Citation for published version:
Singh, P, Kadi, VSR & Namboodiri, VP 2020, 'FALF ConvNets: Fatuous auxiliary loss based filter-pruning for efficient deep CNNs', *Image and Vision Computing*, vol. 93, no. 103857, 103857, pp. 1-14.
https://doi.org/10.1016/j.imavis.2019.103857

DOI:
10.1016/j.imavis.2019.103857

Publication date:
2020

Document Version
Peer reviewed version

Link to publication

Publisher Rights
CC BY-NC-ND

FALF ConvNets: Fatuous Auxiliary Loss based Filter-pruning for Efficient Deep CNNs

Pravendra Singh[a,1,*], Vinay Sameer Raja Kadi[b,1], Vinay P. Namboodiri[a]

[a]Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, India
[b]Carnegie Mellon University, Pittsburgh, United States

Abstract

Obtaining efficient Convolutional Neural Networks (CNNs) are imperative to enable their application for a wide variety of tasks (classification, detection, etc.). While several methods have been proposed to solve this problem, we propose a novel strategy for solving the same that is orthogonal to the strategies proposed so far. We hypothesize that if we add a fatuous auxiliary task, to a network which aims to solve a semantic task such as classification or detection, the filters devoted to solving this frivolous task would not be relevant for solving the main task of concern. These filters could be pruned and pruning these would not reduce the performance on the original task. We demonstrate that this strategy is not only successful, it in fact allows for improved performance for a variety of tasks such as object classification, detection and action recognition. An interesting observation is that the task needs to be fatuous so that any semantically meaningful filters would not be relevant for solving this task. We thoroughly evaluate our proposed approach on different architectures (LeNet, VGG-16, ResNet, Faster RCNN, SSD-512, C3D, and MobileNet V2) and datasets (MNIST, CIFAR, ImageNet, GTSDB, COCO, and UCF101) and demonstrate its generalizability through extensive experiments. Moreover, our compressed models can be used at run-time without requiring any special libraries or hardware. Our model compression method reduces the number of FLOPS by an impressive factor of 6.03X and GPU

*Corresponding author.
Email addresses: `psingh@iitk.ac.in` (Pravendra Singh), `vkadi@andrew.cmu.edu` (Vinay Sameer Raja Kadi), `vinaypn@iitk.ac.in` (Vinay P. Namboodiri)
[1]Equal contribution.

memory footprint by more than 17X for VGG-16, significantly outperforming other state-of-the-art filter pruning methods. We demonstrate the usability of our approach for 3D convolutions and various vision tasks such as object classification, object detection, and action recognition.

Keywords: Filter pruning, Model compression, Convolutional neural network, Image recognition, Deep learning

1. Introduction

After marking themselves as the state of the art solutions in a variety of fields such as vision, NLP, speech, etc., CNNs (convolutional neural networks) are finding their way to be deployed in the real world lately. However, this deployment is not straightforward due to high computation (FLOPS) and memory requirements of CNNs. While it may seem trivial to address this problem by using smaller sized networks, redundancy of parameters seems necessary in aiding highly non-convex optimization during training to find effective solutions. A similar line of works aimed at devising efficient architectures [1, 2] to be trained from scratch on a given task. While they have shown promising results, their generalizability across the tasks is not fully studied. Hence significant efforts are seen in recent days to address model compression. Prominent works in this area [3] have focused on model compression to make CNNs more efficient in terms of computations (FLOPS) and memory requirements (Run Time Memory usage and storage space of the model). These methods first train a large model for a given task and then prune the model until the desired compression is achieved.

Model compression techniques can be broadly divided into the following categories. The first category [4, 5] aims at introducing sparsity in the parameters of the model. While these approaches achieved a good compression rate in model parameters, computations (FLOPS) and Total Runtime Memory (TRM) aren't improved. Such methods also require sparse libraries support to achieve the desired compression [4].

The second category of methods are [4, 6, 7] based on model compression using quantization. Often specialized hardware is required to achieve the required acceleration. These model compression techniques are specially designed for IoT devices.

The third category of methods [8, 9, 10, 3, 11, 12, 13, 14] perform filter level pruning in the model. These approaches prune an entire filter based on some criteria/metrics and hence provide a structured pruning in the model. As for pruning the whole convolutional filter from the model reduces the depth of the feature maps for subsequent layers, these approaches give high compression rate regarding computations (FLOPS) and Total Runtime Memory (TRM). Since the final pruned model using these methods are again CNNs, albeit smaller in size, sparsity and quantization based methods complement these methods to achieve better compression rates.

These methods are mainly differentiated by the ranking mechanism used to identify important filters. The brute force approach has been addressed in [8] to prune the filters from the model. They remove each filter sequentially and rank the importance of the filter based on their corresponding drop in the accuracy. This approach seems to be impractical for large size networks on large-scale datasets.

The other works in this category can be further classified into two classes, Handcrafted metrics for calculating filter ranking and metrics calculated based on data and architecture. [3, 15] fall under the former category. In the work of [3] they use l_1 norm of a filter to identify the filter importance. In the latter category, [11, 16, 17] use data-driven metrics to identify the filter importance. [16] use the Taylor expansion to calculate the filter importance, which is motivated by optimal brain damage [18, 19].

While the previously proposed techniques adopt heuristics such as sparsity or quantization, in this paper we aim to identify the core set of filters that are pertinent for solving a task. Towards obtaining this set of filters, it is important to qualify the importance based on some meaningful criterion. Our approach is based on the following hypothesis: If a set of filters are crucial for performance on a particular task, then if some other fatuous task is added as an auxiliary task, these filters would be minimally perturbed by the network in an effort to solve the two tasks provided to the network, the main task and the fatuous task. This simple observation guides our work, and we show that by allowing the network itself to figure out which weights it can use for the other task, we are provided with the core set of filters necessary for the main task. This approach is orthogonal to all other filter pruning methods or other compression techniques prevalent in the literature. We evaluate our approach on a variety of tasks and show an impressive reduction in FLOPS across different architectures. We further demonstrate the generalizability of our approach by achieving competitive accuracy using a small model pruned for a different task. Our method does not place any restrictions and is augmented with other pruning methods such as quantized weights, Low-rank approximation, connection pruning, etc., to further decrease the FLOPS and memory consumption. We also demonstrate how our method complements the works of efficient design of CNNs, by pruning those architectures to achieve a further efficient model.

In our work [20], we initiated the work by proposing filter pruning using stability based criteria [20] in the classification problem, whereas in this work, we concretize the notion of the requirement of a fatuous auxiliary loss function to understand our method. We further evaluate our proposed filter pruning approach on various vision tasks such as object detection and action recognition. We also demonstrate the usability of our approach to 3D convolutions. As the main contribution is based on adding auxiliary loss function, the choice of auxiliary loss functions is considered in detail in this paper. Further, in ablation study, we analyze the effect of fine-tuning on the compressed model rather than training from scratch and effect of jointly doing pruning and training from scratch on the performance. We experimentally show that the proposed approach is able to compress already compact model such as MobileNet with no loss in accuracy.

Our main contributions are as follows:

- We propose a novel approach based on fatuous task augmentation to obtain a ranking of importance of filters.
- We demonstrate the usability of our approach for various vision tasks such as object classification, object detection, and action recognition.
- We empirically show that the proposed approach works well on both 2D convolutions and 3D convolutions.
- We experimentally show that the proposed approach is able to compress already compact model such as MobileNet with no loss in accuracy.

2. Related Work

The works on model compression can be divided into the following categories.

2.1. Connection Pruning

In the connection pruning, they introduce sparsity in the model by removing unimportant connections (parameters). There are many heuristics proposed to identify the unimportant parameters. Earliest works include Optimal Brain Damage [19] and Optimal Brain Surgeon [18] where they used Taylor expansion to identify the significance of the parameters. Later [4] proposed an iterative method where absolute values of weights below a certain threshold are pruned, and the model is fine-tuned to recover the drop in accuracy. This type of pruning is called unstructured pruning as the pruned connections have no specific pattern. This approach is useful when most of the parameters lie in the FC (fully connected) layers. Often, specialized libraries and hardware are required to leverage the induced sparsity to save computation and memory requirements. However, this does not typically result in any significant reduction in CNN computations (FLOPS based SpeedUp) as most of the calculations are performed in CONV (convolutional) layers. For example, in VGG-16, 90% of the total parameters belong to FC layers, but they contribute to 1% of the overall computations, which implies that CONV layers (having 10% of the total model parameters) are responsible for 99% of the overall calculations.

Other works include [5] where they propose hashing technique to randomly group the connection weights into a single bucket and then fine-tune the model to recover from the accuracy loss.

SBP [21] propose a Bayesian model to provide structured sparsity by injecting noise to outputs of the neurons. The method provides structured sparsity by removing elements with a low SNR from the computation graph. NISP [22] calculate NISP (Neuron Importance Score Propagation) importance scores of final responses to every neuron for neurons pruning and then fine-tuned to recover the performance loss.

