Approximated Oracle Filter Pruning for Destructive CNN Width Optimization

Xiaohan Ding [1] **Guiguang Ding** [1] **Yuchen Guo** [1] **Jungong Han** [2] **Chenggang Yan** [3]

Abstract

It is not easy to design and run Convolutional Neural Networks (CNNs) due to: 1) finding the optimal number of filters (i.e., the width) at each layer is tricky, given an architecture; and 2) the computational intensity of CNNs impedes the deployment on computationally limited devices. Oracle Pruning is designed to remove the unimportant filters from a well-trained CNN, which estimates the filters' importance by ablating them in turn and evaluating the model, thus delivers high accuracy but suffers from intolerable time complexity, and requires a given resulting width but cannot automatically find it. To address these problems, we propose Approximated Oracle Filter Pruning (AOFP), which keeps searching for the least important filters in a binary search manner, makes pruning attempts by masking out filters randomly, accumulates the resulting errors, and finetunes the model via a multi-path framework. As AOFP enables simultaneous pruning on multiple layers, we can prune an existing very deep CNN with acceptable time cost, negligible accuracy drop, and no heuristic knowledge, or re-design a model which exerts higher accuracy and faster inference.

1. Introduction

Convolutional Neural Networks (CNNs) have become an important tool for many real-world applications and related research areas (Collobert & Weston, 2008; LeCun et al., 1990a; 1995). Nowadays, designing a CNN usually means a tiring exploration in a vast design space, which usually includes the usage of non-linearities (ReLU, sigmoid or none), downsampling (max / average pooling or stride-2 convolution), shortcut connections (He et al., 2016), etc. With so many hyper-parameters in consideration, we still have to make a hard decision every time we use a convolutional layer: the number of filters, i.e., the width of the layer. Since an unnecessarily wide conv layer usually leads to meaningless parameters, heavy computational burdens, and overfitting, we wish to set a proper width for each layer, which is inherently tricky. In modern CNN architectures, some practical guidelines on the number of filters are followed. Taking VGG (Simonyan & Zisserman, 2014) for example, when the feature maps are spatially downsampled by $2\times$, the number of filters becomes $2\times$, so that the computational burdens of each layer are kept roughly the same. Apparently, such guidelines leave much room to improve on the layer width for better accuracy and efficiency.

In this paper, destructive CNN width optimization refers to the process which takes a well-trained tidy CNN as input and produces an optimized one where some useless filters are removed. In this context, our method can be categorized into filter pruning, a.k.a. channel pruning (He et al., 2017) or network slimming (Liu et al., 2017), a family of CNN compression techniques, which features three strengths. **1) Universality:** filter pruning can handle any kinds of CNNs, making no assumptions on the application field, the network architecture or the deployment platform. **2) Effectiveness:** filter pruning effectively reduces the floating-point operations (FLOPs) of the network, which serve as the main criterion of computational burdens. When a filter is pruned, its output channel and the corresponding input channels of the following layer are removed. That is, when several conv layers stacked together are pruned respectively, the total FLOPs are reduced quadratically. **3) Orthogonality:** filter pruning simply produces a thinner network with no customized structure or extra operation, which is orthogonal to other model compression and acceleration techniques.

A common paradigm of filter pruning is to evaluate the importance of filters by some means (Polyak & Wolf, 2015; Hu et al., 2016; Li et al., 2016; Molchanov et al., 2016; Abbasi-Asl & Yu, 2017; Anwar et al., 2017; Yu et al., 2018), such that the accuracy is not damaged severely by the removal of the less important filters and can be recovered by finetuning. Apparently, the quality of the filter importance evaluation

[1]Beijing National Research Center for Information Science and Technology (BNRist); School of Software, Tsinghua University, Beijing, China. Email: dxh17@mails.tsinghua.edu.cn. [2]WMG Data Science, University of Warwick, Coventry, United Kingdom [3]InstituteofInformationandControl, Hangzhou Dianzi University, Hangzhou, China. Correspondence to: Yuchen Guo <yuchen.w.guo@gmail.com>, Jungong Han <jungonghan77@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

plays a vital role in the entire pipeline. By recognizing the unimportant filters, we diminish the accuracy drop, such that it becomes easier for the finetuning process to restore the accuracy. In this sense, the importance of filter can be defined by the network's accuracy drop with the filter ablated. If we ablate a filter (i.e., mask out its outputs), test the model on an assessment dataset, and observe a severe accuracy drop, and then the filter can be defined as important.

The most straightforward and accurate algorithm to prune filters greedily by importance, which is referred to as *Oracle Pruning* (Molchanov et al., 2016), can be implemented in a trial-and-error manner. For a specific layer, we ablate a filter, test the model on the assessment dataset, record the accuracy drop as the importance score, i.e., mean accuracy reduction (Abbasi-Asl & Yu, 2017) or loss increase (Molchanov et al., 2016), then restore the filter and move on to the next filter. When all the filters have been tested, we prune the filter with the least importance score. However, when we start to prune the next filter, the relative importance of the remaining filters may have been changed (Sect. 4.1), so they should be tested in the same manner again. In this way, we can slim a layer by always removing the filter with the least importance score until we are satisfied with the trade-off between accuracy and efficiency. However, for today's CNNs where a conv layer can comprise hundreds of filters, the time complexity of Oracle Pruning is intolerable. In order to acquire the importance score of filters with reasonable time cost, some heuristic methods (Li et al., 2016; Hu et al., 2016; Molchanov et al., 2016) have been proposed, which suffer from inferior quality of importance estimation, compared to Oracle Pruning (Fig. 3). Of note is that "oracle" described here is only the most accurate *greedy* pruning method. A better oracle would consider all combinations of pruned filters, but of course, this is extremely expensive.

In this paper, we propose Approximated Oracle Filter Pruning (AOFP), a multi-path training-time filter pruning framework (Fig. 1), where we keep searching for the next filters to prune in a binary search manner and finetuning the model in the meantime, which features high quality of importance estimation, reasonable time complexity and no need for heuristic knowledge. The codes are available at `https://github.com/ShawnDing1994/AOFP`. Our contributions are summarized as follows.

