A Smoothed LASSO Based DNN Sparsification Technique

Basava Naga Girish Koneru, *Student Member, IEEE*, Nitin Chandrachoodan, *Member, IEEE*
and Vinita Vasudevan, *Member, IEEE*

Abstract—Deep Neural Networks (DNNs) are increasingly being used in a variety of applications. However, DNNs have huge computational and memory requirements. One way to reduce these requirements is to sparsify DNNs by using smoothed LASSO (Least Absolute Shrinkage and Selection Operator) functions. In this paper, we show that for the same maximum error with respect to the LASSO function, the sparsity values obtained using various smoothed LASSO functions are similar. We also propose a layer-wise DNN pruning algorithm, where the layers are pruned based on their individual allocated accuracy loss budget determined by estimates of the reduction in number of multiply-accumulate operations (in convolutional layers) and weights (in fully connected layers). Further, the structured LASSO variants in both convolutional and fully connected layers are explored within the smoothed LASSO framework and the tradeoffs involved are discussed. The efficacy of proposed algorithm in enhancing the sparsity within the allowed degradation in DNN accuracy and results obtained on structured LASSO variants are shown on MNIST, SVHN, CIFAR-10, and Imagenette datasets.

Index Terms—Deep Neural Networks, LASSO, smoothing functions, sparsity, structured LASSO

I. INTRODUCTION

Deep Neural Networks (DNNs) are used in a large number of classification and recognition tasks used by commercial, medical and military applications. Having to model complex patterns and predictions for real world problems, DNNs are computational and memory intensive, owing to large number of connections and computational nodes. This creates a bottleneck in deploying DNNs on present-day devices owing to the limited on-board resources available on these devices. This situation necessitates strategies to minimize computational and memory requirements of DNNs without adversely affecting their performance.

The approaches in research fraternity towards this direction can be broadly classified into two clusters: 1) Approximating certain portions of DNNs by means of approximate computing techniques; 2) Pruning the redundant portions of DNNs using sparsifying algorithms. The inherent error-tolerant feature of the DNN applications is leveraged by these approaches towards minimizing the resources required.

Regularization is a widely used technique to help DNNs generalize the examples learnt during training. The main objective during training is to minimize the cost function $C(W)$, where W is the set of all the DNN weights. Typically,

a cost function comprises of error function (such as cross entropy) and a penalty term for regularization.

A detailed survey of various penalty terms used during DNN training can be found in [48]. In general, the most commonly used types of penalty terms are: Ridge(L_2) [18]–[21], Weight elimination [22]–[24], Approximate smoother [25], [26] and LASSO (L_1).

The benefit of using LASSO (Least Absolute Shrinkage and Selection Operator) regularization while training the DNNs is two-fold. It enhances the generalization capabilities of the DNNs for the examples learnt during training and produces sparse DNNs. However, the LASSO function is non-differentiable at origin. Therefore, it does not fit the standard DNN training framework using gradient descent based algorithms involving backpropagation of the gradient information. One widely used and popular approach to overcome this difficulty is to use smoothed LASSO functions ([1]–[3], [7], [29], [30], [35]). Using a non-convex approximation [49] is another way to overcome the drawback of non-differentiability of LASSO function at origin. The non-differentiability of LASSO is also dealt with by using subgradient method. However, subgradient method has a slower rate of convergence than smoothed LASSO based method [13].

In this work, we focus on sparsification of DNNs using smoothed LASSO based regularization techniques.

A. Related works

DNN approximation involves approximating certain DNN portions by various methods such as precision scaling, approximate multipliers and approximate adders. Pruning involves sparsifying a DNN by removing the insignificant (or redundant) connections (or group of connections) from the DNNs.

The approximate/pruning algorithms employed by various works in literature typically comprise of identifying DNN portions to be approximated (or pruned), modifying them accordingly. In many cases, the DNN is healed by means of re-training to compensate for the loss in DNN performance due to approximations or pruning (sparsifying).

1) *DNN approximations*: [5] and [11] rank the various neurons respectively based on their sensitivity. The sensitivity of neurons in these works are determined by calculating the contribution of neurons to the DNN output degradation by using gradient information of neuron output obtained during backpropagation. Those neurons, causing significant degradation in DNN output quality are deemed sensitive. The less sensitive neurons are approximated (by using precision scaling and approximate multipliers) and DNNs are retrained to

possibly recover the loss in accuracy. These steps are repeated iteratively as long as the DNN performance loss is within the tolerable limit. [5] calculates the energy consumption of MAC (Multiply And Accumulate) operations in every layer using Synopsys power compiler. It then chooses the energy-intensive DNN layer in every iteration and identifies the sensitive neurons within that layer. [11] ranks the DNN neurons based on their sensitivity values and those with sensitivity values below a certain threshold are approximated. [8] approximates a DNN using permuted diagonal matrices, where a pre-trained dense DNN is approximated to a permuted diagonal model and the non-zero elements lying along the permuted diagonals alone are re-trained.

2) *DNN Sparsification*: Majority algorithms in DNN sparsification works use LASSO function. These works such as [4], [9], [31]–[34], [36]–[43], [45] use various Deep Learning libraries such as Tensorflow and PyTorch for training the DNNs with LASSO function. These libraries use subgradient methods to deal with non-differentiability of LASSO function at the origin. Subgradient methods are known to have slower convergence [13] due to step change in derivatives near the origin. [14] incorporates a convex optimization based algorithm to sparsify the DNN. However, algorithms based on convex optimization techniques incur higher computational costs.

[1]–[3], [7] cover a wide range of LASSO smoothing functions from polynomial approximations [1] to higher order approximations (logarithmic [2]), (hyperbolic [3]). [29], [30], [35] use quartic smoothing function in the context of $L_{1/2}$ regularization based pruning. [1] and [7] have employed the polynomial based smoothing functions and quadratic smoothing function respectively in the context of group LASSO for pruning neurons in the hidden layer and input layer (features) of feedforward fully connected neural networks respectively. As shown in [3], the poorer convergence of logarithmic smoothing function proposed in [2] limits its usage in LASSO based training of DNNs. The remaining smoothing functions (polynomial class of functions [1], hyperbolic [3]) have demonstrated significant generalization and pruning capabilities for different neural nets with different activation and error functions.

In [6], [12], [46], the DNNs are sparsified by retaining the important connections by means of L_2 regularization, dropout techniques and pruning the unimportant connections. The remnant DNN is retrained (with L_2 regularization and dropout) iteratively until the degradation in DNN accuracy levels are within the acceptable limits.

3) *Pruning algorithms*: [37], [38], [43], [44] use the L_1 norm as a sensitivity metric in the context of pruning filters in convolutional layers. The filters with lower L_1 norm values are deemed less sensitive and are pruned first. [45] prunes one layer at a time for various compression ratios (fraction of connections pruned). After obtaining the DNN accuracy loss characteristics of various pruning ratios for all layers, the set of best individual compression ratios for all layers to achieve the targetted overall DNN compression ratio for minimum possible loss in accuracy is determined by using binary search algorithm. [42] uses layer-wise relevance propagation method [50] where, the contribution of layers to the intended output

of DNN is ascertained by using backpropagation information. [46] uses Monte Carlo analysis for identifying the best subset of weights in the pruned DNN. It also proposed another heuristic where filters with least sum of output values across the testing data are pruned.