Group-wise sparsity is also used in model compression. Lebedev and Lempitsky [23] used group-sparsity regularization to the loss function. [24] explored different types of sparsity from irregular connection pruning to regular filter pruning. [25] proposed the Structured Sparsity Learning (SSL) approach to regularize network structure. The main limitation of methods mentioned above is the loss of the original network structure. Therefore, some dedicated libraries are required to get speed-up in practice.

2.2. Filter Pruning

In our work, we focus on filter level pruning. Most of the works in this category evaluate the importance of an entire filter and prune them based on some criteria followed by re-training to recover the accuracy drop. In the work [8], they calculate the filter importance by measuring the change in accuracy after pruning the filter from the model. [13, 14] perform random filter pruning from deep CNNs. [3] used l_1 norm to calculate the filter importance. [26] calculate the filter importance on a subset of the training data using activation of the output feature map. These approaches are largely based on hand-crafted heuristics. Parallel to these works, ranking filters based on data-driven approaches are proposed. [27] performed the channel level pruning by attaching a learnable scaling factor to each channel and enforcing l_1 norm on those parameters during the training. Recently, group sparsity is also being explored for filter level pruning. CP [28] propose a channel pruning approach to accelerate Deep CNNs using an iterative two-step algorithm. The method uses LASSO regression based channel selection and least square reconstruction to prune each layer in Deep CNNs. [29, 23, 25, 30] explored the filter pruning using group lasso.

RNP (Runtime Neural Pruning) [31] propose a framework to prune deep CNN dynamically at the runtime. The pruning is performed in a bottom-up manner and uses reinforcement learning for training. An agent calculates the importance of each convolutional kernel. ThiNet [32] do filter pruning as an optimization problem based on statistics computed from its next layer. SFP (Soft Filter Pruning) [10] enables the pruned filters to be updated when training the deep model after filter pruning.

GAL [33] proposes a structured pruning approach that jointly prunes filters along with other structures using sparsity regularization. They solve the optimization problem by using generative adversarial learning (GAL). GDP [34] proposes a global and dynamic pruning method to prune unnecessary filters. SSS [35] proposes a framework to learn and prune CNN in an end-to-end manner using a scaling factor parameter, which is used to scale the outputs of specific structures. They safely prune the unnecessary parts of a deep model by forcing some of the scaling factors to zero.

Closest to our work is the work of [16] where they proposed filter rankings using a mean of absolute gradient values and demonstrated that it gives competitive results to the brute-force method of checking loss deviation for each filter. In contrast to their approach, we obtain a fatuous loss based ranking of importance of filters that is semantically more meaningful and shows improved performance over such works.

2.3. Quantization

Quantization based approaches aim to convert and store the network weights into a comparatively low bit configuration. The reduction in memory and computational requirements seems improbable after a certain level. However, these approaches can be used as a complement to filter pruning based approaches to extend the compression rates. Notably, [4] compressed the model by combining pruning, quantization and Huffman coding. In the early works binarization [7] has been used for the model compression. Extending this, [30] used ternary quantization learned from the given data. Recently, [36] conducted network compression based on the float value quantization for model storage.

At times, these quantization methods require specialized library/hardware support to reach desired compression rates. Some of the other notable works using different approaches from quantization include [9, 12] and [37] where they used the low-rank approximation to decompose tensors and reduce the computations.

Our method performs filter pruning using data-driven filter ranking. To the best of our knowledge, our work is a primary effort to relate filter importance to its task importance and does not require any special hardware/software such as cuSPARSE (NVIDIA CUDA Sparse Matrix library).

3. Proposed Approach

3.1. Terminology

Given a pre-trained CNN which contains K convolution layers, let $\mathcal{L}_i$ denote the i^{th} layer where $i \in [1, 2, \dots K]$. The set of filters in layer $\mathcal{L}_i$ is denoted as $\mathcal{F}_{\mathcal{L}_i}$. Where $\mathcal{F}_{\mathcal{L}_i} = \{f_1^i, f_2^i, \dots, f_{n_i}^i\}$ and n_i denotes the number of filters in layer $\mathcal{L}_i$ (which is also the number of output channels for that layer). The dimension of each filter f_j is (h_i, w_i, c_{in}), where h_i, w_i and $c_{in}(= n_{i-1})$ are height, width and number of input channels respectively. $|f_j^i|$ denotes the sum of absolute values of filter f_j^i (j^{th} convolutional filter on i^{th} layer) and $f_{j,l}^i$ denotes the individual value of the filter ($l \in \{1, \dots, h_i w_i c_{in}\}$). Each filter on i^{th} layer is of dimension $h_i w_i c_{in}$.

3.2. Approach

To calculate the filter importance, we propose a method based on the sensitivity of the filter with respect to a fatuous auxiliary loss function. The high sensitivity of a filter implies that it is less important for the current task and vice versa. Let $C(\Theta)$ denote the loss function for the original task, where Θ are the model parameters. To modify the loss function, we introduce an auxiliary loss. This auxiliary loss function can be chosen based on the prior knowledge about the task or from a generic set of functions. We refer the reader to section 5 for more details. One of the fatuous auxiliary loss functions is designed such that it forces the negative values in filters to -1 and positive values in filters to $+1$. The reason for such design will be made clear during the filter ranking discussion in next section. The auxiliary loss for a layer i is given as:

$$S_{\mathcal{L}_i} = \sum_{j=1}^{n_i} \sum_{l=1}^{h_i * w_i * c_{in}} [|(-1 - f_{j,l}^i)|.\mathbb{I}(f_{j,l}^i < 0) \\ + |(1 - f_{j,l}^i)|.\mathbb{I}(f_{j,l}^i \geq 0)] \tag{1}$$

where $\mathbb{I}()$ denotes the function which equals 1 if the condition is satisfied else 0. Now the complete loss can be given as:

$$L(\Theta) = C(\Theta) + \lambda \sum_{i=1}^{K} S_{\mathcal{L}_i} \tag{2}$$

where λ is a hyperparameter controlling the effect of auxiliary loss term. Having defined the auxiliary loss, we now describe the procedure for pruning.

3.2.1. Training the network

As we know that the deep networks have enough complexity to represent any function, the auxiliary loss may interfere with the optimization of an actual loss function. To avoid this possibility, we first train the network using actual loss function $C(\Theta)$. Let the filters at the end of training be denoted by $\mathcal{F}_{\mathcal{L}_i} = \{f_1^i, f_2^i, \ldots, f_{n_i}^i\}$. We then train the network using the total loss function $L(\Theta)$. To prevent the weights from drifting away from optimal weights for the actual task, we train the model using $L(\Theta)$ only for the limited number of epochs (typically 1-3 epochs will be enough as the auxiliary loss is data independent). Let the $\mathcal{M}_{\mathcal{L}_i} = \{m_1^i, m_2^i, \ldots, m_{n_i}^i\}$ be the set of filters at layer $\mathcal{L}_i$ after optimizing equation-2.

3.2.2. Ranking the filter Importance

The sensitivity of a filter to auxiliary loss can be written as the magnitude of difference of filter weights before and after training using auxiliary loss. Mathematically, we define sensitivity of a filter as S(f_j^i) = $|m_j^i$-$f_j^i|$. As the weight values differ in scale for different filters in a layer, we normalize the difference to make the ranking criteria invariant to scaling. Thus the filter ranking $(FI_{\mathcal{L}_i})$ of $\mathcal{L}_i$ is defined as:

$$FI_{\mathcal{L}_i} = \left\{ \frac{|m_j^i - f_j^i|}{|f_j^i|} : \forall j \in \{1, 2, ..., n_i\} \right\} \tag{3}$$

Now, as per the design of auxiliary loss function, if a filter tries to contribute to the minimization of this loss, then its weight values will increase in magnitude compared to their previous value. i.e, if a filter is sensitive to the introduction of auxiliary loss function, then the ratio term in (3) will be high. As per our hypothesis, the filter that has a strong contribution to the original task has the least sensitivity, hence low ratio. Let $P = [p_1, p_2, \ldots, p_K]$ be the number of filters to be pruned form each layer, where K is the number of convolutional layers. Now, based on the filter importance given by equation-3, we select $p_1, p_2, \ldots, p_K$ least important filters from the corresponding

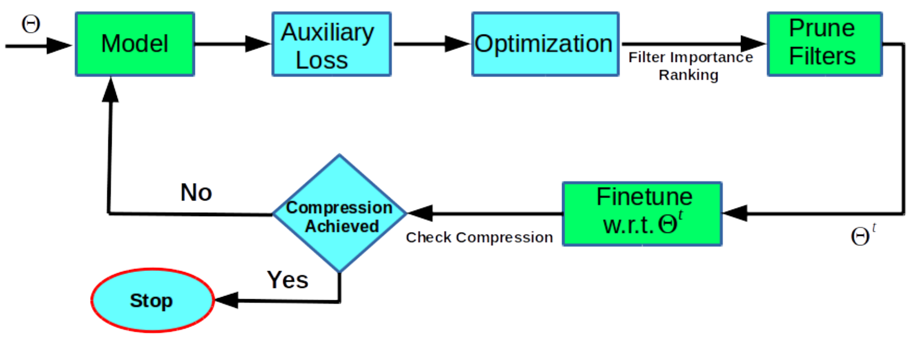

Figure 1: Fatuous auxiliary loss function based filter pruning approach where filters are pruned iteratively based on sensitivity to the auxiliary loss.

layers in the model and prune them. The pruned set is given as:

$$P^t_{set} = \{\sigma_{p_1}(\mathcal{F}_{L_1}), \sigma_{p_2}(\mathcal{F}_{L_2}), \ldots, \sigma_{p_k}(\mathcal{F}_{L_K})\} \tag{4}$$

Here σ is the select operator that selects p_i least important filters from the layer $\mathcal{L}_i$. P^t_{set} is the set of filters that are discarded from the model.