- We improve unimportant filters selection by analyzing the outputs of the next layer only, rather than the final outputs. However, instead of solving a linear regression problem layer-by-layer (Luo et al., 2017; 2018), we ablate the filters randomly, then compute and accumulate the change in the next layer's outputs, which is referred to as Damage Isolation. Doing so enables not only the faster importance estimation but also mutually independent pruning on all the layers simultaneously. We have shown that the structural change of every layer in CNNs can be separately measured using only local information, which may inspire future researches.
- Our experiments on CIFAR-10 and ImageNet have shown the effectiveness of AOFP in significantly reducing the parameters and FLOPs of several very deep off-the-shelf CNNs including ResNet-152.
- We propose Destructive CNN Re-design, a CNN design paradigm, which aims at optimizing the width of convolutional layers in order for higher accuracy and faster inference. E.g., the first two layers of VGG both have 64 filters, but we found out that 44 and 80 work better. This process can be used as a final refining step before a model is publicly released or deployed.

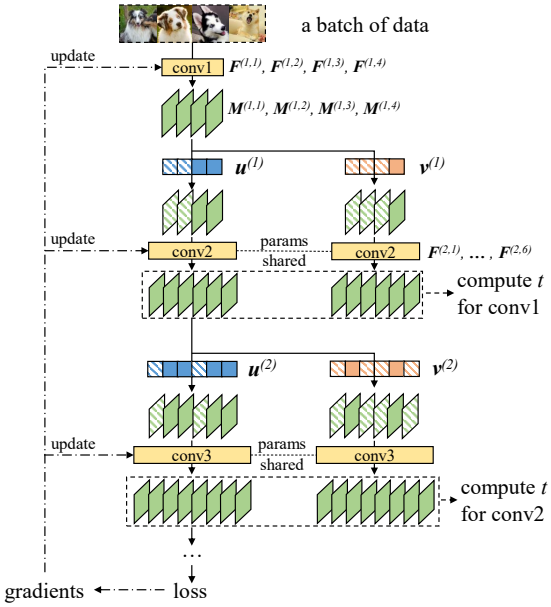

Figure 1. Overview of AOFP, where conv1 and conv2 in a CNN are being pruned simultaneously for example. Filters $\boldsymbol{F}^{(1,1)}$, $\boldsymbol{F}^{(1,2)}$, $\boldsymbol{F}^{(2,1)}$, $\boldsymbol{F}^{(2,4)}$ have already been masked out, and the algorithm is trying to pick the next unimportant one out of $\{\boldsymbol{F}^{(1,3)}, \boldsymbol{F}^{(1,4)}\}$ and two out of $\{\boldsymbol{F}^{(2,2)}, \boldsymbol{F}^{(2,3)}, \boldsymbol{F}^{(2,5)}, \boldsymbol{F}^{(2,6)}\}$.

2. Related Work

Numerous researches (LeCun et al., 1990b; Hassibi & Stork, 1993; Castellano et al., 1997; Han et al., 2015b; Guo et al., 2016) have proved it feasible to remove a large portion of parameters from neural networks without significant accuracy drop. Furthermore, by removing filters instead of sporadic connections from CNNs, we transform the wide convolutional layers into narrower ones to reduce the FLOPs, memory footprint and power consumption significantly.

A straightforward way of filter pruning is to remove unimportant filters from a well-trained model. Some researchers have discussed various metrics to measure filter importance.

E.g., a prior work prunes filters by the accuracy reduction with the filter ablated (Abbasi-Asl & Yu, 2017); magnitude-based pruning (Li et al., 2016) considers the filters with larger magnitude more likely to be important; APoZ-based pruning (Hu et al., 2016) calculates the percentage of zeros in the activated feature maps; some researchers use Taylor-expansions (Molchanov et al., 2016). There are also some inspiring works which pick up filters by no importance score but by the channel contribution variance (Polyak & Wolf, 2015) or the Lasso regression (He et al., 2017).

A major drawback of the existing methods is the requirement for heuristic knowledge. **1)** The filter importance metrics (Li et al., 2016; Molchanov et al., 2016; Hu et al., 2016) are heuristic, as it is not clear why the proposed metrics reflect the inherent importance of filters, and it is hard to judge if a heuristic method outperforms another. **2)** For iterative filter pruning methods, the granularity, i.e., the number of filters pruned at each step, is a key manually set hyper-parameter and a critical trade-off to be solved. The fewer filters are discarded once, the less damage is done to the model, which means the less finetuning time is required for the network to restore the accuracy; but more steps are needed to reach a satisfactory compression rate. **3)** It is difficult to decide when to stop pruning, i.e., the resulting width of each layer. Many works (Li et al., 2016; Hu et al., 2016; He et al., 2017) have shown that some layers can be pruned by a large ratio without accuracy drop, but some layers are sensitive, which makes it hard to set layer-wise termination conditions.

Apart from pruning by importance, some other methods train the model under certain constraints (e.g., group Lasso (Roth & Fischer, 2008)) in order to zero out some filters (Alvarez & Salzmann, 2016; Wen et al., 2016; Ding et al., 2018) or make them identical for removal (Ding et al., 2019).

Moreover, some other CNN compression and acceleration techniques have also been intensively studied, including tensor low rank expansion (Jaderberg et al., 2014), parameter quantization (Han et al., 2015a), knowledge distillation (Hinton et al., 2015), DCT-based fast convolution (Wang et al., 2016), feature map compacting (Wang et al., 2017b), etc. Of note is that AOFP is complementary to these methods.

