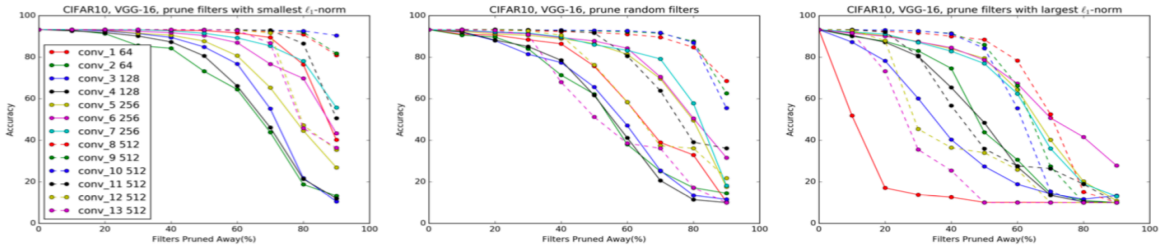


Figure 9:Pruning for smallest filter, pruning random filters, and pruning the largest filters

all the kernel weight filters which are lower than it is pruned away in magnitude-based weight pruning. The authors [29] present a method that accelerates the CNN. The filters having small effect on accuracy is identified and they are pruned. Pruning the filters occurs in a structured way without introducing sparsity. The authors adopt a one-shot pruning and retraining strategy instead of layer-wise fine-tuning. The method of pruning is such that the less useful filters are removed from a well-trained model while minimizing loss in accuracy loss. The relative importance of a filter in each layer is calculated as the sum of its absolute value $\sum |F_{ij}|$. For understanding the sensitivity, each layer is pruned away independently and evaluated on its accuracy on the validation set. A different strategy, greedy pruning strategy, accounts for the filter that have been removed in previous layers [29]. To compensate for the performance degradation after pruning the filters, the network is retrained iteratively. For every pruning layer by layer, the model is retrained before pruning the next layer so that the weights adapt to the changes from the pruning process.

The proposed method is implemented on two networks: VCG-16 on CIFAR-10 and ResNet-56/110 on CIFAR-10 and ResNet-34 on ImageNet. Since the proposed method is based on magnitude, there is significantly less loss in accuracy. Pruning filters with less significance results in less computational costs. This method also does not produce sparsity and thus it does not need the support of sparse convolutional libraries [29] or any specialized hardware. Reducing the number of filters corresponds to a target speed-up. The implementation of this method on VCG-16 resulted in 34% reduction in inference costs and ResNet-110, 38% on CIFAR-10 while having the output accuracy close to the original accuracy.

K. Efficiency of Pruning for Model Compression

Recent research has found model pruning to be an efficient way of reducing the size of a model without compromising accuracy. Zhu et al. [30] looks into the possibility that the baseline models are over-parameterized and the simple alternative would be to decrease the number of hidden units while maintain the model's dense connection structure [30]. A deep study into the effectiveness of model pruning for model compression is done by the authors. The pruning of network's connections is done during training. The Tensorflow [6] framework is used for pruning. From the network connections, the layers to be pruned is chosen. For every layer, a binary mask variable is added. This binary mask is of the same size and shape as the weights of the layer. This helps in determining which weights are important in the network connections. A new pruning algorithm is introduced by the authors. In this algorithm the sparsity is increased from an initial sparsity value s_i, which is usually at 0, to a final sparsity value s_f over a range of n pruning steps. The training step is started at t_0 with a pruning frequency of Δt: [30]

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad for\ t \in \{t_0, t_0 + \Delta t, \dots, t_0 + n\Delta t\} \qquad (1)$$

For every Δt steps, the binary weights of the masks are updated. The network is trained to gradually increase the sparsity of the network while allowing the training steps to recover from any loss in accuracy due to pruning.