3.2.3. Pruning and fine-tuning

$\mathcal{F}^t$ is the set of remaining filters in the model with parameters Θ^t after t^{th} pruning iteration.

$$\mathcal{F}^t = \mathcal{F}^{t-1} \setminus P^t_{set} \tag{5}$$

Where $\setminus$ is the set-difference symbol. In each pruning iteration, after discarding the filter, we observe a small drop in accuracy. To avoid the accumulation of such accuracy drops, we fine-tune the pruned network for 2-5 epochs. During fine-tuning, we use the actual loss (without auxiliary loss). This process of alternative pruning and finetuning is continued until the desired compression rate is achieved as shown in Figure-1.

3.3. Relationship with the previous approaches

Though our approach is based on fatuous auxiliary loss functions for obtaining importance is quite different from the approaches proposed so far, we can always interpret the proposed approach in terms of earlier work such as that by Molchanov *et al.* [16].

In the work by Molchanov et.al. [16], they proposed to prune the channels using

$$|\Delta C(h_i)|(= |C(D, h_i = 0) - C(D, h_i)|)$$

criteria. They used the Taylor series expansion to calculate the metric. They demonstrate the difference between their work and the Optimal Brain Damage (OBD) [19] by arguing that the variance of the gradients serves as an important metric for pruning.

Their argument, in brief, states that after the completion of training, as per OBD,

$$\frac{\partial E_{x\sim p(x)}[C]}{\partial W_i} = E_{x\sim p(x)}\left[\frac{\partial C(x)}{\partial W_i}\right] = 0 \tag{6}$$

Although, the expected value of the gradient of loss w.r.t a parameter, say W_i, may tend to zero, the individual samples need not have their cost function (cost function for that particular sample) indifferent to W_i. This is effectively captured in variance of $\frac{\partial C(x)}{\partial W_i}$. So, if the variance is higher then it is possible that the weight W_i is indeed useful even though $E_{x\sim p(x)}\left[\frac{\partial C(x)}{\partial W_i}\right] = 0$. On the other hand, if the variance is low, and with the expectation also tending to 0, it is evident that the weight is useless and thus can be removed. Now, instead of calculating the variance explicitly, it can efficiently calculated by

$$E_{x\sim p(x)}\left[\left|\frac{\partial C(x)}{\partial W_i}\right|\right] \tag{7}$$

As stated in [16] this term is proportional to the variance of gradients over data distribution and hence can be used to rank filters. This implies that an unconscious assumption is made that $E_{x\sim p(x)}\left[\frac{\partial C(x)}{\partial W_i}\right] = 0$ when they start pruning.

Let us call this assumption A_1 for the rest of the paper. We first describe one scenario where the above method has issues with robustness. In their analysis, they considered that assumption A_1 holds. However, in practical scenarios, this may not hold as practitioners follow different strategies such as early stopping, etc., where they stop training based on validation error. This implies that there is no guarantee that the assumption A_1 holds for the training dataset. So, if we prune the weights according to equation-7 in such scenarios, it may remove important weights since $E_{x\sim p(x)}\left[\frac{\partial C(x)}{\partial W_i}\right]$ may not be zero but $E_{x\sim p(x)}\left[\left|\frac{\partial C(x)}{\partial W_i}\right|\right]$ may be minimum.

Our proposed approach can be observed to remove such disadvantages. In formulating our approach, we provide a general framework which includes the reasoning of [16] as a special case. We argue that as the networks are often over parameterized and it is obvious from the previous works that only a few of them contribute to an actual loss, the rest of the weights gets modified when an auxiliary loss function is added to existing loss function during the training. i.e., important weights for the actual task remain the same whereas the unimportant weights try to fit the auxiliary loss function if it is not related to the present task when trained using both loss functions. We now formulate it mathematically.

Notation:

Let the random variable X denote data distribution, and parameter W denote network weights, and R be a scalar random variable. Let C denote the actual loss function, D denotes the auxiliary loss function, and L be the total loss function.

Formulation:

The total loss function L is given by

$$L = C + R * D \tag{8}$$

Now, the gradient of cost function w.r.t. a parameter, say w_i, depends on two random variables, X, and R. According to our hypothesis, the important weights for the actual task do not change due to the introduction of an auxiliary fatuous loss function, and this implies that for a given data sample X_i, the following holds:

$$E_R\left[\left|\frac{\partial L(X_i)}{\partial w_i}\right|\right] \approx 0 \tag{9}$$

To understand it better, compare it with the argument given by Molchanov et al., where the variance (over the data distribution) of the gradients w.r.t. unimportant weights will be low because they do not contribute to the loss function for the majority of the samples. Whereas the variance of gradients w.r.t. important weights will be high due to their contribution in loss function for all the samples. Here, we follow the same logic but with a minute change of taking the expectation over the joint probability distribution of (X,R). Since the importance weights for the actual task are indifferent to auxiliary loss function (by our hypothesis), they contribute less to the update term during training with an auxiliary loss. So, when the R is varied, the resulting variance (of $\frac{\partial L}{\partial w_i}$) should be low. On the other hand, the unimportant weights for the actual task are the ones who try to fit the auxiliary loss function (by our hypothesis). So, when R is varied, the variance of $\frac{\partial L}{\partial w_i}$ will be high because when $R = 0$, $\frac{\partial L}{\partial w_i}$ =0 and when $R \neq 0$, $\frac{\partial L}{\partial w_i} \neq 0$ (by our hypothesis of unimportant weights). Hence resulting in a high mean and variance w.r.t. R. As stated earlier, we do not train the network until the auxiliary loss is completely minimized as this may affect the actual task. But, as the gradients are proportional to change in weight values, we use the change in weight values criteria for pruning instead of mean of absolute gradient values. In practice, we found this approach to be effective.

4. Results

To evaluate our proposed work, we perform experiments on seven standard models, LeNet-5 [38], VGG-16 [39], ResNet-50 [40] and MobileNet [41] for classification task, Faster-RCNN [42] and SSD [43] for object detection task and C3D [44] for action recognition task (clip level). All the experiments are performed on GTX-1080 Ti GPU and i7-4770 CPU@3.40GHz.

4.1. LeNet-5 on MNIST

MNIST dataset contains 60,000 training images and 10,000 testing images. Two convolutional (20,50) and two fully connected layers (800,500) are present in the LeNet-5 model. The error rate that we obtained on training the model is 0.83%.

We optimized equation-2 for one epoch with $\lambda = 0.00001$ to calculate filter importance in each pruning iteration. Learning rate is varied in the range $[0.001, 0.0001]$ for this experiment. As compared to the previous approaches (Table-1), we have a significantly higher FLOPS compression with the less drop in the accuracy. This proves the effectiveness of our proposed metric for filter importance ranking over the previous methods.

Table 1: Table showing results for the LeNet-5 model on the MNIST dataset. SSL and SBP are proposed by [25] and [21] respectively. Bold values indicate the best results obtained by our method in the comparison.