B. Contributions of this paper

The main contributions of this paper are as follows:

- 1) As opposed to previous studies ([1], [3]), we find that all smoothing functions exhibit similar pruning capabilities, if the maximum error with respect to the LASSO function is set to the same value across all the smoothing functions.
- 2) We propose a novel layer-wise pruning algorithm to enhance the sparsity in DNNs. Based on estimates of the reduction in number of multiply-accumulate operations (in convolutional layers) and weights (in fully connected layers), accuracy loss budgets are assigned to each of them. Individual layers are then pruned accordingly.
- 3) Within the smoothed LASSO framework, we explore structured LASSO variants for the fully connected and convolutional layers separately and discuss the trade-offs involved.

The rest of this paper is as follows: Section II discusses the limitations of the existing algorithms using smoothed LASSO functions. Section III contains the study of smoothing functions based on a fixed maximum error. Section IV presents the proposed novel layer-wise pruning algorithm and Section V contains the detailed discussion on structured LASSO variants. Section VI presents the results of experiments on the proposed pruning algorithm and structured LASSO variants. Section VII concludes the paper. Architecture details of neural networks used in this paper are given in an appendix.

II. LIMITATIONS OF EXISTING ALGORITHMS USING SMOOTHED LASSO FUNCTIONS

In this section, we discuss the limitations of LASSO based algorithms in [1] and [3].

We use the terms “sparsity” to denote the proportion of weights pruned and “neuron sparsity” for the proportion of neurons pruned.

In the presence of a penalty term, cost function $C(W)$ is written as:

$$C(W) = C_o(W) + \lambda P(W) \quad (1)$$

where, W is the set of DNN weights, $C_o(W)$ is the error term, $P(W)$ is the penalty term and λ is the penalty coefficient. When a smoothed LASSO function ($h(w)$) is used as the penalty term, the cost function can be re-written as follows:

$$C(W) = C_o(W) + \lambda \sum_j \sum_{i=1}^{N_j} h(w_i^j) \quad (2)$$

where, w_i^j is the i^{th} weight in j^{th} layer and N_j is the number of weights in j^{th} layer. The existing smoothing functions are listed in Table I (for brevity w_i^j is represented as w).

TABLE I
SMOOTHING FUNCTIONS

Function	$ w < \beta$	$ w \geq \beta$
Sqrt [1]	$\sqrt{w^2 + \beta} + \beta - \sqrt{\beta^2 + \beta}$	$ w $
Quadratic [1]	$\frac{w^2}{2\beta} + \frac{\beta}{2}$	
Quartic [1]	$-\frac{w^4}{8\beta^3} + \frac{3w^2}{4\beta} + \frac{3\beta}{8}$	
Sextic [1]	$\frac{w^6 - 15\beta^2 w^4 + 75\beta^4 w^2 + 35\beta^6}{96\beta^5}$	
Logarithmic [2]	$\frac{2}{\gamma} \ln(1 + e^{\gamma w}) - w$	
Hyperbolic [3]	$\frac{\ln(\cosh(\alpha w))}{\alpha}$	

For the sqrt, quadratic, quartic and sextic approximations, the smoothing parameter (β) lies between 0 and 1. Whereas, in the case of logarithmic and hyperbolic approximations, γ and α (both > 0) are typically of the order of 100. As defined in [3], the error w.r.t. LASSO is given by: $E = ||w| - h(w)|$. Given the poorer convergence of logarithmic smoothing function compared to other smoothing functions [3] and the higher computational expense of square root smoothing function, coupled with its lower pruning capability [1], these two smoothing functions are omitted from further discussions in this paper. The hyperparameters learning rate (η) and penalty coefficient (λ) values for various experiments have been systematically chosen based on the heuristics discussed in [28].

In order to evaluate the performance of various smoothing functions, we carried out a series of experiments using MNIST architecture I and MNIST architecture II. The pixel values in the MNIST dataset have been normalized to $[-1, 1]$ for all the experiments in this paper [1]. To compare with the results published in [1], we had chosen the smoothing parameter (β) for the polynomial class of functions to be 0.003 and smoothing parameter (α) for hyperbolic function to be 634 as discussed in [3]. The pruning threshold for each of these experiments were chosen so as to ensure maximum dip of 0.5% in the testing accuracy levels of the networks.

Table II shows the neuron sparsity values obtained by different smoothing functions using Group LASSO based pruning [1] on MNIST architecture II. The experiments were carried out in two different ways: 1) neurons were pruned during training (as done in [1]); 2) neurons were pruned after training. The neuron sparsity values obtained in Table II support the conclusions in [1] that employing higher order smoothing functions leads to larger sparsity in neural networks. LASSO based pruning [3] was carried out on MNIST architecture I using the different smoothing functions. Table III shows the sparsity values obtained. From the sparsity values, it can be observed that higher order smoothing functions exhibit superior pruning capabilities in conventional LASSO based penalty functions also.

The limitations of the algorithms proposed in [1] and [3] are as follows:

- 1) The pruning algorithms discussed above have been implemented with the assumption that the transition from smoothing approximation to the LASSO term occurred

TABLE II
RESULTS OF GROUP LASSO [1] ON MNIST ARCHITECTURE II WITH A FIXED PRUNING THRESHOLD OF 10^{-6} . $\eta = 0.03$, $\lambda = 10^{-3}$. EXPERIMENTS WERE CARRIED OUT IN TWO WAYS: 1) PRUNING DURING TRAINING (P.D.T.); 2) PRUNING AFTER TRAINING (P.A.T.)

Smoothing Function	β/α	Pruning type	Neuron sparsity (%)		Accu.(%)
			Hidden Layer I	Hidden Layer II	
Quadratic	0.003	P.D.T.	1.96	57.6	97.31
		P.A.T.	1.96	64.6	97.28
Quartic	0.003	P.D.T.	2.35	62.6	97.30
		P.A.T.	2.35	69.3	97.28
Sextic	0.003	P.D.T.	2.35	62.9	97.28
		P.A.T.	2.35	69.1	97.30
Hyperbolic	634	P.D.T.	2.35	63.6	97.30
		P.A.T.	2.35	69.9	97.30

TABLE III
RESULTS OF LASSO BASED PRUNING [3] ON MNIST ARCHITECTURE I WITH A FIXED PRUNING THRESHOLD OF 10^{-3} . $\eta = 0.1$, $\lambda = 0.3$

Smoothing Function	β/α	Sparsity (%)		Accu.(%)
		Hidden Layer	O/P Layer	
Quadratic	0.003	55.2	61.7	97.49
Quartic	0.003	55.6	63.5	97.46
Sextic	0.003	55.9	63.9	97.53
Hyperbolic	634	56.3	64.2	97.58

at same value (β) across all the smoothing functions (Figure 1a). As seen from Figure 1a, this assumption causes significant differences in the maximum error ($e = \max E$) due to each smoothing function. This, in turn affects the sparsity capabilities of the different smoothing functions, as will be seen in the next section.

- 2) From neuron sparsity values in Table II, it is observed that pruning MNIST architecture II after training has resulted in larger neuron sparsity values in hidden layer II compared to pruning during training [1] for similar accuracy values. This can be attributed to the fact that training a DNN fully until convergence with a LASSO based function is more likely to yield larger number of insignificant weights rather than altering the trajectory of some weights by pruning during the training process. Therefore, for all further experiments in this paper, we prune the DNNs only after they have converged.
- 3) The algorithms in [1], [3] train the neural networks using

Fig. 1. (a) Smoothing functions (b) Derivatives of smoothing functions with equal smoothing parameters $\alpha, \beta (= 1)$. Note that for hyperbolic function, maximum error does not occur at origin.

the smoothed Group LASSO/LASSO penalty functions from scratch. However, training the present state-of-art DNNs from scratch is quite time consuming. Also, the present state-of-art DNNs are trained robustly using different penalty terms such as L_2 . Therefore, it is advantageous to take these pre-trained DNNs and train them again with smoothed LASSO based functions to achieve sparsity. [4], [6] also used pre-trained DNNs for pruning channels and weights respectively by means of LASSO and dropout methods. We use this approach of starting with pre-trained DNNs in our proposed smoothed LASSO based pruning algorithm, as discussed in section IV.

