OTOv2: AUTOMATIC, GENERIC, USER-FRIENDLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Only-Train-Once (OTOv1) is recently proposed to drastically simplify and automate the complicated multi-stage procedure for model compression via structured pruning. However, its automation relies on manually conducting zero-invariant groups (ZIGs) partition and slimmer model construction for specific DNNs beforehand, which necessitates numerous engineering efforts and domain-knowledge and prevents its wider applications onto general scenarios. We propose the second generation of Only-Train-Once (OTOv2), which trains and compresses an *arbitrary* DNN only once from scratch to produce a more compact model with competitive performance without fine-tuning. OTOv2 is *automated* and *pluggable* into various deep learning applications, and requires *minimal* engineering efforts from the users. Methodologically, OTOv2 proposes two revolutionary improvements: *(i)* Autonomy: automatically partitions ZIGs and constructs compressed model for *arbitrary* DNNs; and *(ii)* Dual Half-Space Projected Gradient (DHSPG): a novel optimizer to more reliably solve structured-sparsity problems. Numerically, we demonstrate the generality and autonomy of OTOv2 on a variety of model architectures such as VGG, ResNet, DenseNet and StackedUnets, the majority of which cannot be handled by OTOv1 without extensive handcrafting. Together with benchmark datasets including CIFAR10/100, Fashion-MNIST, SVNH and ImageNet, the effectiveness of OTOv2 is validated by achieving competitive or even better results than the state-of-the-arts.

1 INTRODUCTION

Large-scale Deep Neural Networks (DNNs) have demonstrated successful in a variety of applications to typically outperform the slimmer and simplier counterparts (He et al., 2016; Huang et al., 2017). However, how to deploy such heavy DNNs onto resource-constrained environments is facing severe challenges. Consequently, in both academy and industry, compressing full DNNs into slimmer ones with negligible performance regression becomes ubiquitous. Although this area has been explored by numerous methods in the past decade, it is still far away from being solved.

Structured pruning is perhaps the most popular and commonly-used compression method because of its generality, which achieves significant reduction of FLOPs and model size by identifying and removing redundant structures (Gale et al., 2019; Han et al., 2015). However, most existing pruning methods typically proceed a complicated multi-stage procedures as Figure 1, which have apparent limitations *(i)* **Hand-Crafting and User-Hardness**: requires significant engineering efforts and expertise from users to apply the methods onto their own scenarios; *(ii)* **Time Consuming**: conducts DNN training multiple times including foremost pre-training and afterwards fine-tuning; and *(iii)* **Low-Generality**: many methods are designed for specific architectures and tasks.

	OTOv2	OTOv1	Others
Train Cost	Low	Low	High
Autonomy	✓	✓̸	✗
User-Friendly	✓	✗	✗
Generality	✓	✓	✓̸

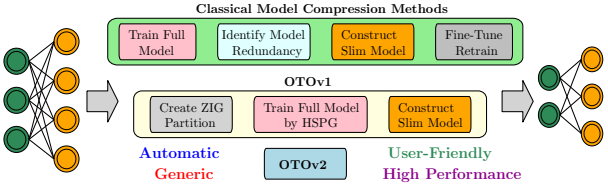


Figure 1: OTOv2 versus existing methods.

We have the perspective that the ideal DNN training and weight pruning method should be the

North Star. *Given an arbitrary DNN architecture, automatically train it only once to achieve both high performance and slimmer model architecture simultaneously without pretrain and fine-tuning.*

To reach the north star, the following key problems need to be resolved systematically. *(i)* What are the removal structures of DNNs? *(ii)* How to identify the redundant removal structures? *(iii)* How to effectively remove redundant structures without deteriorating the model performance to avoid extra fine-tuning? *(iv)* How to make all the above proceeding automatically? Addressing all these problems is challenging in the manner of both algorithmic design and engineering development, thereby is not achieved yet by the existing methods to the best of our knowledge.

To resolve *(i-iii)*, Only-Train-Once (OTOv1) (Chen et al., 2021) proposed a concept so-called Zero-Invariant Group (ZIG), which is a class of minimal removal structure of DNNs consisting of most DNN architectures, and a Half-Space Projected Gradient (HSPG) method to pursue a highly group sparse solution in the manner of ZIGs, wherein zero groups correspond to redundant structures and removing them does not affect model performance thereby avoid fine-tuning. As a result, OTOv1 trains a full DNN only once to compute a slimmer model with competitive performance, and is perhaps the closest to the north star among existing compression competitors.

Nevertheless, the fundamental problem *(iv)* is not addressed in OTOv1, *i.e.*, the ZIGs partition is not automatic and only implemented and applied onto several specific architectures. OTOv1 suffers a lot from requiring extensive hand-crafting efforts and domain knowledge to partition trainable variables into ZIGs and construct smaller models by trimming redundant structures which prevents its broader usages. Meanwhile, the performance of OTOv1 highly depends on the effectiveness of HSPG to yield a solution with both satisfactory performance and high group sparsity, which is sometimes sensitive to regularization coefficient hence need hyper-parameter fine-tuning.

Library Usage

```
from only_train_once import OTO
oto = OTO(model)
optimizer = oto.dhspg()
# Train as normal
optimizer.step()
oto.compress()
```

To overcome the drawbacks of OTOv1 and simultaneously tackle *(i-iv)*, we propose Only-Train-Once v2 (OTOv2), the next-generation one-shot deep neural network training and pruning framework. Given an arbitrary full DNN , OTOv2 is able to train and compress it from scratch into a slimmer DNN with significant FLOPs and parameter quantity reduction. In contrast to others, OTOv2 drastically simplifies the complicated multi-stage procedures; **guarantees performance** more reliably than OTOv1; and is unprecedentedly **generic**, **highly automatic** and **user-friendly**. Our main contributions are summarized as follows.

