Differentiable Transportation Pruning

Yunqiang Li[1] Jan C. van Gemert[2] Torsten Hoefler[3]
Bert Moons[1] Evangelos Eleftheriou[1] Bram-Ernst Verhoef[1]

[1]Axelera AI [2]TU Delft [3]ETH Zurich

Abstract

Deep learning algorithms are increasingly employed at the edge. However, edge devices are resource constrained and thus require efficient deployment of deep neural networks. Pruning methods are a key tool for edge deployment as they can improve storage, compute, memory bandwidth, and energy usage. In this paper we propose a novel accurate pruning technique that allows precise control over the output network size. Our method uses an efficient optimal transportation scheme which we make end-to-end differentiable and which automatically tunes the exploration-exploitation behavior of the algorithm to find accurate sparse sub-networks. We show that our method achieves state-of-the-art performance compared to previous pruning methods on 3 different datasets, using 5 different models, across a wide range of pruning ratios, and with two types of sparsity budgets and pruning granularities.

1. Introduction

As the world is getting smaller, its edge is getting larger: more compute-limited edge devices are used. To unlock deep learning on the edge, deep networks need to be efficient and compact. There is a demand for accurate networks that fit exactly in pre-defined memory constraints.

Pruning is an effective technique [20, 41, 67] to find a sparse, compact model from a dense, over-parameterized deep network by eliminating redundant elements such as filters or weights while retaining accuracy. Pruning is a hard problem because finding a good-performing sub-network demands selecting which part of a network to keep —and which part to prune—, requiring an expensive search over a large discrete set of candidate architectures.

Recent work [23, 52, 40, 25] finds that optimizing a continuously differentiable mask to sparsify network connection while simultaneously learning network parameters facilitates search space exploration and enables gradient-based optimization. Prior methods [23, 40], however, control the sparsity, *i.e.* pruned network size, by an additional penalty term in the loss function that is difficult to optimize and tune, and hampering a precise control over the sparsity.

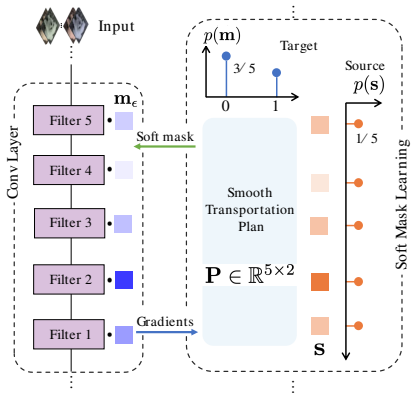


Figure 1. During training we multiply the filters in a layer with soft masks $\mathbf{m}_\epsilon$. The soft masks are obtained by solving an optimal transport problem: minimizing a transportation cost between a uniform source distribution over trainable importance scores $\mathbf{s}$ and a Bernoulli target distribution defined by sparsity ratio *i.e.* $3/5$. During training the soft masks gradually converge to hard masks.

In this paper, as illustrated in Fig. 1, we propose a fully differentiable transportation pruning method that allows precise control of the output network size. We draw inspiration from Xie et al. (2020) [65] who formulate a soft top-k operator based on entropic optimal transport. Recent work [59] shows good pruning results with the soft top-k operator. This soft top-k operator, however, is computationally expensive [10] and relies on implicit, and therefore less accurate, gradients [11] for back-propagation. Instead, we speed up the soft top-k operator with an approximate bi-level optimization which is optimized using accurate automatic differentiation [48]. Moreover, in contrast to [65, 59] we can automatically tune the temperature hyper-parameter.

We make the following contributions: i) A fully differentiable pruning method that allows precise control over output network size; ii) An efficient entropic optimal transportation algorithm that significantly reduces the computational complexity of bi-level optimization; iii) We increase the ease-of-use of the algorithm by means of an automatic temperature annealing mechanism instead of a manually-chosen decay schedule; iv) We show state-of-the-art results on 3 datasets with 5 models for structured and unstructured pruning using layer-wise and global sparsity budgets.

2. Related Work

Pruning in deep learning. Network pruning makes models smaller by removing elements that do not contribute significantly to accuracy. Pruning methods can be subdivided into methods that promote structured (*i.e*. blocked) [42, 27] or unstructured (*i.e*. fine-grained) sparsity [6, 13, 43, 61, 68], see [22] for a review. Filter pruning is a form of structured sparsity by removing entire filters from the network's layers [34, 30]. Filter pruning often achieves practical network compression and significant acceleration as entire feature maps are no longer computed. For these reasons, we here focus on filter pruning.

Selection criteria for pruning. Network pruning can be phrased as a form of neural architecture search [38]. Such an architecture search may involve training and evaluating several random subnetworks based on leave-some-out approaches [5, 58, 32], but this can be quite expensive for large models. More efficient approaches assign an importance score as selection criteria. This score can be based on L1/L2 norm of weights [67], geometric median [20] of filters, Kullback–Leibler divergence [41], Wasserstein barycenter of channel probability vectors [55], prediction error [45], or empirical sensitivity of a feature map [33]. These methods first prune a pretrained model based on the importance scores and subsequently fine-tune the pruned model so the model can adapt to the reduced set of parameters. Once pruned, however, the model is not allowed to explore other subnetworks. Here, we also use importance scores, which we learn as latent variables from which soft masks are computed through optimal transport. The soft mask is automatically annealed to a hard mask during training, allowing the model to explore different subnetworks in the process.

Differentiable continuous relaxation. Recent work aims to select the best sub-architecture by training a binary mask on the network weights [15]. This binary mask, however, is non-differentiable and applying a hard binary mask during the forward pass may impair network performance [23]. Other methods approximate binary masks using a sigmoid function, which gradually converges to a binary step function by means of a decaying temperature hyper-parameter [23, 25, 40, 52]. To control the sparsity ratio, these methods usually introduce a sparsity penalty term in the loss function, which requires an additional difficult-to-tune hyper-parameter if an exact pruning ration is required. Our method mitigates these issues as it allows for an exact pruned network size using optimal transport and decays the temperature variable automatically.

Learning soft mask via entropic optimal transport. Our method draws inspiration from the work of Xie et al. (2020) [65], who formulate a soft top-k operator based on entropic optimal transport [51], optimized via Sinkhorn's algorithm [8]. The authors apply their method to predictive modeling applications such as information retrieval. As remarked by [10], this method can be computationally expensive if applied to pruning, as hundreds of Sinkhorn iterations are needed in the forward pass of each update step. In this paper we phrase network pruning as an efficient optimal transport problem. Compared to [65] our method significantly reduces the number of Sinkhorn iterations required in the forward pass. Furthermore, in contrast to [65], who employ implicit gradients [11], our method relies on accurate automatic differentiation [48]. Finally, we replace the manually-chosen decay schedule of the entropic hyperparameter in [65] with an automatic schedule, thereby simplifying the optimization process.

Bregman divergence based proximal method. Previous work [26, 53] has applied to optimal transport the proximal point algorithm [47] based on the Bregman divergence [4] to alleviate the potential numerical instability of Sinkhorn iterations. Xie *et al*. [66] use the proximal algorithm to optimize optimal transport in deep generative models [2, 16] and achieve good performance, but still performs a large amount of proximal point iterations per mini-batch step, making it computational inefficient for our pruning purposes. In addition, its back-propagation pass [66] relies on the envelope theorem [1] that does not go into proximal point iterations to accelerate training, which causes inaccurate gradients. Instead, we only use a single proximal point iteration, *i.e*. a single Sinkhorn iteration, per mini-batch step and compute gradients via automatic differentiation [48].

Budget aware pruning. Budgeted pruning focuses on compressing the network subject to the prescribed explicit resource constraints, such as targeted amount of layer-wise filters [28], global filters [60], FLOPs [29, 17], or execution latency [54, 24]. Our method can directly control the pruned network size given a sparsity budget. Moreover, we show how it can be easily extended to general budgeted pruning by learning budget-constrained sparsity across layers [46]. Our technique improves the accuracy while using less hyperparameters, *e.g*. ChipNet [60] adds 6 extra hyperparameters while we only introduce a single temperature hyperparameter, which eases hyperparameter tuning effort.

3. Differentiable Transportation Pruning

Given a neural network $f(\mathbf{x}; \mathbf{w})$ with input $\mathbf{x}$ and a set of n filters forming the columns of weight matrix $\mathbf{w} \in \mathbb{R}^{m \times n}$, filter pruning removes filters by applying a binary mask $\mathbf{m} \in \{0, 1\}^n$ to the n filters ($1 =$ kept, $0 =$ pruned).

If we need to keep exactly k out of n filters, then the sparse learning problem can be formulated as:

$$\begin{aligned} &\underset{\mathbf{m},\mathbf{w}}{\arg\min} \;\; \mathcal{L}_{\text{train}}\big(f(\mathbf{x}; \mathbf{w}, \mathbf{m})\big), \\ &\text{s.t.} \;\; \sum\nolimits_{i=1}^{n} \mathbf{m}_i = k, \;\; \mathbf{m} \in \{0, 1\}^n, \end{aligned} \tag{1}$$

where $\mathcal{L}_{\text{train}}$ is the training loss, *e.g*. cross entropy loss.

