Structured Pruning of Deep Convolutional Neural Networks

SAJID ANWAR, KYUYEON HWANG, and WONYONG SUNG, Seoul National University, Republic of Korea

Real-time application of deep learning algorithms is often hindered by high computational complexity and frequent memory accesses. Network pruning is a promising technique to solve this problem. However, pruning usually results in irregular network connections that not only demand extra representation efforts but also do not fit well on parallel computation. We introduce structured sparsity at various scales for convolutional neural networks: feature map-wise, kernel-wise, and intra-kernel strided sparsity. This structured sparsity is very advantageous for direct computational resource savings on embedded computers, in parallel computing environments, and in hardware-based systems. To decide the importance of network connections and paths, the proposed method uses a particle filtering approach. The importance weight of each particle is assigned by assessing the misclassification rate with a corresponding connectivity pattern. The pruned network is retrained to compensate for the losses due to pruning. While implementing convolutions as matrix products, we particularly show that intra-kernel strided sparsity with a simple constraint can significantly reduce the size of the kernel and feature map tensors. The proposed work shows that when pruning granularities are applied in combination, we can prune the CIFAR-10 network by more than 70% with less than a 1% loss in accuracy.

CCS Concepts: • **Computing methodologies → Machine learning approaches;**

Additional Key Words and Phrases: Deep convolutional neural networks, structured pruning, feature map pruning, intra-kernel strided sparsity

ACM Reference Format:
Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. J. Emerg. Technol. Comput. Syst. 13, 3, Article 32 (February 2017), 18 pages.
DOI: http://dx.doi.org/10.1145/3005348

1. INTRODUCTION

Large, deep Convolutional Neural Networks (CNN) have been successfully applied to diverse classification problems including speech and image recognition [Simonyan and Zisserman 2014; Krizhevsky et al. 2012; Hinton et al. 2012]. These networks can produce state-of-the-art results at a high computational cost. For resource-limited machines and real-time applications, it is important to learn the unknown function with a reduced complexity network. Large networks are capable of learning complicated problems but may overfit on the training set. On the other hand, small networks that demand low computational cost usually have limited learning capabilities. It is therefore of prime importance to design high-performance low-complexity neural networks. This

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A1A10056051).
Authors' addresses: S. Anwar, K. Hwang, and W. Sung, Department of Electrical Engineering and Computer Science, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea; emails: {sajid, khwang}@dsp.snu.ac.kr, wysung@snu.ac.kr.

DOI: http://dx.doi.org/10.1145/3005348

goal can be achieved by applying the pruning technique to high-performance large networks, where pruning reduces the computational cost. Furthermore, these lightweight networks can be implemented using only on-chip memory for energy savings because frequent DRAM accesses consume much energy. Pruning induces sparsity in a network and can be categorized as structured and unstructured. Unstructured pruning does not follow a specific geometry or constraint. In most cases, this technique needs extra information to represent sparse locations. It leads to irregular sparsity, which is difficult to exploit for efficient computation. On the other hand, structured sparsity places non-zero parameters at well-defined locations. This kind of constraint enables modern CPUs and Graphics Processing Units (GPUs) to easily exploit computational savings.

Network pruning has been studied by several researchers [Han et al. 2015a, 2015b; Castellano et al. 1997; Collins and Kohli 2014; Stepniewski and Keane 1997; Reed 1993]. The works of Han et al. [2015a, 2015b] have shown that a much bigger portion of weights can be set to zero with minimum or no loss in performance. They train a network with an additional L1/L2 loss function on the weights and gradually prune it. If the weight of a connection is less than a threshold, the connection is dropped. The authors in Han et al. [2015b] further extend this work by quantizing the finally pruned network [Han et al. 2015a]. However, both works have to explicitly locate non-zero weights with sparse representation. Conventionally, sparse representation uses the Compressed Sparse Row/Compressed Sparse Column (CSR/CSC) format, which represents m non-zero numbers with $2m + n + 1$ numbers, where n represents the number of rows or columns. The work of Han et al. [2015a] shows that half of the AlexNet memory space is required for storing the indices of the non-zero parameters. This also doubles memory accesses because each weight fetch now becomes an *(index, weight)* pair. Our proposed work does not demand such extra representation.

In this work, we explore feature map and intra-kernel sparsities as a means of structured pruning. In the feature map pruning, all incoming and outgoing weights to/from a feature map are pruned. The intra-kernel sparsity prunes weights in a kernel. The kernel-level pruning is a special case of intra-kernel sparsity with 100% pruning. These pruning granularities can be applied in various combinations and different orders. The proposed work contains the following contributions:

—We introduce structured pruning at various granularities for maximum pruning benefits. The pruned networks are easily accelerated with very simple sparse representation. Feature map pruning reduces the width of a convolution layer and directly produces a low-complexity network.
—We propose Intra-Kernel Stride Sparsity (IKSS), which can benefit from convolution unrolling [Chellapilla et al. 2006; Chetlur et al. 2014], where convolutions are efficiently implemented as matrix-matrix multiplications by unrolling the feature maps and kernels.
—The proposed work selects pruning candidates with a particle filtering approach in which each particle simulates a possible set of connections or pruning mask. Least likely connection combinations are pruned while the remainder survive the iterative pruning process.

The rest of this article is organized as follows: Section 2 briefly introduces CNN and discusses pruning granularities. Section 3 outlines the pruning criterion, the hybrid evolutionary particle filter, and the activation sum voting scheme. Experimental results and discussions are provided in Section 4 for each pruning granularity and their combinations. Section 5 compares this work with related woks, and Section 6 discusses future research directions and concludes this article.

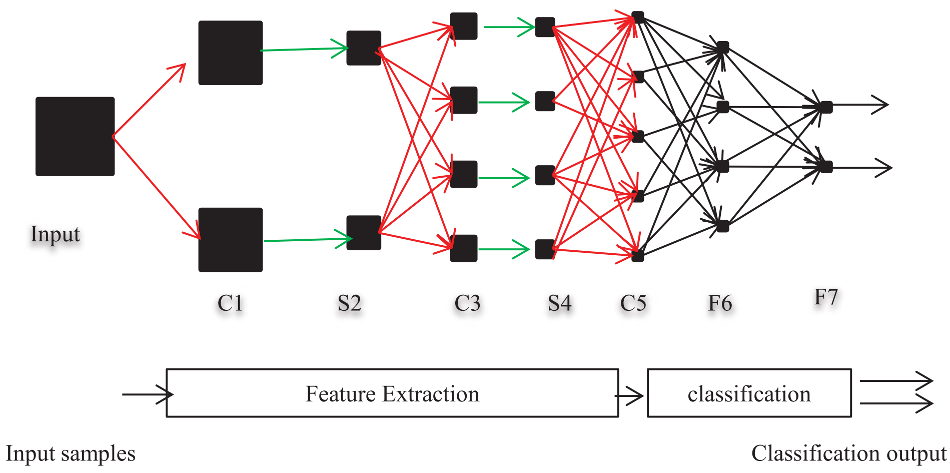


Fig. 1. A convolutional neural network with eight layers is shown here. Layers C1, C3, and C5 are the convolution layers, while S2 and S4 constitute the two pooling layers. F6 and F7 are the two fully connected layers. The convolution and pooling connections are shown in red and green, respectively.

2. PRUNING CONVOLUTIONAL NEURAL NETWORKS

In this section, the CNN network is briefly introduced in the context of pruning. The network can be pruned at various granularities, which are outlined here. Furthermore, we present the proposed IKSS sparsity and its impact on reducing the dimensions of matrices in convolution unrolling.

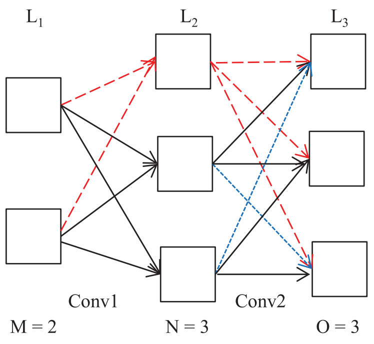
2.1. Convolutional Neural Network

A CNN [LeCun et al. 1998] employs more diverse layer types than a fully connected Deep Neural Network (DNN). It has convolution, pooling, and fully connected layers. The convolution and pooling layers act as feature extractors. At the rear end, near the output layer, a CNN employs fully connected layers for classification. A sample CNN is shown in Figure 1.