3. Approximated Oracle Filter Pruning

3.1. Formulation

Let i be the layer index, $\boldsymbol{M}^{(i)} \in \mathbb{R}^{h_i \times w_i \times c_i}$ be an $h_i \times w_i$ feature map with c_i channels and $\boldsymbol{M}^{(i,j)} = \boldsymbol{M}^{(i)}_{:,:,j}$ be the j-th channel. The parameters of conv layer i with kernel size $r_i \times s_i$ reside in the kernel tensor $\boldsymbol{K}^{(i)} \in \mathbb{R}^{r_i \times s_i \times c_{i-1} \times c_i}$ and the bias term $\boldsymbol{b}^{(i)} \in \mathbb{R}^{c_i}$, so we use $\boldsymbol{P}^{(i)} = (\boldsymbol{K}^{(i)}, \boldsymbol{b}^{(i)})$ to denote the parameters of layer i. In this paper, filter j at layer i refers to the tuple comprising the trained parameters related to the output channel j of layer i, $\boldsymbol{F}^{(i,j)} = (\boldsymbol{K}^{(i)}_{:,:,:,j}, b^{(i)}_j)$, and we denote the set of all such filters at layer i by $\mathcal{F}_i$. This layer takes $\boldsymbol{M}^{(i-1)} \in \mathbb{R}^{h_{i-1} \times w_{i-1} \times c_{i-1}}$ as input and outputs $\boldsymbol{M}^{(i)}$. Let $*$ be the 2-D convolution operator, an arbitrary output channel j is

$$\boldsymbol{M}^{(i,j)} = \sigma^{(i)}\left(\left(\sum_{k=1}^{c_{i-1}} \boldsymbol{M}^{(i-1,k)} * \boldsymbol{K}^{(i)}_{:,:,k,j}\right) + b^{(i)}_j\right), \quad (1)$$

where $\boldsymbol{K}^{(i)}_{:,:,k,j}$ is the k-th input channel of the j-th filter, i.e., a 2-D convolution kernel, function $\sigma^{(i)}$ denotes the possible following operations such as non-linearities. For simplicity, we rewrite this transformation as a function $\zeta^{(i)}$,

$$\boldsymbol{M}^{(i)} = \zeta^{(i)}(\boldsymbol{M}^{(i-1)}, \mathcal{F}_i)\,. \quad (2)$$

Importance-based filter pruning methods define the importance of filters in terms of some measures, score the filters by some means, then prune the unimportant parts and reconstruct the network using the remaining parameters. Let T be the filter importance score value, δ be the threshold and $\mathcal{I}_i$ be the filter index set of layer i (e.g., if conv5 has four filters, then $\mathcal{I}_5 = \{1, 2, 3, 4\}$), the remaining set, i.e., the index set of the filters which survive the pruning, is $\mathcal{R}_i = \{j \in \mathcal{I}_i \mid T(\boldsymbol{F}^{(i,j)}) > \delta\}$. We prune the other filters by reconstructing the network using the remaining parameters sliced from the original kernel and bias term. That is,

$$\boldsymbol{P}^{(i)} \leftarrow (\boldsymbol{K}^{(i)}_{:,:,:,\mathcal{R}_i}, \boldsymbol{b}^{(i)}_{\mathcal{R}_i})\,. \quad (3)$$

If the conv layer is followed by a batch normalization (Ioffe & Szegedy, 2015) layer, its parameters should be handled in the same way as the bias term $\boldsymbol{b}$. The input channels of the following layer corresponding to the pruned filters should also be discarded,

$$\boldsymbol{P}^{(i+1)} \leftarrow (\boldsymbol{K}^{(i+1)}_{:,:,\mathcal{R}_i,:}, \boldsymbol{b}^{(i+1)})\,. \quad (4)$$

3.2. Rethinking Oracle Pruning

In this subsection, we focus on the situation where we prune q filters from a layer which originally has c filters. Oracle Pruning assesses a filter's importance by looking at the model's accuracy drop when the filter is ablated. Formally, let $\mathcal{F}$ be the original filter set of the CNN, $L(x, y, \mathcal{F})$ be the accuracy-related loss value (e.g., cross-entropy loss for classification tasks) generated with the given filter set, X and Y be the examples and labels of the assessment dataset, which is a subset of the training dataset, the *scoring process* aims to obtain the importance score for each filter $\boldsymbol{F}$ by

$$T(\boldsymbol{F}) = \sum_{(x,y)\in(X,Y)} \left(L(x, y, \mathcal{F} - \boldsymbol{F}) - L(x, y, \mathcal{F})\right), \quad (5)$$

where $L(x, y, \mathcal{F} - \boldsymbol{F})$ is the loss value computed without filter $\boldsymbol{F}$, i.e., with the corresponding feature map channel

removed or equivalently masked out. In this way, we can slim a layer by always removing the filter with the least T value and re-scoring the remaining filters for q times. Compared to the heuristic approaches, where T is computed in other ways, an obvious strength of Oracle Pruning is the accuracy, while its weakness is the high time complexity. Specifically, using an assessment dataset of γ examples, the time complexity of Oracle Pruning is $O(cq\gamma)$, because we need to ablate every remaining filter in turn ($O(c)$) then test on the assessment dataset ($O(\gamma)$) to pick up a single filter to prune, and this scoring process is conducted for q times.

A straightforward way to alleviate the computational burdens is to prune several filters once at a time, trading accuracy for efficiency. However, as all the filters at a conv layer compose a highly non-linear system (Mozer & Smolensky, 1989), removing a filter can affect the relative importance of other filters, inevitably resulting in poor accuracy (Fig. 3). We refer to the number of filters pruned at a time as the *granularity* g. Except for lower accuracy, another downside of granular pruning is that we introduce an extra hyper-parameter g, which may require heuristic knowledge and human efforts to tune. For example, we can estimate the redundancy of a layer by pruning some filters and observing the accuracy drop in advance, such that we set g to a larger value to reduce the time cost if the layer seems to be highly redundant, or adopt a smaller g to prune more carefully.