In the following sections, we carry out systematic studies on various smoothing LASSO approximations and LASSO variants to acquire deeper insights into DNN pruning.

III. STUDY OF SMOOTHING FUNCTIONS FOR A FIXED MAXIMUM ERROR

In this section, we look at the effect of fixing the maximum value of error ($e = \max_w E$) to the same value in all smoothing functions. The smoothing functions are re-written in terms of e in Table IV.

Figure 2a shows the different smoothing functions, when the maximum error value (e) is set at same value for all the smoothing functions. From Figure 2b, it can be seen that the derivatives of different smoothing functions now match more closely when compared to those in Figure 1b (where the smoothing functions have same transition points). Since, the update of weights are based on derivatives of the smoothing functions, it is expected that setting maximum error (e) to the same value in all cases will result in relatively similar weight updates across all the cases.

For proper performance (pruning capability) of smoothing functions, the point of transition from smoothing function to $|\cdot|$ function can at most be in the order of 10^{-2} [7]. As this transition point depends on maximum error (e) value, it is necessary to arrive upon a proper choice of e value based on systematic experiments.

Towards this direction, we performed a series of experiments on various neural networks using various smoothing functions and e values. Based on the results obtained, we determine that quadratic smoothing function (with $e = 10^{-3}$) is the best choice for sparsifying DNNs. The details of these

TABLE IV
SMOOTHING FUNCTIONS IN TERMS OF e .

Function	r	$ w < r$	$ w \geq r$
Quadratic	$2e$	$\frac{w^2}{4e} + e$	$ w $
Quartic	$\frac{8e}{3}$	$-\frac{27w^4}{4096e^3} + \frac{9w^2}{32e} + e$	
Sextic	$\frac{96e}{35}$	$\frac{w^6 - (15(\frac{96e}{35})^2 w^4) + (75(\frac{96e}{35})^4 w^2)}{\frac{96^6 e^5}{35^5}} + e$	
Hyperbolic	-	$\frac{\ln(\cosh((\frac{\ln 2}{e})w))}{(\frac{\ln 2}{e})}$

Fig. 2. (a) Smoothing functions (b) Derivatives of smoothing functions with equal maximum error $e (= 1)$.

experiments will be discussed in the results section (Section VI) of this paper.

IV. PROPOSED LAYER-WISE PRUNING ALGORITHM

A. On pruning threshold

The earlier works on pruning [1], [4], [6] pruned the DNNs with a fixed threshold. Given a pruning threshold, there is a trade-off between the sparsity levels and the performance (accuracy) of the DNNs. The maximum permissible pruning threshold cannot be determined a priori. Therefore, it is necessary to increase the pruning threshold and test the DNN performance iteratively in a systematic manner (dynamic pruning) as long as the degradation in DNN output quality is within the acceptable limit. A pruning algorithm along this direction was proposed in [3] and was tested on different fully connected networks. It was shown to result in increased sparsity. To check the performance of this algorithm for convolutional layers, we used the algorithm to sparsify the convolutional layers of SVHN CNN I. Table V compares the smoothed LASSO based sparsity results of SVHN CNN I obtained by fixed threshold with those obtained by dynamic threshold on the convolutional layers. It can be observed that using dynamic pruning threshold leads to increase of sparsity values for similar accuracy levels.

B. Layer-wise pruning

In the state-of-art DNNs, there is a significant difference between each layer. For example, CNNs consist of a large number of convolutional layers, each differing significantly in filter dimensions, number of filters and channel widths. In such scenario, pruning weights of various layers with the same pruning threshold (Algorithm 1) results in ineffective pruning. Therefore, it is necessary to come up with layer-wise heuristics for effective pruning.

TABLE V
COMPARISON OF SPARSITY RESULTS ON CONVOLUTIONAL LAYERS FOR SVHN CNN I OBTAINED BY FIXED PRUNING THRESHOLD WITH THOSE OBTAINED BY DYNAMIC PRUNING THRESHOLD. $\eta = 3 \times 10^{-5}$, $\lambda = 0.3$

Pruning threshold	Sparsity (%)			Accu.(%)
	Conv. I	Conv. II	Conv. III	
Fixed (0.01)	12.5	27.3	31.5	87.84
Dynamic	15.71	33.57	38.72	87.85

Algorithm 1 Dynamic pruning: same threshold for all the layers.

Input: W (DNN weights), ΔA_{max} , p_i , Δp
Output: *Remnant weights*

```

1:  $p_u = p_i$ 
2: while  $\Delta A \leq \Delta A_{max}$  do
3:   for  $w_i^j \in W$  do
4:     if  $|w_i^j| \leq p_u$  then
5:       Prune  $w_i^j$ 
6:     end if
7:   end for
8:    $p_u = p_u + \Delta p$ 
9: end while

```

As discussed in Section IA, [37], [38], [42]–[46] carried out DNN pruning by using various layer-wise heuristics. [37], [38], [43], [44] use L_1 norm to determine the DNN parts for pruning. [42] uses a layer-wise relevance propagation method [50] to determine the layers for pruning. [46] uses Monte Carlo analysis for pruning the connections and identifies the insignificant filters based on their output sum values over the testing data. The compression ratio procedure used in [45] causes a significant overhead as every layer need to be pruned completely to identify the suitable pruning threshold. The heuristics in these works are good enough to identify the least sensitive DNN parts for pruning. However, they do not identify the energy-intensive layers in the DNNs.

Based on [51], the energy used in convolutional and fully connected layers primarily depends on number of MAC operations and number of weights respectively. Therefore, we include these two parameters in the heuristics used for layer-wise pruning in our proposed algorithm. Towards this direction, we introduce a metric termed “accuracy loss budget” for each layer. The procedure to determine accuracy loss budget for each layer is as follows:

Layers are pruned one at a time, while keeping all other layers intact, until the maximum allowed DNN accuracy loss (ΔA_{max}) is reached. The number of connections pruned in every layer (Sp_j) is determined. This count is the maximum sparsity that can be obtained in a particular layer. Unlike in [45], every layer need not be pruned completely, thereby reducing the overhead involved. Since, energy is based on number of MAC operations and weights for convolutional layers and fully connected layers respectively, we define the Sensitivity ratio (S_j) for each of these layers as follows:

- 1) Convolutional layers: S_j for a convolutional layer j is calculated in terms of maximum decrease in number of MAC operations (MSp_j) that can be obtained in that layer.

$$s_j = MSp_j = Sp_j \times h_{oj} \times w_{oj}$$

$$S_j = s_j / \left(\sum_{j=1}^{N_l} s_j \right)$$

where, h_{oj} and w_{oj} are height and width respectively of layer j output. Given the use of MSp_j for calculating S_j in convolutional layers, the sparsity metric here

should also reflect the reduction in MAC operations. Therefore, we introduce the term MAC sparsity, defined as the proportion of MAC (Multiply And Accumulate) operations that can be skipped due to pruning of weights. We use MAC sparsity to gauge the pruning results of all convolutional layer based experiments.