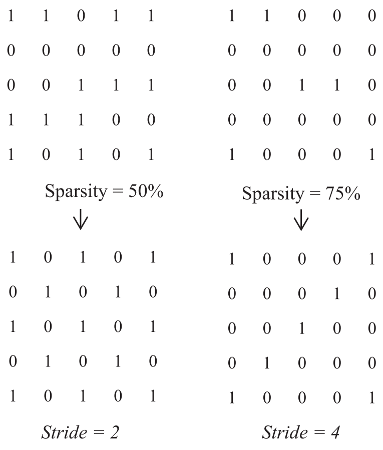
The convolution layer convolves the $k \times k$ kernels with n feature maps of the previous layer. If the next layer has m feature maps, then $n \times m$ convolutions are performed and $n \times m \times (H \times W \times k \times k)$ Multiply-Accumulate (MAC) operations are needed, where W and H represents the feature map width and height of the next layer. Thus, the convolution layers demand a large number of arithmetic operations in many cases. Furthermore, the default memory access pattern of convolution layers is not computationally friendly [Strigl et al. 2010]. It is therefore highly desirable to reduce the complexity of convolution layers. In the literature, there have been various works to reduce the computational complexity of convolution layers. The work of Chellapilla et al. [2006] and Chetlur et al. [2014] converts convolutions into matrix-matrix multiplications and speeds up the computation by 3–4 times [Chellapilla et al. 2006]. However, redundant data and kernels storage incurs its own cost of extra memory usage. Our work proposes to reduce this complexity with structured pruning. We show that the proposed intra-kernel pruning is helpful in reducing the size of matrices in convolution unrolling [Chellapilla et al. 2006]. The next section discusses this in further detail.

2.2. Pruning Granularities

Taking computational advantage by using randomly scattered unstructured sparsity in a network is very difficult. It demands many conditional operations and extra representation to denote the indices of zero or non-zero parameters. In this subsection,



(a) Feature map and kernel wise pruning



(b) Intra-kernel strided pruning

Fig. 2. (a) The red dashed line shows feature map pruning. When we prune all the incoming kernels to a feature map, all the outgoing kernels are also pruned. The blue dotted line depicts pruning $k \times k$ kernels. (b) Shows intra-kernel sparsity for both structured and unstructured cases. Kernel-level pruning (blue dotted line) is a special case of intra-kernel pruning, used when the kernel level sparsity rate is 100%.

we describe three levels of structured pruning: feature map, kernel, and IKSS pruning. Generally, the convolution layers of a non-pruned network employ fully connected convolution connections. In Figure 2(a), the layers L1, L2, and L3 contain 2, 3, and 3 feature maps, respectively. The number of convolution connections between L1 and L2 is $2 \times 3 = 6$ and that between the L2 and L3 is $3 \times 3 = 9$. Each feature map in L2 has a $k \times k$ convolution connection from each feature map in L1. Thus, the pruning exploiting the largest granularity is deleting one or more feature maps. If a feature map in a layer is removed, all the incoming and outgoing kernels are also pruned. Figure 2 shows 5 pruned kernels with red dashed lines. Considering the configuration in Figure 2(a), the 2-3-3 architecture is reduced to 2-2-3. The next level pruning is intra-kernel pruning, in which each kernel represents one whole convolution. The kernel level pruning is depicted with blue dotted lines in Figure 2(a). Figure 2(b) shows examples of intra-kernel sparsity for both the structured and unstructured cases. Kernel-level pruning (blue dotted line) is a special case of intra-kernel pruning, used when the sparsity ratio is 100%.

The finest pruning granularity is intra-kernel sparsity, which forces some weights into zero. In the previous works, the intra-kernel level pruning is usually conducted by zeroing small-valued weights [Han et al. 2015a, 2015b]. We particularly explore intra-kernel level pruning using sparsity at well-defined locations (IKSS). Figure 2(b) depicts this idea. The starting index (offset) for the first non-zero element is randomly assigned in the range of $[0, stride - 1]$. Therefore, the IKSS associates an offset as the starting index and the stride size with each kernel. The stride size is also randomly sampled and depends on how well pruning proceeds. In our experiments, we start with a stride size of two. If, during the gradual iterative pruning, the pruned network has similar or better performance when compared with the unpruned network, we use bigger strides and prune the connections more aggressively (use higher pruning ratios). We monitor how well the pruning proceeds after each prune-retrain step. If the last pruning increases the Misclassification Rate (MCR) by more than a small tolerance number, then we decrease the intra-kernel pruning rate and stride size. In normal pruning mode, a majority of the kernels use small strides. The evolutionary particle filter evaluates several possible combinations of the offset, stride, and feature maps and selects the best combination based on likelihood. In this way, we induce sparsities in the network. The retraining procedure uses the finally selected pruning mask. We found this mask using an evolutionary approach and the validation dataset.

The computational complexity of a network is mainly decided by its depth, width, and connectivity pattern. The proposed work does not affect the depth of the network. However, both IKSS and feature map alter the connectivity pattern. Furthermore, feature map pruning directly reduces the width of a convolution layer. Therefore, it is quite straightforward to exploit the computational savings of this technique. This lighter network can then be implemented with convolution unrolling [Chellapilla et al. 2006], FFTs [Mathieu et al. 2013], or in conventional ways. The IKSS can further increase the pruning ratio. As mentioned, the IKSS associates an offset and a stride with each kernel. These two numbers are enough to locate the non-zero weights. We further show that if we constrain each outgoing convolution connection from a source feature map to have similar stride and offset, it will result in only two extra parameters (offset, stride) per feature map. In this way, the IKSS can be computationally exploited to avoid MAC operations with zeroed weights. The convolution layer can be computed as matrix-matrix multiplication by unrolling the source layer feature maps and kernels [Chellapilla et al. 2006; Chetlur et al. 2014]. Figure 3(a) explains this idea in greater detail. Figure 3(b) shows how the proposed IKSS and a simple constraint can reduce the size of feature and kernel matrices. The constraint bounds each outgoing convolution kernel from the source feature maps to have the same stride and offset. The constraint is shown with similar background colors for kernels. These pruning granularities have the potential to bridge the gap between pruning and its computational advantages.

3. PRUNING CANDIDATE SELECTION

While pruning a network, it is important to select good pruning candidates. The pruning candidate selection criterion must identify less important and redundant connections. Exhaustive search to find the best pruning patterns takes too much time in most cases. We therefore propose two heuristics: the particle filter and the activation sum voting. Both schemes are elaborated and compared in this section.