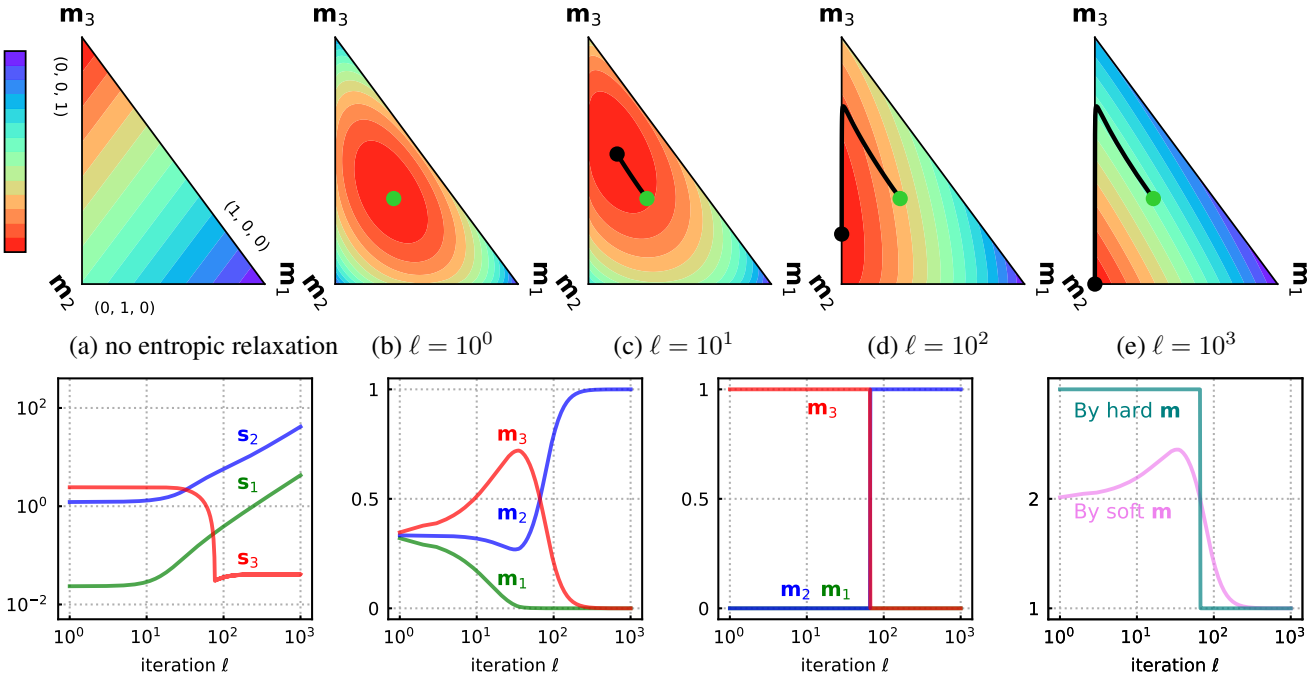


Figure 2. **Illustrative example:** The figure is best understood after Eq. (8) and it illustrates how our method can prune 2 weights out of 3 by learning a binary mask $\mathbf{m} \in \{0,1\}^3$. We can visualize in 2D because the simplex $\sum_3$ can be shown as a triangle in two dimensions. In this example we chose to favor the 2nd weight, and thus define some loss on the weights as $\mathcal{L}_{\text{train}} = \mathbf{w}^\top \mathbf{m}$ where we define $\mathbf{w} = [2, 1, 3]$. To keep just a single weight, we set k to 1, yielding the constraint $\sum_{i=1}^{3} \mathbf{m}_i = 1$ from Eq. (1), where the optimal solution is $\mathbf{m} = (0, 1, 0)$. *The top row (a)-(e)* illustrates the solution space, where the corners of the triangle are the hard masks: $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. Specifically, (a), (b) denote the sparse and non-sparse solution space without or with entropy regularization at $\ell = 1$. The color bar indicates low (red) and high (violet) loss values. A green dot in the center is the initial position; and the black dot is the current position. *the bottom row (f)-(i)*: training curve of importance score, soft mask, discrete mask and training loss for one thousand training steps. The hard mask in (h) is obtained by Eq. (6). *(a)*: the discrete optimal transportation loss at $\ell = 1$ just after initializing the importance scores $\mathbf{s}$. Note the ordering of $\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3}$ in (f) at $\ell = 1$, which explains the low loss value for $\mathbf{m_3}$. *(b)-(e)*: Snapshots at iterations $\ell = 10^0, 10^1, 10^2, 10^3$ after making the discrete optimal transport problem differentiable by entropy regularization in Eq. (7) where $\varepsilon = 10$. Note how the black dot moves to the correct solution. Note the training loss $\mathcal{L}_{\text{train}}$ in (i), as it pushes $\mathbf{s}_2$ towards top-1 in (f), and minimizing transportation loss pushes the soft mask $\mathbf{m}_2$ towards 1 in (g). The minimum value of the training loss is 1, due to how we chose the dot product loss $\mathcal{L}_{\text{train}}$.

The binary mask $\mathbf{m}$ is discrete and therefore difficult to optimize with gradient descent. To overcome this, we make the binary mask depend on a continuous latent importance score $\mathbf{s}$ which can be optimized by gradient descend. By sorting these importance scores we allow exact control over the pruned output network size by keeping only the top-k importance scores. Specifically, the top-k operator first sorts the importance scores $\mathbf{s}$, and then assigns the top k importance scores to 1 and the remaining importance scores to 0, that is:

$$\mathbf{m}_i = \begin{cases} 1, & \text{if } \mathbf{s}_i \text{ is in the top-}k \text{ of } \mathbf{s}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The top-k operator can be parameterized in terms of the solution of an optimal transport problem [65]. This allows us to parameterize the pruning mask in terms of optimal transport, which is the problem of efficiently moving probability mass from a source distribution to a target distribution. In our case, we know the target Bernoulli distribution over the binary values, which is given by the fraction of the k filters out of the n total filters to keep: $P(1) = \frac{k}{n}$ and $P(0) = \frac{n-k}{n}$. Then, the optimization problem reduces to minimizing the transportation costs of moving probability mass from the trainable importance scores $\mathbf{s}$ to this target distribution. See Fig. 1 for an illustration.

The two probability distributions used in optimal transport are over the importance scores $\mathbf{s}$ and over the two binary values. We define them by considering two discrete measures $\mathbf{a} = \sum_{i=1}^{n} \mathbf{a}_i \delta_{\mathbf{s}_i}$ and $\mathbf{b} = \sum_{j=1}^{2} \mathbf{b}_j \delta_{\mathbf{q}_j}$ supported on $\{\mathbf{s}_i\}_{i=1}^{n}$ and $\{\mathbf{q}_j : \{0,1\}\}$ respectively where δ_x is the Dirac at location x, intuitively a unit of mass which is infinitely concentrated at location x, the $\mathbf{a}_i$ and

$\mathbf{b}_j$ are the probability mass. Following [65], we define $\mathbf{a}$ as an empirical discrete uniform distribution (source) with $\mathbf{a}_i = 1/n$. Our binary mask variable is distributed as $\mathbf{b} = [1-(k/n), k/n]^\mathsf{T}$ (target).

These two distributions now allow optimal transportation optimization where probability mass is moved according to the cost it takes to move one unit of probability mass. We define the cost matrix $\mathbf{C} \in \mathbb{R}^{n\times 2}$ which aligns n filters (source) to the 2 binary options (target). The elements of the cost matrix are squared Euclidean distances and are denoted as:

$$\mathbf{C}_{i1} = \mathbf{s}_i^2,\ \mathbf{C}_{i2} = (\mathbf{s}_i - 1)^2,\ i = 1, 2, ..., n \tag{3}$$

where the targets are binary (0 or 1) and thus the target value for $\mathbf{C}_{i1}$ is 0, and the target value for $\mathbf{C}_{i2}$ is 1.

With the source distributions, target distribution and cost matrix defined, the optimal transportation plan can be formulated as:

$$\mathbf{P}^* = \underset{\mathbf{P}\in\mathcal{U}(\mathbf{a},\mathbf{b})}{\arg\min} \langle \mathbf{C}, \mathbf{P}\rangle, \tag{4}$$

where $\mathcal{U}(\mathbf{a},\mathbf{b}) = \{\mathbf{P} \in \mathbb{R}^{n\times 2} : \mathbf{P}\mathbb{1}_2 = \mathbf{a},\ \mathbf{P}^\mathsf{T}\mathbb{1}_n = \mathbf{b}\}$ is a set of coupling measures satisfying marginal constraints $\mathbf{P}\mathbb{1}_2 = \mathbf{a}$ and $\mathbf{P}^\mathsf{T}\mathbb{1}_n = \mathbf{b}$, $\mathbb{1}_n$ and $\mathbb{1}_2$ denote vectors with n ones and 2 ones, and $\mathbf{P} \in \mathbb{R}^{n\times 2}$ denotes the general probabilistic coupling matrix, *i.e.* the transportation plan, and $\langle\cdot,\cdot\rangle$ is the Frobenius dot-product.