- **Infrastructure for Arbitrary DNN One-Shot Training and Compression.** We propose and develop perhaps the first generic and automatic framework to compress an arbitrary DNN and ensure both excellent model performance and substantial complexity reduction in terms of FLOPs and model cardinality. OTOv2 only trains the DNN once, neither pre-training nor fine-tuning is a necessity. Furthermore, OTOv2 is likely the most user-friendly compression framework that requires minimal engineering efforts to apply onto generic DL tasks as library usage. The success of OTOv2 relies on the breakthroughs from both algorithmic designs and infrastructure development.
- **Automatic ZIG Partition and Compressed Model Construction.** We propose a novel graphic algorithm to automatically exploit and partition the variables of an arbitrary DNN into Zero-Invariant Groups (ZIGs), *i.e.*, the most common minimal removal structures. We further propose a novel algorithm to automatically construct the compressed model by following the hierarchy of DNN's trace graph and eliminating the structures corresponding to ZIGs as zero. Both algorithms are delicately designed, and works effectively with low time and space complexity.
- **Novel Structured-Sparsity Optimization Algorithm.** We propose a novel optimization algorithm to train an arbitrary DNN once from scratch to effectively achieve competitive performance and high group sparsity in the manner of ZIGs, which solution is further leveraged into the above automatic compression. DHSPG formulizes a constrained sparse optimization problem and solves it by constituting a direction within the intersection of dual half-spaces to guarantee the progress to both the objective convergence as well as the identification of redundant groups. DHSPG outperforms the HSPG in OTOv1 in terms of enlarging search space, fewer hyper-parameter fine-tuning efforts and guarantee ultimate sparsity-level more reliably.
- **Experimental Results.** We apply OTOv2 onto a variety of DNNs, many of which structures exhibit complicated connectivity, and extensive benchmark datasets, including CIFAR10, CIFAR100, SVNH, Fashion-MNIST and ImageNet. OTOv2 effectively trains and compresses

various DNNs simultaneously from scratch without fine-tuning for significant inference speedup and parameter reduction, and achieve state-of-the-art results on compression benchmarks.

2 Related Work

Structured Pruning. To compute compact architectures for efficient model inference and storage, structured pruning identifies and prunes the redundant structures in a full model (Gale et al., 2019; Han et al., 2015). The general procedure can be largely summarized as: (*i*) train a full model; (*ii*) identify and remove the redundant structures to construct a slimmer DNN based on various criteria, including (structured) sparsity (Lin et al., 2019; Wen et al., 2016; Li et al., 2020b; Zhuang et al., 2020; Chen et al., 2020; Gao et al., 2020; Zhuang et al., 2020; Meng et al., 2020; Yang et al., 2019), Bayesian pruning (Zhou et al., 2019; Louizos et al., 2017; van Baalen et al., 2020), ranking importance (Li et al., 2020a; Luo et al., 2017a; Hu et al., 2016; He et al., 2018a; Li et al., 2019; Zhang et al., 2018), reinforcement learning (He et al., 2018b; Chen et al., 2019), lottery ticket (Frankle & Carbin, 2018; Frankle et al., 2019; Renda et al., 2020), etc.; (*iii*) (iteratively) retrain the pruned model to regain the accuracy regression during pruning. These methods have to conduct a complicated and time-consuming procedure to trains the DNN multiple times and requires a good deal of domain knowledge to manually proceed every individual step. OTOv1 (Chen et al., 2021) is recently proposed to avoid fine-tuning and end-to-end train and compress the DNN once, whereas its automation relies on spending numerous handcrafting efforts on creating ZIGs partition and slimmer model construction for the specific target DNNs in advance, thereby is semi-auto.

Automated Machine Learning (AutoML). OTOv2 fills into a vital gap within AutoML domain regarding given an arbitrary DNN architecture, how to automatically train and compress it into a slimmer one with competitive performance and significant FLOPs and parameter quantity reduction. The existing AutoML methods focus on *(i)* automated feature engineering (Kanter & Veeramachaneni, 2015; Katz et al., 2016), *(ii)* automated hyper-parameter setting (Klein et al., 2017), *(iii)* neural architecture search (NAS) (Elsken et al., 2018; Brock et al., 2017). NAS searches a DNN architecture with satisfactory performance from a prescribed fixed full graph wherein the connection between two nodes (tensors) is searched from a pool of prescribed operators. NAS itself has no capability to slim and remove redundancy from the searched architectures due to the pool being fixed and is typically time-consuming. As a result, NAS may serve as a prerequisite step to search a target network architecture as the input to OTOv2. We leave the incorporation between OTOv2 and NAS as future work to achieve jointly automated neural architecture search and compression.

3 OTOv2

OTOv2 has nearly reached the north star of model compression via weight pruning, which outline is stated in Algorithm 1. In general, given an arbitrary neural network $\mathcal{M}$ to be trained and compressed, OTOv2 first automatically figures out its minimal removal structures and partitions its trainable variables into zero-invariant groups (ZIGs) (Algorithm 2). ZIGs $(\mathcal{G})$ are then fed into a structured sparsity optimization problem, which is solved by a Dual Half-Space Projected Gradient (DHSPG) method to yield a solution x^*_{DHSPG} with competitive performance as well as high group sparsity in the view of ZIGs (Algorithm 3). The compressed slimmer model $\mathcal{M}^*$ is ultimately constructed via removing the redundant structures corresponding to the ZIGs being zero (Algorithm **??**). $\mathcal{M}^*$ returns the exact same inferences as the full model $\mathcal{M}$ parameterized as x^*_{DHSPG}, thereby no further fine-tuning is a necessity. The whole procedure can be proceeded automatically and easily plugged into various applications and consume minimal engineering efforts from the users.

Algorithm 1 Outline of OTOv2.

1: **Input.** An arbitrary full model $\mathcal{M}$ to be trained and compressed (no need to be pretrained).
2: **Automated ZIG Partition.** Partition the trainable parameters of $\mathcal{M}$ into $\mathcal{G}$.
3: **Train $\mathcal{M}$ by DHSPG.** Seek a highly group-sparse solution $\boldsymbol{x}^*_{\text{DHSPG}}$ with high performance.
4: **Automated Compressed Model $\mathcal{M}^*$ Construction.** Construct a slimmer model upon $\boldsymbol{x}^*_{\text{DHSPG}}$.
5: **Output:** Compressed slimmed model $\mathcal{M}^*$.