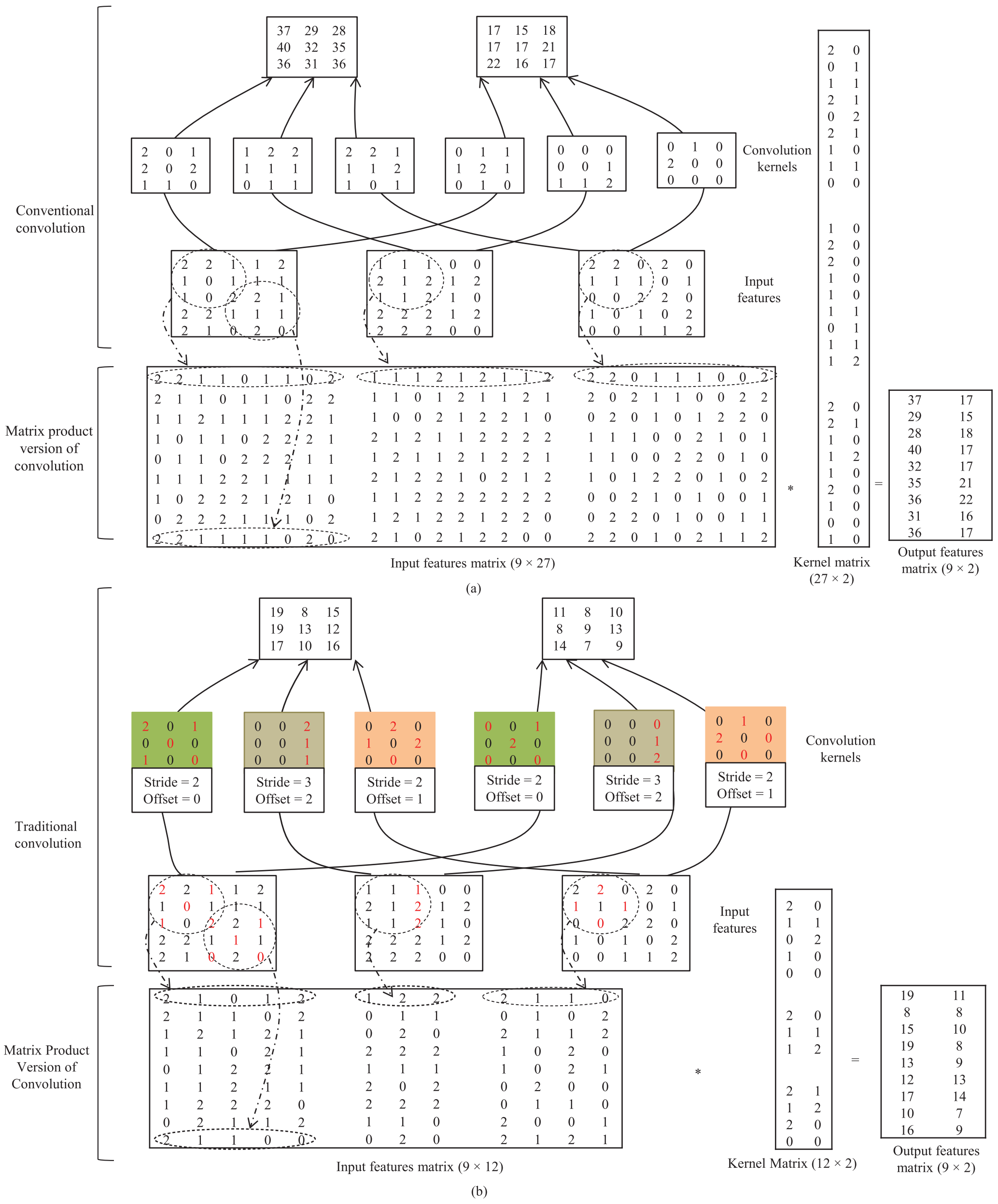
3.1. Evolutionary Particle Filter

The pruning process needs to select less important connection combinations. These connections, when pruned, produce the least adversarial effects on network performance, which can be compensated with retraining. Note that considering all the pruning patterns is too complex in most cases. In this work, we propose to locate pruning candidate connections with the sequential Monte Carlo (SMC) method also known as particle filters [Gordon et al. 1993]. Particle Filtering (PF) finds its applications in several fields [Arulampalam et al. 2002; Nummiaro et al. 2003; Nakamura et al. 2009; Carpenter et al. 1999]. With a set of weighted particles, the PF represents the filtering distribution [Vermaak et al. 2003]. A particle filter is usually applied to the system model shown in Equations (1) and (2).

$$\mathbf{x_k} = \mathbf{f}(\mathbf{x_{k-1}}) + \mu_\mathbf{k} \tag{1}$$

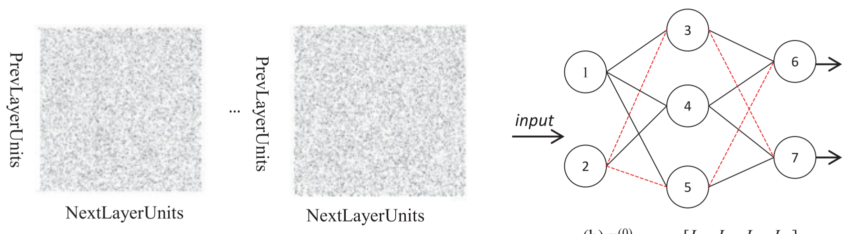
$$\mathbf{z_k} = \mathbf{h}(\mathbf{x_k}) + \mathbf{v_k}, \tag{2}$$

where k shows the time step, $\mathbf{z}$ represents the observation vector, $\mathbf{v}$ shows the observation noise, and μ is the process noise. The state vector is represented by $\mathbf{x}$, and an example is shown in Figure 4(b). The observation function is represented by $h()$, whereas $f()$ represents the transition function. When pruning n connections, the possible combinations are on the order of $\mathcal{O}(2^n)$, which means that exhaustive search is not



Conventional convolution
Convolution kernels
Input features
Matrix product version of convolution
Input features matrix (9 × 27)
Kernel matrix (27 × 2)
Output features matrix (9 × 2)
(a)
Traditional convolution
Stride = 2
Offset = 0
Stride = 3
Offset = 2
Stride = 2
Offset = 1
Matrix Product Version of Convolution
Input features matrix (9 × 12)
Kernel Matrix (12 × 2)
Output features matrix (9 × 2)
(b)

Fig. 3. (a) An example of the convolution unrolling idea introduced in Chellapilla et al. [2006] and Chetlur et al. [2014]. (b) Our proposed idea constrains each outgoing convolution connection for a source feature map to have the same stride and offset. The offset shows the index of first non-zeroed weight. The constraint is shown with similarly colored background squares. This significantly reduces the size of both features matrix and kernel matrix. The first 9 columns in row 1 of the input feature matrix changes from [2 $\underline{2}$ 1 $\underline{1}$ 0 $\underline{1}$ 1 $\underline{0}$ 2] to [2 1 0 1 2] with the underlined elements pruned. Only the red elements in the feature maps and kernels survive, and the rest are pruned. For this example, the size of the feature matrix is reduced from 9 × 27 to 9 × 12 and the kernel matrix size is reduced from 27 × 2 to 12 × 2.



(a) Example of pruning candidates paths

(b) $\mathbf{x}^{(0)}_{\text{inp-h1}} = [I_{13}, I_{14}, I_{15}, I_{24}]$
$\mathbf{x}^{(0)}_{\text{h1-out}} = [I_{36}, I_{46}, I_{47}, I_{57}]$

Fig. 4. (a) The dotted matrices as pruning masks for weights between two layers. The $\mathbf{x}$ state vector represents the inverse of the pruning mask. Nonexisting entries in the state vector are pruned. In (b), one example of the state vector is provided. The circles represent the neurons, while I_{ij} shows the index of the weight going from neuron i to j. The red dotted connections are pruned while the black solid lines survive the pruning process.

feasible in most cases. With N particles, we simulate several possible connection combinations. The state vector contains a set of possible connections through which the inputs are forward-propagated. Thus, the likelihood computation is simulated with pruning masks. The trained network is used as the observation function, which is noisy because the classification error rate is greater than 0%. Thus, each particle simulates a set of connections as the possible candidate for the most likely path through the network. The likelihood to each particle is assigned based on the MCR on the validation set. In this way, importance weight is computed by $1-MCR$ and shows the likelihood of keeping the connection. Thus, connections with high importance survive while the rest are pruned. The MCR likelihood criterion compares the network-assigned label with the true one, which, through iterative pruning, guides the network to learn the true labels. Once all particles are assigned probabilities, we construct the Cumulative Distribution Function (CDF) and resample with Sequential Importance Resampling (SIR) [Arulampalam et al. 2002]. The transition function $f()$ is simulated by perturbing the pruning mask.

Due to the finite number of samples, PF suffers from degeneracy and impoverishment problems when the highly likely particles replace the less likely ones [Li et al. 2014]. Kwok et al. suggested the Evolutionary Particle Filter (EPF) and proposed a hybrid approach where a Genetic Algorithm (GA) is combined with PF to solve these problems. The authors argue that particles are similar to chromosomes, and survival of the fittest has equivalence to the resampling algorithm [Kwok et al. 2005]. With the hybrid approach, the aim is to increase the fitness of the whole population. An important merit of the GA-augmented approach is that it not only maintains the highly likely genes in its chromosome but also redefines particles in the less likely region. EPF reduces computational cost because it requires fewer particles than do conventional particle filters [Kwok et al. 2005]. This property suits us because it will reduce the cost of finding pruning candidates. We also believe that SMC techniques have more potential usages in exploring the network parameter space. One usage can be finding class-specific routes in a DNN.