The optimal transportation plan $\mathbf{P}^*$ can be computed as:

$$\mathbf{P}^*_{\sigma_i,1} = \begin{cases} 1/n, & \text{if } i \le k \\ 0, & \text{otherwise} \end{cases}, \mathbf{P}^*_{\sigma_i,2} = \begin{cases} 0, & \text{if } i \le k \\ 1/n, & \text{otherwise} \end{cases} \tag{5}$$

with σ being the sorting permutation, *i.e.* $\mathbf{s}_{\sigma_1} < \mathbf{s}_{\sigma_2} < \cdots < \mathbf{s}_{\sigma_n}$. Given $\mathbf{P}^*$, the binary mask can be parameterized as a function of the optimal transport plan:

$$\mathbf{m} = n\mathbf{P}^* \cdot [0, 1]^\mathsf{T}. \tag{6}$$

Differentiable transportation relaxation. So far, the derived hard mask in Eq. (6) is not differentiable everywhere with respect to importance scores, which can be seen from the sorting permutation applied in Eq. (5) and also in the lack of a gradient when using masking with a hard zero.

To make the method differentiable, we derive a soft mask by smoothing the optimal transportation plan with entropic regularization. By adding an entropy regularization term we extend the solution space by allowing non-sparse solutions of the transportation plan. That is, we now allow soft masks.

Define the discrete entropy of a coupling matrix as $\mathcal{H}(\mathbf{P}) = -\sum_{ij}\mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1)$, an entropic regularization is used to obtain smooth solutions to the original transport problem in Eq. (4):

$$\mathbf{P}^*_\varepsilon = \underset{\mathbf{P}\in\mathcal{U}(\mathbf{a},\mathbf{b})}{\arg\min} \langle \mathbf{C}, \mathbf{P}\rangle - \varepsilon\mathcal{H}(\mathbf{P}). \tag{7}$$

This objective is an ε-strongly convex function which has a unique solution. As $\varepsilon \to 0$ the unique solution $\mathbf{P}_\varepsilon$ of (7) converges to the original optimal sparse solution of (4), *i.e.* with hard masks.

Our soft masks $\mathbf{m}_\epsilon$ are now obtained by inserting $\mathbf{P}^*_\varepsilon$ into (6), as:

$$\mathbf{m}_\epsilon = n\mathbf{P}^*_\varepsilon \cdot [0, 1]^\mathsf{T}. \tag{8}$$

Interlude: illustrative example. We have now described the main ingredients of our method. This allows us to give an illustrative example of the main setting before we describe how to solve the optimization problem. In Fig. 2 we show a 2D visualization for pruning $n = 3$ importance scores to just a single score.

Dual formulation for optimization. To facilitate the optimization problem in (7), we make use of Lagrangean duality. That is, by introducing two dual variables $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^2$ for each marginal constraint of the transport plans in $\mathcal{U}(\mathbf{a},\mathbf{b})$, the Lagrangian of (7) is optimized, resulting in an optimal $\mathbf{P}_\varepsilon$ to the regularized problem given by:

$$\mathbf{P}_\varepsilon = e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}. \tag{9}$$

where $\odot$ is element wise product (see details in A.1). Note that the $\mathbf{P}_\varepsilon$ in Eq. (9) is differentiable with respect to $\mathbf{s}$.

Bi-level optimization. Now that we have relaxed the problem, we aim to jointly learn the dual variables $\mathbf{f}$, $\mathbf{g}$ and the model variables $\mathbf{s}$, $\mathbf{w}$. The dual variables are optimized by minimizing the Sinkhorn divergence as in Eq. (7). This is equivalent to maximizing its dual problem, denoted as $\mathcal{L}^\varepsilon_{\text{dual}}(\mathbf{f},\mathbf{g},\mathbf{s}) = \langle \mathbf{f}, \mathbf{a}\rangle + \langle \mathbf{g}, \mathbf{b}\rangle - \varepsilon\left\langle e^{\mathbf{f}/\varepsilon}, e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \cdot e^{\mathbf{g}/\varepsilon}\right\rangle$ (See details in A.2). Finally the model variables are obtained by minimizing the training loss $\mathcal{L}_{\text{train}}$.

This implies a *bi-level optimization problem* [7], *i.e.* an optimization problem which contains another optimization problem as a constraint. The model variables $\mathbf{s}$, $\mathbf{w}$ appear as the upper-level variables, and the dual variables $\mathbf{f}$, $\mathbf{g}$ as the lower-level variables. We rewrite the original problem in Eq. (1) as the following bi-level optimization problem:

$$\min_{\mathbf{s},\mathbf{w}}\ \mathcal{L}_{\text{train}}\Big(f\big(\mathbf{x}; \mathbf{w}, \mathbf{m}(\mathbf{P}^*_\varepsilon(\mathbf{s}))\big)\Big), \tag{10}$$

$$\text{s.t.}\ \ \mathbf{f}^*, \mathbf{g}^* = \arg\max_{\mathbf{f},\mathbf{g}}\ \mathcal{L}^\varepsilon_{\text{dual}}(\mathbf{f},\mathbf{g},\mathbf{s}), \tag{11}$$

where $\mathbf{P}^*_\varepsilon(\mathbf{s})$ is obtained by inserting the outcome from (11) into (9), and the mask $\mathbf{m}_\varepsilon$ in (10), dubbed soft mask, is computed by inserting $\mathbf{P}^*_\varepsilon(\mathbf{s})$ into Eq. (8).

Nonetheless, this bi-level optimization problem is expensive to train with Sinkhorn's algorithm: Firstly, per mini-batch the inner optimization in Eq. (11) needs to perform Sinkhorn's iterative algorithm for hundreds of iterations to converge, which is computationally inefficient. Secondly, during the back-propagation pass, the gradient of $\mathbf{P}^*_\varepsilon(\mathbf{s})$ with respect to the importance scores $\mathbf{s}$ is computed by differentiating [48] through all Sinkhorn iterations, which is

Algorithm 1: Differentiable Transportation Pruning

Probabilities $\mathbf{a} = \mathbb{1}_n/n$, $\mathbf{b} = [p, 1-p]^\mathsf{T}$ on supports $\{\mathbf{s}_i\}_{i=1}^n$, $\{\mathbf{m}_j\}_{j=1}^2$, weights: $\mathbf{w}$, training loss: $\mathcal{L}_{\text{train}}$, training iterations: L, learning rate: α, initialization: ε, $\mathbf{g}^{(1)} \leftarrow \mathbb{1}_2$, $\mathbf{P}^{(1)} \leftarrow \frac{1}{n}\mathbb{1}_n\mathbb{1}_2^\mathsf{T}$

for $\ell = 1, 2, ..., L$ **do**

- Cost matrix $\mathbf{C}$ with element $\mathbf{C}_{ij} = (\mathbf{s}_i^{(\ell)} - \mathbf{m}_j)^2$; Gibbs kernel $\mathbf{K} = e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}$
- Update dual variables: $\mathbf{f}^{(\ell+1)} = \varepsilon \log \mathbf{a} - \varepsilon \log (\mathbf{K}e^{\mathbf{g}^{(\ell)}/\varepsilon})$; $\mathbf{g}^{(\ell+1)} = \varepsilon \log \mathbf{b} - \varepsilon \log (\mathbf{K}^\mathsf{T}e^{\mathbf{f}^{(\ell+1)}/\varepsilon})$
- Update transportation plan $\mathbf{P}^{(\ell+1)} = e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \mathbf{K} \odot e^{\mathbf{g}^{(\ell+1)}/\varepsilon}$
- Update model variables with SGD: $\mathbf{s}^{(\ell+1)} = \mathbf{s}^{(\ell)} - \alpha\nabla_{\mathbf{s}}\mathcal{L}_{\text{train}}$; $\mathbf{w}^{(\ell+1)} = \mathbf{w}^{(\ell)} - \alpha\nabla_{\mathbf{w}}\mathcal{L}_{\text{train}}$

Derive pruned architecture based on soft mask $\mathbf{m} = n\mathbf{P}^{(L)} \cdot [0, 1]^\mathsf{T}$ for finetuning.

expensive (see details in A.3). Finally, we have also introduced an additional hyperparameter ε to decay, which makes the problem even harder to optimize.

Approximate bi-level optimization. We note that in our bi-level optimization the target distribution is a fixed Bernoulli distribution with parameter defined by the sparsity. The source distribution changes only when the trainable importance score updates via SGD. This differs from previous approaches [65, 66] involving a stochastic sampling of the target or source distribution and where the inner optimization in Eq. (11) is expected to converge per mini-batch step. We therefore propose to approximate $\mathbf{P}_\varepsilon^*(\mathbf{s})$ in Eq. (11) using only a single Sinkhorn step, *i.e.*, without optimizing the inner optimization problem until convergence per mini-batch step.

Given the ℓ-th mini-batch update step, we approximate the training loss in Eq. (10) with:

$$\begin{aligned}&\mathcal{L}_{\text{train}}\Big(f\big(\mathbf{x}; \mathbf{w}, \mathbf{m}(\mathbf{P}_\varepsilon^*(\mathbf{s}))\big)\Big)\\ \approx&\mathcal{L}_{\text{train}}\Big(f\big(\mathbf{x}; \mathbf{w}, \mathbf{m}(\mathbf{P}_\varepsilon^{(\ell+1)}(\mathbf{s}))\big)\Big),\end{aligned} \tag{12}$$

where $\mathbf{P}_\varepsilon^{(\ell+1)}(\mathbf{s})$ is obtained by applying a single Sinkhorn update step. Similar techniques for solving bi-level optimization have been used in architecture search [36], hyperparameter tuning [39] and meta-learning [12].