This algorithm has been tested on two types of model: (1) A large model is trained pruned to obtain a sparse model which contains a smaller number of non-zero parameters; and (2) A small dense model is trained and the size of the model is similar to the large-sparse model [30]. The large-sparse model achieves higher accuracy than the small dense models. Is it observed that in pruning, the number of non-zero valued connections in the network are reduced. This paper presents the trade-off between model size and accuracy in pruned neural networks. The algorithm proposed by the authors encourage the use of model pruning as a tool for compressing neural networks when implementing on embedded systems in resource-constrained environments.

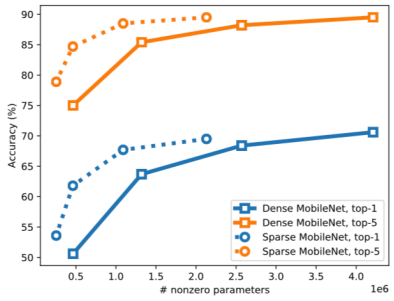


Figure 10: Sparsity vs No. of parameters

III. CONCLUSION AND FUTURE WORK

A survey has been successfully performed on recent techniques to compress DNNs so as to deploy them on resource-constrained environment such as embedded systems, mobile devices, and FPGAs. From the survey, we have found that pruning and sharing contribute to a better compression rate of the original model with a very small loss in output accuracy. They have a better performance rate while having a stable model. It is possible to combine more than one technique to get the optimal results.

There are a few challenges still faced in this domain. Most of the DNN models have limited freedom to change the configuration, thus not much change in size can be made while keeping the accuracy loss small. There are hardware constraints that still cannot deploy the compressed DNN model to its full accuracy. How to efficiently compress the DNN model is still a challenge that needs to be researched on. Pruning is found to be an effective method to compress the model, although pruning eliminates connections completely from the network to reduce the model size. This may affect the input to the next input layer and affect the output accuracy.

Further research can be done on this topic for model compression and hardware acceleration. The authors of [31] [32] solve hyperparameter problem by providing a framework that allows the model to automatically learn on exploiting structure in the problem of interest. [33] Uses reinforcement learning to design the compressed model efficiently. Wang et al. [34] proposed a hardware aware approach which uses Hardware-Aware Quantization (HAQ) to have a hardware accelerators feedback in the design loop.

Future work can be done on applications with larger deep nets like videos or on deep natural language models.

IV. REFERENCES

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet Classification with deep convolutional neural networks," in *Neural Information Processing Systems* , 2012.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang and A. Ng, "Large scale distibuted deep networks," in *NIPS*, 2012.

[3] C. Zhu, S. Han, H. Mao and W. J. Dally, "Trained Ternary Quantization," in *arXiv:1612.01064v1*, 2016.

[4] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," 2009.

[5] O. Russakovsky, J. Deng, h. Su, J. Krause, S. Satheesh, S. Ma, A. Karpathy, A. Khosla, M. Bernstein, A. Berg and L. Fei-Fei, "Imagenet Large Scale Visual Recongnition Challenge," in *Internationla Journal of Computer Vision (IJCV)*, 2015.

[6] M. Abadi and e. al., "Tensorflow: Large-Scale machine learning on heterogenous systems," in *URL https://www.tensorflow.org*, 2015.

[7] J. Yangqing and e. al., "Caffe: Convolutional architectue for fast feature embeddeding," in *arXiv preprint. arXiv: 1408.5903*, 2014.

[8] Y. Ging, L. Liu, M. Yang and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," in *arXiv:1412.6115v1*, 2014.

[9] E. Denton, W. Zaremba, J. Bruna, Y. LeCun and R. Fergus, "Exploiting Linear Structure within convolutional networks for efficient evaluation," in *Neural Information Processing Systems*, 2014.

[10] L. WAN, M. Zeiler, S. Zhang, Y. Lecun and R. Fergus, "Regularization of Neural Networks using Dropconnect," in *ICML*, 2013.

[11] H. Jegou, M. Douze and C. Schmid, "Product Quantization for Nearest Neighbor Search," in *IEEE TPAMI*, 2011.

[12] Y. Chen, T. Guan and C. Wang, "Approximate Nearest Neighbor Search by Residual Vector Quantization," in *Sensors*, 2010.