- 2) Fully Connected Layers: S_j for a fully connected layer is obtained directly from Sp_j value.

$$s_j = Sp_j$$

$$S_j = s_j / \left(\sum_{j=1}^{N_l} s_j \right)$$

For fully connected layers, we retain the “sparsity” measure defined earlier (in Section II) to gauge the results of all related experiments.

The accuracy loss budget (ΔA_j) of a layer j is then calculated as follows:

$$\Delta A_j = \Delta A_{max} \times S_j \quad (3)$$

Once ΔA_j is obtained for all layers, each layer is ranked based on its ΔA_j value. Then, each layer is sequentially pruned by varying the pruning threshold dynamically until the loss in DNN accuracy equals ΔA_j value of the layer. The layers with higher ΔA_j values are pruned first. In case of multiple layers with equal ΔA_j , those layers closer to the input layer are pruned first [38] (Note: In our pruning algorithm, we prune convolutional layers and fully connected layers separately).

C. Healing after pruning

[5], [11] exploit the natural error-healing ability of DNN training process to curtail degradation in the output quality of DNNs after approximation. We also use a similar approach where the sparsified DNNs are trained again to improve DNN testing accuracy.

D. Algorithm description

Based on the above discussed sub-sections, the proposed algorithm (Algorithm 2) systematically develops sparse DNNs using three stages: **Retraining**; **Iterative pruning- testing** and **Healing**.

During **Retraining** (line 1), a pre-trained DNN is re-trained with the cost function comprising both error function and the smoothed LASSO based penalty function.

This is followed by the **iterative pruning-testing** stage (lines 2-26), where the weights falling below the pruning threshold are pruned. For each layer, the pruning threshold is increased until the DNN accuracy loss (ΔA) equals the maximum allowed degradation in DNN accuracy (ΔA_{max}) and s_j for every layer is ascertained (lines 2-16). Then the accuracy loss budget of every layer (ΔA_j) is calculated (lines 17-20) and the layers are ranked based on their ΔA_j values (line 21). Layers with higher ΔA_j values are pruned first (lines 22-26). The pruning threshold for each layer is incremented

dynamically with a step size of Δp until the DNN accuracy loss (ΔA) equals ΔA_j for the layer.

These set of steps enhance the sparsity in the DNNs by allowing higher DNN degradation proportions and higher sequential priority for layers which are likely to contribute more to the DNN sparsity.

Finally, in the **healing** stage (line 27), the DNN (only unpruned connections) is re-trained for M epochs. The cost function in this stage does not contain the LASSO penalty term.

The stages of retraining and healing will definitely cause a rise in the overall training time. However, as this entire process is only a one-time affair, the overhead incurred here is a favourable trade-off for the significant benefits of sparse DNNs during inference.

The experiments performed on the proposed algorithm and results obtained will be discussed in Section VI.

Algorithm 2 Proposed algorithm.

Input: *DNN* (Pre-trained), *TrData*, ΔA_{max} , p_i , Δp
Output: *Sparse DNN*

```

1:  $W, \text{Accu.} = \text{train}(DNN, TrData, N \text{ epochs}, LASSO)$ 

   { $s_j$  calculation}
2: for  $j = 1$  to  $N_l$  do
3:    $w_{temp} = w^j$ 
4:   while  $\Delta A \leq \Delta A_{max}$  do
5:      $p_u = p_i$  {Pruning threshold}
6:     for  $w_i^j \in w^j$  do
7:       if  $|w_i^j| \leq p_u$  then
8:         Prune  $w_i^j$ 
9:          $Sp_j = Sp_j + 1$ 
10:      end if
11:    end for
12:     $p_u = p_u + \Delta p$ 
13:  end while
14:  Calculate  $s_j$ 
15:   $w^j = w_{temp}$ 
16: end for

   { $\Delta A_j$  calculation}
17: for  $j = 1$  to  $N_l$  do
18:    $S_j = s_j / (\sum_{j=1}^{N_l} s_j)$ 
19:    $\Delta A_j = \Delta A_{max} \times S_j$ 
20: end for

   {Layer wise pruning}
21:  $Pru\_Seq = \text{index}(\text{Descending sort}(\{\Delta A_j\}_{j=1}^{N_l}))$ 
22: for  $j$  in  $Pru\_Seq$  do
23:   while  $\Delta A \leq \Delta A_j$  do
24:     Repeat 5 to 12
25:   end while
26: end for

27:  $Sparse\ DNN = \text{train}(DNN, TrData, M \text{ epochs})$ 
28: return Sparse DNN

```

V. STRUCTURED LASSO VARIANTS

Consider a $m \times n$ matrix with b_v being the number of bits required to store each entry in the matrix. The memory requirement to store the entire matrix is $b_v mn$ bits.

Assume that this matrix (weights of a layer) has been sparsified by the conventional LASSO technique with a sparsity of f (fraction of entries which are zeros). A sparse matrix in TensorFlow requires a $1D$ array to store the values (with b_v bits each) and a $2D$ array to store the indices in `INT64` format. Therefore, each non zero entry in the sparse matrix requires b_v and 2×64 bits for storing their values and indices respectively.

For the conventional LASSO to be an efficient technique of reducing memory requirement, the sparse matrix memory requirements must be less than the normal matrix memory requirements, leading to the following inequality:

$$(1 - f)mn(b_v + (2 \times 64)) < b_v mn$$

$$\implies f > \frac{128}{b_v + 128}$$

If the matrix entries are stored in `FLOAT64` format, a minimum of 67% sparsity is required for conventional LASSO to be useful. In case of `FLOAT32` format, it is 80%. Clearly, very high sparsity values are required for the conventional LASSO based pruning to be a useful technique in terms of achieving memory savings on the account of irregularity and indexing overhead it incurs.

This necessitates the use of a smoothed LASSO based structured LASSO penalty term to prune groups of weights in DNNs for achieving structuredness in DNN sparsity. However, structured LASSO penalty function is not purely a L_1 type function. It is an intermediate mode between L_1 and L_2 . Therefore, in comparison to LASSO function, structured LASSO function is expected to generate less sparse DNNs. There is a trade-off between sparsity levels and structuredness in the sparse DNNs as we switch from LASSO to structured LASSO penalty function. The associated cost function in the context of structured LASSO is as follows:

$$C = C_o + \lambda_{fc} P_{fc} + \lambda_{conv} P_{conv}$$

where, λ_{fc} and λ_{conv} are the penalty coefficients associated with Fully Connected (FC) layers and convolutional (conv.) layers respectively. P_{fc} and P_{conv} are the quadratic smoothing function based structured LASSO penalty terms for fully connected layers and convolutional layers respectively. The advantage of using our formulation with smoothed LASSO is that it brings the convolutional and fully connected layers within a common framework. When we want to prune a particular type of layer (either FC or conv.), the penalty coefficient corresponding to the other layer type can be set to zero. These two types of layers differ significantly in their structures and associated bottlenecks. The following subsections carry out structured sparsity study on these two types of layers independently.

Fig. 3. A neuron (in hidden FC layer) with input and output weights grouped into m and n clusters respectively.

A. In Fully Connected layers

[1], [7], [9] had used the smoothed Group LASSO function and [32]–[34], [36] carried out Group LASSO experiments using the LASSO function to prune redundant neurons.

In this paper, we carry out a detailed study on using structured LASSO within the smoothed LASSO framework for fully connected layers and discuss the trade-offs involved. Structured LASSO with different cluster sizes (from pruning individual weights using conventional LASSO to pruning neurons using Group LASSO) are explored. The input and output connections of neurons in a hidden layer are grouped into different clusters. Figure 3 depicts the clustering of input and output weights of a neuron in a hidden layer. Based on the pruning threshold, these clusters are pruned as a whole. When all the input (and/or output) connections of a neuron are grouped into a single cluster, the entire neuron can be pruned as a whole when the pruning criterion is satisfied. This is identical to that of Group LASSO technique proposed in [1].