To enhance the convergence guarantee using just a *single Sinkhorn inner iteration*, we use generalized proximal point iteration based on Bregman divergence [53, 66] (see details in A.4), which can be viewed as applying iteratively the Sinkhorn algorithm [51] with a $e^{-\frac{\mathbf{C}}{\varepsilon/\ell}}$ kernel (see details in A.5). Thus, a proximal point iteration keeps track of the previous Sinkhorn update like Momentum [57] that incorporating a moving average of previous gradients, which accelerates the convergence and improves optimization stability. We therefore compute $\mathbf{P}_\varepsilon^{(\ell+1)}(\mathbf{s})$ in Eq. (12) as,

$$\mathbf{P}_\varepsilon^{(\ell+1)}(\mathbf{s}) = e^{\frac{\mathbf{f}^{(\ell+1)}}{\varepsilon}} \odot \big(e^{-\frac{\mathbf{C}(\mathbf{s})}{\varepsilon}} \odot \mathbf{P}^{(\ell)}\big) \odot e^{\frac{\mathbf{g}^{(\ell+1)}}{\varepsilon}}, \tag{13}$$

using the transportation plan $\mathbf{P}^{(\ell)}$ from the previous step. Let $\mathbf{K} = e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot \mathbf{P}^{(\ell)}$ denote the Gibbs kernel in the Sinkhorn algorithm, and the dual variables can be computed using a single Sinkhorn iteration, *i.e.* a proximal point iteration with block coordinate ascent as (see details in A.6):

$$\mathbf{f}^{(\ell+1)} = \varepsilon \log \mathbf{a} - \varepsilon \log (\mathbf{K}e^{\mathbf{g}^{(\ell)}/\varepsilon}); \tag{14}$$

$$\mathbf{g}^{(\ell+1)} = \varepsilon \log \mathbf{b} - \varepsilon \log (\mathbf{K}^\mathsf{T}e^{\mathbf{f}^{(\ell+1)}/\varepsilon}). \tag{15}$$

A side advantage of the updates in Eq. (13) is that it can iteratively decay the regularization parameter through ε/ℓ (see details in A.5). This means we can perform an **automatic decaying** on the regularization temperature, and as $\ell \to \infty$ the solution gradually approaches the optimal sparse transportation plan of the original objective (1).

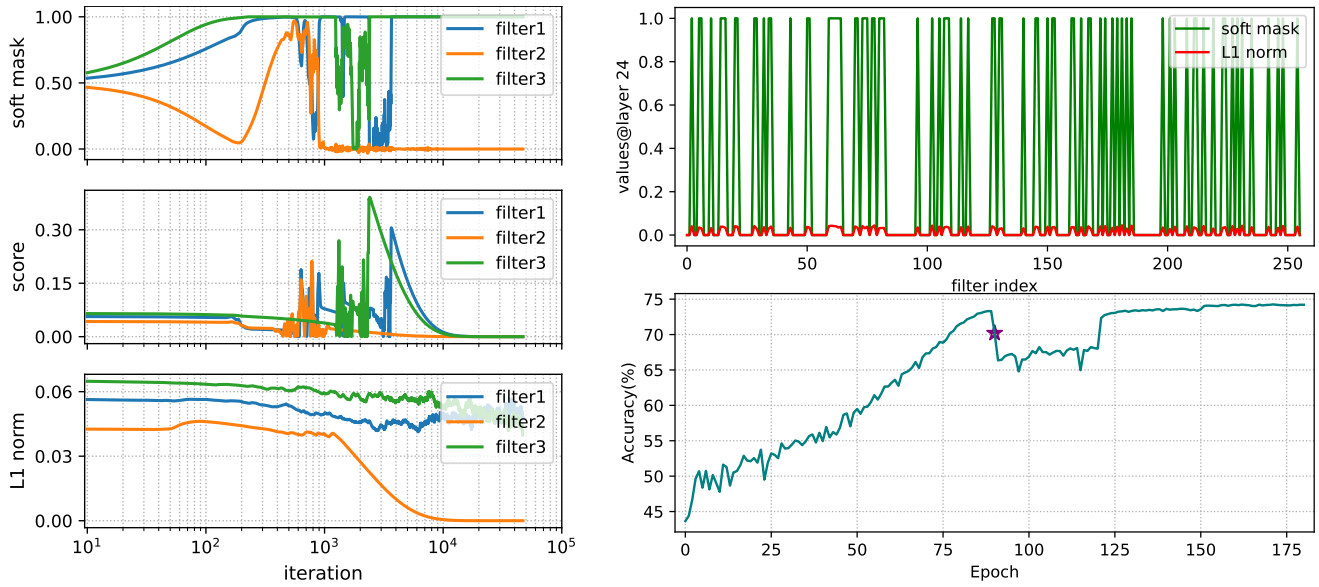
The gradient of Eq. (13) with respect to the importance scores $\mathbf{s}$ can be obtaind using automatic differentiation. We can therefore use normal SGD to jointly train the model weights $\mathbf{w}$ and importance scores $\mathbf{s}$ in Eq. (12).

Our method results in fast training and converges well as shown by our experiments in Section 4. After training, we derive the sparse architecture based on the soft masks $\mathbf{m} = n\mathbf{P}^{(L)} \cdot [0, 1]^\mathsf{T}$ and finetune the model. We outline the iterative training procedure in Algorithm 1.

4. Experiments

We validate our proposed method mainly on filter pruning using different models and datasets, and then extend to unstructured pruning. Filter pruning is a form of structured pruning that removes entire filters from the layers of a network, thereby compressing the network, and potentially accelerating the network by reducing the number of FLOPs needed to execute it.

Datasets and networks. We first conduct our experiments on CIFAR-10 with ResNet-56 and on CIFAR-100 with VGG-19. Following [63], for the CIFAR datasets we train our baseline models with accuracy comparable to those in the original papers. We then evaluate our method on the large-scale ImageNet dataset with ResNet-34 and ResNet-50. Following [63], we use the PyTorch [49] pretrained models as our baseline models. We also apply our method to prune lightweight architecture MobileNetV2.



(a) On CIFAR-10 (b) On ImageNet

Figure 3. Convergence analysis experiments. *(a).* Evolution of the soft masks associated with three filters from ResNet-56 trained on CIFAR-10 with $\varepsilon = 1$. Early in training our method explores different filter compositions. In later training phases the soft masks have converged to hard masks. For each mask, the soft mask, importance score and L1 norm of the respective filter are correlated. *(b).* Experiments with ResNet-50 on ImageNet with $\varepsilon = 0.25$. After training the soft mask has converged to a discrete hard mask solution. The L1 norms of the filters are highly correlated with the soft masks. Bottom: Validation accuracy during training with differential transportation pruning and during finetuning. The star point shows test accuracy for derived pruned architecture by soft mask before finetuning.

Table 1. Various pruning ratios with ResNet-56/VGG-19 on CIFAR-10/CIFAR-100: Accuracy comparison with GReg-1 [63] and L_1+one-shot [30]. Baseline accuracy for ResNet-56 is 93.36% and for VGG-19 is 74.02%. Each experiment is repeated 3 times with random initialization, mean and std accuracies are reported.

ResNet-56 (CIFAR-10)	Pruning ratio (%)	50	70	90	92.5	95
	Sparsity (%) / Speedup	49.82/1.99×	70.57/3.59×	90.39/11.41×	93.43/14.76×	95.19/19.31×
	L_1+one-shot (Acc.%) [30]	92.97±0.15	91.88±0.09	87.34±0.21	87.31±0.28	82.79±0.22
	GReg-1 (Acc. %) [63]	93.06±0.09	92.23±0.21	89.49±0.23	**88.39±0.15**	85.97±0.16
	Ours (Acc. %)	**93.46±0.18**	**92.46±0.10**	**89.84±0.14**	88.03±0.33	**86.18±0.06**
VGG-19 (CIFAR-100)	Pruning ratio (%)	50	60	70	80	90
	Sparsity (%) / Speedup	74.87/3.60×	84.00/5.41×	90.98/8.84×	95.95/17.30×	98.96/44.22×
	L_1+one-shot (Acc.%) [30]	71.49±0.14	70.27±0.12	66.05±0.04	61.59±0.03	51.36±0.11
	GReg-1 (Acc.%) [63]	71.50±0.12	70.33±0.12	67.35±0.15	63.55±0.29	57.09±0.03
	Ours (Acc. %)	**71.56±0.19**	**70.41±0.06**	**67.74±0.13**	**63.98±0.25**	**57.21±0.09**

Table 2. Unstructured pruning on ImageNet for ResNet-50.

Method	Baseline Acc.(%)	Pruned Acc.(%)	Droped Acc.(%)	Sparsity (%)
GSM [9]	75.72	74.30	1.42	80.00
Sparse VD [44]	76.69	75.28	1.41	80.00
DPF [35]	75.95	74.55	1.40	82.60
WoodFisher [56]	75.98	75.20	0.78	82.70
GReg-1 [63]	76.13	75.45	0.68	82.70
GReg-2 [63]	76.13	75.27	0.86	82.70
Ours	76.13	**75.50**	**0.63**	82.70

Table 3. MobileNetV2 on CIFAR-10/100 with global sparsity ratio over network.