Method	Filter	Error (%)	FLOPS	Pruned (%)
Baseline	20,50	0.83	4.40×10^6	–
SSL [25]	5,19	0.80	5.97×10^5	86.42
SBP [21]	–	0.86	–	90.47
SSL [25]	3,12	1.00	2.89×10^5	93.42
Prun-1 (ours)	4,14	**0.79**	$\mathbf{3.97 \times 10^5}$	**90.98**
Prun-2 (ours)	3,8	**0.92**	$\mathbf{2.14 \times 10^5}$	**95.14**

4.2. VGG-16 on CIFAR-10

We experiment with the VGG-16 model on the CIFAR-10 dataset. We use the same VGG-16 model and settings as mentioned in [3], and after each layer, batch normalization is deployed. The model is trained from scratch. Layerwise FLOPS distribution is shown in Figure-2. It is clear from Figure-2 that CONV1_2, CONV2_2, CONV3_2, CONV3_3, CONV4_2, CONV4_3 layers have much higher FLOPS as compared to the remaining layers. Hence, to compress FLOPS, we need to remove more filters from

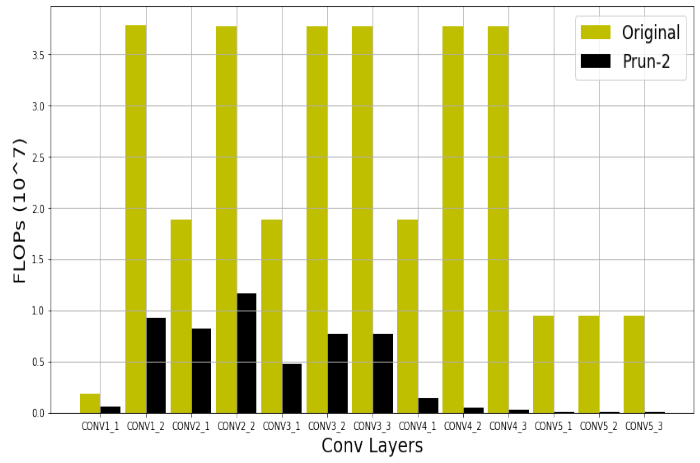

Figure 2: Figure shows the original and pruned model layer-wise FLOPS for the VGG-16 model on the CIFAR-10 dataset.

Table 2: Table shows the layer-wise pruning results and pruned model details for VGG-16 model on CIFAR-10 dataset.

		Baseline	VGG-16 Prun 1	VGG-16 Prun 2
Input Size		32x32x3	32x32x3	32x32x3
Layers	CONV1_1	64	31	20
	CONV1_2	64	53	50
	CONV2_1	128	84	71
	CONV2_2	128	84	71
	CONV3_1	256	146	116
	CONV3_2	256	146	116
	CONV3_3	256	146	116
	CONV4_1	512	117	87
	CONV4_2	512	62	42
	CONV4_3	512	62	42
	CONV5_1	512	62	42
	CONV5_2	512	62	42
	CONV5_3	512	62	42
	FC6	512	512	512
	FC7	10	10	10
Total parameters		15.0M	1.0M (15X)	0.62M (24.2X)
Model Size		60.0 MB	4.1 MB (14.6X)	2.5 MB (24X)
Accuracy		93.49	93.43	93.02
FLOPS		313.7M	78.0M (4.02X)	52.0M (6.03X)

such layers. We optimized equation-2 for one/two epochs with $\lambda = 0.00001$ to calculate filter importance ranking in each pruning iteration. We vary the learning rate in the range $[0.001, 0.0001]$ for this experiment. We get our first pruned model (Prun-1) after 82 epochs.

Table-2 shows the detailed results for VGG-16 pruning. Table-3 shows the comparison of our pruned model with previous approaches. Our method prunes 95.9% of parameters on CIFAR10, significantly larger than 64.0% pruned by [3]. Furthermore, our method reduces the FLOPS by 83.43% compared to 34.2% pruned by [3]. Layer-

Table 3: Table shows the FLOPS pruning result for VGG-16 on the CIFAR-10 dataset. Weight-Sum and SBP are proposed by [3] and [21] respectively. SSS* is the results based on [33] implementation. Bold values indicate the best results obtained by our method in the comparison.

Method	Error (%)	Parameters Pruned (%)	FLOPS Pruned (%)
Baseline	6.51	–	–
Weight-Sum [3]	6.60	64.0	34.20
SSS* [35, 33]	6.37	66.7	36.30
GAL-0.05 [33]	6.23	77.6	39.60
SSS* [35, 33]	6.98	73.8	41.60
GAL-0.1 [33]	6.58	82.2	45.20
SBP [21]	7.50	–	56.52
SBP [21]	9.00	–	68.35
Prun-1 (ours)	**6.57**	**93.3**	**75.14**
Prun-2 (ours)	**6.98**	**95.9**	**83.43**

wise FLOPS distribution for the original and pruned model are shown in the Figure-2.

4.3. Ablation study

In this section, we present various ablation studies to show the importance of the proposed criteria to evaluate filter importance. Next, we are training a compressed model with randomly initialized weights to show the effectiveness of the proposed approach. Since our approach starts with the pretrained model, we also show that pruning and training can be done jointly, but it results in some drop in accuracy.

4.3.1. Ablation study on filter importance ranking criteria

We next show an ablation study on VGG-16 to demonstrate the effectiveness of the proposed filter importance ranking. Here, we pruned filters from 6 layers; Conv4_1 to Conv5_3 simultaneously. Since each layer from Conv4_1 to Conv5_3 contains 512 filters, therefore, a total of 512*6 filters are available for pruning. If we remove X filters in each layer from Conv4_1 to Conv5_3, then a total of 6*X filters gets pruned from the model. Figure-3 horizontal axis shows the 6*X prune filters, and the vertical axis

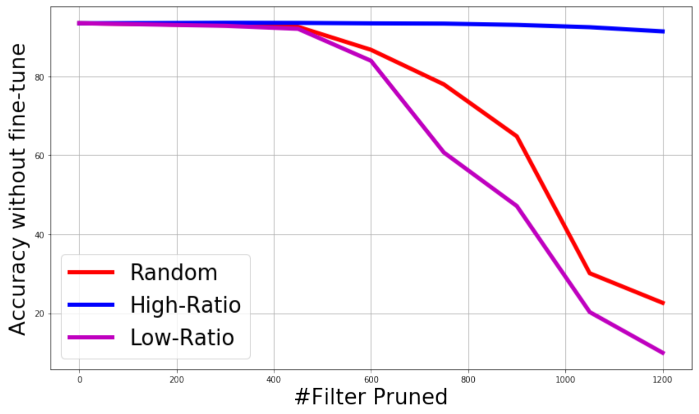

Figure 3: Effect of filter pruning with respect to accuracy for VGG-16. Filters are pruned from 6 Layers CONV4_1 to CONV5_3 simultaneously.

shows the accuracy without fine-tuning. We optimize equation-2 for three epochs with $\lambda = 0.00001$ to calculate filter importance ranking. Figure-3 shows that if we prune filters from the low ratio (important filters), there is a sharp accuracy drop. A similar pattern is observed if we prune a filter randomly. In contrast, if we prune the filters from the high ratio (unimportant filters), then it results in a small accuracy drop even when we prune 1200 filters.

4.3.2. Training compressed model with randomly initialized weights

To show the importance of pruning, we analyze the performance difference between models, which are obtained by training the compact/pruned model (with weights reset to random values) from scratch and finetuning the compact/pruned model. On CIFAR 10 dataset, using the two models VGG-16 Prun 1 and VGG-16 Prun 2 (same architecture as given in Table-2), we observed that the errors obtained, 6.89% and 7.58% respectively, by training from scratch are high compared to finetuning the pruned model as shown in Table-4. A similar trend is also observed in [3]. We surmise that the involvement of a highly non-convex optimization problem, for which a certain degree of parameter redundancy is required during training, is the reason for this behavior. But after the training, as the role of these redundant parameters is accomplished, they can be removed without affecting the performance.

Table 4: Table shows the Parameters Pruned (PP), and FLOPS Pruned (FP) results for VGG-16 on the CIFAR-10 dataset in different setups.

Model	Error (%)	Method	PP (%)	FP (%)
Baseline	6.51	–	–	–
Prun-1	6.89	training from scratch	93.3	75.14
Prun-2	7.58	training from scratch	95.9	83.43
Prun-1 (ours)	6.57	pretrained model used	93.3	75.14
Prun-2 (ours)	6.98	pretrained model used	95.9	83.43

Table 5: Table shows the results for VGG-16 on the CIFAR-10 dataset in jointly doing pruning and training from scratch (PP: Parameters Pruned, FP: FLOPS Pruned).

Model	Error (%)	Method	PP (%)	FP (%)
Baseline	6.51	–	–	–
Prun-1	6.75	jointly pruning-training	93.3	75.14
Prun-2	7.27	jointly pruning-training	95.9	83.43
Prun-1 (ours)	6.57	pretrained model used	93.3	75.14
Prun-2 (ours)	6.98	pretrained model used	95.9	83.43

4.3.3. Jointly doing pruning and training from scratch

We can also do jointly pruning and training from scratch but, the errors are only 6.75%, 7.27% for Prun-1, Prun-2 models respectively, which are much worse than our pruned models as shown in Table-5. The reason for the same is straight forward. Our filter pruning criteria are solely based on sensitivity towards a fatuous auxiliary loss and since the model is not optimal during training; hence our criteria is not well suited for jointly pruning and training from scratch.