3.1 AUTOMATED ZIG PARTITION

Background. Before describing how to proceed ZIG partition automatically, let us review necessary concepts beforehand. A structure of DNN is called a removal structure (RS) if and only if the DNN after removing it still serves as a valid DNN. Consequently, a removal structure is minimal (MRS) if and only if it can be not decomposed into multiple removal structures. ZIG is first proposed by OTOv1 (Chen et al., 2021), describes a class of minimal removal structures, *i.e.*, the ones which outputs equal to zero if their parameters being zero, and dominates most DNN structures, *e.g.*, layers as `Conv`, `Linear` and `MultiHeadAtten`. By the definition, the existence of ZIGs is deterministic given a DNN. However, due to the hierarchy and complicated connectivity, the topology of ZIG varies a lot. Combining with the lack of API, all result in the severe challenges to automatically exploit ZIGs of an arbitrary DNN in terms of algorithmic design and engineering development.

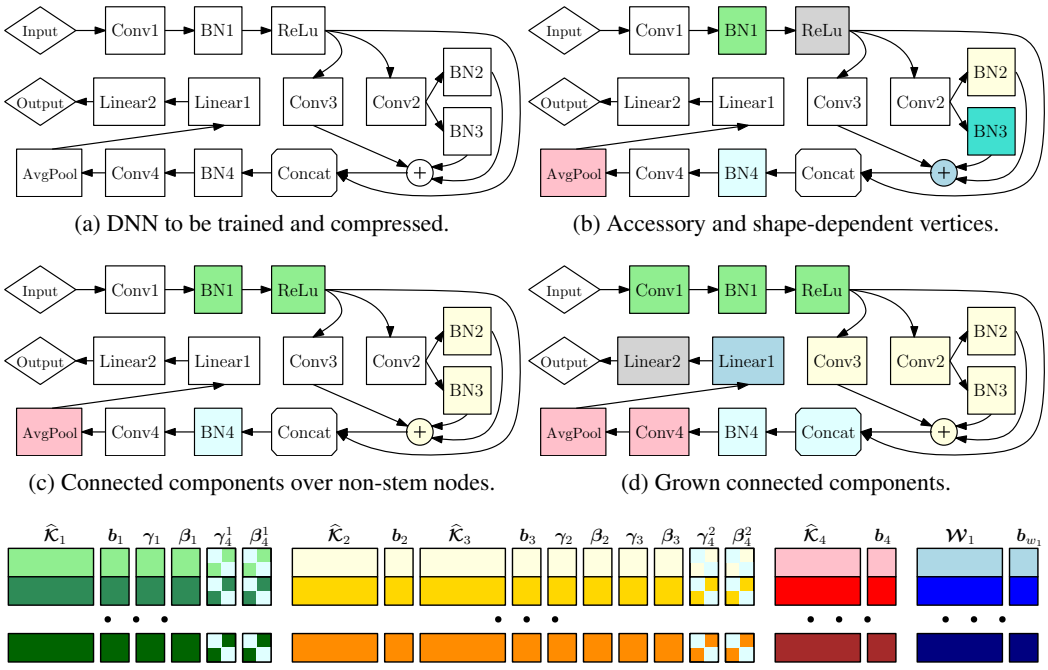
Algorithm 2 Automated Zero-Invariant Group Partition.

1: **Input:** A DNN $\mathcal{M}$ to be trained and compressed.
2: Get the trace graph $(\mathcal{E}, \mathcal{V})$ of $\mathcal{M}$.
3: Find connected components $\mathcal{C}$ over all accessory, shape-dependent joint and unknown vertices.
4: Grow $\mathcal{C}$ till incoming nodes are either stem or joint vertices.
5: Merge connected components in $\mathcal{C}$ if any intersection.
6: Find the dependent stem vertices for the affiliated vertices of shape-dependent-free vertices.
7: Group pairwise parameters of vertices in the same connected components associated with parameters from affiliated vertices of shape-dependent-free vertices if any into $\mathcal{G}$.
8: **Return** the zero-invariant group partition $\mathcal{G}$ and the connected component set $\mathcal{C}$.

Algorithmic Outline. To resolve the autonomy of ZIG partition, we present a novel, effective and efficient algorithm, outlined as Algorithm 2. For more intuitive illustration, we provide a small but complicated DemoNet along with explanations about its ground truth minimal removal structures (ZIGs) in Figure 2. We now perform Algorithm 2 to automatically recover the ground truth ZIGs.

Graph Construction. In particular, we first establish the trace graph $(\mathcal{E}, \mathcal{V})$ of the target DNN, wherein the each vertex in $\mathcal{V}$ refers to a specific operator, and the edges in $\mathcal{E}$ describe how they connect (line 2). We categorize the vertices into stem, joint, accessory or unknown. Stem vertices equip with trainable parameters and have capacity to transform their input tensors into other shapes, *e.g.*, `Conv` and `Linear`. Joint vertices aggregate multiple input tensors into a single output such as `Add`, `Mul` and `Concat`. Accessory vertices operate a single input tensor into a single output and may possess trainable parameters such as `BatchNorm` and `ReLu`. The remaining unknown vertices proceed some uncertain operations. Apparently, stem vertices compose most of the DNN parameters. Joint vertices establish the connections cross different vertices, which existence dramatically brings hierarchy and intricacy of DNN. Naturally, the joint vertices introduce the majority of challenges of automatically figuring out ZIGs since due to the complicated connectivity, the minimal removal structures should be carefully constructed to keep the validness of the joint vertices. Furthermore, we call joint vertices being input shape dependent (SD) if requiring inputs in the same shapes such as `Add`, otherwise being shape-dependent-free (SDF) such as `Concat`.

Construct Connected Components of Dependency. Now, we need to figure out the exhibiting dependency across the vertices to seek the minimal removal structures of the target DNN. To proceed, we first connect accessory, SD joint and unknown vertices together if adjacent to form a set of connected components $\mathcal{C}$ (see Figure 2c and line 3 of Algorithm 2). This step is to establish the skeletons for finding vertices that depend on each other when considering removing hidden structures. The underlying intuitions of this step in depth are *(i)* the accessory vertices operate and are subject to the same ancestral stem vertices if any; *(ii)* SD joint vertices force their ancestral stem vertices dependent on each other to yield tensors in the same shapes; and *(iii)* unknown vertices introduce uncertainty hence finding potential affected vertices is necessary. We then grow $\mathcal{C}$ till all their incoming vertices are either stem or SDF joint vertices and merge the connected components if any intersection as line 4-5. Remark here that the newly added stem vertices are affiliated by the accessory vertices, such as Conv1 for BN1-ReLu and Conv3+Conv2 for BN2|BN3 in Figure 2d. In addition, the SDF joint vertices introduce dependency between their affiliated accessory vertices and incoming connected components, *e.g.*, Concat-BN4 depends on both Conv1-BN1 and Conv3+Conv2-BN2|BN3 since BN4 normalizes the concatenated tensors from them as line 6.