[13] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard and L. Jackel, "Backpropogation applied to hand written zip code recognition," in *Neural Computation, 1(4):541-551*, 1989.

[14] J. Wu, C. Leng, Y. Wang, Q. Hu and C. Jian, "Qunatized Convolutional Neural Networks for Mobile Devices," in *arXiv:1512.06473v3*, 2016.

[15] W. Chen, T. Wsilson, S. Tyree, K. Weinberger and Y. Chen, "Compressing Neural Network with the Hashin Trick," in *Internationakl Conference on Machine Learning (ICML)*, 2015.

[16] Y. LeCun, L. Bottou, Bengio.Y. and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceeedings of IEEE,86(11):2278-2324*, 1998.

[17] Y. LeCun, Y. Bengio and G. Hinton, "Deep Leaning," in *Nature 521(7553):436-444*, 2015.

[18] L. Hou, Q. Yao and J. T. Kwok, "Loss Aware Binarization of Deep Networks," in *ICLR. arXiv:1611.01600v3*, 2018.

[19] J. Lee, Y. Sun and M. and Saunders, "Proximal Newton-type Methods for Minimizing Composite Functions.," in *SIAM Journal on Optimization, 24(3):1420-1443*, 2014.

[20] M. Rategari, V. Ordonez, J. Redmon and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[21] M. Courbariaux, Y. Bengio and J. David, "BinaryConnect: Training deep neural networks with binary weights during propogations.," in *NIPS*, 2015.

[22] A. Novikov, D. Podoprikhin, A. Osokin and D. Vetrov, "Tensorizing Neural Networks," in *arXiv:1509.06569*, 2015.

[23] I. Oseledets, "Tensor-Train Decomposition," in *SIAM J. Scientific Computing*, 2011.

[24] D. Rumelhart, G. Hinton and R. WILLIAMS, "Learning Representations by Back-Propagating Errors," in *Nature*, 1986.

[25] S. Han, P. Jeff, J. Tran and W. J. Dallu, "Learning both Weights and Connections for Efficient Neural Networks," in *arXiv:1506.02626*.

[26] M. Kim and P. Smaragdis, "Bitwise Neural Network," in *arXiv:1601.0607v1*, 2016.

[27] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram and M. A. D. W. J. Harowitz, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *arXiv:1602.01528v2*, 2016.

[28] w. Wen, C. Wu, Y. Wang, Y. Chen and H. Li, "Learning Structured Sparsity in Deep Neural Networks," in *arXiv:1608.03665v4*, 2016.

[29] H. Li, K. Asim, I. Durdanovic, H. Samet and H. P. Graf, "Pruning Filter for Efficient ConvNets," in *ICLR. arXiv:1608.08710v3*, 2017.

[30] M. H. Zhu and S. Guta, "To prune or not to prune: exploring the efficacy of pruning for model compression," in *arXiv: 1710.01878v2*, 2017.

[31] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. Hoffman, D. Pfau, T. Schaul and N. d. Freitas, "Learning to learn by gradient descent by gradient descent," in *Neural Information Processing Systems (NIPS)*, 2016.

[32] D. Ha, A. Dai and Q. Le, "Hypernetworks," in *International Conference on Learning Representations*, 2016.

[33] y. He, J. Lin, Z. Liu, H. Wang, L. Li and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *The European Conference on Computer Vision*, 2018.

[34] K. Wang, Z. Liu, Y. Lin, J. Lin and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *IEEE conference on Computer Vision and Pattern Recgnotion (CVPR)*, 2019.

[35] H. Cai, L. Zhu and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Internation Conference on Learning Representations*, 2019.

[36] S. Ioffe and C. Szegedy, "Batch Normalization: Acclerating Deep Network trsaining by Reducing Internal Covariate Shift," in *arXiv:1502.03167v3*, 2015.

[37] Y. Gong, L. Liu, M. Yang and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," in *CoRR,abs/1412.6115*, 2014.