4.4. ResNet-56 on CIFAR-10

We experiment on ResNet-56 model [40] over CIFAR-10 dataset. The ResNet-56 architecture contains three stages of the convolutional layer of size 16-32-64 where each convolution layer in each stage contains the same 2.36M FLOPS. We trained the

Table 6: Pruning results for ResNet-56 architecture on CIFAR-10 dataset. Bold values indicate the best results obtained by our method in the comparison.

Method	Error (%)	FLOPS	FLOPS Pruned (%)
Baseline	6.91	1.26×10^8	–
Weight-Sum [3]	6.90	1.12×10^8	10.40
Weight-Sum [3]	6.94	9.04×10^7	27.60
GAL-0.6 [33]	6.62	7.83×10^7	37.60
NISP [22]	6.99	–	43.61
CP [28]	8.20	–	50.00
GAL-0.8 [33]	8.42	4.99×10^7	60.20
Prun-1 (Ours)	**6.95**	$\mathbf{4.08 \times 10^7}$	**67.62**

model from scratch using the same parameters and settings proposed by [40, 3] and achieve the error rate of 6.91%.

Our method significantly outperforms various state-of-the-art approaches on ResNet-56 model over CIFAR-10 dataset. The results are shown in Table 6. Our compressed model (Prun-1) contains three stages of the convolutional layer of size 9-18-36. We achieve high pruning rate 67.62% with the 6.95% error rate, while channel pruning CP [28] has the error rate of 8.20% with only 50.00% FLOPS pruning.

4.5. VGG-16 on ImageNet

We now turn our attention to the performance of our algorithm on models that are trained on large scale datasets. We first experiment with VGG-16 network which is trained on ILSVRC-2012 [45] dataset which contains 1000 classes with 1.5 million images.

To enable a fair comparison, we follow the same setting as [32]. We perform conventional data augmentation and pre-processing techniques such as random cropping to obtain 224 x 224 images and random horizontal flipping. We use the Stochastic Gradient Descent (SGD) optimizer with momentum value as 0.9. Our experiments on VGG-16 [39] using ImageNet dataset [45] shows the state-of-art results over the other

Table 7: Pruning results for the VGG-16 model on ImageNet dataset. Our approach has minimal accuracy drop compare to state-of-art pruning approach. We use the result reported in MatConvNet: http://www.vlfeat.org/matconvnet/pretrained/. Bold values indicate the best results obtained by our method in the comparison.

Method	Top-5 Accu. (%)	Pruned FLOPS (%)
Baseline	90.10	–
GDP-0.7 [34]	89.16	51.61
GDP-0.6 [34]	88.77	58.71
Taylor criterion [16]	87.00	62.86
RNP (3X)[31]	87.57	66.67
Taylor criterion [16]	84.50	74.20
ThiNet-Conv-2 [32]	88.86	77.66
CP [28]	88.20	77.30
Prun-1 (Ours)	**89.22**	**80.00**

approaches for model compression as shown in Table-7, where Prun-1 (compressed model) is obtained after 45 epochs.

4.6. ResNet-50 on ImageNet

We perform experiment on the large-scale ImageNet [45] dataset for the ResNet-50 model. The results are shown in the Table-8 for the compressed model. Our pruned model (Prun-1) achieved 44.45% FLOPS compression while the previous method, ThiNet-70 [32], achieved 36.9% FLOPS compression with similar accuracy. Compared to ThiNet-70 we have significant better FLOPS compression.

Presence of identity mapping (skip connection) in ResNet model restrict pruning on the few layers. Since the output ($output = f(x) + x$) involves addition of x and $f(x)$, hence x and $f(x)$ need to be of same dimensions. This is the reason for pruning only two convolutional layers in each block as shown in Figure-4.

We pruned ResNet-50 from block 2a to 5c iteratively. The number of remaining filters from each layer in block 2, 3, 4 and 5 are 40, 80, 160 and 320 respectively in the pruned model. If a filter is pruned, then the corresponding channels in the batch-

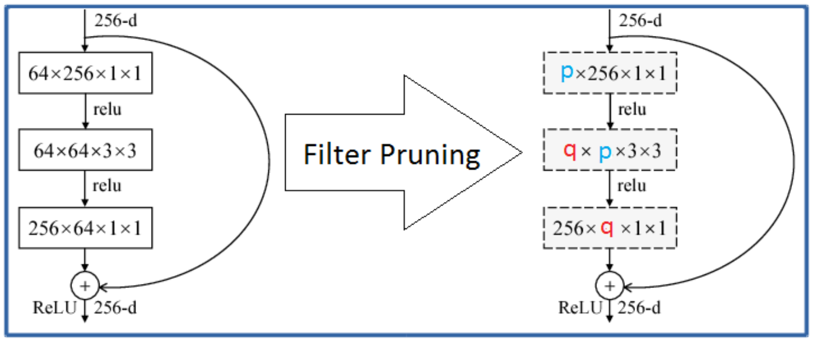

Figure 4: Figure shows our ResNet pruning strategy, where we perform pruning on the first two convolutional layers in each block to maintain the consistency over identity mapping.

normalization layer and all dependencies to that filter are also removed. We optimize equation-2 for one epoch with $\lambda = 0.000005$ to calculate filter importance ranking in each pruning iteration. We vary the learning rate in the range $[0.001, 0.00001]$ for this experiment. Our pruned model (Prun-1) is obtained after 65 epochs. Our results on ResNet pruning are shown in Table-8.

4.7. SpeedUp and Memory Size

The theoretical FLOPS based SpeedUp is not the same as practical GPU/CPU SpeedUp. The practical SpeedUp depends on intermediate layers parallelization bottleneck, the speed of I/O data transfer, etc. TRM (Total Run-time Memory) depends on the number of parameters in the final compressed model, feature maps (FM) generated at run-time, batch-size (BS), the dynamic library used by Cuda, and all supporting header-file. But from the theoretical point of view, only model parameters size and feature maps size are considered in the TRM calculations. Hence TRM can be calculated as follows:

$$TRM = MPS + (FM * 4 * BS) \tag{10}$$

Here we don‘t have control over all the parameters barring model parameters size (MPS), FM and BS. We experiment VGG-16 on the CIFAR-10 dataset to show the practical SpeedUp and Memory size. SpeedUp and TRM results are shown in the Figure-6, 5 respectively.

Table 8: Table shows the comparison of our pruned model with [32, 10] for ResNet-50 FLOPS compression on the Imagenet dataset. The accuracy of ResNet-50 is reported over validation set using 1-crop setting (https://github.com/KaimingHe/deep-residual-networks). Bold values indicate the best results obtained by our method in the comparison.

Model	Top-5 (%)	Parameters	Pruned FLOPS (%)
Baseline	92.6	25.56M	–
SSS-32 [35]	91.9	18.6M	31.1
CP [28]	90.8	–	33.3
ThiNet-70 [32]	92.1	16.94M	36.9
SFP [10]	92.0	–	41.8
GDP-0.7 [34]	91.1	–	42.0
SSS-26 [35]	90.8	15.6M	43.0
GAL-0.5 [33]	90.9	21.2M	43.0
GDP-0.6 [34]	90.7	–	51.3
ThiNet-50 [32]	90.9	12.38M	55.8
Prun-1 (Ours)	**92.2**	**15.10M**	**44.45**
Prun-2 (Ours)	**91.7**	**12.38M**	**55.83**

As shown in the above equation, TRM grows linearly with respect to Batch size. Also, TRM linearly depends on FM; hence FM is the most critical factor for compressing the run-time memory. Filter pruning methods compress the model parameters as well as the depth of the feature maps hence filter level pruning methods achieves good compression for TRM. On the other hand, approaches based on inducing sparsity in the model only reduce the MPS and the size of the FM remains the same making batch size as the bottleneck. If we have constraints on batch size, this minimizes the parallelism on the GPU which results in a drop in speed. Figure-5 explains that if we increase BS then TRM increases. Therefore we cannot afford large batches. The Figure-6 explains that for the small batch sizes, SpeedUp is degraded. Therefore for SpeedUp, we have to select a bigger BS, but then GPU or CPU memory bottleneck is there. Hence in the proposed method, we are pruning at filter level to compress FM memory.