(a) DNN to be trained and compressed.

(b) Accessory and shape-dependent vertices.

(c) Connected components over non-stem nodes.

(d) Grown connected components.

(e) Zero-Invariant Groups.

Figure 2: Automatic ZIG partition illustration. $\widehat{\mathcal{K}}_i$ and $\boldsymbol{b}_i$ are the flatten filter matrix and bias vector of Conv$i$, where the jth row of $\widehat{\mathcal{K}}_i$ represents the jth 3D filter. $\boldsymbol{\gamma}_i$ and $\boldsymbol{\beta}_i$ are the weight and bias vectors of BNi. The ground truth ZIGs are present in Figure 2e. Since the output tensors of Conv2 and Conv3 are added together, both layers associated with the subsequent BN2 and BN3 must remove the same number of filters from $\widehat{\mathcal{K}}_2$ and $\widehat{\mathcal{K}}_3$ and scalars from $\boldsymbol{b}_2, \boldsymbol{b}_3, \boldsymbol{\gamma}_2, \boldsymbol{\gamma}_3, \boldsymbol{\beta}_2$ and $\boldsymbol{\beta}_3$ to keep the addition valid. Since BN4 normalizes the concatenated outputs from Conv1-BN1-ReLu and Conv3+Conv2-BN2|BN3, the corresponding scalars in $\boldsymbol{\gamma}_4, \boldsymbol{\beta}_4$ need to be removed simultaneously.

Form ZIGs. Finally, we form ZIGs based on the connected components of dependency as Figure 2d. The pairwise trainable parameters across all individual stem vertices in the same connected component need to be first grouped together as Figure 2e, wherein the parameters of the same color represents one group. Later on, the accessory vertices insert their trainable parameters if applicable into groups of their dependent stem vertices accordingly. Remark here that some accessory vertex such as BN4 depends on multiple groups because of SDF joint vertex, thereby its trainable parameters $\boldsymbol{\gamma}_4$ and $\boldsymbol{\beta}_4$ need to be partitioned and separately added into corresponding groups, *e.g.*, $\boldsymbol{\gamma}_4^1, \boldsymbol{\beta}_4^1$ and $\boldsymbol{\gamma}_4^2, \boldsymbol{\beta}_4^2$. In addition, the connected components that are adjacent to the output of DNN are excluded from forming ZIGs since the output shape should be fixed such as Linear2. For safety, the connected components that possess unknown vertices are excluded as well due to uncertainty, which further guarantees the generality of framework applying onto DNNs with customized operators.

Complexity Analysis. The proposed automated ZIG partition Algorithm 2 is a series of customized graphic algorithms delicately composed together. Its overall complexity equals to the summation of each individual step. In depth, every individual sub-algorithm is achieved by depth-first-search recursively traversing the trace graph of DNN and conduct step-specific operations, which has time complexity as $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ and space complexity as $\mathcal{O}(|\mathcal{V}|)$ in the worst case. The former one is computed by discovering all neighbors of each vertex by traversing the adjacency list once in linear time; and the latter one is due to the fact that the trace graph of DNN is acyclic thereby the memory cache consumption is up to the length of possible longest path for acyclic graph as $|\mathcal{V}|$. Therefore, automated ZIG partition can be efficiently complete within linear time.

3.2 Dual Half-Space Projected Gradient (DHSPG)

Given the constructed ZIGs by Algorithm 2, the next step is to jointly identify which groups are redundant to be further removed and train the remaining groups to achieve high performance.

Target Problem. Structured sparsity inducing optimization problem is a natural choice to seek a group sparse solution with high performance, wherein the zero groups refer to the redundant structures and the non-zero groups exhibits the prediction power to maintain competitive performance to the full model. We formulate an optimization problem with group sparsity constraint in the form of ZIGs $\mathcal{G}$ as (1) and propose a novel Dual Half-Space Projected Gradient (DHSPG) to solve it.

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}),\quad \text{s.t. Card}\{g\in\mathcal{G}|[\boldsymbol{x}]_g=0\}\leq K, \tag{1}$$

where K is the target group sparsity level. Larger K indicates higher group sparsity in the solution, hence typically results more aggressive FLOPs and parameter quantiy reduction.

Related Optimizers. To solve such constrained problem, ADMM converts it into a min-max problem, but can not tackle the non-smooth and non-convex hard constraint of sparsity without hurting the objective, thus necessitates extra fine-tuning afterwards (Lin et al., 2019). HSPG in OTOv1 (Chen et al., 2021) and proximal methods (Xiao & Zhang, 2014) relax it into a non-constrained mixed ℓ_1/ℓ_2 regularization problem, but can not guarantee the sparsity constraint since they are sometimes sensitive to the regularization coefficient. In addition, the augmented regularizer penalize the magnitude of the entire trainable variables which restricts the search space to be surrounding the origin point, then may miss the local optima with higher generalization, see Figure 3.

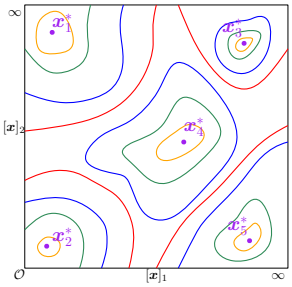


Figure 3: Local optima $\boldsymbol{x}^*\in\mathbb{R}^2$ distribution.

Algorithm Outline for DHSPG. To resolve the drawbacks of the existing optimization algorithms for solving (1). We propose a fresh and novel algorithm, named Dual Half-Space Projected Gradient (DHSPG), stated as Algorithm 3, with two takeaways.