Considering the case of feature map pruning, we explain here the EPF approach to candidate selection with N particles. We represent each particle with $P[i]$ and the state vector as $\mathbf{x}^{(i)}$. Suppose that we are pruning a layer L with F non-pruned feature maps. Let's further suppose that the current pruning rate is denoted with pr. We compute the dimension d of the state vector with $d = F \times (1 - pr)$. In order to make the likelihood computation connections count-invariant, the state vectors of all the particles have the same dimension d for a given value of pr. Let's consider a simple example where F is

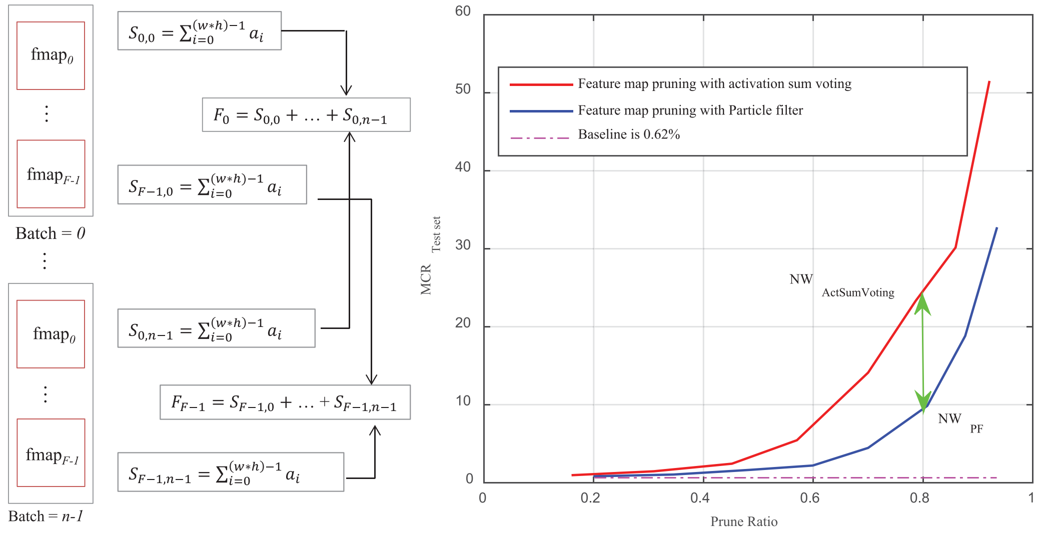


Fig. 5. The activation sum voting and its comparison with the PF-based method. Suppose there are F feature maps with w, h dimensions and rectified linear unit as an activation function. We compute the sum for each feature map across the whole mini-batch of size n on the validation set. Experimental results are shown in the plot on the right side of the figure with the MNIST [LeCun et al. 1998] dataset. The network architecture is $1 \times 16(C5) - MP2 - 32(C5) - MP2 - 64(C5) - 120 - 10)$. It can be observed that, for similar pruning ratios, the particle filter-based method selects less adversarial pruning candidates than the simple method.

10, the pruning rate is 25%, and N is 3. The d in this case will be $\lfloor 10 \times (1 - 0.25) \rfloor = 7$. The state vectors $\mathbf{x}^{(i)}$ for each particle are randomly assigned and may be as follows: $\mathbf{x}^{(1)} = [10\ 2\ 9\ 7\ 1\ 3\ 6]$, $\mathbf{x}^{(2)} = [10\ 3\ 2\ 7\ 4\ 5\ 9]$, $\mathbf{x}^{(3)} = [2\ 5\ 1\ 8\ 6\ 7\ 3]$. Each particle is only allowed to compute the likelihood with the set of feature maps listed in its state vector. Other feature maps not included in the set are zeroed. For example, for the particle $P[3]$, the pruning mask is [1 1 1 0 1 1 1 1 0 0], where the feature maps [4 9 10] are zeroed. In this way, likelihood is computed for each particle. These N particles then undergo the GA-based SIR explained in Section 2.3. Thus, EPF helps reduce the adversarial effect of pruning.

There can be a concern regarding the computational cost of this approach. Two reasons, outlined here, mitigate this concern. First, likelihood computation only uses the feed-forward computation with the validation set. The validation set is 10–20% of the training set and is thus not very large. Second, the hybrid approach (GA + particle filter) in particle filtering allows us to select the pruning candidate with few particles. In all our experiments, we used not more than 100 particles. The next subsection introduces a simple pruning candidate selection scheme and compares it with the EPF- based method.

3.2. Activation Sum Voting

We propose a simple and low-complexity scheme called activation sum voting for feature map pruning. This criterion is inspired by max pooling, in which the magnitude of a neuronal output determines its importance. To the best of our knowledge, this method has not been reported earlier and is depicted in Figure 5. Each feature map is summed across the whole mini-batch on the validation set, and the minimum summation feature maps are pruned. Experimental evaluations for the two methods are reported for the MNIST dataset [LeCun et al. 1998] in Figure 5. The network is first

Table I. Specifications of the Three CIFAR-10 Networks

Network	Architecture	Baseline MCR(%)
CNN_{small}	$32C5$-MP2-$32C5$-AP2-$64C5$ -$10Softmax$	25.71
CNN_{large}	2x128C3-MP2-2x128C3-MP2-2x256C3-256FC-10Softmax	16.6
$CNN_{verylarge}$	2x128C3-MP2-2x256C3-MP2-2x256C3-1x512C3-1024FC-1024FC-10Softmax	9.41

trained to the baseline MCR of 0.69%. We then select pruning candidates for various pruning ratios. This plot does not consider the effect of retraining the pruned networks. The NW_{PF} and $NW_{ActSumVoting}$ show the pruned network obtained with PF and the activation sum voting method, respectively. The pruning plot shows that the particle PF-based method performs better especially at higher pruning ratios by selecting less adversarial pruning candidates. Further in the plot, a green arrow shows the difference between the classification performance of NW_{PF} and $NW_{ActSumVoting}$. Network retraining may recover some of the pruning losses. However, NW_{PF} starts better than the $NW_{ActSumVoting}$. Thus, we show that the higher computational complexity of the PF-based method is justified because it selects better pruning candidates.

4. EXPERIMENTAL RESULTS

The pruning process starts with a pretrained network. The EPF-guided pruning process may degrade network performance, which is compensated for by retraining. Retraining is important to keep network performance closer to the non-pruned network. In the spirit of Han et al. [2015b], we train the network with L2 regularization in the absence of batch normalization [Ioffe and Szegedy 2015]. We consider the pruning limit for each layer depending on its parameter count and learning capacity. Usually, the first convolution layer has fewer parameters than the following layers. As well, it directly operates on the input layer and is therefore more sensitive to pruning [Han et al. 2015b]. We present experimental results with the CIFAR-10 and SVHN datasets of Krizhevsky [2009] and Netzer et al. [2011]. The CIFAR-10 dataset consists of a 10-class classification problem [Krizhevsky 2009]. The dataset includes samples from 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The training set consists of 50,000 RGB samples. Test set contains 10,000 samples. Each sample has 32×32 resolution. During training and pruning, we use the Stochastic Gradient Descent (SGD) with a mini-batch size of 128 and RMSProp [Tieleman and Hinton 2012].

We present experimental results with three networks on the CIFAR-10 dataset: CNN_{small}, $CNN_{large,}$ and $CNN_{verylarge}$. Details about these three networks are provided in Table 1, and the network architectures are represented with alphanumeric strings as reported in Courbariaux et al. [2015]. The $(2 \times 128C3)$ represents two convolution layers, with each having 128 feature maps and 3×3 convolution kernels. $MP2$ and $AP2$ represents 2×2 max and average pooling layers, respectively. The CNN_{small} is trained with the original CIFAR-10 dataset without any preprocessing and data augmentation. The CNN_{large} is inspired by Courbariaux et al. [2015] and is trained with batch normalization [Ioffe and Szegedy 2015]. We preprocess the original CIFAR-10 dataset with global contrast normalization followed by ZCA whitening. The validation set uses 10,000 training samples. Batch normalization accelerates training by normalizing each layer input and also reduces the impact of weight scale [Ioffe and Szegedy 2015], [Courbariaux et al. 2015]. The latter property suits pruning because we prefer small weights [Han et al. 2015b]. CNN_{small} has three convolution layers and is smaller than $CNN_{large,}$ which has six convolution layers. Furthermore, CNN_{small} applies a bigger receptive field of 5×5 convolution kernels whereas CNN_{large} has 3×3 kernels.