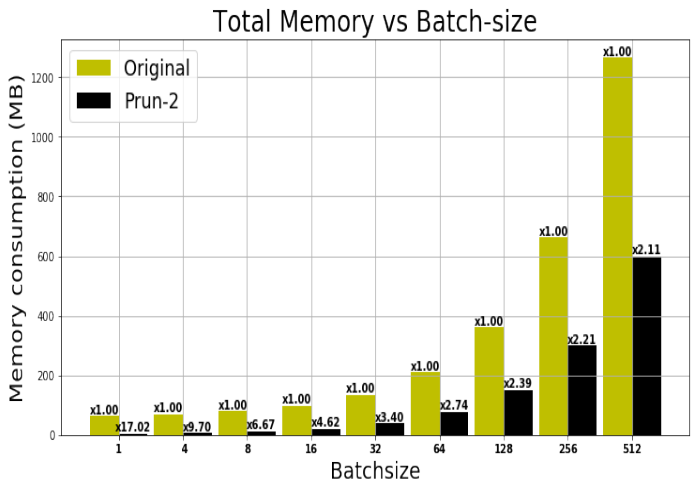

Figure 5: Figure shows Total Run Time (TRM) memory with respect to the batch size for VGG-16 models on CIFAR-10 dataset.

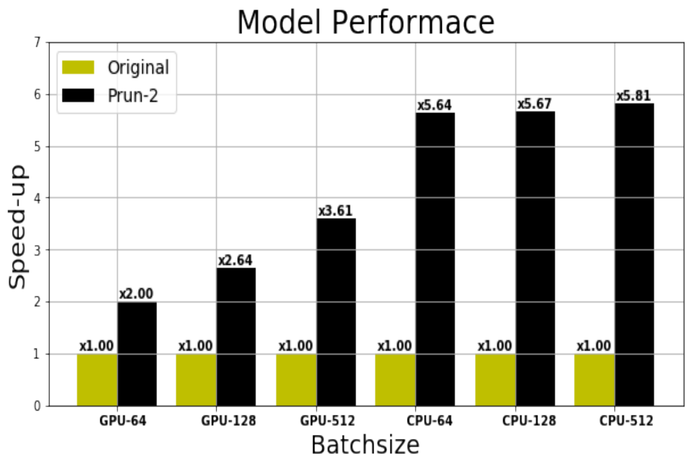

Figure 6: Figure shows the practical speed-up for the VGG-16 model on the CIFAR-10 dataset. Where i7-4770 CPU@3.40GHz CPU and TITAN GTX-1080 Ti GPU is used to calculate speed-up.

The result for CPU and GPU SpeedUp over the different batch-size is shown in the Figure-6. It is clear from the Figure-6 that with the increase in batch size, GPU has sharp SpeedUp, since on the small batch there it is not using its full parallelization capability. Although there are a lot of cores, only a few are used because the available data is limited whereas, on the bigger batch sizes, GPU uses its full parallelization capability. On the VGG-16 with 512 batch size, we have achieved 3.61X practical GPU SpeedUp while the FLOPS base theoretical SpeedUp is 6.03X. This gap is very close to CPU, and our approach gives the 5.81X practical CPU SpeedUp compare to 6.03X theoretical FLOPS base SpeedUp.

4.8. Generalization Ability

To show the generalization ability of our compressed model, we experimented on the object detection task. We have done experiments on two popular object detector SSD [43] on GTSDB data-set and Faster RCNN [42] on MS-COCO [46]. In SSD, we achieve $\sim$45$\times$ compression regarding model size with a slight improvement in AP.

In second experiment, we use the standard object detector Faster-RCNN [42] over large-scale MS-COCO [46] dataset. We use ResNet-50 as the base network for Faster RCNN.

4.8.1. SSD512 on German traffic detection benchmarks

It is well known that CNNs learn generalized features that make the prominent in applications like transfer learning. One potential doubt that could arise after pruning is that does the compact network still generalize to other tasks or has it become dataset-specific (the dataset which is used in pruning). To address this question, we empirically show that our algorithm preserves the generalization ability of the original models. We first evaluate VGG-16 Prun-2, which is compressed on CIFAR-10, as shown in Table-2. Specifically, we first train SSD512 on German traffic detection benchmarks (GTSDB) [47] dataset with ImageNet pre-trained base network. Then we substitute our pruned/compressed VGG-16 Prun 2 model as a base network and evaluate the performance.

SSD accounts for multiple object scales by adding a connection from shallow layers to the final layer. Generally, initial layers are responsible for detecting the smaller object as their receptive field is small, and the deeper layers are responsible for detecting bigger objects. As the object sizes are small in the GTSDB dataset, we found that the model overfits badly after training because of deeper layer feature maps were

Table 9: Class wise AP for SSD512-original(O) and SSD512-pruned(P) model on GTSDB dataset. Bold values indicate the best results obtained by our method in the comparison.

Model	AP				Size	Parameters
	prohibitory	mandatory	danger	mAP		
SSD512-O	96.8	86.9	87.1	90.27	98.7 MB	24.7M
SSD512-P	97.2	87.3	87.5	**90.67**	**2.2 MB (44.9**$\times$**)**	**0.56M (2.3%)**

unable to capture the objects for detection. Therefore, in our pruned SSD512 model, we discover the object from only the CONV4_3 layer (first detection layer in SSD512). We observed a slight improvement in the mAP and $\sim 45\times$ compression in model size as shown in Table 9.

4.8.2. Faster RCNN on COCO

We performed experiments on the large-scale COCO detection dataset which contain 80 object categories [46]. Here all the 80k train images and a 35k val images are used for training (trainval35K) [48]. We are reporting the detection accuracies over the 5k unused validation images (also known as minival). We trained Faster-RCNN with the image-net pre-trained ResNet-50 as the base model to get F-RCNN original as shown in Table-10.

For F-RCNN pruned, we used our pruned ResNet-50 model (Prun-1) as given in Table-8 as a base network in Faster-RCNN. It is clear from Table-10 that F-RCNN pruned model shows similar performance in all cases. However, some minor improvement in detection accuracies can be seen due to the reduction in over-fitting because of filter pruning. We used ROI Align and the stride 1 for the last block of the convolutional layer (layer4) in the base network (ResNet-50) in the Faster-RCNN implementation. Table-10 show the results in detail.

Table 10: Table shows the generalization results for Faster-RCNN on the MS-COCO dataset. In Faster-RCNN, we use our pruned ResNet-50 model (ResNet-50 Prun-1) as a base model.

Model	data	Avg. Precision, IoU:			Avg. Precision, Area			Avg. Recall, #Dets:			Avg. Recall, Area:		
		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
F-RCNN original	trainval35K	30.3	51.3	31.8	13.8	34.6	42.6	27.3	41.3	42.4	22.4	47.9	58.5
F-RCNN pruned	trainval35K	30.6	51.0	32.2	14.7	34.7	42.5	27.7	42.0	43.2	23.8	48.1	58.9

4.9. C3D on UCF101

While most of the works on pruning have concentrated on 2D CNN architectures, pruning performance on 3D CNNs are not analyzed. We show the effectiveness of our algorithm by pruning C3D architecture using UCF 101 dataset. C3D [44] is a popular architecture using 3D convolutions with temporal dimension as the 3^{rd} dimension. It takes 16 frames (let us call it as clip) of a video and outputs the activity going on in that clip. These clip predictions are further processed through either feedforward networks, simple consensus or pipeline methods like SVM for video action classification. C3D can come with different base architectures such as VGG, ResNet, etc., where it replaces 2D filters with 3D filters. We used VGG as a base network, and due to computation limitations, we constrained ourselves to clip level action classification task in which 16 frames constitutes a single clip. The network architecture details are shown in Table 11.

UCF 101 dataset contains 13320 total videos, and we used split 1 in our experiments. We used a publicly available pre-trained model with 80.2% test accuracy. For finetuning and rank computation, we used sgd optimizer with momentum. During pruning, we remove the entire 3D kernel whose gradients vary by a large amount. Considering the additional complexity in this task due to the temporal dimension in filters, our model performed very well resulting in 32.56% reduction in flops. The accuracy of the pruned model after marginal finetuning is 80% as shown in Table 11.

4.10. MobileNet on CIFAR-100

The experiments till now have shown the performance of our algorithm on considerably large models. We now show the adaptability of our algorithm even on architectures designed specifically to be compact. We picked MobileNet V2 for our experiments on

Table 11: Table shows the layer-wise pruning results and pruned model details for C3D on UCF101 (split1).

		Baseline	C3D Prun
Input Size		112x112x3x16	112x112x3x16
Layers	CONV1_1	64	31
	CONV1_2	64	64
	CONV2_1	128	128
	CONV3_1	256	176
	CONV3_2	256	176
	CONV4_1	512	352
	CONV4_2	512	352
	CONV5_1	512	352
	CONV5_2	512	512
	FC6	4096	4096
	FC7	4096	4096
	FC8	4096	101
Total parameters		78.4M	65.6M
Model Size		313.6 MB	262.6 MB
Accuracy		80.2%	80%
FLOPS		3857M	2601M

CIFAR100 dataset. To obtain the initial trained model, we used color jittering, horizontal flip, and rotation as preprocessing steps. We used sgd with momentum and learning rate scheduler during optimization. Our trained model was able to achieve 66.86% test accuracy as shown in Table 12.