Method	Sparsity (%)	Acc. CIFAR-10	Acc. CIFAR-100
Unpruned	0	93.70	72.80
ChipNet [60]	60	91.83	66.61
Ours		**91.93**	**70.45**
ChipNet [60]	80	90.41	52.96
Ours		**91.49**	**68.90**

Training settings. The importance scores corresponding to each layer's filters are initialized to the L2-norm of the respective pretrained filter weights. For fair comparisons, we adopt the same pruning rate schedules as [63]. Also following [63], for each pruning schedule we denote the model speedup as the pruning-induced reduction in the number of floating point operations (FLOPs) relative to the original model. The whole training procedure consists of two stages: optimal transportation pruning and post-pruning finetuning.

Our main focus is on how to train the soft masks in first stage. An important hyperparameter in this respect is the regularization constant ε. For ResNet-56 and MobileNetV2, we use $\varepsilon = 1$, for all other networks we set $\varepsilon = 0.25$ (see A.7). The model variables and importance scores are trained with standard Stochastic Gradient Descent (SGD) with a momentum of 0.9. For CIFAR datasets, a weight decay of 5×10^{-4} and a batch size of 256 are used. On ImageNet we use a weight decay of 10^{-4} and train on

4 Tesla V100 GPUs with a batch size of 64 per GPU. In all experiments except for MobileNetV2 that follows the training settings of [60], the initial learning rate is set as 0.1 and a cosine learning rate decay is applied. In finetuning stage, we employed the same training settings as [63].

4.1. Effect of training soft masks

How do soft masks converge? We show how our soft masks converge to discrete masks during training in this section. In Fig. 3(a), we trained ResNet-56 on CIFAR-10 with a regularization hyperparameter $\varepsilon = 1$ and a pruning rate of 0.5, and visualize the evolution during training of the soft mask, importance score and L1 norm values for three filters. Early in training, all filters have similar L1 norms and soft masks. Subsequently, however, the algorithm starts to explore different filter compositions: The soft mask of filter1 first converges to one, then fluctuates between zero and one, after which it moves back towards one, its final value. The soft masks corresponding to filter2 and filter3 display similar exploratory behavior. In later training phases, the soft masks converge to hard masks ($\approx$0 or $\approx$1), and the L1 norm of filters with mask values near zero have gradually decreased to zero, a result of the weight decay term in the loss. Since the mask encodes the neurons' connection information, one could interpret the exploration-exploitation process as performing architecture search.

Training overhead with vs without pruning masks. We measure the wall-clock time for a ResNet-50 on a single A100 GPU for 300 iterations using a batch size of 64 with or without training a pruning mask. The training overhead increases by $\sim$0.5%, which is generally negligible.

4.2. Accuracy vs. number of Sinkhorn steps

To improve the computational efficiency of our method, we propose an optimal transportation algorithm that requires only a single Sinkhorn iteration to converge. To verify that a single Sinkhorn iteration indeed suffices, we performed experiments on CIFAR-10 with ResNet-56 using a single GPU. In Fig. 4 we plot the accuracy and training time as a function of the number of innner Sinkhorn steps. A larger number of Sinkhorn steps significantly increases training time without markedly affecting accuracy. Thus a single Sinkhorn iteration suffices.

4.3. Results for ResNet-56/VGG-19 on CIFAR

We first explore the effect of applying different pruning schedules on the accuracy of ResNet-56 on CIFAR-10 and VGG-19 on CIFAR-100. We compare our method to two other pruning methods: L1-norm based one-shot pruning [30] and growing regularization "GReg-1" [63] where the results are taken from the literature [63]. We use a simple uniform pruning ratio scheme, where the pruning ratio is identical across all pruned layers. Following common practice [14], we do not prune the first layer, and for ResNet-56 we only prune the first layer in a residual block (as a result of the restriction imposed by the residual addition) [63, 30]. We explore a broad range of pruning ratios, covering Speedups between $2\times$ and $44\times$. For fair comparisons, the finetuning scheme (*e.g.*, learning rate schedule, number of epochs, etc.) is identical across all methods.

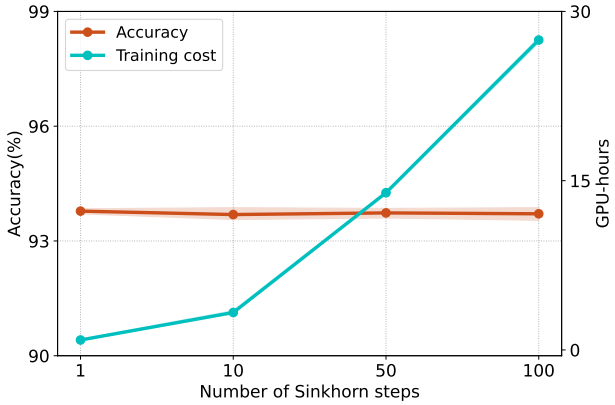


Figure 4. Training cost as a function of the number of inner Sinkhorn steps on CIFAR-10 with ResNet-56 using a single GPU. Each experiment is repeated 3 times with random initialization. The training cost for a Sinkhorn step is measured in GPU-hours. Increasing the number of Sinkhorn steps, significantly increases the training cost, but hardly affects the accuracy. A single Sinkhorn step is sufficient for our method.

The results are presented in Table 1. Note that all pruning methods result in exactly the same pruned network size. Therefore different observed accuracies between methods result from *how* the filters are pruned. Our proposed method first performs exploration followed by exploitation. In doing so, our method searches for best architecture while gradually removing the effect of pruned filters on the network output. Our method outperforms L1+one-shot in all examined conditions and GReg-1 for all conditions except one.

4.4. Results for ResNet-34/50 on ImageNet

We next evaluate our method with ResNet-34 and ResNet-50 on the large-scale ImageNet dataset. We compare our results to previous pruning methods, including L1 Norm [30], TaylorFO [45], IncReg [62], SFP [19], HRank [34], Factorized [31], DCP [69], CCP-AC [50], LFPC [18], GReg-1 and GReg-2 [63], SRR [64], MetaPruning [37], AutoPruner [40], PGMPF [3] and Random Pruning [32]. For fair comparisons, following [63], we use the official PyTorch ImageNet training example to assure that implementation details such as data augmentation, weight decay, momentum, etc. match between methods.

We first explore the behavior of our method by training ResNet-50 on ImageNet with the pruning setting cor-

Table 4. Acceleration comparison for structured filter pruning on ImageNet. Speedup reflects the reduction on FLOPs.

Method	Backbone	Baseline Acc (%)	Pruned Acc (%)	Acc Drop (%)	Speed Up
L1 (pruned-B) [30]	ResNet-34	73.23	72.17	1.06	1.32×
Taylor-FO [45]		73.31	72.83	0.48	1.29×
GReg-1 [63]		73.31	73.54	-0.23	1.32×
GReg-2 [63]		73.31	73.61	-0.30	1.32×
Ours		73.31	**74.28**	**-0.97**	1.32×
SFP [19]	ResNet-50	76.15	74.61	1.54	1.72×
HRank [34]		76.15	74.98	1.17	1.78×
Factorized [31]		76.15	74.55	1.60	2.33×
DCP [69]		76.01	74.95	1.06	2.25×
CCP-AC [50]		76.15	75.32	0.83	2.18×
SRR [64]		76.13	75.11	1.02	2.27×
AutoPruner [40]		76.15	74.76	1.39	2.05×
MetaPruning [37]		76.60	75.40	1.20	2.05×
PGMPF [3]		76.01	75.11	0.90	2.20×
Random Prune [32]		76.13	75.13	1.00	2.04×
GReg-1 [63]		76.13	75.16	0.97	2.31×
GReg-2 [63]		76.13	75.36	0.77	2.31×
Ours		76.13	**75.55**	**0.58**	2.31×
LFPC [18]	ResNet-50	76.15	74.46	1.69	2.55×
GReg-1 [63]		76.13	74.85	1.28	2.56×
GReg-2 [63]		76.13	74.93	1.20	2.56×
Ours		76.13	**75.24**	**0.89**	2.56×
IncReg [62]	ResNet-50	75.60	71.07	4.53	3.00×
Taylor-FO [45]		76.18	71.69	4.49	3.05×
GReg-1 [63]		76.13	73.75	2.38	3.06×
GReg-2 [63]		76.13	73.90	2.23	3.06×
Ours		76.13	**74.26**	**1.87**	3.06×

responding to 3.06× speedup [63] (see detail in A.7). In the top row of Fig. 3(b), we visualize the soft mask and L1 norm of 256 filters from layer 24 of the trained model. Note how all soft masks have converged to hard masks and that the L1 norms of the filters are highly correlated with their corresponding mask values. In the bottom row of Fig. 3(b), we plot the validation accuracy of ResNet-50 during training. We first use our method to prune the model for 90 epochs and, following previous work [63], finetune the pruned model for another 90 epochs. During the first stage, our method searches for the best architecture and reduces the importance of the pruned filters. After 90 epochs we can safely prune the filters without a large impact on the accuracy, in the star point we test the accuracy for the derived sparse model before finetuning, after which finetuning helps to further improve the accuracy. At the beginning of finetuning we observe a small accuracy drop since we use a larger learning rate to start with.