The associated smoothing function based structured LASSO penalty term (P_{fc}) is as follows:

$$P_{fc} = \sum_j \sum_{\overline{w}_{i,k} \in W_j, \overline{u}_{i,k} \in W_{j+1}} \sum_i (h(\overline{w}_{i,k}) + h(\overline{u}_{i,k})) \quad (4)$$

where, W_j is the set of weights in j^{th} hidden layer, $\overline{w}_{i,k}$ and $\overline{u}_{i,k}$ are the i^{th} input weights cluster and output weights cluster of k^{th} neuron in hidden layer j .

Note that, in case of clustered LASSO, L_2 norm is used instead of absolute value $|\cdot|$ for calculating $h(\cdot)$ and groups of weights as a whole are pruned based on the type of clustered LASSO being used. The (m, n) cluster configuration notation used in this section refers to the number of clusters used for grouping input weights (m) and output weights (n) respectively (refer Figure 3). It is to be noted here that, (1,1) corresponds to Group LASSO.

B. In convolutional layers

A typical convolutional layer comprises of filters, channels (across which the convolution operation is carried out), pooling layers and activation functions. In a convolutional layer, a weight is denoted as follows: $W(n_f, n_c, h, w)$, where n_f , n_c , h , w are the dimensions along the axes of filter, channel, filter height and filter width respectively. Figure 4 depicts an example of a typical convolutional layer. [9], [47] highlight the importance of structured sparsity in convolutional layers

Fig. 4. A typical convolutional layer with n_f , n_c , h , w being the dimensions along the axes of filter, channel, filter height and filter width respectively.

for quicker inference. Several works such as [37]–[41], [43] carried out structured sparsity such as filter or channel pruning using LASSO function. To the best of our knowledge, there have been no works till date on implementation of structured LASSO in convolutional layers using the smoothed LASSO functions. In this paper, we compare various structured LASSO types in the smoothed LASSO framework and discuss the trade-offs involved.

The associated smoothing function based structured LASSO penalty term (P_{conv}) is as follows:

$$P_{conv} = \sum_j \sum_{i, \overline{w}_i \in W_j} h(\overline{w}_i) \quad (5)$$

where, W_j is the set of weights in j^{th} layer and $\overline{w}_i$ is the i^{th} cluster. As in the case of FC layers, here also L_2 norm is used instead of absolute value $|\cdot|$ while calculating $h(\cdot)$ and the cluster of weights are pruned instead of a single weight.

The various types of structured LASSO clusters in the convolutional layers are as follows:

- 1) **Channel wise:** The weights belonging to a channel across all the filters are grouped together. In Equation (5), $\overline{w}_i = w(:, i, :, :)$, $i \in 1$ to N_c (number of channels).
- 2) **Filter- 3D wise:** The weights belonging to a filter across all the channels are grouped together. In Equation (5), $\overline{w}_i = w(i, :, :, :)$, $i \in 1$ to N_f (number of filters).
- 3) **Filter- 2D wise:** The weights belonging to a filter of a channel are grouped together. In Equation (5), $\overline{w}_i = w(j, k, :, :)$, $j \in 1$ to N_f , $k \in 1$ to N_c and $i \in 1$ to $N_f \times N_c$.
- 4) **Row wise:** The weights belonging to a row (along the width) of a filter in a channel are grouped together. In Equation (5), $\overline{w}_i = w(j, k, m, :)$, $j \in 1$ to N_f , $k \in 1$ to N_c , $m = 1$ to H (number of rows (filter height)) and $i \in 1$ to $N_f \times N_c \times H$.
- 5) **Column wise:** The weights belonging to a column (along the height) of a filter in a channel are grouped together. In Equation (5), $\overline{w}_i = w(j, k, :, m)$, $j \in 1$ to N_f , $k \in 1$ to N_c , $m = 1$ to W (number of columns (filter width)) and $i \in 1$ to $N_f \times N_c \times W$.

VI. RESULTS

In this section, we discuss various experiments and their results pertaining to the study of smoothing functions, proposed algorithm and structured LASSO. The simulations have

Fig. 5. Sparsity in hidden layer 1 (H_1), hidden layer 2 (H_2) and output layer (O) of **MNIST architecture II** for different e values with pruning threshold values of 10^{-4} and 10^{-3} .

Fig. 6. Sparsity in fully connected layers FC_1 and FC_2 of **SVHN CNN I** for different e values with pruning threshold values of 10^{-3} and 10^{-2} .

Fig. 7. Sparsity in convolutional layers (C_1 , C_2 and C_3) on **SVHN CNN I** for different e values with pruning threshold values of 10^{-3} and 10^{-2} .

been implemented in the TensorFlow 2.2, using Python 3.5.2 language on Intel i7 CPUs.

We have reported our results on various datasets such as MNIST, SVHN, CIFAR-10 on standard architectures and Imagenette [16] on VGG-16 [15]. We use the Imagenette dataset (10 classes) instead of the full Imagenet (1000 classes) [17] in order to reduce the training time required. Since we start with a VGG-16 network that was pre-trained on the full Imagenet dataset and then re-train it on the reduced set of Imagenette classes, the resulting (baseline) accuracy is in the range of 97%.

A. Choice of smoothing functions

We performed a series of experiments on MNIST architecture II and SVHN CNN I architecture using the various smoothing functions with various maximum error (e) values. The (η, λ) values for MNIST and SVHN networks

are $(0.03, 10^{-3})$ and $(3 \times 10^{-5}, 0.3)$ respectively. Due to convergence issues, the maximum error (e) was not decreased below 10^{-4} . Each of the benchmarks were subjected to conventional LASSO based pruning for two different pruning threshold values. Figures 5-7 show sparsity values obtained. The following conclusions are drawn:

- 1) Figs. 5-7 show the sparsity values as a function of maximum error (e) in different layers of MNIST architecture II and SVHN CNN I for various smoothing functions with two pruning thresholds each. The testing accuracy values (across all smoothing functions and pruning thresholds) are about 96.6% (MNIST architecture II) and 87.8% (SVHN CNN I). The difference in accuracy values between various smoothing functions (for both pruning thresholds) was in the range of $\pm 0.1\%$ for both the networks. Similar sparsity values can be observed across all the smoothing functions, independent of maximum error (e) and pruning threshold values. Therefore, for a fixed e value, we conclude that the choice of smoothing function has a relatively low impact on sparsity. Hence, quadratic smoothing function, which is simplest in terms of computation is the best choice.

Fig. 8. Testing accuracy during LASSO based training of MNIST architecture II for e values of 10^{-4} and 10^{-3} .

- 2) From Figures 5-7, a significant dip in sparsity is observed across all smoothing functions for e values greater than 10^{-3} . This is because, sparsity is expected to fall for e values larger than pruning threshold. Also, at larger e values, smoothed LASSO functions lose their LASSO behaviour, thereby causing the decline in DNN sparsity. Larger and similar sparsity values are observed across all smoothing functions for e values of 10^{-4} and 10^{-3} . Figure 8 shows the variation of testing accuracy with epochs while training MNIST architecture II for $e = 10^{-3}, 10^{-4}$. It can be observed that the network converges faster when e is set at 10^{-3} . This is expected as pushing e to smaller values results in the smoothed LASSO function becoming sharper at origin, resembling the (unsmoothed) LASSO function, thereby slowing down convergence. Given this, we empirically find 10^{-3} to be a suitable value for e and use it for all further experiments.