Algorithm 3 Dual Half-Space Projected Gradient (DHSPG)

1: **Input:** an initial variable $\boldsymbol{x}_0\in\mathbb{R}^n$, an initial learning rate α_0, and target group sparsity level K.
2: **for** $t=0,1,2\cdots$, **do**
3: **if** $\text{mod}(t,T)=0$ **then**
4: Partition ZIGs to $\mathcal{G}_p$ and $\mathcal{G}_{np}$ as (2).
5: Compute gradient estimate $\nabla f(\boldsymbol{x}_t)$ or its variant.
6: Update variables for $\mathcal{G}_{np}$,

$$[\boldsymbol{x}_{t+1}]_{\mathcal{G}_{np}}\leftarrow[\boldsymbol{x}_t]_{\mathcal{G}_{np}}-\alpha_t[\nabla f(\boldsymbol{x}_t)]_{\mathcal{G}_{np}}.$$

7: Select proper λ_g given $\boldsymbol{x}_t$ and $\nabla f(\boldsymbol{x}_t)$.
8: Compute trial variables for $g\in\mathcal{G}_p$,

$$[\tilde{\boldsymbol{x}}_{t+1}]_g\leftarrow[\boldsymbol{x}_t]_g-\alpha_t\left([\nabla f(\boldsymbol{x}_t)]_g+\lambda_g\frac{[\boldsymbol{x}_t]_g}{\|[\boldsymbol{x}_t]_g\|_2}\right).$$

9: Compute $\boldsymbol{x}_{t+1}$ by Half-Space projection given

$$[\boldsymbol{x}_{t+1}]_{\mathcal{G}_p}=\text{Proj}_{\text{Half-Space}}([\tilde{\boldsymbol{x}}_{t+1}]_{\mathcal{G}_p},[\boldsymbol{x}_t]_{\mathcal{G}_p}).$$

10: Update α_{t+1}.
11: **Return** the final iterate $\boldsymbol{x}^*$.

Partition Groups. To avoid trapping in the local optima near the origin point, we further partition the groups in $\mathcal{G}$ into two subsets, one of which magnitudes of variables being penalized $\mathcal{G}_p$, and the other one being not $\mathcal{G}_{np}$, *i.e.*, $\mathcal{G}=\mathcal{G}_p\bigcup\mathcal{G}_{np}$. Different criteria can be applied here to construct the above partition based on salience score, *e.g.*, cosine-similarity $\cos(\theta_g)$ between the projection direction $-[\boldsymbol{x}]_g$ and the negative gradient or its estimation $-[\nabla f(\boldsymbol{x})]_g$. Higher cos-similarity over $g\in\mathcal{G}$ indicates that projecting the group of variables in g onto zeros is more likely to make progress to the optimality of f (considering the descent direction from the perspective of optimization). The magnitude of variables over $[\boldsymbol{x}]_g$ then need to be penalized. Therefore, we compute $\mathcal{G}_p$ by picking up the ZIGs with top-K highest salience scores and $\mathcal{G}_{np}$ as the complementary of $\mathcal{G}_p$ as follows. The partition can be periodically proceeded during the training process.

$$\mathcal{G}_p=(\text{Top-K})\underset{g\in\mathcal{G}}{\arg\max}\,\text{salience-score}(g),\ \text{and}\ \mathcal{G}_{np}=\mathcal{G}\backslash\mathcal{G}_p. \tag{2}$$

Update Variables. For the groups of variables in $\mathcal{G}_{np}$ of which magnitudes are not penalized, we proceed vanilla gradient descent or its variants, such as Adam (Kingma & Ba, 2014), *i.e.*, $[\boldsymbol{x}_{k+1}]_{\mathcal{G}_{np}}\leftarrow[\boldsymbol{x}_k]_{\mathcal{G}_{np}}-\alpha_k[\nabla f(\boldsymbol{x}_k)]_{\mathcal{G}_{np}}$. For the groups of variables in $\mathcal{G}_p$ to penalize magnitude, we seek to find out redundant groups as zero, but instead of directly projecting them onto zero as ADMM which easily destroys the progress to the optimum, we formulate a relaxed non-constrained subproblem as (3) to gradually reduce the magnitudes without deteriorating the objective and project

groups onto zeros only if the projection serves as a descent direction during the training process.

$$\underset{[\boldsymbol{x}]_{\mathcal{G}_p}}{\text{minimize}}\ \psi([\boldsymbol{x}]_{\mathcal{G}_p}) := f\left([\boldsymbol{x}_{\mathcal{G}_p}]\right) + \sum_{g\in\mathcal{G}_p} \lambda_g \left\|[\boldsymbol{x}]_g\right\|_2, \tag{3}$$

where λ_g is a group-specific regularization coefficient and needs to be delicately chosen to guarantee decrease of both variable magnitude for g as well as objective f. Intuitively, the search direction $[\boldsymbol{d}(\boldsymbol{x})]_{\mathcal{G}_p} := -[\nabla f(\boldsymbol{x})]_{\mathcal{G}_p} - \sum_{g\in\mathcal{G}_p} \lambda_g [\boldsymbol{x}]_g / \max\{\|[\boldsymbol{x}]_g\|_2, \tau\}$, *i.e.*, a negative subgradient of ψ, serves as a descent direction for both f and $\|[\boldsymbol{x}]_g\|$, *i.e.*, $[\boldsymbol{d}(\boldsymbol{x})]_{\mathcal{G}_p}^\top [-\nabla f(\boldsymbol{x})]_{\mathcal{G}_p} > 0$ and $[\boldsymbol{d}(\boldsymbol{x})]_{\mathcal{G}_p}^\top [-\boldsymbol{x}]_{\mathcal{G}_p} > 0$. In other words, $[\boldsymbol{d}(\boldsymbol{x})]_g$ falls into the intersection between the dual half-spaces with normal directions as $[-\nabla f]_g$ and $-[\boldsymbol{x}]_g$ for any $g \in \mathcal{G}_p$ as shown in Figure 4. It further indicates that λ_g locates in the interval $(\lambda_{\text{min},g}, \lambda_{\text{max},g}) := \left(-\cos(\theta_g)\left\|[\nabla f(\boldsymbol{x})]_g\right\|_2, -\frac{\|[\nabla f(\boldsymbol{x})]_g\|_2}{\cos(\theta_g)}\right)$ if $\cos(\theta_g) < 0$ otherwise can be arbitrary positive constant. The above selection of λ_g guarantees decrease of both objective function and variable magnitude. We then compute a trial iterate and feed it into the Half-Space projector which outperforms proximal operator (Chen et al., 2021) to yield group sparsity more productively without hurting objective as line 8-9 in Algorithm 3. Remark here that OTOv1 utilizes a global regularization coefficient λ for all groups, thus lacks sufficient capability to guarantee both aspects for each individual group in accordance.