Table 4 gives an overview of all comparisons. We group the methods with similar speedup together for easy comparison. We observe that our method consistently achieves the best result among all approaches, for various speedup comparisons on different architectures. On ResNet-34, our method improves the baseline (unpruned) model by 0.97% accuracy. Previous work [22] explained how pruning can improve generalization performance to improve accuracy for unseen data drawn from the same distribution. Such improved accuracy after pruning has been observed in earlier work [21], but has been more apparent for smaller datasets like CIFAR. In agreement with [63] we also observe improvements over the baseline accuracy on the much more challenging ImageNet benchmark. With larger speedups, the advantage of our method becomes more obvious. For example, our method outperforms Taylor-FO [45] by 1.05% top-1 accuracy at the $2.31\times$ setting, while at $3.06\times$, ours is better by 2.57% top-1 accuracy. Previous work has made considerable progress defining importance scores based on the networks weights [30] [18]. In contrast to those studies our method learns the importance scores used to prune the network in an end-to-end fashion. Our results indicate this can improve the accuracy of pruned networks.

Unstructured pruning on ImageNet. Thus far we focused on structured filter pruning, but previous studies have also explored unstructured pruning. In Table 2, we show that our method can also be applied to unstructured pruning use cases. Our method compares favorably to other advanced unstructured pruning methods for ResNet-50 on ImageNet.

Global sparsity for MobileNetV2. Thus far we have focused on normal convolutions, but networks based on depthwise convolutions may be harder to prune. In Table 3, we compare to ChipNet [60] using the global sparsity ratio for the lightweight MobileNetV2 architecture on CIFAR-10/100. Learning with a global sparsity ratio may distribute different sparsity ratios across layers, while respecting the preset sparsity across the network. For both datasets, we rerun the experiments of ChipNet using its released hyperparameters and training settings. Under different sparsity ratios, our method consistently outperforms ChipNet.

We also further extend our method to general budgeted pruning with *e.g.* a certain FLOPs budget (see detail in A.9).

5. Conclusion

In this paper we propose a differentiable sparse learning algorithm based on entropy regularized optimal transportation that achieves exact control over the sparsity ratio. We formulate a differentiable bi-level learning objective to jointly optimize the soft masks and the network parameters using standard SGD methods combined with Sinkhorn optimization. Optimizing this bi-level problem is computational intensive, but we show that by using a Bregman divergence we can improve the efficiency, *i.e.* requiring only a single Sinkhorn iteration, while still guaranteeing algorithmic convergence. We demonstrated the effectiveness of our method on different datasets for different pruning rates.

Acknowledgements. This work is funded by EU's Horizon Europe research and innovation program under grant agreement No. 101070374.

References

[1] SN Afriat. Theory of maxima and the method of lagrange. *SIAM Journal on Applied Mathematics*, 20(3):343–357, 1971.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] Linhang Cai, Zhulin An, Chuanguang Yang, Yangchun Yan, and Yongjun Xu. Prior gradient mask guided pruning-aware fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 1, 2022.

[4] Yair Censor and Stavros Andrea Zenios. Proximal minimization algorithm with d-functions. *Journal of Optimization Theory and Applications*, 73(3):451–464, 1992.

[5] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

[6] Daiki Chijiwa, Shin'ya Yamaguchi, Yasutoshi Ida, Kenji Umakoshi, and Tomohiro Inoue. Pruning randomly initialized neural networks with iterative randomization. *Advances in Neural Information Processing Systems*, 34:4503–4513, 2021.

[7] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

[8] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[9] Xiaohan Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu, et al. Global sparse momentum sgd for pruning very deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Konstantin Ditschuneit and Johannes S Otterbach. Autocompressing subset pruning for semantic image segmentation. In *Pattern Recognition: 44th DAGM German Conference*, pages 20–35. Springer, 2022.

[11] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, Florian Bernard, and Daniel Cremers. A unified framework for implicit sinkhorn differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 509–518, 2022.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.

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

[15] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908, 2020.

[16] Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*, pages 1608–1617. PMLR, 2018.

[17] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1539–1547, 2020.

[18] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2009–2018, 2020.

[19] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *IJCAI*, 2018.

[20] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019.

[21] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[22] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.

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

[24] Ryan Humble, Maying Shen, Jorge Albericio-Latorre, Eric Darve, and Jose M Alvarez. Soft masking for cost-constrained channel pruning. *ECCV*, 2022.

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

[26] Jeffrey J Kosowsky and Alan L Yuille. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural networks*, 7(3):477–490, 1994.

[27] Ronny Krashinsky, Olivier Giroux, Stephen Jones, Nick Stam, and Sridhar Ramaswamy. Nvidia ampere architecture in-depth. *NVIDIA blog: https://devblogs. nvidia. com/nvidia-ampere-architecture-in-depth*, 2020.

[28] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. Structured pruning of neural networks with budget-aware regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9108–9116, 2019.

[29] Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *ECCV 2020*, pages 639–654. Springer, 2020.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[31] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019.

[32] 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*, pages 191–201, 2022.

[33] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

[34] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020.

[35] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020.

[36] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

[37] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305, 2019.

[38] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *ICLR*, 2019.

[39] Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pages 2952–2960. PMLR, 2016.

[40] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 107:107461, 2020.

[41] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1458–1467, 2020.

[42] Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning filter in filter. *Advances in Neural Information Processing Systems*, 33:17629–17640, 2020.

[43] 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*, 2022.

[44] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

[45] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[46] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In *ECCV 2020*, pages 592–607. Springer, 2020.

[47] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

[48] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[50] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122. PMLR, 2019.

[51] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[52] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. *Advances in Neural Information Processing Systems*, 33:11380–11390, 2020.

[53] Bernhard Schmitzer. Stabilized sparse scaling algorithms for entropy regularized transport problems. *arXiv preprint arXiv:1610.06519*, 2016.

[54] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose Alvarez. Structural pruning via latency-saliency knapsack. In *Advances in Neural Information Processing Systems*, 2022.

[55] Yucong Shen, Li Shen, Hao-Zhi Huang, Xuan Wang, and Wei Liu. Cpot: Channel pruning via optimal transport. *arXiv preprint arXiv:2005.10451*, 2020.

[56] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

[57] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[58] Kenji Suzuki, Isao Horiba, and Noboru Sugie. A simple neural network pruning algorithm with application to filter synthesis. *Neural processing letters*, 13(1):43–53, 2001.

[59] Kai Sheng Tai, Taipeng Tian, and Ser Nam Lim. Spartan: Differentiable sparsity via regularized transportation. *Advances in Neural Information Processing Systems*, 35:4189–4202, 2022.

[60] Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations*, 2021.

[61] Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. *ICLR*, 2022.

[62] Huan Wang, Xinyi Hu, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convolutional neural networks via incremental regularization. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):775–788, 2019.

[63] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *ICLR*, 2021.

[64] Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14913–14922, 2021.

[65] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. *Advances in Neural Information Processing Systems*, 33:20520–20531, 2020.

[66] Yujia Xie, Xiangfeng Wang, Ruijia Wang, and Hongyuan Zha. A fast proximal point method for computing exact wasserstein distance. In *Uncertainty in artificial intelligence*, pages 433–453. PMLR, 2020.

[67] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

[68] Shuai Zhang, Meng Wang, Sijia Liu, Pin-Yu Chen, and Jinjun Xiong. Why lottery ticket wins? a theoretical perspective of sample complexity on sparse neural networks. *Advances in Neural Information Processing Systems*, 34:2707–2720, 2021.

[69] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

A. Supplementary Material

A.1. Optimal coupling to regularized problem

Here we prove the optimal coupling to the regularized problem in Eq. (7) is:

$$\mathbf{P}_{\varepsilon} = e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}. \tag{16}$$

Proof. Introducing two dual variables $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^2$ for marginal constraints $\mathbf{P}\mathbb{1}_2 = \mathbf{a}$ and $\mathbf{P}^{\mathsf{T}}\mathbb{1}_n = \mathbf{b}$, given the discrete entropy of a coupling matrix $\mathcal{H}(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1)$, the Lagrangian of Eq. (7) is optimized as follows:

$$\xi(\mathbf{P}, \mathbf{f}, \mathbf{g}) = \langle \mathbf{C}, \mathbf{P} \rangle - \varepsilon \mathcal{H}(\mathbf{P}) - \langle \mathbf{f}, \mathbf{P}\mathbb{1}_2 - \mathbf{a} \rangle - \big\langle \mathbf{g}, \mathbf{P}^{\mathsf{T}}\mathbb{1}_n - \mathbf{b} \big\rangle. \tag{17}$$

First order conditions then yield:

$$\frac{\partial \xi(\mathbf{P}, \mathbf{f}, \mathbf{g})}{\partial \mathbf{P}_{ij}} = \mathbf{C} + \varepsilon \log(\mathbf{P}_{ij}) - \mathbf{f}_i - \mathbf{g}_j = 0, \tag{18}$$

which result for an optimal coupling to the regularized problem, in the matrix expression shown in Eq. (16).