The pruning process is guided by EPF, which locates and identifies pruning candidates. The EPF evaluates several possible pruning candidates and selects the one that

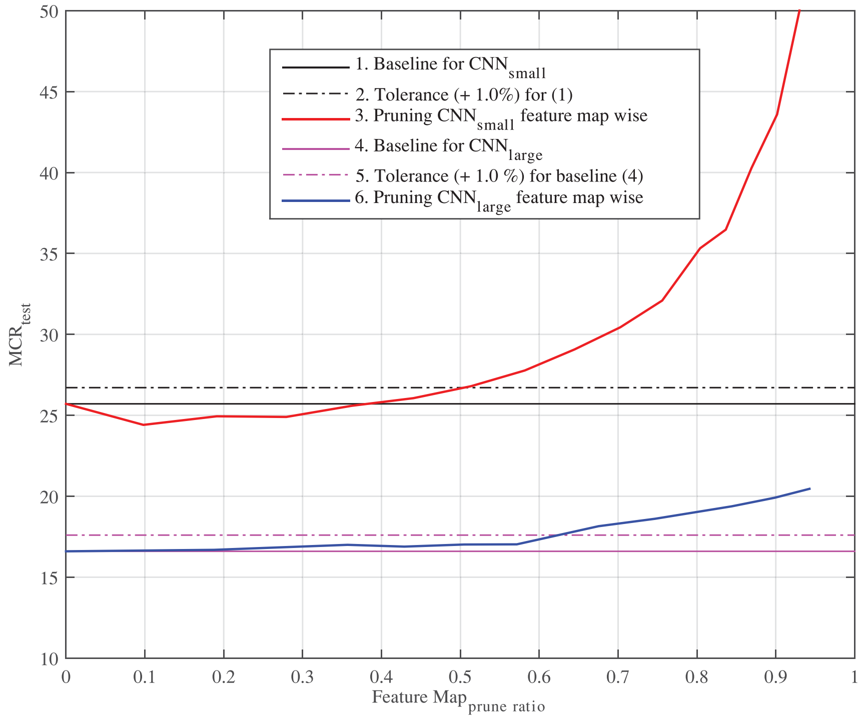


Fig. 6. This plot shows feature map pruning results for CNN_{small} and CNN_{large}. This pruning granularity can be induced at higher rates for CNN_{large} than CNN_{small} due to bigger width. The pruning ratios are computed for convolutional layers. It is important to mention that the pruned networks are retrained to compensate for loss in performance.

has the least adversarial effect on the network when pruned. The count of potential pruning candidates is dependent on the pruning rate and granularity. Smaller pruning rates and the IKSS constraint of similar stride and offset limit this count. Furthermore, retraining the network may compensate for loss in accuracy due to pruning.

4.1. Feature Map Pruning

In this section, we present feature map pruning guided by EPF. We discuss experimental results with CNN_{small} and CNN_{large}. We do not prune the first convolution layer in both networks due to fewer convolution connections and higher pruning sensitivity. Our experiments show that 100 particles are enough to select the pruning candidates. The pruning plots are provided in Figure 6. By feature map pruning the CNN_{small}, under a tolerance of 1% increase in MCR, the numbers of convolution connections are reduced to $672 (32 \times 21 = 672)$ and $798\ (21 \times 38 = 798)$ in the 2nd and 3rd convolution layers, respectively. This means that 35% $(= (1024 - 672)/(32 \times 32))$ and 62%$(= (2048 - 798)/(32 \times 64))$ of the convolution connections are dropped with less than a 1% increase in MCR. Increasing the feature map level sparsity beyond this point increases the MCR by more than 1%.

Next, the CNN_{large} is feature map pruned. At a 57% pruning rate, the CNN_{large} is reduced to $(128C3 - 83C3)$-MP2-$(83C3 - 84C3)$-MP2-$(166C3 - 166C3)$-$256FC$-$10Softmax$ with less than a 0.6% increase in MCR. Comparing the feature map pruning of the two networks, we observe that CNN_{small} cannot be pruned as much as the CNN_{large} due to modest width. The CNN_{large} is wider and has more room for

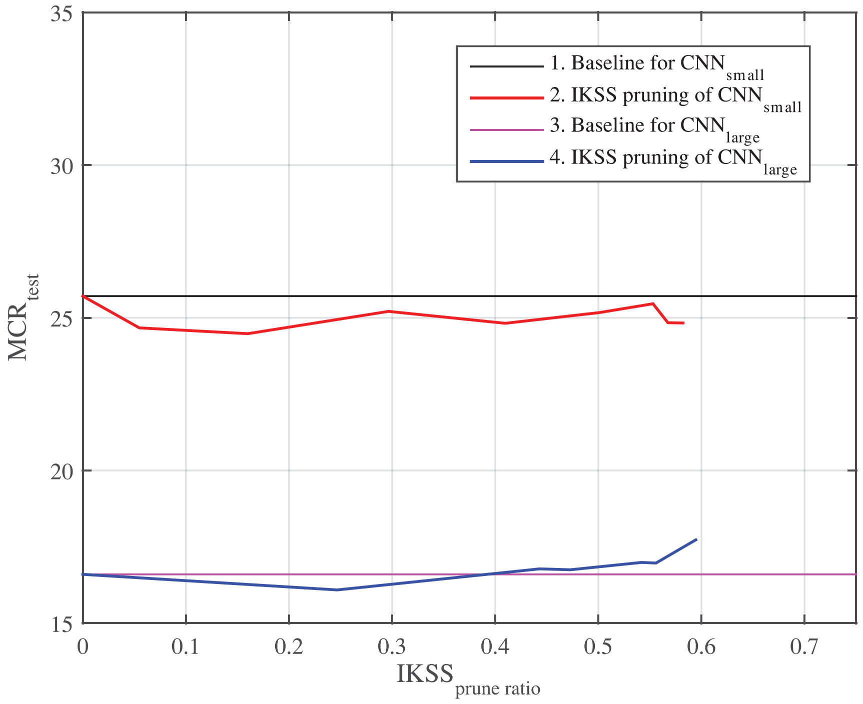


Fig. 7. This plot shows intra-kernel pruning results for CNN_{small} and CNN_{large}. The CNN_{small} uses 5×5 kernels while CNN_{large} uses 3×3 kernels. The IKSS can be induced at higher rates for CNN_{small} than CNN_{large} due to bigger kernels. The pruning ratios are computed for convolutional layers only. It is important to mention that the pruned networks are retrained to compensate for loss in accuracy.

feature map pruning. Feature map pruning the CNN_{large} beyond 62% decreases the network's baseline performance by more than 1%.

4.2. Intra-Kernel Pruning

In this section, we present IKSS for CNN_{small} and CNN_{large} and discuss the experimental results shown in Figure 7. It can be observed in Figure 7 that the NW_{small} can achieve more intra-kernel sparsity than NW_{large}, which cannot be pruned at higher rates. This is attributed to the difference in kernel sizes. Although NW_{large} is deeper and wider, it uses a smaller kernel size of 3×3. Therefore, we infer that IKSS is more related to the kernel size than to the width of the network.