3.3. Damage Isolation

The essence of Oracle Pruning is to observe the consequences of the temporary removal of filters, i.e., to observe the feedback of many pruning attempts, which is generated by computing the final loss value. In this way, even when we are trying to prune a low-level layer, we still need to feed the input data through the entire network to obtain the feedback. Even worse, using such a feedback loop, we can only deal with one layer at a time, because as we ablate the filters in a specific layer, the subsequent information flow of the network is changed, such that the scoring processes of the higher-level layers are affected. Therefore, we seek to shorten the feedback loop for faster inference and mutually independent parallel filter scoring on every layer.

Our proposed approach is based on an intuition that a CNN can be viewed as a state machine, where the feature maps (states) are transformed by the operations performed by conv layers (Eq. 2). So essentially, the change in the filters at layer i, i.e., the change in $\boldsymbol{M}^{(i)}$, is isolated by the subsequent layer $i+1$, because layer $i+2$ and the higher-level layers cannot see the change in $\boldsymbol{M}^{(i)}$. Taking the extreme case for example, if we prune some filters at layer i, but observe no difference in $\boldsymbol{M}^{(i+1)}$, we can claim that the pruning does no damage to the model because the input states to the remaining part of the network are not changed.

Inspired by this, we propose to calculate the approximated importance score T' based on the output of the next layer,

$$T'(\boldsymbol{F}) = \frac{1}{|X|} \sum_{x \in X} t(\boldsymbol{F}, x)\,, \tag{6}$$

where $\boldsymbol{F}$ is a filter at layer i, t is the *isolated damage sample* which reflects how much the output of layer $i+1$ on input example x is deviated by the pruning attempt on $\boldsymbol{F}$,

$$t(\boldsymbol{F}, x) = \frac{||\boldsymbol{M}^{(i+1)}(x) - \zeta^{(i+1)}(\boldsymbol{M}^{(i)}_{\boldsymbol{F}}(x), \mathcal{F}^{(i+1)})||_2^2}{||\boldsymbol{M}^{(i+1)}(x)||_2^2}\,. \tag{7}$$

Here $\boldsymbol{M}^{(i)}_{\boldsymbol{F}}(x)$ is the output of layer i derived without $\boldsymbol{F}$,

$$\boldsymbol{M}^{(i)}_{\boldsymbol{F}}(x) = \zeta^{(i)}(\boldsymbol{M}^{(i-1)}(x), \mathcal{F}_i - \boldsymbol{F})\,. \tag{8}$$

Except for Euclidean distance, other distance functions may work as well, which are beyond the scope of this paper.

3.4. Multi-path Training-time Pruning Framework

It is common to finetune the whole model after each time of pruning (Li et al., 2016; Molchanov et al., 2016; Hu et al., 2016; Abbasi-Asl & Yu, 2017), i.e., the scoring and finetuning processes are serial. To reduce the time cost, we propose a multi-path training-time pruning framework (Fig. 1) to parallelize the scoring and finetuning.

Specifically, when we prune a certain conv layer i, the computation flow after it is split into two paths, which are referred to as the base path and the scoring path, respectively. E.g., Fig. 1 shows two scoring paths which each contain only one conv layer (conv2, conv3) as we are pruning conv1 and conv2 simultaneously. The base path forwards the outputs of layer i through a *base mask* $\boldsymbol{u}^{(i)} \in \mathbb{R}^{c_i}$ initialized as $\mathbf{1}$. The j-th channel of the output of the next layer becomes

$$\boldsymbol{M}^{(i+1,j)} = \sigma^{(i+1)}((\sum_{k=1}^{c_i} u_k^{(i)} \boldsymbol{M}^{(i,k)} * \boldsymbol{K}^{(i+1)}_{:,:,k,j}) + b_j^{(i+1)})\,. \tag{9}$$

It is obvious that setting $u_k^{(i)} = 0$ is equivalent to removing the k-th filter at layer i. At the endpoint of the base path, the original loss value is calculated, the gradients are derived and the model parameters are updated. Meanwhile, the scoring path goes through a *scoring mask* $\boldsymbol{v}^{(i)} \in \mathbb{R}^{c_i}$, the masked $\boldsymbol{M}^{(i)}$ is fed into layer $i+1$, then the isolated damage sample t is computed and stored in memory.

During the training process, for each batch of input data, we randomly set some bits in the scoring mask to zero, such that the corresponding filters are ablated on the scoring path but still kept on the base path. The t value is computed by comparing the corresponding feature maps on the base and scoring paths, and if it is large, we learn that the ablated filters are important for the current batch of data. With more

and more samples collected, we become more and more confident to tell which filters are the least important. When enough samples have been collected, we approximate T' for each filter j in the layer by

$$\hat{T}(\boldsymbol{F}^{(i,j)}) = mean(\mathcal{T}^{(i,j)}), \tag{10}$$

where $\mathcal{T}^{(i,j)}$ is a set which records all the t values collected with filter j ablated.

With all the filters scored, we pick up g filters with the lowest $\hat{T}$ value, fix the corresponding bits in $\boldsymbol{u}^{(i)}$ and $\boldsymbol{v}^{(i)}$ to zero, such that the filters are masked out permanently. Of note is that changing some bits in $\boldsymbol{u}^{(i)}$ affects the back-propagation through the base path, thus the network's accuracy will be restored by finetuning. In the meantime as finetuning, we choose the next g filters to prune through the scoring path. We refer to the process of choosing one or more filters to prune (in the meantime as recovering the damage caused by the last pruning) as a *move*. When some termination conditions have been met, we remove the filters according to the base mask by Eq. 3, 4 with $\mathcal{R}_i = \{j|u_j^{(i)} = 1\}$, such that the layer is slimmed with no further accuracy drop.

Such a multi-path framework enables parallel scoring and pruning on multiple layers, i.e., we can prune layer i according to the outputs of layer $i+1$ and prune layer $i+1$ according to layer $i+2$ simultaneously. Scoring a low-level conv layer does not affect the higher-level ones because every scoring process compares the outputs of the scoring path with the base path unaffected by the pruning attempts (i.e., the changes of the scoring masks) on the previous layers.