Due to the structure of the Inverse Residual module in MobileNet V2, we can prune filters from the middle layer. Since the middle layer used group wise convolutions, removing a specific filter in the middle layer will lead to the removal of the corresponding filter in the previous layer. Upon pruning, we achieved an impressive reduction in flops by 36.8% even on such a compact model. We finetuned the pruned model for 1 epoch after every pruning iteration. Refer to Table 12 for more details on compressed model.

Table 12: Table shows the layer-wise pruning results and pruned model details for MobileNetV2 on CIFAR100 dataset.

		Baseline	MobileNetV2 Prun
Input Size		32x32x3	32x32x3
Layers	CONV1	32	32
	InvResidual-1	(32,32,16)	(32,32,16)
	InvResidual-2	(96,96,24)	(82,82,24)
	InvResidual-3	(144,144,24)	(117,117,24)
	InvResidual-4	(144,144,32)	(127,127,32)
	InvResidual-5	(192,192,32)	(127,127,32)
	InvResidual-6	(192,192,32)	(119,119,32)
	InvResidual-7	(192,192,64)	(162,162,64)
	InvResidual-8	(384,384,64)	(168,168,64)
	InvResidual-9	(384,384,64)	(178,178,64)
	InvResidual-10	(384,384,64)	(175,175,64)
	InvResidual-11	(384,384,96)	(192,192,96)
	InvResidual-12	(576,576,96)	(238,238,96)
	InvResidual-13	(576,576,96)	(245,245,96)
	InvResidual-14	(576,576,160)	(318,318,160)
	InvResidual-15	(960,960,160)	(448,448,160)
	InvResidual-17	(960,960,160)	(420,420,160)
	InvResidual-18	(960,960,320)	(462,462,320)
	CONV2	1280	1280
	FC/1x1CONV	100	100
Total parameters		2.41M	1.47M
Model Size		9.9 MB	6 MB
Accuracy		66.86%	67.03%
FLOPS		22.9M	14.4M

5. Choice of auxiliary loss functions

In this section, we revisit the equation (8) and analyze the choice of auxiliary loss functions. We address in detail the derivation of ranking criteria and show empirically how the choice of different loss functions affects the pruning.

5.1. Auxiliary loss function

We broadly classify the choice of auxiliary loss functions into two classes namely data dependent loss functions which depend on the distribution of input data and data independent loss functions which are functions of only weight values. Before explaining the pros and cons of these two classes, we formulate the terms required to compute the ranking.

As we are interested in calculating the variance of the gradients, the variance of gradients of the total loss function (L) w.r.t a specific weight (w_i) in equation (8) is given by:

$$\frac{\partial L}{\partial w_i} = \frac{\partial C}{\partial w_i} + R\frac{\partial D}{\partial w_i} \tag{11}$$

$$\begin{aligned} Var_{(X,R)}\left(\frac{\partial L}{\partial w_i}\right) = Var_{(X,R)}\left(\frac{\partial C}{\partial w_i}\right) + Var_{(X,R)}\left(R\frac{\partial D}{\partial w_i}\right) \\ +2 * Cov_{(X,R)}\left(\frac{\partial C}{\partial w_i}, R\frac{\partial D}{\partial w_i}\right) \end{aligned} \tag{12}$$

From the above equation, the first term becomes independent of R upon marginalization. For a given distribution of X and R, if the covariance is high in the last term, then the overall variance is dominated by the last term and verifying our hypothesis that if the addition of auxiliary loss function increases the overall variance or not become difficult (as the covariance term dominates those terms). So, an auxiliary loss functions with zero covariance in the last term will be ideal. Now, if we take D to be task dependent loss function, then it is hard to ensure the zero covariance property as we don't know the exact distribution of input data. On the other hand, any data independent loss function can easily satisfy that property.

Before moving on to show how to compute this variance efficiently, we address one more important factor. Since, our algorithm involves training the model again (with auxiliary loss) using mini-batches, the order in which the mini batches are passed can result in different weight updates and hence a different ranking of filters at the end of training with auxiliary loss which in turn affects the pruning. To address this, we make a slight change in modeling the variance above to incorporate the ordering of mini-batches. Let us say that the dataset contains n samples; then we define a vector of random variables (X_1, X_2, ..., X_n) denoted by Y. Each ordering of mini-batches thus becomes a sample of Y. This type of modeling accounts for variance due to data distribution and ordering of mini-batches. When the effective gradient of weight (weight after passing all the mini-batches - weight before) follows a normal distribution, we can rewrite equation (12) as

$$\begin{aligned} Var_{(Y,R)}\left(\frac{\partial L}{\partial w_i}\right) &= Var_{(Y,R)}\left(\frac{\partial C}{\partial w_i}\right) + Var_{(Y,R)}\left(R\frac{\partial D}{\partial w_i}\right) \\ &\quad +2 * Cov_{(Y,R)}\left(\frac{\partial C}{\partial w_i}, R\frac{\partial D}{\partial w_i}\right) \end{aligned} \tag{13}$$

The above decomposition of variance is used for analysis purpose, but in practice, there is an effective method for computing variance using folded normal distribution property as stated in [16]. Specifically, the mean of the folded normal distribution is given by:

$$\mu_{fold} = \sigma\sqrt{\frac{2}{\pi}}e^{\left(-\mu^2/2\sigma^2\right)} + \mu\left(1 - 2\Phi\left(-\frac{\mu}{\sigma}\right)\right) \tag{14}$$

where μ and σ are the mean and variance of original distribution respectively. Now, μ_{fold} tends to $\sigma\sqrt{\frac{2}{\pi}}$ if either of the conditions are met:

1. μ tends to 0
2. μ non-zero but $\frac{\mu}{\sigma}$ tends to 0 due to high variance

In our case, we are interested in the distribution of gradients. While, theoretically, it is safe to assume that the mean of gradients tends to zero by the end of the training, in practice, this need not be the scenario due to some regularization methods like early stopping. In such cases, the condition (2) helps in approximating the variance and is in line with our hypothesis/loss function formulation. Hence, we use the expectation of absolute values of gradients to approximate the variance. As the gradients are proportional to change in weights and from the modeling used in equation (13), we formulate the ranking criteria for any general (data independent) loss function as:

$$FI_{\mathcal{L}_i} = \left\{ \frac{|m_j^i - f_j^i| + \sum_{perm} |u_j^i - v_j^i|}{|f_j^i|} : \forall j \in \{1, \ldots, n_i\} \right\}$$

where v_j^i is filter before training with certain permutation of the dataset and u_j^i is filter after training with certain permutation of dataset. The denominator in the above formulation is to compensate for the weight magnitude differences among the filters in a layer. In practice, if the training converges for the pretrained model, then only the first term in the numerator remains. With a little manipulation, it can be seen that the ranking criteria in equation (3) is a special case/approximation when the auxiliary loss function is chosen as in equation (1) and when training converges.

5.2. Loss function experiments

While any task independent loss should ideally work for the role of the auxiliary loss function, we observed that some loss functions are more robust compared to others in terms of the coefficient tuning of auxiliary loss term. We show our findings on VGG16 for CIFAR 10 dataset. We plot the accuracies for each cycle, where a cycle is a combination of 1 pruning iteration followed by 3 finetuning iterations. We used three loss functions l1, l2, and l3, where l1 loss is defined in equation (1), l2 loss pushes all weight values to 1, and l3 loss pushes all weight values to 0 (same as L_1 norm). We analyze the effect of these loss functions using three different values of λ as shown in fig 7, 8 and 9.

From the above plots, one can observe the consistency of l1 auxiliary loss function across different beta values whereas l2 and l3 loss functions perform poorly compared to l1 on increasing the beta value. We hypothesize that this may be due to activation preserving nature (in case of ReLU) of l1 auxiliary loss whereas l2 has high potential to drag zero activations (negative input values of ReLU) to positive values. As the role of an ideal auxiliary function is to preserve the accuracy of the original task while perturbing the unimportant weights, such change of activations (in case of l2) would affect the performance on the original task and hence lower accuracy as the number of

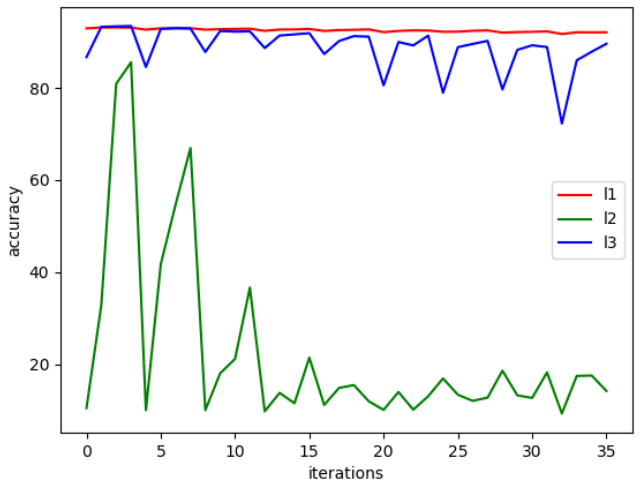

Figure 7: Pruning results for λ = 1e-2.

