BALANCING TRAINING TIME VS. PERFORMANCE WITH BAYESIAN EARLY PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning is an approach to alleviate overparameterization of *deep neural network* (DNN) by zeroing out or pruning DNN elements with little to no efficacy at a given task. In contrast to related works that do pruning before or after training, this paper presents a novel method to perform *early* pruning of DNN elements (e.g., neurons or convolutional filters) *during the training process* while preserving performance upon convergence. To achieve this, we model the future efficacy of DNN elements in a Bayesian manner conditioned upon efficacy data collected during the training and prune DNN elements which are predicted to have low efficacy after training completion. Empirical evaluations show that the proposed Bayesian early pruning improves the computational efficiency of DNN training. Using our approach we are able to achieve a 48.6% faster training time for ResNet-50 on ImageNet to achieve a validation accuracy of 72.5%.

1 INTRODUCTION

Deep neural networks (DNNs) are known to be overparameterized (Allen-Zhu et al., 2019) as they usually have more learnable parameters than needed for a given learning task. So, a trained DNN contains many ineffectual parameters that can be safely pruned or zeroed out with little/no effect on its predictive accuracy. *Pruning* (LeCun et al., 1989) is an approach to alleviating overparameterization of a DNN by identifying and removing its ineffectual parameters while preserving its predictive accuracy on the validation/test dataset. Pruning is typically applied to the DNN *after training* to speed up *testing-time evaluation*. For standard image classification tasks with MNIST, CIFAR-10, and ImageNet datasets, it can reduce the number of learnable parameters by up to 50% or more while maintaining test accuracy (Han et al., 2015; Li et al., 2017; Molchanov et al., 2017).

In particular, the overparameterization of a DNN also leads to considerable training time being wasted on those DNN elements (e.g., connection weights, neurons, or convolutional filters) which are eventually ineffectual after training and can thus be safely pruned. Our work in this paper considers *early pruning* of such DNN elements by identifying and removing them *throughout the training process* instead of after training.[1] As a result, this can significantly reduce the time incurred by the training process without compromising the final test accuracy (upon convergence) much.

Recent work (Section 5) in foresight pruning (Lee et al., 2019; Wang et al., 2020) show that pruning heuristics applied at initialization work well to prune connection weights without significantly degrading performance. In contrast to these work, we prune throughout the training procedure, which improves performance after convergence of DNNs, albeit with somewhat longer training times.

In this work, we pose early pruning as a constrained optimization problem (Section 3.1). A key challenge in the optimization is accurately modeling the future efficacy of DNN elements. We achieve this through the use of *multi-output Gaussian process* which models the *belief* of future efficacy conditioned upon efficacy measurements collected during training (Section 3.2). Although the posed optimization problem is NP-hard, we derive an efficient *Bayesian early pruning* (BEP) approximation algorithm, which appropriately balances the inherent training time vs. performance tradeoff in pruning prior to convergence (Section 3.3). Our algorithm relies on a measure of network element efficacy, termed saliency (LeCun et al., 1989). The development of saliency functions is an active area of research with no clear optimal choice. To accomodate this, our algorithm is agnostic, and therefore

[1] In contrast, foresight pruning (Wang et al., 2020) removes DNN elements *prior* to the training process.

flexible, to changes in saliency function. We use BEP to prune neurons and convolutional filters to achieve practical speedup during training (Section 4).[2] Our approach also compares favorably to state-of-the-art works such as *SNIP* (Lee et al., 2019), *GraSP* (Wang et al., 2020), and *momentum based dynamic sparse reparameterization* (Dettmers & Zettlemoyer, 2019).

2 Background and Notations

Pruning. Consider a dataset of D training examples $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_D\}, \mathcal{Y} = \{y_1, \ldots, y_D\}$ and a neural network $\mathcal{N}_{\boldsymbol{v}_t}$ parameterized by a vector of M *pruneable* network elements (e.g. weight parameters, neurons, or convolutional filters) $\boldsymbol{v}_t \triangleq [v_t^a]_{a=1,\ldots,M}$, where $\boldsymbol{v}_t$ represent the network elements after t iterations of stochastic gradient descent (SGD) for $t = 1, \ldots, T$. Let $\mathcal{L}(\mathcal{X}, \mathcal{Y}; \mathcal{N}_{\boldsymbol{v}_t})$ be the loss function for the neural network $\mathcal{N}_{\boldsymbol{v}_t}$. Pruning aims at refining the network elements $\boldsymbol{v}_t$ given some sparsity budget B and preserving the accuracy of the neural network after convergence (i.e., $\mathcal{N}_{\boldsymbol{v}_T}$), which can be stated as a constrained optimization problem (Molchanov et al., 2017):