Note that though we randomly mask out channels in a dropout-like (Srivastava et al., 2014; Molchanov et al., 2017) manner, we do not rescale the remaining parts for compensation as we do when using dropout for regularization.

3.5. Binary Filter Search

In this subsection, we discuss and solve three problems of the proposed framework. **1)** On a specific layer, the finetuning process makes the parallel scoring inaccurate. When g filters have been masked out permanently, i.e., the corresponding bits in the two masks have been fixed to zero, the network needs a period to recover, during which the filter importance assessment is not accurate. Namely, since the remaining filters vary during the finetuning period after the last pruning, the t values obtained in this period do not accurately reflect the actual importance of the filters which are being scored. **2)** The optimal value of the granularity g is hard to resolve. **3)** It is heuristic to decide when to stop pruning, as we do not know the optimal resulting width.

Inspired by the idea of *incremental refinement*, we propose to search for the least important filters in a binary search manner. Concretely, at the beginning of each move, all the remaining filters compose the search space $\mathcal{A}$. We first score every filter in $\mathcal{A}$ and pick up $|\mathcal{A}|/2$ filters as the picked set $\mathcal{B}$ which are most likely to be unimportant. Though the network has not become stable (if this is not the first move), the assessment is not accurate indeed, but accurate enough for such a coarse-grained search. When $\mathcal{B}$ is obtained, the network finetuned through the base path has become more stable, so we abandon the collected samples and start a more fine-grained search by letting $\mathcal{A} \leftarrow \mathcal{B}$, searching for the less important half of the picked set (i.e., a quarter of the last search space). As we use imprecise samples to do coarse searches and high-quality samples to search accurately, the samples collected in the meantime as finetuning are fully utilized, and the accuracy of importance scoring is guaranteed. We refer to the number of collected t samples needed to complete one step of binary search as the *search cost* ϕ.

Except for accurate assessment, Binary Filter Search also helps the decision of the granularity g and the judgment of the termination conditions in a natural way, freeing us from heavy works on layer sensitivity analysis experiments (He et al., 2017; Li et al., 2016; Hu et al., 2016) and heuristic

Algorithm 1 Approximated Oracle Filter Pruning

Input: the target layer i of the original CNN, refinement threshold θ, search cost ϕ
Base mask $\boldsymbol{u} \leftarrow \mathbf{1}$
while True **do**
  Search space $\mathcal{A} \leftarrow \{j|u_j = 1\}$
  repeat
    Loss record set $\mathcal{T}^{(i,j)} \leftarrow \{\}, \forall\, j \in \mathcal{A}$
    repeat
      Randomly choose $|\mathcal{A}|/2$ elements out of $\mathcal{A}$ as the ablated filter index set $\mathcal{H}$
      Initialize $\boldsymbol{v} \leftarrow \boldsymbol{u}$, let $v_j \leftarrow 0, \forall\, j \in \mathcal{H}$
      Generate and forward a batch of input data, compute the t value as Eq. 7, record it for the ablated filters by $\mathcal{T}^{(i,j)} \leftarrow \mathcal{T}^{(i,j)} \cup \{t\}, \forall\, j \in \mathcal{H}$
      Back-prop gradients, update parameters
    until ϕ batches have been forwarded
    Compute $\hat{T}$ for each filter $j \in \mathcal{A}$ as Eq. 10
    Pick up $|\mathcal{A}|/2$ filters with the smallest $\hat{T}$ value as the picked set $\mathcal{B}$
    Max damage $p = max(\{\hat{T}(\boldsymbol{F}^{(i,j)}) \,|\, \forall\, j \in \mathcal{B}\})$
    Let $\mathcal{A} \leftarrow \mathcal{B}$
  until $p < \theta$ or $|\mathcal{B}| = 1$
  if $p < \theta$, **then**
    Let $\boldsymbol{u}_j \leftarrow 0, \forall\, j \in \mathcal{B}$ // prune the picked filters
  else
    break // $p \geq \theta$ and $|\mathcal{B}| = 1$, stop refining
  end if
end while
Prune layer i by Eq. 3, 4 with $\mathcal{R}_i = \{j|u_j = 1\}$
Return

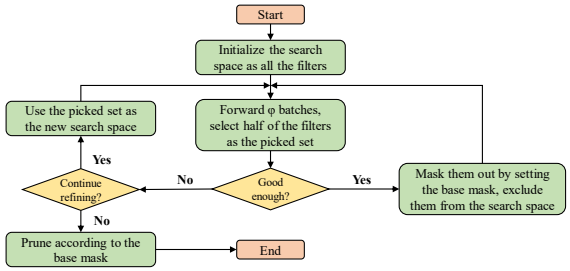

Figure 2. Flow chart of AOFP on a single layer.

manually set controlling conditions. Essentially, at each step of binary search, the picked set can be regarded as the least important $|\mathcal{B}|$ filters. If we finish the current move by pruning them, $|\mathcal{B}|$ serves exactly as the granularity g. So in the context of binary search, the problem of deciding g and the termination conditions can be simply solved as follows:

- If the current picked set $\mathcal{B}$ is good enough, finish the current move with $g = |\mathcal{B}|$ (i.e., permanently mask out the filters in $\mathcal{B}$ and start a new move); otherwise, see if it is possible to continue refining ($|\mathcal{B}| > 1$ or $|\mathcal{B}| = 1$).
- If $|\mathcal{B}| > 1$, we continue refining by letting $\mathcal{A} \leftarrow \mathcal{B}$; otherwise, it suggests that the single least important filter is still too important, so we stop pruning the layer.