Next, we discuss the selection of the stride and the offset for IKSS. We introduce a new term for each feature map, OKFF, which stands for the set of outgoing kernels from a feature map. Thus, each feature map has one OKFF. As mentioned, all kernels in an OKFF have the same stride and offset. We further suppose that the feature maps in layer L have F non-pruned OKFFs and that the pruning rate is pr. The EPF has to select the pruning candidates with N particles. The state vector dimension is represented by $d = F$ and contains the stride size for each OKFF. The OKFFs associated with $\lfloor (F \times (1 - pr)) \rfloor$ are not pruned and are assigned zero offset and the stride of 1. If F is 10 and pr is 25%, seven OKFFs are not pruned while the remaining three undergo pruning. A sample state vector for this example may look like this: [1 2 1 3 1 1 1 1 2 1]. The strides are assigned from a random distribution but take feedback from how well the pruning proceeds. If pruning is proceeding well, we choose higher strides and induce IKSS more aggressively. Furthermore, the earlier discussions suggest that

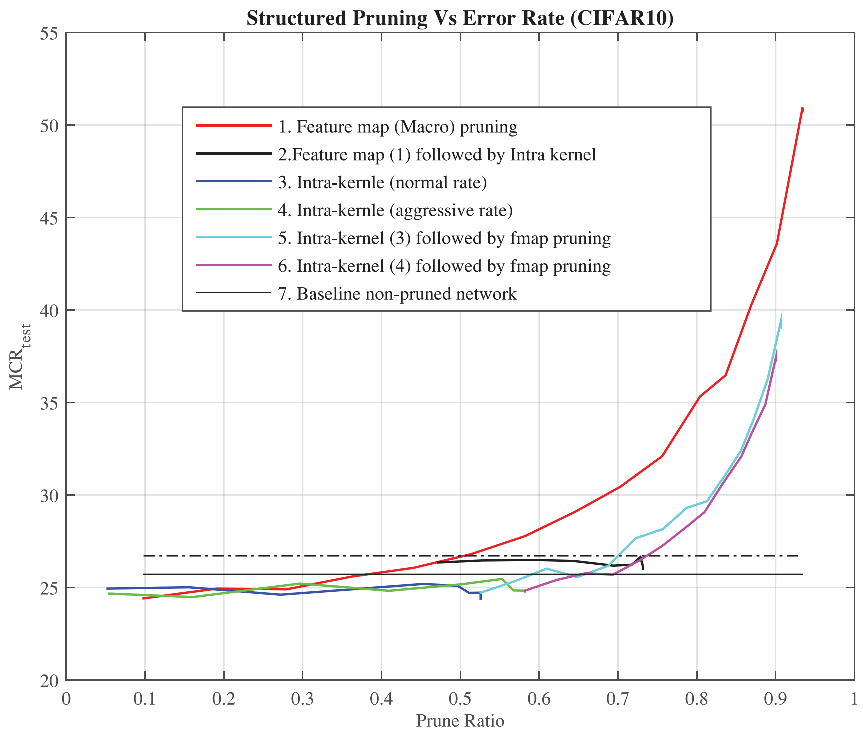


Fig. 8. The plot shows pruning applied in various combinations to NW_{small}. It is observed that feature map pruning followed by intra-kernel pruning provides the best result. The pruning ratios are computed for convolutional layers only.

intra-kernel sparsity can be induced at higher rates in 5×5 kernels compared to 3×3 kernels. Therefore, the stride assignment also considers kernel size. We experimentally fine-tune the hyper parameters with the validation set. Each particle's likelihood is also computed with the validation set. The offset represents the index of the first non-zero weight in a kernel and is randomly assigned in the range of $[0, stride - 1]$. Thus, two OKFFs having similar stride size but different offsets result in two different intra-kernel pruning masks. In this way, the EPF guides the IKSS to approximately select the best pruning candidates.

4.3. Pruning Granularities Applied in Different Combinations

In this section, we evaluate several combinations of pruning granularities applied to CNN_{small} and CNN_{large}. We first present experimental results with $CNN_{small,}$ and the results are shown in Figure 8. We observe that due to modest network depth, higher feature map pruning ratios cannot be achieved. Intra-kernel sparsities are induced at normal and aggressive rates. Choosing an aggressive rate increases the pruning ratio with bigger strides but also increases the MCR, as shown in Figure 8. We find that the MCR does not increase much when IKSS follows the feature map pruning. In Figure 8, the solid black line extending from the red line shows the best pruning results. This is achieved with feature map pruning (46%) followed by IKSS (shown in Figure 8 by a black solid line extending from the red solid line). Our experimental results show that we can reduce the size of the CNN_{small} network by 72% with less than a 1% loss in accuracy.

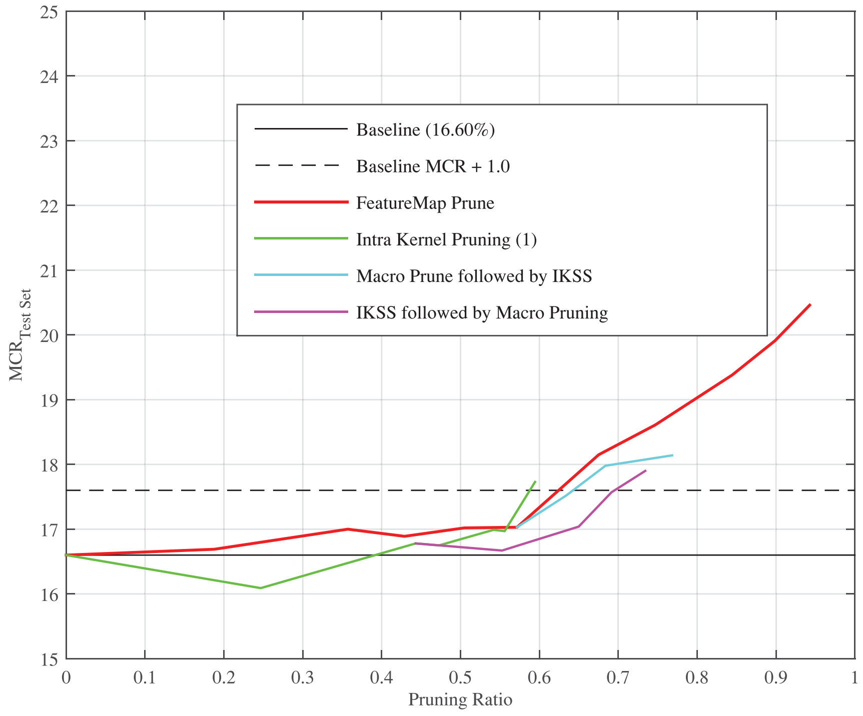


Fig. 9. The plot shows different pruning granularities applied to CNN_{large} in various combinations. For this network, we achieve the best pruning result when intra-kernel pruning is followed by feature map pruning.

The pruning plots for CNN_{large} are provided in Figure 9. It can be observed that inducing intra-kernel sparsities beyond 55% increases MCR by more than 1%. However, feature maps can be pruned by more than 62% in the same network. This is due to smaller kernel sizes of 3×3 and a higher width of the CNN_{large} network. From the figure, we obtain the best result when the intra-kernel sparse network is followed by feature map pruning. From these results, we can infer that wider networks have more chances for feature map-level pruning whereas bigger kernels are suitable for intra-kernel pruning. In both networks, we achieve the maximum pruning ratio when the better pruning granularity is applied at the later stage.

We conduct another experiment with the $CNN_{verylarge}$ and achieve better baseline accuracy than with CNN_{large}. The $CNN_{verylarge}$ has seven convolution layers and the architecture is reported as $(2 \times 128C3)$-MP2-$(2 \times 256C3)$-MP2-$(2 \times 256C3)$-$(1 \times 512C3)$-$1024FC$-$1024FC$-$10Softmax$. The network is trained with data augmentation, where we extract 28×28 patches and apply random flipping, rotation, translation, and scaling to the input samples. We randomly crop 28×28 regions from the 32×32 input image. At test time, we crop the sample from the four corners and the center. We further apply horizontal flips to each sample. In this way, we obtain 10 samples, and we take the average of their predictions to decide the final label [Krizhevsky et al. 2012]. This improves the prediction accuracy, and the network obtains 90.61% accuracy on the test set. We train the network with batch normalization, as reported earlier. The corresponding pruning plots are reported in Figure 10.