TABLE VI

COMPARISON OF MAC SPARSITY LEVELS OBTAINED BY QUADRATIC AND HYPERBOLIC SMOOTHING FUNCTIONS BY USING ALGO. 2 ON (A) CIFAR 10 CNN (PRUNING THRESHOLD=0.02) AND (B) IMAGENETTE VGG-16 CNN (PRUNING THRESHOLD=0.01).

Smoothing Function	MAC Sparsity (%)										Accu. (%)
	1	2	3	4	5	6	7	8	9	Total	
Quadratic	31.48	40.21	39.36	42.56	48.33	21.3	17.13	12.05	22.03	39.72	91.22
Hyperbolic	31.48	40.21	39.36	42.56	48.33	21.3	17.13	12.05	22.03	39.72	91.22

(a)

Smoothing Function	MAC Sparsity (%)														Accu.(%)
	1	2	3	4	5	6	7	8	9	10	11	12	13	Total	
Quadratic	6.94	66.92	67.07	53.3	21.64	68.07	32.9	16.91	63.48	19.72	57.18	30.35	31.88	46.78	96.6
Hyperbolic	7	67.34	67.65	54.17	22.17	69.22	33.7	17.54	64.77	20.46	58.45	31.07	32.63	47.62	96.5

(b)

B. Sparsity results based on accuracy loss budget (ΔA_j) for each layer

Table VI compares MAC sparsity values obtained by quadratic smoothing function with those obtained by hyperbolic smoothing function on CIFAR-10 CNN and Imagenette VGG-16 CNN, using the proposed algorithm (Algorithm 2). It can be seen that both the smoothing functions exhibit similar pruning capabilities within the framework of Algorithm 1 also, thereby, further validating the choice of quadratic smoothing function for DNN pruning. Quadratic function is used in all further experiments.

Table VII compares the sparsity results obtained on convolutional layers of CIFAR 10 CNN by using Algorithm 1 with proposed algorithm (Algorithm 2). Algorithm 2 has been implemented using both Sp_j and MSp_j for calculating accuracy loss budget (ΔA_j) values. Algorithm 2 has also been implemented using L_1 norm ([37], [38], [43], [44]), where, the L_1 norm value of weights of a layer is normalized by dividing with number of weights in that layer (similar to that used in [37]). The accuracy loss budget (ΔA_j) values are then assigned proportional to the normalized L_1 norm value.

TABLE VII

COMPARISON OF CIFAR 10 CNN SPARSITY LEVELS OF DIFFERENT LAYERS OBTAINED BY ALGORITHM 1 WITH PROPOSED ALGORITHM (ALGORITHM 2). $\eta = 0.3$, $\lambda = 3 \times 10^{-5}$.

Layer no.	Prop. of weights (%)	Prop. of MAC op. (%)	Sparsity/ MAC sparsity (%)			
			Algo. 1	Algo. 2 using Sp_j	Algo. 2 using MSp_j	Algo. 2 using L_1
1	0.2	1	45	26	31.48	47.11
2	6	34	25	32	40.21	21.91
3	6	7	23	19	39.36	15
4	12.1	14.7	21	37	42.56	27.26
5	24.2	29.4	20	35	48.33	26.2
6	24.2	6.3	19	51	21.3	33.91
7	24.2	6.3	20	52	17.13	25.89
8	3	0.7	19	24	12.05	28.73
9	0.1	0.03	20	17	22.03	27.24
Total sparsity (%)			20.43	42	31.37	27.31
Total MAC sparsity (%)			22.18	34.86	39.72	24.66
Accuracy (%)			91.23	91.22	91.22	91.2

From the sparsity values in Table VII, it can be seen that all variants of Algorithm 2 give larger sparsity values than Algorithm 1. However, Algorithm 2 using L_1 value gives

TABLE VIII

COMPARISON OF IMAGENETTE VGG-16 CNN SPARSITY LEVELS OF DIFFERENT CONVOLUTIONAL LAYERS OBTAINED BY ALGORITHM 1 WITH PROPOSED ALGORITHM (ALGORITHM 2). $\eta = 10^{-3}$, $\lambda = 10^{-3}$.

Layer no.	Prop. of weights (%)	Prop. of MAC op. (%)	Sparsity/ MAC sparsity (%)			
			Algo. 1	Algo. 2 using Sp_j	Algo. 2 using MSp_j	Algo. 2 using L_1
1	0.01	0.56	1.8	6.2	6.94	17.13
2	0.3	12.1	14	50.47	66.92	13.96
3	0.5	6	19.2	30.9	67.07	8.13
4	1	12.1	24.1	59.3	53.3	10.43
5	2	6	27.9	21.6	21.64	39.59
6	4.01	12.1	33.6	33.6	68.07	33.63
7	4.01	12.1	32.9	25.42	32.9	46.39
8	8.02	6	39	30.35	16.91	60.8
9	16.03	12.1	46.5	70.32	63.48	46.49
10	16.03	12.1	45.2	45.2	19.72	19.72
11	16.03	3	41.2	41.2	57.18	41.17
12	16.03	3	39.1	39.1	30.35	16.83
13	16.03	3	41	57.1	31.88	17.74
Total sparsity (%)			40.9	46.7	39.37	31.82
Total MAC sparsity (%)			32.56	43.53	46.78	29.52
Accuracy %			96.9	96.7	96.6	96.8

lower sparsity than the other two types. Among the other two types, using Sp_j in Algorithm 2 gives lower MAC sparsity than using MSp_j . This is because using Sp_j targets those layers with larger number of weights (layers 4,5,6,7) and using MSp_j targets those layers with larger number of MAC operations (layers 2,4,5) during pruning. Therefore, given the energy intensive nature of MAC operations in convolutional layers, it is necessary to use MSp_j while allocating the ΔA_j values to individual convolutional layers.

Table VIII shows similar results on convolutional layers of Imagenette VGG-16 architecture. It can be observed that using Algorithm 2 with MSp_j value has resulted in a considerable increase in total MAC sparsity by targeting those layers with larger number of MAC operations.

C. Structured LASSO

The proposed algorithm (Algorithm 2) has been used for all experiments on structured LASSO.

1) *In fully connected layers:* Table IX shows the sparsity levels obtained by using structured LASSO with various cluster sizes on MNIST architecture II. It is seen that, decrease in cluster size (i.e. moving from Group LASSO to conventional LASSO) results in increase of sparsity in bulkier layers (Hid. layer I & II). This is expected as using the L_1 norm (conventional LASSO) yields sparse solutions than a mixture of $L_1 - L_2$ (structured LASSO variants). However, this trend is not observed in the O/P layer. This is due to the insignificant accuracy budget allocated for O/P layer, as it accounts for only 1% of DNN weights.

Table X shows similar results on FC layers of Imagenette VGG16 CNN. It can be seen that sparsity of bulkier layers (Hidd. layers I & II) is larger for smaller cluster sizes.

TABLE IX

SPARSITY LEVELS ACHIEVED BY STRUCTURED LASSO BASED PRUNING ALGORITHM WITH VARIOUS CLUSTERINGS OF INPUT AND OUTPUT WEIGHTS OF HIDDEN NEURONS ON MNIST ARCHITECTURE II. $p_i = 10^{-4}$, $\Delta p = 10^{-4}$, $\Delta A_{max} = 0.5\%$, $\eta = 0.03$, $\lambda = 10^{-3}$. PROP. REFERS TO THE PROPORTION OF WEIGHTS IN A GIVEN LAYER.