We introduce a global hyper-parameter, the *refinement threshold* θ, which is used to compare with the max $\hat{T}$ value (Eq. 10) of the filters in $\mathcal{B}$ to judge if the picked set is good (unimportant) enough. We say $\mathcal{B}$ is good enough if

$$max(\{\hat{T}(\boldsymbol{F}^{(i,j)}) \,|\, \forall\, j \in \mathcal{B}\}) < \theta\,. \tag{11}$$

Intuitively, θ indicates the upper limit of the damage we can endure for a single step of pruning. E.g., with $\theta = 0.02$, we consider it acceptable to prune one or more filters with 2% isolated damage. With a larger θ, we tend to prune with larger granularity, and vice versa.

In this way, another design concern is settled naturally, that is, how many filters to randomly ablate for a batch of input data. We randomly ablate $|\mathcal{A}|/2$ filters out of the search space $\mathcal{A}$ at a time, and the reason is simple: according to our discussions above, the collected t values should reflect the expected damage if we prune the current picked set, i.e., the t values should be derived with $|\mathcal{B}|$ filters ablated, and $|\mathcal{B}| = |\mathcal{A}|/2$. The AOFP algorithm on a single layer is outlined in Fig. 2 and formally described in Alg. 1. In practice, we apply AOFP on every layer simultaneously.

4. Experiments

4.1. Comparison of Filter Pruning Metrics

In this subsection, we present a comparison of Oracle Pruning, AOFP and other heuristic methods using AlexNet (Krizhevsky et al., 2012) on ImageNet (Deng et al., 2009).

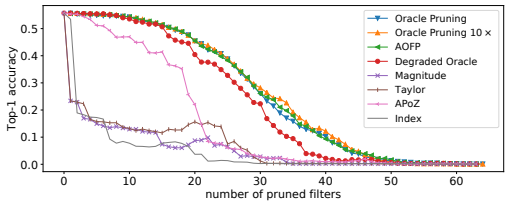

Figure 3. Comparison of filter pruning metrics on AlexNet.

We use the simplified version of AlexNet (GoogLe, 2017a), which is composed of five stacked conv layers and three fully-connected layers with no LRU or cross-GPU connections. For faster convergence, batch normalization (Ioffe & Szegedy, 2015) is applied on every conv layer. We compare these metrics by pruning filters one by one from the first layer without finetuning. Fig. 3 shows the Top-1 accuracy on the validation set with varying number of pruned filters. For Oracle Pruning, APoZ-based and Taylor-expansion-based, we use randomly sampled 10,000 training examples as the assessment dataset. For AOFP, we set the search cost ϕ such that the total number of examples consumed equals that of Oracle Pruning. As we are pruning only one layer, we apply Binary Filter Search to collect the final loss value instead of the isolated damage. For Oracle Pruning $10\times$, we use 100,000 examples to score a filter. For Degraded Oracle, no re-scoring processes are conducted, i.e., the importance scores collected at the very beginning are used to guide the pruning till the end. For the curve labeled as Index, we prune filters from the first one to the 64th, which is essentially equivalent to random guess.

Our observations are as follows. **1)** By comparing Degraded Oracle and Oracle Pruning, we conclude that re-computing importance scores after each step of Oracle Pruning is essential, as a CNN is a highly non-linear system, and the removal of a filter can affect the relative importance of other filters. This discovery is consistent with the observations of prior works (Mozer & Smolensky, 1989; Sharma et al., 2017) that neural networks do not distribute the learning representation equitably across neurons. **2)** AOFP is almost as good as Oracle Pruning. **3)** The extra costs of Oracle Pruning $10\times$ bring marginal accuracy improvement.

4.2. AOFP for Automatic CNN Compression

Abundant experiments are conducted using several common networks on CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet, including AlexNet, VGG, and ResNets, to validate the effectiveness of AOFP, which is measured by the FLOPs and accuracy of the pruned model. For reproducibility and comparability, we use the same VGG version as other works (Li et al., 2016; Liu et al., 2017), and the same ResNet structures as the official tensorflow/slim models (GoogLe, 2017b). All the original models are trained from scratch, and the FLOPs of every model are calculated

Table 1. Pruned VGG on CIFAR-10. The resulting models with different FLOPs are labeled from A1 to A5.

Result	Top-1	FLOPs	FLOPs↓%
base	93.38	313M	-
AOFP-A1	**93.81**	**215M**	**31.32**
(Li et al., 2016)	93.40	206M	34.20
AOFP-A2	**94.03**	**186M**	**40.51**
(Liu et al., 2017)	93.80	-	51.00
(Huang et al., 2018)	91.67	-	55.20
AOFP-A3	**93.84**	**124M**	**60.17**
(Xu et al., 2018)	93.29	120M	61.46
AOFP-A4	**93.47**	**108M**	**65.27**
(Zhou et al., 2018)	92.33	-	74.81
AOFP-A5	**93.28**	**77M**	**75.27**

in the same manner as (He et al., 2016). Of note is that we perform AOFP on every target layer simultaneously. Concretely, for a specific layer, when the returning condition in Alg. 1 is satisfied, the AOFP process is restarted, i.e., we finish the current move without changing the base mask and start the next move. After each move on any conv layer, we calculate the reduced global FLOPs based on the model architecture and the current values of all the base masks, and stop pruning when the reduced FLOPs reach a target level (e.g., above 60% for the model labeled as AOFP-A3 in Table. 1). Then we reconstruct a narrower network following Eq. 3, 4 for every layer, finetune the model, and test it on the validation dataset using a single central crop.

On CIFAR-10, we use VGG-16 for a quick sanity check. The base model is trained from scratch for 600 epochs to ensure the convergence, with the standard data augmentation, i.e., padding to 40×40, random cropping and flipping. We use a batch size of 64 and a learning rate initialized to 5×10^{-2} then decayed by 0.1 every 200 epochs. We perform AOFP on all of the 13 layers with search cost $\phi = 20,000$, $\theta = 0.01$ and a constant learning rate of 1×10^{-3}. On ImageNet, we first prune all the conv layers of AlexNet with $\phi = 4,000$, $\theta = 0.02$ and a learning rate of 1×10^{-3}. For ResNets, since there are more layers being simultaneously pruned, we increase the search cost to $\phi = 8,000$ for better filter scoring and damage recovery. These hyper-parameters are casually set without careful tuning. Of note is that, on ResNets, due to the constraints of the shortcut connections, only the internal layers (i.e., the first and second layers in each block which are not directly added to the identity mapping) are pruned, as a common practice (Luo et al., 2017; Luo & Wu, 2018; Wang et al., 2017a).