A.2. Dual problem computation

Here we prove that minimizing the regularized optimal transport distance in Eq. (7) is equivalent to maximizing its dual problem:

$$\max_{\mathbf{f} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^2} \langle \mathbf{f}, \mathbf{a} \rangle + \langle \mathbf{g}, \mathbf{b} \rangle - \varepsilon \left\langle e^{\mathbf{f}/\varepsilon}, e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \cdot e^{\mathbf{g}/\varepsilon} \right\rangle \tag{19}$$

Proof. We start from the result in Eq. (16), and substitute it in the Lagrangian $\xi(\mathbf{P}, \mathbf{f}, \mathbf{g})$ of Eq. (17), where the optimal $\mathbf{P}$ is a function of $\mathbf{f}$ and $\mathbf{g}$, we obtain that the Lagrange dual function equals:

$$\begin{aligned} \mathbf{f}, \mathbf{g} \mapsto & \left\langle e^{\mathbf{f}/\varepsilon}, \big(e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot \mathbf{C}(\mathbf{s})\big) e^{\mathbf{g}/\varepsilon} \right\rangle \\ & - \varepsilon \mathcal{H}\big(e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}\big) \end{aligned} \tag{20}$$

The negative entropy of $\mathbf{P}$ scaled by ε, namely $\varepsilon \langle \mathbf{P}, \log \mathbf{P} - \mathbb{1}_{n\times 2} \rangle$, can be stated explicitly as a function of $\mathbf{f}, \mathbf{g}, \mathbf{C}$:

$$\begin{aligned} & \varepsilon \langle \mathbf{P}, \log \mathbf{P} - \mathbb{1}_{n\times 2} \rangle \\ = & \left\langle e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}, \mathbf{f}\mathbb{1}_2^{\mathsf{T}} + \mathbb{1}_n \mathbf{g}^{\mathsf{T}} - \mathbf{C}(\mathbf{s}) - \varepsilon \mathbb{1}_{n\times 2} \right\rangle \\ = & - \left\langle e^{\mathbf{f}/\varepsilon}, \big(e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot \mathbf{C}(\mathbf{s})\big) e^{\mathbf{g}/\varepsilon} \right\rangle + \langle \mathbf{f}, \mathbf{a} \rangle + \langle \mathbf{g}, \mathbf{b} \rangle - \\ & \varepsilon \left\langle e^{\mathbf{f}/\varepsilon}, e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \cdot e^{\mathbf{g}/\varepsilon} \right\rangle \end{aligned}$$

therefore, the first term in Eq. (20) cancels out with the first term in the entropy above. The remaining terms are those appearing in Eq. (19).

A.3. Expensive to train with Sinkhorn's algorithm

The bi-level optimization problem in Eq. (10) and Eq. (11) is expensive to train with Sinkhorn's algorithm: During forward pass, per mini-batch the inner optimization in Eq. (11) needs to perform Sinkhorn's iterative algorithm for hundreds of iterations to converge, which is computationally inefficient. During the back-propagation pass, the gradient of $\mathbf{P}^*_{\varepsilon}(\mathbf{s})$ with respect to the importance scores $\mathbf{s}$ is computed by differentiating [48] through all Sinkhorn iterations, which is expensive. We display the forward and backward of Sinkhorn algorithm in Fig. 5.

A.4. Bregman divergence based optimization

In this paper we use similar Bregman Divergence D_h as [66] based on entropy function $\mathcal{H}(\mathbf{x}) = -\sum_i x_i \big(\log(x_i) - 1\big)$ as:

$$D_{\mathcal{H}}(\mathbf{x}, \mathbf{y}) = -\sum_i x_i \log \frac{x_i}{y_i} + \sum_i x_i. \tag{21}$$

Based on the defined Bregman Divergence, a single proximal point iteration for problem (4) can be written as:

$$\begin{aligned} \mathbf{P}^{(\ell+1)} &= \min_{\mathbf{P} \in \mathcal{U}(\mathbf{a}, \mathbf{b})} \langle \mathbf{C}, \mathbf{P} \rangle - \varepsilon D_{\mathcal{H}}(\mathbf{P}, \mathbf{P}^{(\ell)}) \\ &= \min_{\mathbf{P} \in \mathcal{U}(\mathbf{a}, \mathbf{b})} \Big\langle \mathbf{C} - \varepsilon \log \mathbf{P}^{(\ell)}, \mathbf{P} \Big\rangle - \varepsilon \mathcal{H}(\mathbf{P}). \end{aligned} \tag{22}$$

Denote $\mathbf{C}' = \mathbf{C} - \varepsilon \log \mathbf{P}^{(\ell)}$. Note that for optimization problem 22, $\mathbf{P}^{(\ell)}$ is a fixed value that is not relevant to optimization variable $\mathbf{P}$. Comparing to Eq. (7), the problem in Eq. (22) can be solved by Sinkhorn iteration by replacing Gibbs kernel $\mathbf{K}$ by $\mathbf{K}' = e^{-\frac{\mathbf{C}'}{\varepsilon}} = e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}$. With this reorganization, we can solve it with Sinkhorn algorithm. [66] have shown both theoretically and empirically that a *single Sinkhorn inner iteration* is sufficient to converge under a large range of fixed ε, we therefore compute $\mathbf{P}^{(\ell+1)}$ as:

$$\mathbf{P}^{(\ell+1)} = e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \big(e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}\big) \odot e^{\mathbf{g}^{(\ell+1)}/\varepsilon}, \tag{23}$$

which is the expression of Eq. (13).

A.5. Proximal point iteration as iterative Sinkhorn

We refer to the proof in Chapter 4.2 of the book [51]. The general setting for proximal point iteration is to define Gibbs kernel as $\mathbf{K} = e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}$, the proximal point iterations thus have the form:

$$\begin{aligned} \mathbf{P}^{(\ell+1)}_{\varepsilon}(\mathbf{s}) =& e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \big(e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}\big) \odot e^{\mathbf{g}^{(\ell+1)}/\varepsilon} \\ =& \big(e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \cdot\cdot \odot e^{\mathbf{f}^{(1)}/\varepsilon}\big) \odot \big(e^{-\frac{(\ell+1)\mathbf{C}}{\varepsilon}} \\ & \odot \mathbf{P}^{(\ell)}\big) \odot \big(e^{\mathbf{g}^{(\ell+1)}/\varepsilon} \odot \cdot\cdot \odot e^{\mathbf{g}^{(1)}/\varepsilon}\big). \end{aligned} \tag{24}$$

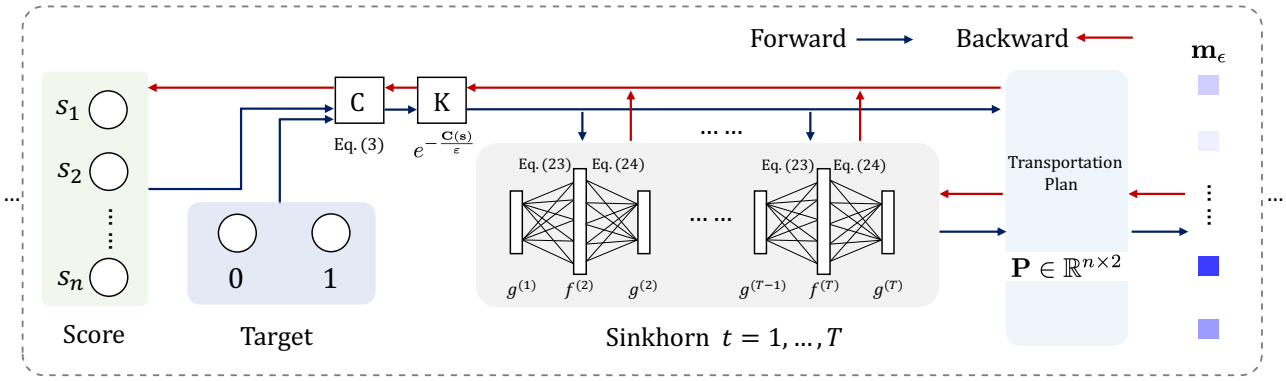


Figure 5. The display of training with Sinkhorn's algorithm. We show the forward and backward pass over one single layer: given importance score $\mathbf{s}$, the cost matrix is computed by Eq. (3), we thus get $\mathbf{K}$; in forward pass we input $\mathbf{K}$ to iteratively compute $\mathbf{f}^{(t+1)}$ with Eq. (27) and $\mathbf{g}^{(t+1)}$ with Eq. (28) using Sinkhorn algorithm for T iterations; during back-propagation pass, the gradient of soft mask with respect to the importance scores is computed by differentiating through all Sinkhorn iterations. A large T makes it expensive to train.

Table 5. Training setting: For the SGD optimizer, in the parentheses are the momentum and weight decay. For ImageNet, batch size is 64 per GPU. We learn the soft mask using cosine learning rate schedule.

Settings	CIFAR	ImageNet
Optimizer	SGD (0.9, 5e-4)	SGD (0.9, 1e-4)
LR schedule (soft mask)	Cosine LR schedule (0.1)	
LR schedule (finetune)	Multi-step (0:1e-2, 60:1e-3, 90:1e-4)	Multi-step (0:1e-2, 30:1e-3, 60:1e-4, 75:1e-5)
Training Epoch	120 + 120	90 + 90
Batch size	256	256

The proximal point iteration iteratively applies Sinkhorn's algorithm with a $e^{-\frac{\mathbf{C}}{\varepsilon/\ell}}$ kernel, *i.e.* with a decaying regularization parameter ε/ℓ, therefore it can perform an automatic decaying schedule on the regularization as $\ell \to \infty$ to gradually approach the optimal discrete plan.