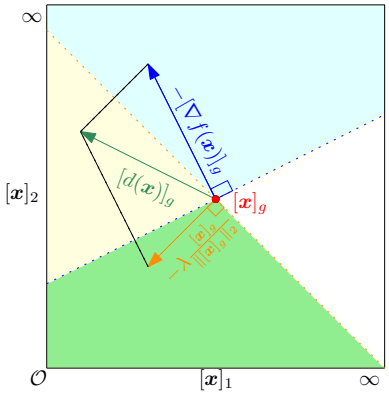


Figure 4: Search direction in DHSPG.

Convergence and Complexity Analysis. DHSPG converges to the solution of (1) $\boldsymbol{x}^*_{\text{DHSPG}}$ in the manner of both theory and practice. In fact, the theoretical convergence relies on the the construction of dual half-space mechanisms which yield sufficient decrease for both objective f and variable magnitude, see Lemma 2 and Corollary 1 in Appendix B. Together with the Half-Space projector, DHSPG effectively computes a solution with desired group sparsity, see Theorem 1 in Appendix B. In addition, DHSPG consumes the same time complexity $\mathcal{O}(n)$ as other first-order methods, such as SGD and Adam, since all of its operations can be finished within linear time.

3.3 Automated Compressed Model Construction

In the end, given the solution $\boldsymbol{x}^*_{\text{DHSPG}}$ with both high performance and group sparsity, we now automatically construct a compact model which is a manual step with unavoidable substantial engineering efforts in OTOv1. In general, we traverse all vertices with trainable parameters, then remove the structures in accordance with ZIGs being zero, such as the dotted rows of $\widehat{\mathcal{K}}_1, \widehat{\mathcal{K}}_2, \widehat{\mathcal{K}}_3$ and scalars of $\boldsymbol{b}_2, \boldsymbol{\gamma}_1, \boldsymbol{\beta}_1$ as illustrated in Figure 5. Next, we erase the redundant parameters that affiliates with the removed structures of their incoming stem vertices to keep the operations valid, *e.g.*, the second and third channels in g_5 are removed though g_5 is not zero. The automated algorithm is promptly complete in linear time via performing two passes of depth-first-search and manipulating parameters to produce a more compact model $\mathcal{M}^*$. Based on the property of ZIGs, $\mathcal{M}^*$ returns the same inference outputs as the full $\mathcal{M}$ parameterized as $\boldsymbol{x}^*_{\text{DHSPG}}$ thus no further fine-tuning is necessary.

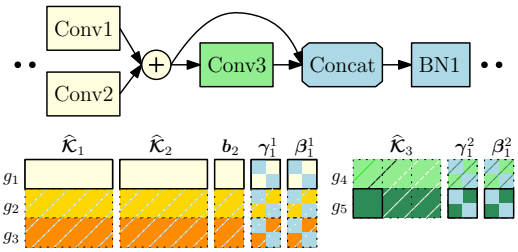


Figure 5: Automated compressed model construction. $\mathcal{G} = \{g_1, g_2, \cdots, g_5\}$ and $[\boldsymbol{x}^*_{\text{DHSPG}}]_{g_2\cup g_3\cup g_4} = \boldsymbol{0}$.

4 Numerical Experiments

In this section, we numerically demonstrate the end-to-end effectiveness of OTOv2 framework by training and compressing arbitrary DNNs into slimmer networks with significant FLOPs and parameter quantity reduction without fine-tuning over a variety of complex model architectures and benchmark datasets. We first verify the correctness of automated ZIG partition and compact model

construction by employing OTOv2 onto several DNNs which structures are even more complicated than the DNNs used in the benchmark experiments. Then, we compare OTOv2 with the existing methods on the benchmark experiments to show the competitive even superior performance. Together with autonomy, user-friendliness and generality, OTOv2 becomes the new state-of-the-state.

Sanity of Automated ZIG and Compression The foremost step is to validate the correctness of the whole framework including both algorithm designs and infrastructure developments. We select three DNNs with complex topological structures, *i.e.*, StackedUnets, CARN (Ahn et al., 2018) and DenseNet121 (Huang et al., 2017), and DemoNet, all of which are not easily to be compressed via the existing non-automatic methods unless having sufficient domain knowledge and spending extensive handcrafting efforts. Remark here that StackedUnets consumes two input tensors, and is constructed by stacking two standard Unets (Ronneberger et al., 2015) with different downsamplers and aggregate the outputs together. To more intuitively illustrate their complicated structural hierarchies, we provided the visualizations of the grown connected components for ZIG partitions here. To quantitatively measure the performance of OTOv2, we further employ these model architectures onto a variety of benchmark datasets, *e.g.*, Fashion-MNIST(Xiao et al., 2017), SVNH (Netzer et al., 2011) and CIFAR100 (Krizhevsky & Hinton, 2009). The main results are present in Table 1.

Table 1: OTOv2 on extensive DNNs and datasets.

Backend	Dataset	Method	FLOPs	# of Params	Top-1 Acc.
DemoNet	Fashion-MNIST	Baseline	100%	100%	**84.8%**
DemoNet	Fashion-MNIST	**OTOv2**	25.3%	24.9%	84.6%
DemoNet	Fashion-MNIST	**OTOv2**	**2.2%**	**2.3%**	82.9%
StackedUnet	SVNH	Baseline	100%	100%	94.8%
StackedUnet	SVNH	**OTOv2**	**26.4%**	**17.0%**	**95.0%**
DenseNet121	CIFAR100	Baseline	100%	100%	**77.0%**
DenseNet121	CIFAR100	**OTOv2**	**22.2%**	**26.7%**	75.4%

Compared with the baselines trained by vanilla SGD, under the same amount of training cost, OTOv2 automatically reaches not only the competitive performance but also remarkable speed up in terms of FLOPs (time complexity) and parameter quantity (space complexity) reduction. In particular, as displayed in Table 1, the slimmer DemoNet computed by OTOv2 negligibly regresses the top-1 accuracy by 0.2% but reduces the FLOPs and number of parameters by 74.7% and 75.1% respectively. Furthermore, the slimmer DemoNet under more aggressive target group sparsity utilizes merely 2.3% of the original parameters to drastically reduce 97.8% FLOPs with a slight 1.9% accuracy sacrifice. For StackedUnet, the compressed model speeds up the inference by 3.8X in terms of FLOPs and even surpasses the baseline accuracy by 0.2%. Consistent phenomena also holds for DenseNet121 that the slimmer architecture is about 4-5 times more efficient than the original full model but with competitive top-1 accuracy. Therefore, the success of OTOv2 on these complicated model architectures serve as strong evidence to validate the sanity of the proposed algorithms as well as the infrastructure developments.