As it turns out (Table. 1, 2), these networks can be pruned significantly even with an increase in accuracy due to the optimized network structure, which is consistent with the observations in other works (Liu et al., 2017; Li et al., 2016). And if we wish to trade accuracy for efficiency, AOFP can reduce the computational burdens by a large margin, demonstrating not only higher pruning ratios but also less accuracy drop than other methods. Moreover, the increased network depth does not hinder the application of AOFP, because we simultaneously prune every layer in a mutually interdependent manner, and do not suffer from the notorious problem of error propagation and amplification in filter importance estimation (Yu et al., 2018), thanks to Damage Isolation.

Table 2. Pruning on ImageNet. The competitors include ThiNet (Luo et al., 2017), NISP (Yu et al., 2018), Channel Pruning (He et al., 2017), SPP (Wang et al., 2017a), Autopruner (Luo & Wu, 2018), ISTA-based (Ye et al., 2018), C-SGD (Ding et al., 2019).

	Result	Top-1	Top-5	FLOPs	↓%
Alex	base	55.71	79.45	838M	-
Alex	**AOFP-B1**	**56.54**	**79.95**	**578M**	**30.98**
Alex	**AOFP-B2**	**56.17**	**79.53**	**492M**	**41.33**
Res50	base	75.34	92.56	3.85G	-
Res50	**AOFP-C1**	**75.63**	**92.69**	**2.58G**	**32.88**
Res50	ThiNet-70	72.04	90.67	2.44G	36.75
Res50	NISP	0.89↓	-	-	44.01
Res50	Chan-Pr	-	90.80	-	50.00
Res50	SPP	-	90.40	-	50.00
Res50	Autopr	74.76	92.15	-	51.21
Res50	ThiNet-50	71.01	90.02	1.70G	55.76
Res50	C-SGD-50	74.54	92.09	1.70G	55.76
Res50	**AOFP-C2**	**75.11**	**92.28**	**1.66G**	**56.73**
Res101	base	76.63	93.29	7.57G	-
Res101	**AOFP-D1**	**76.88**	**93.49**	**5.29G**	**30.11**
Res101	ISTA	75.27	-	4.47G	40.95
Res101	**AOFP-D2**	**76.40**	**93.07**	**3.77G**	**50.19**
Res152	base	77.37	93.52	11.28G	-
Res152	**AOFP-E1**	**77.47**	**93.76**	**6.12G**	**45.72**
Res152	**AOFP-E2**	**77.00**	**93.49**	**4.13G**	**63.36**
Res152	**AOFP-E3**	**76.40**	**93.02**	**2.85G**	**74.69**

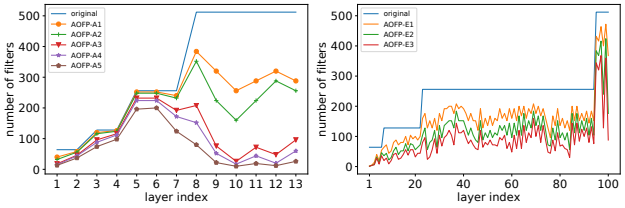

Figure 4. Layer width of pruned models. Left: VGG on CIFAR-10. Right: ResNet-152 on ImageNet (only the pruned layers).

Of note is that AOFP automatically detects the easy-to-prune layers without heuristic knowledge or manually set control conditions, which is a significant strength compared to other approaches where we have to empirically decide the width of every layer in advance (Li et al., 2016; Luo et al., 2017; Wang et al., 2017a). By visualizing the structure of the pruned networks in Fig. 4, we learn that conv2,4,5,6 of VGG are more sensitive to pruning, but conv1 and the top six layers can be pruned dramatically for free, as AOFP chooses to prune these layers aggressively to achieve high

Table 3. AOFP pruned v.s. uniformly slimmed VGG. All the models are tested on an Nvidia GTX 1080Ti GPU or E5-2680 CPU with batch size 64, measured in examples/sec.

	Top-1	FLOPs	CPU	GPU	Speedup
VGG base	93.38	313M	343	6336	-
AOFP-A3	**93.84**	**124M**	**683**	**13903**	**2.19×**
Uniform	92.88	126M	560	10224	1.61×

pruning ratios. Similarly, AOFP converts the original tidy ResNet-152 to a more efficient one without human intervention. One concern about the irregularly shaped models is that the varying width of layers may cause GPU memory bottlenecks, so it may not result in real acceleration, though the FLOPs are reduced. However, our pruned VGG outperforms a uniformly slimmed counterpart in both accuracy and speed (Table. 3). Concretely, we construct a VGG model where every layer is 69% of its original width, such that the FLOPs becomes 126M, which is comparable to our pruned model labeled as AOFP-A3. We train it from scratch for 600 epochs and test it on both CPU and GPU. It is not clear why AOFP-A3 runs faster than the counterpart, but evidently, the discrimination towards irregularly shaped CNNs is just a kind of stereotype.

4.3. AOFP for Global Progressive Pruning

Binary Filter Search enables not only the full use of the low-quality samples but also the adaptive pruning granularity. We present in Fig. 5 the width of each layer of ResNet-152 (AOFP-E1). We pick the first layer in each of the four stages as the representatives, which originally have 64, 128, 256 and 512 filters, respectively. As AOFP proceeds, we show the remaining percentage of filters. Then we plot the remaining width of each layer every 20,000 batches. It can be observed that: **1)** AOFP automatically figures out that the first layer in stage2 can be pruned significantly, and chooses to prune it with large granularity (8 filters every time) at the beginning, then gradually reduces the granularity in order for more fine-grained pruning. However, AOFP always prunes 16 filters from the first layer in stage5. **2)** The adaptive granularity enables global progressive pruning, i.e., the reduction in the total FLOPs does not come from the extreme squeeze of several layers, nor pruning some at the beginning and others at the end. Instead, the network structure shrinks globally, steadily and progressively, which is more likely to result in high accuracy.