We attribute the better baseline performance of $CNN_{verylarge}$ to data augmentation and the bigger network. The pruning plots in Figure 10 show similar trends to those reported in Figure 9. For this network, we also demonstrate the pruning ratios with

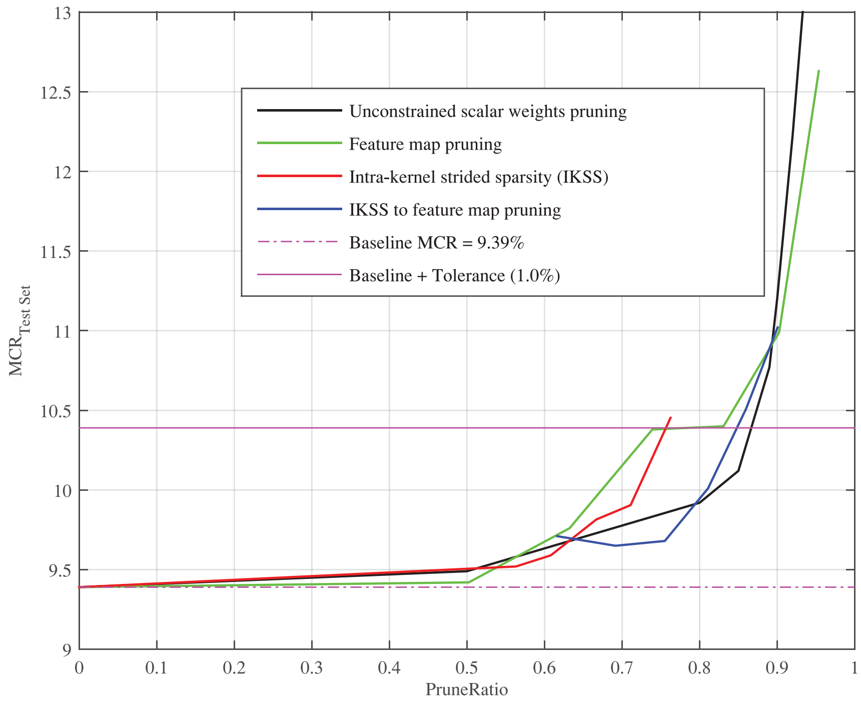


Fig. 10. The plot shows different pruning granularities applied to $CNN_{verylarge}$ in various combinations. Furthermore, we also show the case when we apply constraintless unstructured pruning of the network. The pruning ratios are computed for convolutional layers only.

constraint-free unstructured pruning. It can be observed that higher pruning ratios can be induced if there is no regularity constraint imposed on pruning. However, higher pruning ratios may not directly translate into reduced complexity. We need to identify the indices of non-zero weights with CSR/CSC sparse representation, as mentioned earlier. Furthermore, unstructured pruning is not good for utilizing parallel architectures such as SIMD. Thus, the effectiveness and cost of sparse representation also needs to be taken into consideration. The good side of constrained pruning is that we need very simple sparse representation, and we can achieve higher pruning ratios by pruning the network with mixed granularities, as shown in Figure 10. If we compare the pruning plots of Figures 8, 9, and 10, it can be observed that higher pruning ratios can be induced with the increasing size of the network. Thus, we show that the proposed pruning techniques have good scalability. In fact, we argue that a large network is more resilient to pruning than a small one because pruning followed by retraining makes the network learn adversarial effects [Sung et al. 2015].

4.4. SVHN Dataset

The SVHN dataset consists of 32×32 RGB images of house numbers [Netzer et al. 2011]. This dataset bears similarity with the MNIST dataset [LeCun et al. 1998] but is more challenging. The goal is to identify a digit in the center of a patch because there may be more than one digit in a sample. The dataset consists of 73,257 digits for training, 26,032 for testing, and 53,1131 extra for training. The extra set consists of easy samples and is used along with the training set. We generate a validation set consisting of 10×400 samples from the training set and 10×200 samples from the extra [Sermanet et al. 2012]. Thus, we have 6,000 validation samples. This

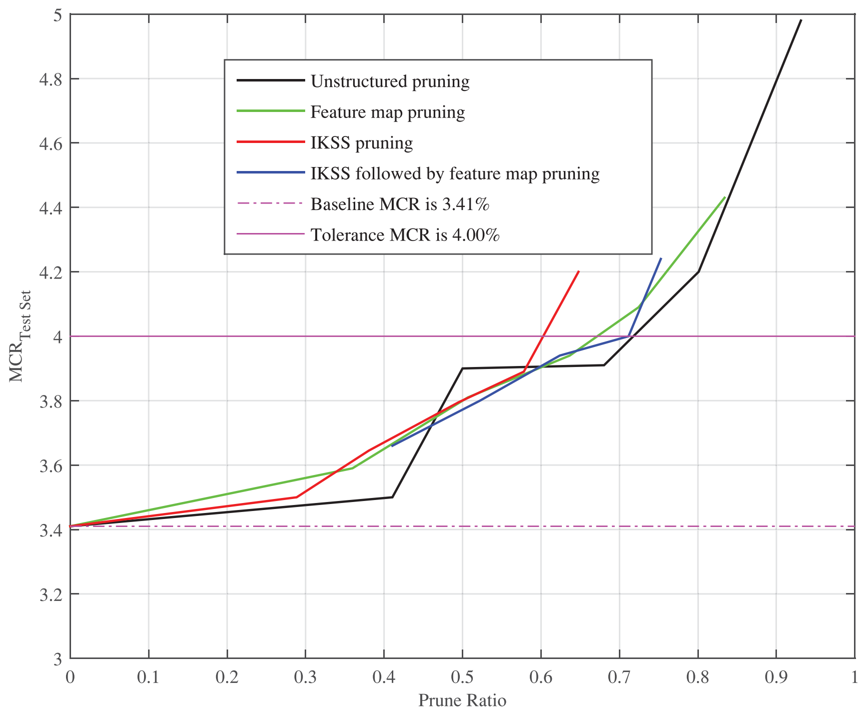


Fig. 11. The plot shows different pruning granularities applied to the SVHN dataset. Furthermore, we also show the pruning granularities applied in different combinations.

criterion is also used in Sermanet et al. [2012]. The network architecture is reported as $(2 \times 64C3)$-MP2-$(2 \times 128C3)$-MP2-$(2 \times 128C3)$ -512FC-512FC-10$Softmax$. This network is trained with batch normalization and RmsProp. This network achieves a baseline MCR of 3.41% on the SVHN test set. The corresponding pruning plots are reported in Figure 11. Due to the small kernel size (3×3), we cannot achieve higher pruning ratios with IKSS. However, the constraintless intra-kernel pruning can induce more than 70% pruning in the network. The network can be pruned with feature map granularity by more than 65%, and thus the layer width is reduced. We achieve the best pruning ratio with structured pruning when we first prune the convolutional layers with IKSS followed by feature map pruning. In this way, we show that the network can be pruned by more than 70% with structured sparsity, which requires very simple sparse representation. Thus, we infer that the conclusions drawn in the previous sections generalize well to datasets other than the CIFAR-10.

4.5. Execution Time Savings

In this section, we report and compare the execution time of pruning granularities. For feature map pruning, we conduct convolutions as matrix multiplications with the BLAS library [Chellapilla et al. 2006]. We first unroll the feature maps and kernels and conduct matrix multiplication on CPU. The execution time statistics are profiled for the aforementioned CNN_{small} network and reported in Figure 12. The execution time for the feed-forward path matters at real time, and we profile it with various batch sizes (1, 4, 16, 64, 128, 256, 512, and 1024). We can observe from Figure 12 that macro pruning causes an acceleration of the second convolution layer by 1.3 times, where 32×32 convolution connections are reduced to 32×20. This macro-pruned network