Benchmark Experiments The secondary step is to demonstrate the effectiveness of OTOv2 by comparing the performance with other state-of-the-arts on benchmark compression experiments, *i.e.*, common architectures such as VGG16 (Simonyan & Zisserman, 2014) and ResNet50 (He et al., 2016) as well as datasets CIFAR10 (Krizhevsky & Hinton, 2009) and ImageNet (Deng et al., 2009).

VGG16 on CIFAR10. We consider two versions of VGG16(-BN) with(out) batch normalization layer appending after every convolutional layer. OTOv2 automatically exploits the minimal removal structures of VGG16 and partitions the trainable variables into ZIGs. DHSPG is then triggered over the partitioned ZIGs to train the model from scratch to find a solution with high group sparsity and high performance. Finally, a slimmer VGG16(-BN) is automatically constructed without any fine-tuning. As shown in Table 2, the slimmer VGG16(-BN) leverages only 2.5%-3.6% of parameters to dramatically reduce the FLOPs by 83.7%-85.6% with the competitive top-1 accuracy to the full model, wherein the 91.4% outperforms previous state-of-the-arts. OTOv2 compressed VGG16-BN to maintain the baseline accuracy by the fewest 4.9% of parameters and 23.7% of FLOPs. Though SCP and RP reach higher accuracy, they requires 43%-100% more FLOPs than that of OTOv2.

ResNet50 on CIFAR10. To systematically compare with a few representative automatic pruning methods AMC and ANNC, we conduct their intersecting experiment. AMC establishes a reinforcement learning agent to guide layer-wise compression, while it only achieves autonomy over a few prescribed specific mod-

Table 3: ResNet50 for CIFAR10.

Method	FLOPs	# of Params	Top-1 Acc.
Baseline	100%	100%	93.5%
AMC (He et al., 2018b)	–	60.0%	93.6%
ANNC (Yang et al., 2020)	–	50.0%	**95.0%**
PruneTrain (Lym et al., 2019)	30.0%	–	93.1%
N2NSkip (Sharma et al., 2020)	–	10.0%	94.4%
OTOv1 (Chen et al., 2021)	12.8%	8.8%	94.4%
OTOv2	2.2%	1.2%	93.0%
OTOv2	16.9%	6.1%	94.5%

Table 2: VGG16 and VGG16-BN for CIFAR10. Convolutional layers are in bold.

Method	BN	Architecture	FLOPs	# of Params	Top-1 Acc.
Baseline	✗	**64-64-128-128-256-256-256-512-512-512-512-512-512**-512-512	100%	100%	91.6%
SBP (Neklyudov et al., 2017)	✗	**47-50-91-115-227-160-50-72-51-12-34-39-20**-20-272	31.1%	5.9%	91.0%
BC (Louizos et al., 2017)	✗	**51-62-125-128-228-129-38-13-9-6-5-6-6**-6-20	38.5%	5.4%	91.0%
RBC (Zhou et al., 2019)	✗	**43-62-120-120-182-113-40-12-20-11-6-9-10**-10-22	32.3%	3.9%	90.5%
RBP (Zhou et al., 2019)	✗	**50-63-123-108-104-57-23-14-9-8-6-7-11**-11-12	28.6%	2.6%	91.0%
OTOv1 (Chen et al., 2021)	✗	**21-45-82-110-109-68-37-13-9-7-3-5-8**-170-344	16.3%	**2.5%**	91.0%
OTOv2	✗	**22-43-73-116-162-123-45-15-10-8-5-8**-134-186	19.7%	3.6%	**91.4%**
OTOv2	✗	**22-30-56-102-142-101-28-11-6-6-5-5**-101-127	**13.4%**	**2.5%**	91.0%
Baseline	✓	**64-64-128-128-256-256-256-512-512-512-512-512-512**-512-512	100%	100%	93.2%
EC (Li et al., 2016)	✓	**32-64-128-128-256-256-256-256-256-256-256-256-256**-512-512	65.8%	37.0%	93.1%
Hinge (Li et al., 2020b)	✓	–	60.9%	20.0%	93.6%
SCP (Kang & Han, 2020)	✓	–	33.8%	7.0%	93.8%
OTOv1 (Chen et al., 2021)	✓	**22-56-93-123-182-125-95-45-27-21-10-13-19**-244-392	26.8%	5.5%	93.3%
RP (Li et al., 2022)	✓	–	47.9%	42.1%	**93.9%**
CPGCN (Di Jiang & Yang, 2022)	✓	–	26.9%	6.9%	93.1%
OTOv2	✓	**14-51-77-122-183-146-92-41-16-13-8-11-14**-107-183	**23.7%**	**4.9%**	93.2%

els and requires multiple-stage training costs. Simple pruning methods such as ANNC and SFW-pruning (Miao et al., 2021) do not construct slimmer models besides merely projecting variables onto zero. OTOv2 overcomes all these drawbacks and first realizes end-to-end autonomy to only train arbitrary DNNs once and compress it into a slimmer counterpart with high performance. Furthermore, OTOv2 achieves the state-of-the-art results on this intersecting ResNet50 on CIFAR10 experiment. In particular, as shown in Table 3, under the most aggressive group sparsity level, OTOv2 utilizes only 1.2% parameters and 2.2% FLOPs to reach 93.0% top-1 accuracy with negligible 0.5% regression. Under moderate group sparsity level, OTOv2 achieves competitive accuracy as 94.5% to other pruning methods but makes use of substantially fewer parameters and FLOPs.