Cluster config.	Sparsity (%)			Accu. %
	Hid. layer I (Prop: 43%)	Hid. layer II (Prop: 56%)	O/P layer (Prop: 1%)	
Group LASSO	0	81.8	72.9	97.67
(2, 2)	5.6	97.2	89.26	97.49
(4, 4)	88.2	99.4	95.3	97.38
(8, 4)	94.7	99.7	95.8	97.16
LASSO	96.6	98.9	70.4	97.33

TABLE X

SPARSITY LEVELS ACHIEVED BY STRUCTURED LASSO BASED PRUNING ALGORITHM WITH VARIOUS CLUSTERINGS OF INPUT AND OUTPUT WEIGHTS OF HIDDEN NEURONS ON FULLY CONNECTED (FC) LAYERS OF IMAGENETTE VGG-16 CNN. $p_i = 10^{-4}$, $\Delta p = 10^{-4}$, $\Delta A_{max} = 0.5\%$, $\eta = 10^{-5}$, $\lambda = 0.06$. PROP. REFERS TO THE PROPORTION OF WEIGHTS IN A GIVEN LAYER.

Cluster config.	Sparsity (%)			Accu. %
	Hid. layer I (Prop: 83%)	Hid. layer II (Prop: 14%)	O/P layer (Prop: 3%)	
Group LASSO	44.6	25.5	14.6	96.6
(2, 2)	45.4	17.4	3.2	96.5
(4, 4)	48.5	47.9	16.9	96.5
(8, 8)	66.5	41.2	29.8	96.62
LASSO	67.6	38.7	20.1	96.8

2) *In convolutional layers:* Table XI shows the MAC sparsity values obtained by using structured LASSO on convolutional layers of SVHN CNN II [10]. It is observed that MAC sparsity of conv. II layer is larger for smaller sizes of groupings. An opposite trend is observed in conv. I layer, which is of little concern as this layer accounts for only 1.18% of MAC operations in convolutional layers. However, it is observed that using filter-2d grouping yields larger MAC sparsity than row and column wise groupings.

Table XII shows similar results on convolutional layers of Imagenette VGG-16 CNN. It can be clearly observed that total sparsity is larger for smaller groupings of structured LASSO. However, as observed in the previous example on SVHN CNN, filter-2d wise grouping gives larger MAC sparsity than row and column wise groupings.

TABLE XI

STRUCTURED LASSO PRUNING ON CONVOLUTIONAL LAYERS OF SVHN CNN II [10]. $p_i = 10^{-4}$, $\Delta p = 10^{-4}$, $\Delta A_{max} = 0.5\%$, $\eta = 10^{-3}$, $\lambda = 0.01$. MAC PROP. REFERS TO THE PROPORTION OF MAC OPERATIONS IN A GIVEN LAYER.

Structured LASSO type	MAC sparsity (%)		Accu.(%)
	Conv. I (MAC prop: 1.18%)	Conv II (MAC prop: 98.82%)	
Channel	0	0	93.06
Filter -3D	68.75	38.89	92.74
Filter - 2D	81.25	85.49	92.92
Row wise	79.58	82.20	92.74
Column wise	43.33	82.86	92.60

VII. CONCLUSION

Based on the detailed study of smoothing functions with a fixed maximum error ϵ value across all the functions, validated using various benchmarks, we determine that the quadratic smoothing function is most suitable to sparsify DNNs. The results on CIFAR-10 and Imagenette datasets demonstrate the efficacy of the proposed layer-wise pruning algorithm in enhancing DNN sparsity by targeting appropriate layers during pruning. The various structured LASSO types for fully connected and convolutional layers have been implemented using the quadratic smoothing function. Results obtained on MNIST, SVHN, Imagenette datasets show that using structured LASSO variants with smaller group sizes result in higher sparsity values than using those with larger group sizes.

APPENDIX

The architecture details of various neural networks used in this paper are as follows:

- 1) **MNIST architecture I:** $SF - L_2$ with ReLU from [3].
- 2) **MNIST architecture II:** 784 (input layer) - 1020 (ReLU without bias) - 1020 (ReLU without bias) - 10 (output layer).
- 3) **SVHN CNN I:** The CNN comprises of 3 convolutional layers with 32, 64, 256 (5×5) filters respectively followed by a fully connected layer of 500 neurons. ReLU activation function was used. The input pixel values have been normalized to [0,1] across each of RGB channels.
- 4) **SVHN CNN II:** Refer [10].
- 5) **CIFAR 10 CNN:** contains 9 convolutional layers (with ReLU), followed by a softmax at the end. The details are: $3 \times 3 \times 3 \times 96c - 3 \times 3 \times 96 \times 96c - 3 \times 3 \times 96 \times 96c - 3 \times 3 \times 96 \times 192c - 3 \times 3 \times 192 \times 192c - 3 \times 3 \times 192 \times 192c - 3 \times 3 \times 192 \times 192c - 1 \times 1 \times 192 \times 192c - 1 \times 1 \times 192 \times 10c - 6 \times 6p - 10$. c refers to convolutional layer, p refers to average pooling.
- 6) **VGG-16:** Refer [15].

TABLE XII

STRUCTURED LASSO PRUNING ON CONVOLUTIONAL LAYERS OF IMAGENETTE VGG-16 CNN. $p_i = 10^{-4}$, $\Delta p = 10^{-4}$, $\Delta A_{max} = 0.5\%$, $\eta = 10^{-3}$, $\lambda = 10^{-3}$. MAC PROP. REFERS TO THE PROPORTION OF MAC OPERATIONS IN A GIVEN LAYER.

Layer no.	MAC prop. (%)	MAC sparsity (%)					
		Channel	Filter -3D	Filter -2D	Row wise	Column wise	LASSO
1	0.58	0	0	0	0.93	0.52	6.94
2	12.05	39.06	21.88	71	25.34	25.19	66.92
3	6.03	0	0	9.72	1.92	1.82	67.07
4	12.05	0	0	48.78	15.75	15.74	53.3
5	6.03	0	0	2.83	8.53	8.19	21.64
6	12.05	0	0	32.33	24.03	24.03	68.07
7	12.05	0	0	31.57	3.56	3.15	32.9
8	6.03	0	0	0.31	14.93	14.43	16.91
9	12.05	0	0	13.37	7.18	6.41	63.48
10	12.05	0	0	11.18	6.6	5.86	19.72
11	3.01	0	0	6.73	3.46	0.54	57.18
12	3.01	0	0	7.9	10.1	0.02	30.35
13	3.01	0	0	12.38	17.62	16.02	31.88
Total MAC sparsity (%)		4.71	2.64	26.57	12.36	10.49	46.78
Testing Accuracy (%)		96.8	96.8	96.5	96.5	96.5	96.6