4.4. AOFP for Destructive CNN Re-design

The above experiments are focused on pruning an existing mature CNN architecture for compression and acceleration. We then seek to use AOFP to re-design the CNN in order to reach a higher level of accuracy with the same computational budgets. To this end, we first train a scaled network from scratch and then apply AOFP until its FLOPs are reduced to the same level as the original model. In this way, we obtain a network where some layers are wider than the original architecture and some are narrower, so we call this process CNN Re-design. Concretely, we first scale the width of VGG and ResNet-50 (only the internal layers) by 1.5× and 1.25×, respectively, and apply AOFP using the same hyper-parameters as Sect. 4.2. Though the pruned models outperform the baselines by a clear margin (Table. 4), we still need to figure out whether the improvement is due to the better structure or the parameters initialized using the scaled model, so a counterpart with the same structure is trained from scratch, which delivers an accuracy higher than the baseline but lower than the pruned model. In this way, we safely claim the superiority of our optimized models over the tidy baselines. We present the discovered structures in the appendix to encourage further studies.

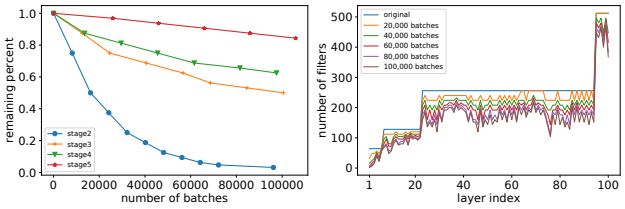

Figure 5. Left: the remaining percentage of filters at the first layer in the four stages respectively. Right: the remaining width of all the pruned layers every 20,000 batches with a batch size of 128.

Table 4. Results of CNN Re-design by AOFP.

	Top-1	FLOPs	CPU	GPU
VGG base	93.38	313M	343	6336
Scaled 1.5×	93.95	703M	205	3228
Re-design-pruned	94.03	312M	351	8099
Re-design-scratch	**93.69**	**312M**	-	-
Res50 base	75.34	3.85G	14.4	437
Scaled 1.25×	76.60	5.28G	11.2	353
Re-design-pruned	76.47	3.83G	14.2	430
Re-design-scratch	**76.30**	**3.83G**	-	-

5. Conclusion

We proposed Approximated Oracle Filter Pruning (AOFP), which features high quality of importance estimation, reasonable time complexity and no need for heuristic knowledge. We proposed a new CNN design paradigm, where we scale the network and apply AOFP to optimize its width to reach a higher level of accuracy without extra computational budgets, which can be used for refinement before a CNN is released. We empirically found out that the structural change in CNNs can be analyzed with local information only, which may inspire further theoretical researches.

Acknowledgement

This work was supported by the National Key R&D Program of China (No. 2018YFC0807500), National Natural Science Foundation of China (No. 61571269), National Postdoctoral Program for Innovative Talents (No. BX20180172), and the China Postdoctoral Science Foundation (No. 2018M640131). We sincerely thank the reviewers for their comments.

References

Abbasi-Asl, R. and Yu, B. Structural compression of convolutional neural networks based on greedy filter pruning. *arXiv preprint arXiv:1705.07356*, 2017.

Alvarez, J. M. and Salzmann, M. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2270–2278, 2016.

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

Castellano, G., Fanelli, A. M., and Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Ding, X., Ding, G., Han, J., and Tang, S. Auto-balanced filter pruning for efficient convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ding, X., Ding, G., Guo, Y., and Han, J. Centripetal sgd for pruning very deep convolutional networks with complicated structure. *arXiv preprint arXiv:1904.03837*, 2019.

GoogLe. Tensorflow-alexnet. https://github.com/tensorflow/models/blob/master/research/slim/nets/alexnet.py, 2017a.

GoogLe. Tensorflow-slim. https://github.com/tensorflow/models/tree/master/research/slim, 2017b.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015b.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, pp. 6, 2017.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Huang, Q., Zhou, K., You, S., and Neumann, U. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990a.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990b.

LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763. IEEE, 2017.

Luo, J.-H. and Wu, J. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

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

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

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2498–2507. JMLR. org, 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. 2016.

Mozer, M. C. and Smolensky, P. Using relevance to reduce network size automatically. *Connection Science*, 1(1): 3–16, 1989.

Polyak, A. and Wolf, L. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

Roth, V. and Fischer, B. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. In *Proceedings of the 25th international conference on Machine learning*, pp. 848–855. ACM, 2008.

Sharma, A., Wolfe, N., and Raj, B. The incredible shrinking neural network: New perspectives on learning representations through the lens of pruning. *arXiv preprint arXiv:1701.04465*, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Wang, H., Zhang, Q., Wang, Y., and Hu, H. Structured probabilistic pruning for convolutional neural network acceleration. *arXiv preprint arXiv:1709.06994*, 2017a.

Wang, Y., Xu, C., You, S., Tao, D., and Xu, C. Cnnpack: Packing convolutional neural networks in the frequency domain. In *Advances in neural information processing systems*, pp. 253–261, 2016.

Wang, Y., Xu, C., Xu, C., and Tao, D. Beyond filters: Compact feature map for portable deep model. In *International Conference on Machine Learning*, pp. 3703–3711, 2017b.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Xu, K., Wang, X., Jia, Q., An, J., and Wang, D. Globally soft filter pruning for efficient convolutional neural networks. 2018.

Ye, J., Lu, X., Lin, Z., and Wang, J. Z. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

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

Zhou, Z., Zhou, W., Hong, R., and Li, H. Online filter weakening and pruning for efficient convnets. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6. IEEE, 2018.