ResNet50 on ImageNet. We finally employ OTOv2 to ResNet50 on ImageNet. Similarly to other experiments, OTOv2 first automatically partitions the trainable variables of ResNet50 into ZIGs, then trains it once by DHSPG, and automatically construct slimmer models without fine-tuning. We report a performance portfolio of the slimmer ResNet50s under various target group sparsity ranging from 40% to 70% and compare with other state-of-the-art methods in Table 4. Remark here that more reliably controlling the ultimate sparsity level to meet various deployment environments is a significant superiority of DHSPG to the HSPG. It is noticeable that increasing target group sparsity results in more FLOPs and parameter quantity reduction, meanwhile sacrificing more accuracy. In particular, under 70% group sparsity, the slimmer ResNet50 by OTOv2 achieves the fewest FLOPs (14.5%) and parameters (21.3%) among all while its top-1 accuracy 70.4% is still competitive to SFP and RBP especially under 3x fewer FLOPs. The one with 72.5% top-1 accuracy under group sparsity as 60% is competitive to CP, DDS-26 and RRBP, but 2-3 times more efficient by consuming only 20.7% FLOPs and 28.6% parameters. In addition, the slimmer ResNet50s under 40% and 50% group sparsity achieves the accuracy milestone, *i.e.*, around 75% top-1 accuracy. Meanwhile, the FLOPs and parameter quantity of both settings reduction outperforms most of other state-of-the-arts. Combining with autonomy and only-train-once, OTOv2 exhibits the state-of-the-art end-to-end performance.

Table 4: ResNet50 for ImageNet.

Method	FLOPs	# of Params	Top-1/5 Acc.
Baseline	100%	100%	76.1% / 92.9%
CP (He et al., 2017)	66.7%	–	72.3% / 90.8%
ThiNet-50 (Luo et al., 2017b)	44.2%	48.3%	71.0% / 90.0%
DDS-26 (Huang & Wang, 2018)	57.0%	61.2%	71.8% / 91.9%
SFP (He et al., 2018a)	41.8%	–	74.6% / 92.1%
RBP (Zhou et al., 2019)	43.5%	48.0%	71.1% / 90.0%
RRBP (Zhou et al., 2019)	45.4%	–	73.0% / 91.0%
GBN-60 (You et al., 2019)	59.5%	68.2%	76.2% / 92.8%
Group-HS (Yang et al., 2019)	52.9%	-	**76.4% / 93.1%**
Hinge (Li et al., 2020b)	46.6%	–	74.7% / —
SCP (Kang & Han, 2020)	45.7%	-	74.2% / 92.0%
ResRep (Ding et al., 2021)	45.5%	-	76.2% / 92.9%
OTOv1 (Chen et al., 2021)	34.5%	35.5%	74.7% / 92.1%
RP (Li et al., 2022)	49.0%	54.1%	75.1% / 92.5%
GNN-RL(Yu et al., 2022)	46.0%	–	74.3% / —
CHEX(Hou et al., 2022)	25.0%	–	76.0% / —
OTOv2 (70% group sparsity)	**14.5%**	**21.3%**	70.4% / 89.6%
OTOv2 (60% group sparsity)	20.7%	28.6%	72.5% / 90.7%
OTOv2 (50% group sparsity)	28.7%	37.4%	74.5% / 92.1%
OTOv2 (40% group sparsity)	39.2%	50.1%	75.3% / 92.3%

5 Conclusion

We propose OTOv2 that first reach the north star of the model compression task via pruning, *i.e.*, automatically trains an arbitrary DNN only once and compresses it into a more compact counterpart without pre-training or fine-tuning to significantly reduce FLOPs and parameter quantity. The success of OTOv2 stems from two revolutionary improvements compared with OTOv1: *(i)* automated ZIG partition and compressed model construction; and *(ii)* DHSPG method to more reliably solve structured-sparsity problem. We leave the incorporation with NAS as future work.

REFERENCES

Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 252–268, 2018.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. 2019.

Tianyi Chen, Bo Ji, Yixin Shi, Tianyu Ding, Biyi Fang, Sheng Yi, and Xiao Tu. Neural network compression via sparse optimization. *arXiv preprint arXiv:2011.04868*, 2020.

Tianyi Chen, Bo Ji, Tianyu Ding, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only train once: A one-shot neural network training and pruning framework. In *Advances in Neural Information Processing Systems*, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Yuan Cao Di Jiang and Qiang Yang. On the channel pruning using graph convolution network for convolutional neural network acceleration. 2022.

Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Lossless cnn channel pruning via decoupling remembering and forgetting. *Proceedings of the IEEE International Conference on Computer Vision*, 2021.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Shang-Hua Gao, Yong-Qiang Tan, Ming-Ming Cheng, Chengze Lu, Yunpeng Chen, and Shuicheng Yan. Highly efficient salient object detection with 100k parameters. In *European Conference on Computer Vision*, pp. 702–721. Springer, 2020.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018a.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018b.

Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. Chex: Channel exploration for cnn model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12287–12298, 2022.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018.

Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pp. 5122–5131. PMLR, 2020.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pp. 1–10. IEEE, 2015.

Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 979–984. IEEE, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pp. 528–536. PMLR, 2017.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European Conference on Computer Vision*, pp. 639–654. Springer, 2020a.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8018–8027, 2020b.

Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 191–201, 2022.

Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2800–2809, 2019.

Shaohui Lin, Rongrong Ji, Yuchao Li, Cheng Deng, and Xuelong Li. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE transactions on neural networks and learning systems*, 31(2):574–588, 2019.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in neural information processing systems*, pp. 3288–3298, 2017.

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

Jianhao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pp. 5058–5066, 2017b.

Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–13, 2019.

Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning filter in filter. *arXiv preprint arXiv:2009.14410*, 2020.

Lu Miao, Xiaolong Luo, Tianlong Chen, Wuyang Chen, Dong Liu, and Zhangyang Wang. Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining. In *International Conference on Learning Representations*, 2021.

Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pp. 6775–6784, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Avinash Sharma, Arvind Subramaniam, and . N2nskip: Learning highly sparse networks using neuron-to-neuron skip connections. In *BMVC*, 2020.

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

Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. *arXiv preprint arXiv:2005.07093*, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *arXiv preprint arXiv:1608.03665*, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2178–2188, 2020.

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*, 2019.

Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 184–199, 2018.

Yuefu Zhou, Ya Zhang, Yanfeng Wang, and Qi Tian. Accelerate cnn via recursive bayesian pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3306–3315, 2019.

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in Neural Information Processing Systems*, 33, 2020.