REFERENCES

- [1] Wang, Jian, et al. "A Novel Pruning Algorithm for Smoothing Feedforward Neural Networks Based on Group Lasso Method." *IEEE transactions on neural networks and learning systems* 29.5 (2018): 2012-2024.
- [2] Mukherjee, Subhadip, and Chandra Sekhar Seelamantula. "Convergence rate analysis of smoothed LASSO." in *Proc. National Conference on Communication (NCC)*. IEEE, 2016.
- [3] Koneru, Basava Naga Girish, and Vinita Vasudevan. "Sparse Artificial Neural Networks Using a Novel Smoothed LASSO Penalization." *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.5 (2019): 848-852.
- [4] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." *Proc. of the IEEE International Conference on Computer Vision*. 2017.
- [5] Venkataramani, Swagath, et al. "AxNN: energy-efficient neuromorphic systems using approximate computing." *Proc. of the international symposium on Low power electronics and design*. 2014.
- [6] Han, Song, et al. "Learning both weights and connections for efficient neural network." *Advances in neural information processing systems*. 2015.
- [7] Zhang, Huaqing, et al. "Feature Selection for Neural Networks Using Group Lasso Regularization." *IEEE Transactions on Knowledge and Data Engineering* v.32, n.4 (2020).
- [8] Deng, Chunhua, et al. "PermdNN: Efficient compressed DNN architecture with permuted diagonal matrices." *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018.
- [9] Wen, Wei, et al. "Learning structured sparsity in deep neural networks." *Advances in neural information processing systems*. 2016.
- [10] Sermanet, Pierre, Soumith Chintala, and Yann LeCun. "Convolutional neural networks applied to house numbers digit classification." *Proc. of the 21st International Conference on Pattern Recognition (ICPR)*. 2012.
- [11] Zhang, Qian, et al. "ApproxANN: An approximate computing framework for artificial neural network." *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015.
- [12] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." *International Conference on Learning Representations (ICLR)*, 2016.
- [13] Nesterov, Yu. "Smooth minimization of non-smooth functions." *Mathematical programming* 103.1 (2005): 127-152.
- [14] Aghasi, Alireza, et al. "Net-trim: Convex pruning of deep neural networks with performance guarantee." *Advances in Neural Information Processing Systems*. 2017.
- [15] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *International Conference on Learning Representations (ICLR)*, 2015.
- [16] Howard, Jeremy. "Imagenette, 2019." URL: <https://github.com/fastai/imagenette>.
- [17] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision* 115.3 (2015): 211-252.
- [18] G. E. Hinton, "Deterministic Boltzmann learning performs steepest descent in weight-space," *Neural Comput.*, vol. 1, no. 1, pp. 143-150, Mar. 1989
- [19] J. Sum, C.-S. Leung, and K. Ho, "Convergence analyses on online weight noise injection-based training algorithms for MLPs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 11, pp. 1827-1840, Nov. 2012.
- [20] J. P.-F. Sum, C.-S. Leung, and K. I.-J. Ho, "On-line node fault injection training algorithm for MLP networks: Objective function and convergence analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 2, pp. 211-222, Feb. 2012
- [21] J. Wang, W. Wei, and J. M. Zurada, "Computational properties and convergence analysis of BPNN for cyclic and almost cyclic learning with penalty," *Neural Networks*, v. 33, pp. 127-135, Sep. 2012.
- [22] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination applied to currency exchange rate prediction," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 1991, pp. 837-841.
- [23] P. May, E. Zhou, and C. W. Lee, "A comprehensive evaluation of weight growth and weight elimination methods using the tangent plane algorithm," *Adv. Comput. Sci. Appl.*, vol. 4, no. 6, pp. 149-156, 2013.
- [24] M. Z. Iskandarani, "A novel approach to system security using derived odor keys with weight elimination neural algorithm (DOK-WENA)," *Trans. Mach. Learn. Artif. Intell.*, vol. 2, pp. 20-31, Apr. 2014.
- [25] J. E. Moody and T. S. Rognvaldsson, "Smoothing regularizers for projective basis function networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 1997, pp. 585-591.
- [26] Z. Chen and S. Haykin, "On different facets of regularization theory," *Neural Comput.*, vol. 14, no. 12, pp. 2791-2846, 2002.
- [27] King, Eric J., and E. E. Swartzlander. "Data-dependent truncation scheme for parallel multipliers." *Asilomar Conference on Signals, Systems and Computers*. Vol. 2. IEEE, 1997.
- [28] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning (chap. 5). Cambridge: MIT press; 2016.
- [29] Alemu, Habtamu Zegeye, et al. "Group $L_{1/2}$ regularization for pruning hidden layer nodes of feedforward neural networks." *IEEE Access* 7 (2019): 9540-9557.
- [30] Wu, Wei, et al. "Batch gradient method with smoothing $L_{1/2}$ regularization for training of feedforward neural networks." *Neural Networks* 50 (2014): 72-78.
- [31] Li, Feng, et al. "Input layer regularization of multilayer feedforward neural networks." *IEEE Access* 5 (2017): 10979-10985.
- [32] Alemu, Habtamu Zegeye, Wei Wu, and Junhong Zhao. "Feedforward neural networks with a hidden layer regularization method." *Symmetry* 10.10 (2018): 525.
- [33] Ochiai, Tsubasa, et al. "Automatic node selection for deep neural networks using group lasso regularization." *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017.
- [34] Scardapane, Simone, et al. "Group sparse regularization for deep neural networks." *Neurocomputing* 241 (2017): 81-89.
- [35] Li, Feng, Jacek M. Zurada, and Wei Wu. "Smooth group $L_{1/2}$ regularization for input layer of feedforward neural networks." *Neurocomputing* 314 (2018): 109-119.

- [36] Zhao, Lei, Qinghua Hu, and Wenwu Wang. "Heterogeneous feature selection with multi-modal deep neural networks and sparse group lasso." *IEEE Transactions on Multimedia* 17.11 (2015): 1936-1948.
- [37] Salama, Abdullah, et al. "Prune Your Neurons Blindly: Neural Network Compression through Structured Class-blind Pruning." *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.
- [38] Li, Hao, et al. "Pruning filters for efficient convnets." *International Conference on Learning Representations (ICLR)*, 2017.
- [39] Yang, Chen, et al. "Structured Pruning of Convolutional Neural Networks via L1 Regularization." *IEEE Access* (2019): 106385-106394.
- [40] Wang, Zongyue, et al. "Pruning Blocks for CNN Compression and Acceleration via Online Ensemble Distillation." *IEEE Access* (2019): 175703-175716.
- [41] Wang, Huan, et al. "Structured Pruning for Efficient Convolutional Neural Networks via Incremental Regularization." *IEEE Journal of Selected Topics in Signal Processing* (2019).
- [42] Aketi, Sai Aparna, et al. "Gradual Channel Pruning while Training using Feature Relevance Scores for Convolutional Neural Networks." *IEEE Access* 8 (2020): 171924-171932.
- [43] Roy, Sourjya, et al. "Pruning filters while training for efficiently optimizing deep learning networks." *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020.
- [44] Hubens, Nathan, et al. "An Experimental Study of the Impact of Pre-Training on the Pruning of a Convolutional Neural Network." *Proc. of the 3rd International Conference on Applications of Intelligent Systems*. 2020.
- [45] Liu, Yunfeng, Huihui Kong, and Peihua Yu. "Automatic Compression Ratio Allocation for Pruning Convolutional Neural Networks." *Proc. of the 3rd International Conference on Vision, Image and Signal Processing*. 2019.
- [46] Anwar, Sajid, Kyuyeon Hwang, and Wonyong Sung. "Structured pruning of deep convolutional neural networks." *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017): 1-18.
- [47] Rumi, Masuma Akter, et al. "Accelerating Sparse CNN Inference on GPUs with Performance-Aware Weight Pruning." *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 2020.
- [48] Reed, Russell. "Pruning algorithms- a survey." *IEEE transactions on Neural Networks* 4.5 (1993): 740-747.
- [49] Ma, Rongrong, et al. "Transformed L1 regularization for learning sparse deep neural networks." *Neural Networks* 119 (2019): 286-298.
- [50] Bach, Sebastian, et al. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." *PLoS ONE* 10.7 (2015): e0130140.
- [51] Horowitz, M. "Computing's energy problem (and what we can do about it)," *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014.