A.6. Sinkhorn iterations on dual problem

A simple approach to solving the unconstrained maximization problem in Eq. (19) is to use an exact block coordinate ascent strategy, namely to update alternatively $\mathbf{f}$ and $\mathbf{g}$ to cancel the respective gradients in these variables of the objective of (19). Indeed, one can notice after a few elementary computations that, writing $\mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s})$ for the objective of (19), we have the gradients, w.r.t., $\mathbf{f}$ and $\mathbf{g}$ as follows:

$$\nabla_{\mathbf{f}} \mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s}) = \mathbf{a} - e^{\mathbf{f}/\varepsilon} \odot (\mathbf{K} e^{\mathbf{g}/\varepsilon}), \tag{25}$$

$$\nabla_{\mathbf{g}} \mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s}) = \mathbf{b} - e^{\mathbf{g}/\varepsilon} \odot (\mathbf{K} e^{\mathbf{f}/\varepsilon}), \tag{26}$$

where $\mathbf{K} = e^{-\mathbf{C}(\mathbf{s})/\varepsilon}$ is defined as the Gibbs kernel in Sinkhorn's algorithm. Block coordinate ascent can therefore be implemented in a closed form by applying successively the following updates, starting from any arbitrary $\mathbf{g}^{(1)}$, for $t \geq 1$:

$$\mathbf{f}^{(t+1)} = \varepsilon \log \mathbf{a} - \varepsilon \log \left(\mathbf{K} e^{\mathbf{g}^{(t)}/\varepsilon}\right); \tag{27}$$

$$\mathbf{g}^{(t+1)} = \varepsilon \log \mathbf{b} - \varepsilon \log \left(\mathbf{K}^{\mathsf{T}} e^{\mathbf{f}^{(t+1)}/\varepsilon}\right). \tag{28}$$

A.7. Experimental setting details

How is the ε selected. The ε controls a trade-off between exploration and exploitation. A large ε leads to a greater exploration by a softer mask. We verify that soft masks converge to hard masks by training for few epochs and measuring the average of the squared differences between them for different ε candidates. As is common to hyperparameter tuning, this requires some effort. Therefore for similar architectures we used the same ε, *e.g.*, $\varepsilon = 1$ for lightweight ResNet-56 and MobileNetV2.

Training setting. In Table 5 we summarize the detailed training settings of this work. For easier comparison, we use same training settings as [63] in finetuning. In our soft mask learning phase, we use a cosine learning rate schedule for updating the importance scores and network weights with SGD.

Pruning ratio. We use same pruning ratio as in [63] for fair comparison. In Table 6 we give the specific pruning ratio used for our experiments in the paper. We here briefly explain how we set the pruning ratio. We use two different ar-

Table 6. Training setting: For the SGD optimizer, in the parentheses are the momentum and weight decay. For ImageNet, batch size is 64 per GPU. We learn the soft mask using cosine learning rate schedule.

Datasets	Backbone	Speedup	Pruning ratio
CIFAR-10	ResNet-56	−−	[0, p, p, p], $p \in \{0.5, 0.7, 0.9, 0.925, 0.95\}$
CIFAR-100	VGG-19	−−	[0:0, 1-15:p], $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$
ImageNet	ResNet-34	1.32×	[0, 0.50, 0.60, 0.40, 0]
ImageNet	ResNet-50	2.31×	[0, 0.60, 0.60, 0.60, 0.21]
ImageNet	ResNet-50	2.56×	[0, 0.74, 0.74, 0.60, 0.21]
ImageNet	ResNet-50	3.06×	[0, 0.68, 0.68, 0.68, 0.50]

chitectures: single-branch (*i.e*. VGG-19) and multi-branch (*i.e*. ResNet). *(i)* For VGG19, we use the following pruning ratio setting. As an example, "[0:0, 1-9:0.3, 10-15:0.5]" means "for the first layer (index starting from 0), the pruning ratio is 0; for layer 1 to 9, the pruning ratio is 0.3; for layer 10 to 15, the pruning ratio is 0.5". *(ii)* For a ResNet, if it has N stages, we will use a list of N floats to represent its pruning ratios for the N stages. For example, ResNet-56 has 4 stages in conv layers, then "[0, 0.7, 0.7, 0.7]" means "for the first stage (the first conv layer), the pruning ratio is set as 0; the other three stages have pruning ratio of 0.7". Besides, since we do not prune the last conv layer in a residual block, which means for a two-layer residual block (for ResNet-56), we only prune the first layer; for a three-layer bottleneck block (for ResNet-34 and ResNet-50), we only prune the first and second layers.

Unstructured pruning. For unstructured pruning, each individual weight needs to couple an importance score which may cause higher memory cost, therefore we follow previous works that use the magnitude of the weight parameter as an importance score.

A.8. Training recipe variants

Our primary goal was to compare pruning methods as fairly as possible. The reported results for ResNet-50 used the *baseline training recipe* from CReg for fair comparisons. Here, we performed additional analyses, pruning ResNet-50 on ImageNet with Label Smoothing, TrivialAugment, Random Erasing, Mixup, and Cutmix to form a *stronger training recipe*. We did not apply other effective techniques such as Long Training (*e.g*. 600 epochs), LR optimizations, EMA, Weight Decay tuning, and Inference Resize tuning, which may further improve by ∼3% accuracy as shown in PyTorch. We trained on a large batch size of 4,096 on 8 NVIDIA A100 GPUs, scaling up the learning rate by 5×. As expected, Table 7 shows increased pruning accuracy, showing that our method can be augmented with different training recipes.

A.9. Extension to FLOPs and latency budgets

The FLOPs or latency budget pruning can be viewed as a learnable differentiable sparsity allocation problem. We formulate it in the following.

Table 7. Training recipes for ResNet50 under structured pruning on ImageNet-1K. Our method is flexible for different training recipes.

Recipe	Speed Up	Acc.
Baseline	2.31×	75.54
Stonger		77.16

Table 8. FLOPs budgets for WideResNet-26 on CIFAR-100

Method	Budget (%)	Acc.
Unpruned	100	80.21
ChipNet [60]	40	76.88
Ours		78.63
ChipNet [60]	20	77.15
Ours		78.17

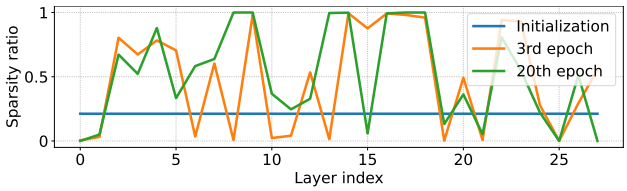


Figure 6. Learned sparsity ratios over layers under FLOPs budget.

Formulation. Given budget $B_{\mathcal{F}}$, the network weights as $\mathbf{w}$, the optimization problem of kept ratios $\mathcal{A} = \{\alpha^{(l)}\}_{l=1,\dots,L}$ (*i.e*., 1−sparsity ratio) of L layers can be written as:

$$\begin{aligned} &\arg\min_{\mathbf{m},\mathbf{w},\mathcal{A}} \ \mathcal{L}_{\text{train}}\big(f(\mathbf{x};\mathbf{w},\mathbf{m},\mathcal{A})\big), \\ &\text{s.t.} \ \ \frac{1}{n}\sum\nolimits_{i=1}^{n} \mathbf{m}_i^{(l)} = \alpha^{(l)}, \ \ \mathbf{m}^{(l)} \in \{0,1\}^n, \\ &\qquad \mathcal{F}(\mathcal{A}) \le B_{\mathcal{F}}, \ \ 0 \le \mathcal{A} \le 1. \end{aligned} \tag{29}$$

where $\mathcal{L}_{\text{train}}$ is training loss. $\mathcal{F}(\mathcal{A})$ is the consumed resource corresponding to the kept ratios. If $\mathcal{A}$ is predefined layer-wise or global kept ratio, the formulation degrades to our original formulation in Eq. (1). For FLOPs or latency budgets, we learn the kept ratios. Note $\mathcal{F}(\mathcal{A})$ can be denoted as a function of kept ratio, see [60] and [54]. By setting $\mathcal{A} = \text{Sigmoid}(\Theta)$ where Θ is a group of learnable parameters where $0 \le \mathcal{A} \le 1$ is satisfied naturally, we optimize Θ to learn kept ratio $\mathcal{A}$ by minimizng $\mathcal{L}_{\text{train}}$ and a budgets

penalty loss $||\mathcal{F}(\mathcal{A}) - B_{\mathcal{F}}||^2$. Our optimal transport method dynamically aligns to learned ratios $\mathcal{A}$.

We add the experimental comparisons in Table 8 with FLOPs budget for WideResNet-26 for CIFAR-100.

FLOPs budget. We use the same way as [60] to calculate the FLOPs budget that assumes a sliding window is used to achieve convolution and the nonlinear computational overhead is ignored. We define FLOPs budget as:

$$\mathcal{F}(\mathcal{A}) = \frac{\sum_{j=1}^{\mathcal{N}}(K_j \cdot n_{j-1} \cdot \alpha^{(j-1)} + 1) \cdot n_j \cdot \alpha^{(j)} \cdot A_j}{\sum_{j=1}^{\mathcal{N}}(K_j \cdot n_{j-1} + 1) \cdot n_j \cdot A_j}, \tag{30}$$

where $\mathcal{N}$ denotes the number of convolutional layers in the network, K_j denotes area of the kernel, and A_j and n_j denote area of the feature maps and the channel count, respectively, in the j^{th} layer.

We note that the FLOPs budget is formulated as a function of kept ratios $\alpha^{(j)}$. Also the the latency cost of each layer can be approximated by the kept ratios of previous layer and current layer as shown in [24].