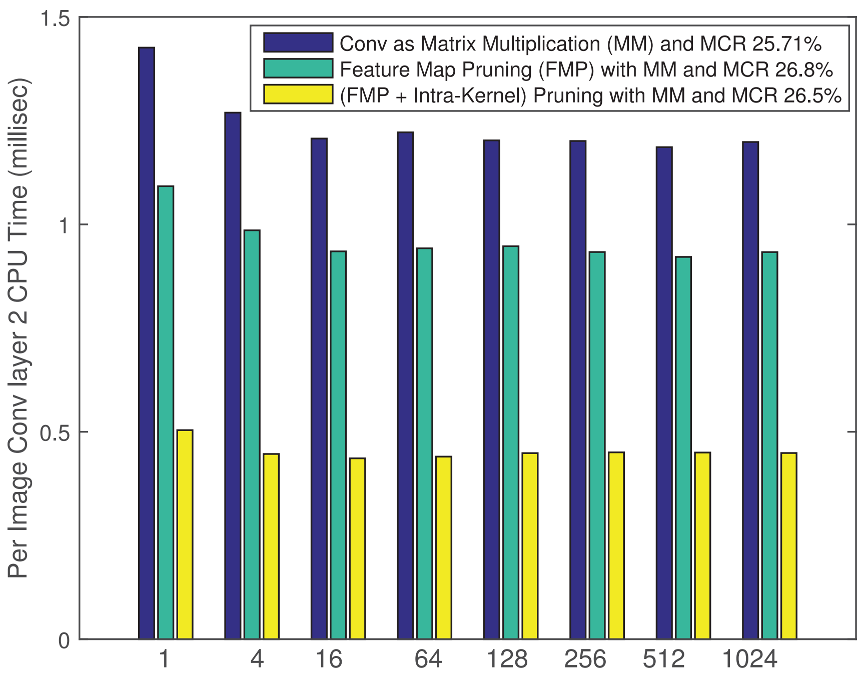


Fig. 12. The plot shows the acceleration due to feature map and intra-kernel pruning granularities for the second convolution layer of CNN_{small}. The x-axis shows the batch size, whereas the y-axis shows the per image CPU time. The convolution connections are reduced from $32 \times 32(C5)$ to $32 \times 20(C5)$ with feature map pruning. Intra-kernel sparsity is then induced in the feature map pruned network, which further accelerates the network.

then undergoes IKSS. This further accelerates the macro-pruned network by 2 times. Overall, the matrix-matrix multiplication of the unpruned network is accelerated by 2.67 times. Furthermore, Figure 12 shows that the speedups are in conformity with the theoretical numbers.

5. COMPARISON WITH PREVIOUS RELATED WORKS

Several previous works have used pruning techniques to reduce the computational cost of DNNs. Computational complexity is reduced with sparse connectivity in convolution and fully connected layers in Collins and Kohli [2014]. Stepniewski and Keane [1997] pruned multilayered feed-forward networks with a GA and simulated annealing [Stepniewski and Keane 1997]. A survey on pruning techniques is reported in Reed [1993]. These works [Castellano et al. 1997; Collins and Kohli 2014; Stepniewski and Keane 1997; Reed 1993] utilize unstructured sparsity in feed-forward neural networks. A recently published work induces channel-wise sparsity in a network [Polyak and Wolf 2015]. Compared to Polyak and Wolf [2015], the proposed work explores sparsity at multiple levels using an evolutionary approach. Dropout [Srivastava et al. 2014] and Dropconnect [Wan et al. 2013] zero neuron outputs and weights only during training, and the network architecture does not change at evaluation time. Both techniques train different subsets of network parameters during training, which results in better generalization. The proposed work drops parameters permanently and yields networks with fewer parameters at test time. Convolutions are converted to matrix-matrix multiplication in Chellapilla et al. [2006], which follows from the same logic that two bigger-sized matrix multiplications are better than several small ones

[Chetlur et al. 2014]. Units in the hidden layers are pruned in Castellano et al. [1997] for a feed-forward DNN.

The reference work of Lebedev and Lempitsky [2015] bears similarity with the proposed work for the case of intra-kernel sparsity. Lebedev and Lempitsky [2015] is mainly aimed at implementing convolutions as matrix-matrix multiplications. In our proposed work, we explicitly apply various pruning granularities, and the lightweight network obtained from feature map pruning can be implemented in a conventional way, through convolution unrolling or convolution with FFTs [Mathieu et al. 2013]. Second, the group-wise sparsification in Lebedev and Lempitsky [2015] is not necessarily strided. Lebedev and Lempitsky [2015] learns the pruning mask with group-wise sparsification, whereas our proposed approach finds the pruning mask with an evolutionary particle filter. Furthermore, this same work states that setting the regularization parameter is complicated.

6. CONCLUSION

In this work, we explored structured sparsity in deep convolutional neural networks as a means of reducing the computational complexity of the convolution layers. The sparsity in the feature map and kernel levels is explained along with the intra-kernel strided one. The selection of the best pruning mask is guided by the evolutionary PF algorithm. We found that feature map level pruning is limited by the width of a layer, while the intra-kernel sparsities are much affected by kernel size. The proposed work has further showed that the IKSS along with convolution unrolling can significantly reduce the computational complexity of convolutions. Moreover, with three different CNN architectures and baseline performances, we showed that the proposed approach scales well to various network sizes. In our future work, we are exploring knowledge-based pruning.

REFERENCES

M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. 2002. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 50, 2 (2002), 174–188.

James Carpenter, Peter Clifford, and Paul Fearnhead. 1999. Improved particle filter for nonlinear problems. In *IEEE Proceedings on Radar, Sonar and Navigation* 146. IET, 2–7.

Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks* 8, 3 (1997), 519–531.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High performance convolutional neural networks for document processing. In *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. Cudnn: Efficient primitives for deep learning. *Arxiv Preprint Arxiv:1410.0759* (2014).

Maxwell D. Collins and Pushmeet Kohli. 2014. Memory bounded deep convolutional networks. *Arxiv Preprint Arxiv:1412.1442* (2014).

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems*. 3105–3113.

Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. 1993. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEEE Proceedings on F-Radar and Signal Processing* 140. IET, 107–113.

Song Han, Huizi Mao, and William J. Dally. 2015a. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. *Arxiv Preprint Arxiv:1510.00149* (2015).

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*. 1135–1143.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and others. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Arxiv Preprint Arxiv:1502.03167* (2015).

A. Krizhevsky. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*. 1097–1105.

Ngai Ming Kwok, Gu Fang, and Weizhen Zhou. 2005. Evolutionary particle filter: Re-sampling from the genetic algorithm perspective. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*. IEEE, 2935–2940.

Vadim Lebedev and Victor Lempitsky. 2015. Fast convnets using group-wise brain damage. *Arxiv Preprint Arxiv:1506.02515* (2015).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.

Tiancheng Li, Shudong Sun, Tariq Pervez Sattar, and Juan Manuel Corchado. 2014. Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Systems with Applications* 41, 8 (2014), 3944–3954.

Michael Mathieu, Mikael Henaff, and Yann LeCun. 2013. Fast training of convolutional networks through FFTs. *Arxiv Preprint Arxiv:1312.5851* (2013).

Kazuyuki Nakamura, Ryo Yoshida, Masao Nagasaki, Satoru Miyano, and Tomoyuki Higuchi. 2009. Parameter estimation of in silico biological pathways with particle filtering towards a petascale computing. In *Proceedings of the Pacific Symposium on Biocomputing* 14. 227–238.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Katja Nummiaro, Esther Koller-Meier, and Luc Van Gool. 2003. An adaptive color-based particle filter. *Image and Vision Computing* 21, 1 (2003), 99–110.

Adam Polyak and Lior Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access* 3 (2015), 2163–2175.

Russell Reed. 1993. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks* 4, 5 (1993), 740–747.

Pierre Sermanet, Soumith Chintala, and Yann LeCun. 2012. Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*. IEEE, 3288–3291.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *Arxiv Preprint Arxiv:1409.1556* (2014).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

Slawomir W. Stepniewski and Andy J. Keane. 1997. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural Computing & Applications* 5, 2 (1997), 76–98.

Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. 2010. Performance and scalability of GPU-based convolutional neural networks. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 317–324.

Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. 2015. Resiliency of deep neural networks under quantization. *Arxiv Preprint Arxiv:1511.06488* (2015).

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* 4 (2012), 2.

Jaco Vermaak, Arnaud Doucet, and Patrick Pérez. 2003. Maintaining multimodality through mixture tracking. In *Proceedings of the 9th IEEE International Conference on Computer Vision, 2003*. IEEE, 1110–1116.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 1058–1066.

Received March 2016; revised August 2016; accepted October 2016