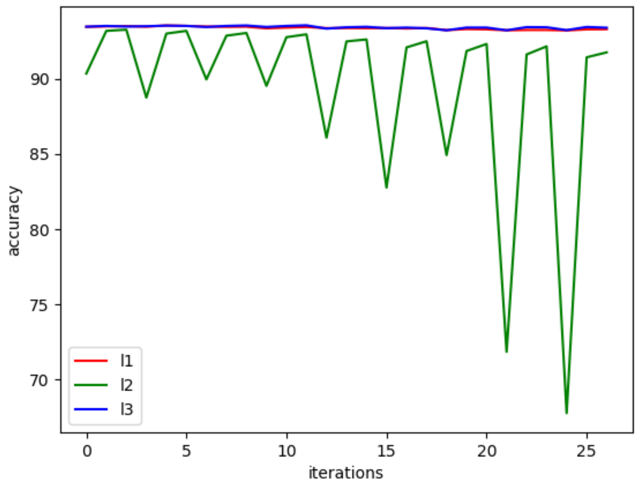

Figure 8: Pruning results for λ = 1e-3.

pruning iterations increase. The effect of l3 is similar to l1 as it also preserves activation sign barring a few instances (when the weights are close to 0) where optimizer factors like momentum can alter the activations.

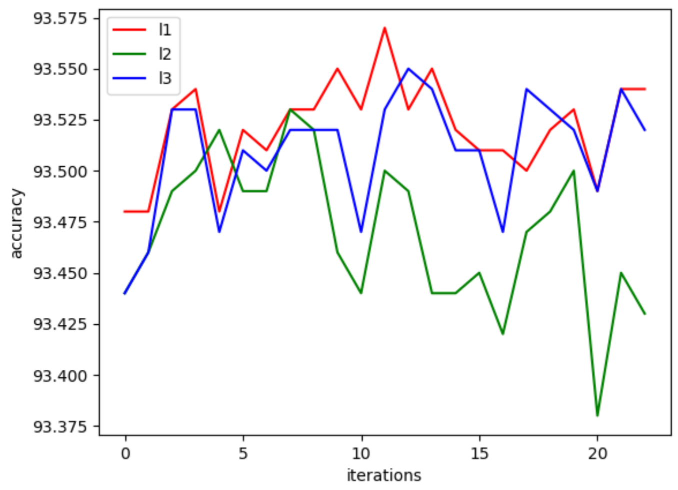

Figure 9: Pruning results for λ = 1e-4.

6. Conclusion

In this work, we have proposed a novel strategy that allows for the pruning of filters using the idea of the ranking of filters using fatuous auxiliary loss functions. This method we believe obtains a meaningful measure of filter importance that is based on the robustness of the filters to the fatuous loss function. We have demonstrated a significant compression in terms of FLOPS and Run Time GPU memory footprint. We have evaluated our method on various architectures like LeNet, VGG, Resnet, MobileNet, SSD512, Faster-RCNN, and C3D. Our method can be used in conjunction with other pruning methods such as binary/quantized weights, and Low-rank approximation to get further boost in SpeedUp. The experimental results show that our method achieves state-of-art results on LeNet, ResNet and VGG architecture. Moreover, we demonstrated that our pruning method generalizes well across tasks by pruning an architecture on one task and achieving competitive results using the same pruned model on another (but related) task. The use of data independent loss function allows for our approach to be flexible and can be easily adapted for a new task by using the fatuous auxiliary loss functions described in the paper.

References

[1] G. Huang, S. Liu, L. Van der Maaten, K. Q. Weinberger, Condensenet: An efficient densenet using learned group convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2752–2761.

[2] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size, arXiv preprint arXiv:1602.07360.

[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, in: International Conference on Learning Representations, 2017.

[4] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149.

[5] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: International Conference on Machine Learning, 2015, pp. 2285–2294.

[6] C. Louizos, K. Ullrich, M. Welling, Bayesian compression for deep learning, in: Advances in Neural Information Processing Systems, 2017, pp. 3288–3298.

[7] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision, Springer, 2016, pp. 525–542.

[8] R. Abbasi-Asl, B. Yu, Structural compression of convolutional neural networks based on greedy filter pruning, arXiv preprint arXiv:1705.07356.

[9] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in neural information processing systems, 2014, pp. 1269–1277.

[10] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, AAAI Press, 2018, pp. 2234–2240.

[11] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 5058–5066.

[12] X. Zhang, J. Zou, X. Ming, K. He, J. Sun, Efficient and accurate approximations of nonlinear convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and pattern Recognition, 2015, pp. 1984–1992.

[13] D. Mittal, S. Bhardwaj, M. M. Khapra, B. Ravindran, Recovering from random pruning: On the plasticity of deep convolutional neural networks, in: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2018, pp. 848–857.

[14] D. Mittal, S. Bhardwaj, M. M. Khapra, B. Ravindran, Studying the plasticity in deep convolutional neural networks using random pruning, Machine Vision and Applications 30 (2) (2019) 203–216.

[15] J.-H. Luo, J. Wu, An entropy-based pruning method for cnn compression, arXiv preprint arXiv:1706.05791.

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, in: International Conference on Learning Representations, 2017.

[17] J. Ye, X. Lu, Z. Lin, J. Z. Wang, Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers, in: International Conference on Learning Representations, 2018.

[18] B. Hassibi, D. G. Stork, Second order derivatives for network pruning: Optimal brain surgeon, in: Advances in neural information processing systems, 1993, pp. 164–171.

[19] Y. LeCun, J. S. Denker, S. A. Solla, Optimal brain damage, in: Advances in neural information processing systems, 1990, pp. 598–605.

[20] P. Singh, V. S. R. Kadi, N. Verma, V. P. Namboodiri, Stability based filter pruning for accelerating deep cnns, in: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2019, pp. 1166–1174.

[21] K. Neklyudov, D. Molchanov, A. Ashukha, D. P. Vetrov, Structured bayesian pruning via log-normal multiplicative noise, in: Advances in Neural Information Processing Systems, 2017, pp. 6775–6784.

[22] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, L. S. Davis, Nisp: Pruning networks using neuron importance score propagation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9194–9203.

[23] V. Lebedev, V. Lempitsky, Fast convnets using group-wise brain damage, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2554–2564.

[24] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, W. J. Dally, Exploring the granularity of sparsity in convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017, pp. 13–20.

[25] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: Advances in neural information processing systems, 2016, pp. 2074–2082.

[26] H. Hu, R. Peng, Y.-W. Tai, C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, arXiv preprint arXiv:1607.03250.

[27] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744.

[28] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[29] J. M. Alvarez, M. Salzmann, Learning the number of neurons in deep networks, in: Advances in Neural Information Processing Systems, 2016, pp. 2270–2278.

[30] H. Zhou, J. M. Alvarez, F. Porikli, Less is more: Towards compact cnns, in: European Conference on Computer Vision, Springer, 2016, pp. 662–677.

[31] J. Lin, Y. Rao, J. Lu, J. Zhou, Runtime neural pruning, in: Advances in Neural Information Processing Systems, 2017, pp. 2181–2191.

[32] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, W. Lin, Thinet: pruning cnn filters for a thinner net, in: IEEE transactions on pattern analysis and machine intelligence, IEEE, 2018.

[33] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, Towards optimal structured cnn pruning via generative adversarial learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2790–2799.

[34] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, B. Zhang, Accelerating convolutional networks via global & dynamic filter pruning., in: IJCAI, 2018, pp. 2425–2432.

[35] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 304–320.

[36] H. Miao, A. Li, L. S. Davis, A. Deshpande, Towards unified data and lifecycle management for deep learning, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 571–582.

[37] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, arXiv preprint arXiv:1405.3866.

[38] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[39] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[40] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[41] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[42] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, in: Advances in neural information processing systems, 2015, pp. 91–99.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, Ssd: Single shot multibox detector, in: European conference on computer vision, Springer, 2016, pp. 21–37.

[44] D. Tran, L. Bourdev, R. Fergus, L. Torresani, M. Paluri, Learning spatiotemporal features with 3d convolutional networks, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 4489–4497.

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, International journal of computer vision 115 (3) (2015) 211–252.

[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft coco: Common objects in context, in: European conference on computer vision, Springer, 2014, pp. 740–755.

[47] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, C. Igel, Detection of traffic signs in real-world images: The german traffic sign detection benchmark, in: The

2013 international joint conference on neural networks (IJCNN), IEEE, 2013, p. 1288.

[48] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature pyramid networks for object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2117–2125.