$$\min_{\boldsymbol{m} \in \{0,1\}^M} |\mathcal{L}(\mathcal{X}, \mathcal{Y}; \mathcal{N}_{\boldsymbol{m} \odot \boldsymbol{v}_T}) - \mathcal{L}(\mathcal{X}, \mathcal{Y}; \mathcal{N}_{\boldsymbol{v}_T})| \quad \text{s.t.} \quad ||\boldsymbol{m}||_0 \leq B \tag{1}$$

where $\odot$ is the Hadamard product and $\boldsymbol{m}$ is a pruning mask. Note that we abuse the Hadamard product for notation simplicity: For $a = 1, .., M$, $m^a \times v_T^a$ corresponds to pruning v_T^a if $m^a = 0$, and keeping v_T^a otherwise. Pruning a network element refers to zeroing the network element or the weight parameters which compute the network element. Any weight parameters which reference the output of the pruned network element are also zeroed since the element outputs a constant 0.

The above optimization problem is difficult due to the NP-hardness of combinatorial optimization. This leads to the approach of using saliency function s which measures efficacy of network elements at minimizing the loss function. A network element with small saliency can be pruned since it's not *salient* in minimizing the loss function. Consequently, pruning can be done by maximizing the saliency of the network elements given the sparsity budget B:

$$\max_{\boldsymbol{m} \in \{0,1\}^M} \sum_{a=1}^{M} m^a s(a; \mathcal{X}, \mathcal{Y}, \mathcal{N}_{\boldsymbol{v}_T}, \mathcal{L}) \quad \text{s.t.} \quad ||\boldsymbol{m}||_0 \leq B \tag{2}$$

where $s(a; \mathcal{X}, \mathcal{Y}, \mathcal{N}_{\boldsymbol{v}_T}, \mathcal{L})$ measures the saliency of v_T^a at minimizing $\mathcal{L}$ after convergence through T iterations of SGD. Typically s is selected from a family of saliency functions such as L_1 norm (Li et al., 2017), mean activation (Polyak & Wolf, 2015), among others. The above optimization problem can be efficienctly solved by selecting the B most *salient* network elements in $\boldsymbol{v}_T$.

Saliency function. Next, we introduce a first-order Taylor-series saliency function proposed by Molchanov et al. (2017) which is used in this work. Our design (Section 3) will remain flexible to allow usage of arbitrary saliency functions in a plug-n-play basis. We partition a DNN of L layers, where each layer ℓ contains C_ℓ convolutional filters, into a sequence of convolutional filters $[z_{\ell,c}]_{\ell=1,\ldots,L}^{c=1,\ldots,C_\ell}$. Each filter $z_{\ell,c} : \mathbb{R}^{C_{\ell-1} \times W_{\ell-1} \times H_{\ell-1}} \to \mathbb{R}^{W_\ell \times H_\ell}$ can be considered as one *network element* in $\boldsymbol{v}_T$ and $z_{\ell,c}(\mathbf{P}_{\ell-1}) \triangleq \mathcal{R}(\mathbf{W}_{\ell,c} * \mathbf{P}_{\ell-1} + b_{\ell,c})$ where $\mathbf{W}_{\ell,c} \in \mathbb{R}^{C_\ell \times O_\ell \times O'_\ell}$, $b_{\ell,c}$ are kernel weights and bias.with receptive field $O_\ell \times O'_\ell$, '$*$' represents the convolution operation, $\mathcal{R}$ is the activation function, $\mathbf{P}_{\ell-1}$ represents the output of $\boldsymbol{z}_{\ell-1} \triangleq [z_{\ell-1,c'}]_{c'=1,\ldots,C_{\ell-1}}$ with $\mathbf{P}_0$ corresponding to an input $\mathbf{x}_d \in \mathcal{X}$, and W_ℓ, H_ℓ are width and height dimensions of layer ℓ for $\ell = 1, \ldots, L$. Let $\mathcal{N}_{\boldsymbol{z}_\ell : \boldsymbol{z}_{\ell'}} \triangleq \boldsymbol{z}_{\ell'} \circ, \ldots, \circ \boldsymbol{z}_\ell$ denote a *partial* neural network of layers $[\ell, \ldots, \ell']_{1 \leq \ell \leq \ell' \leq L}$. The Taylor-series saliency function on the convolutional filter $z_{\ell,c}$ denoted as $s([\ell, c])$ is defined[3]:

$$s([\ell, c]) \triangleq \frac{1}{D} \sum_{d=1}^{D} \left| \frac{1}{W_\ell \times H_\ell} \sum_{j=1}^{W_\ell \times H_\ell} \frac{\partial \mathcal{L}(\mathbf{P}_\ell^{(\mathbf{x}_d)}, y_d; \mathcal{N}_{\boldsymbol{z}_{\ell+1} : \boldsymbol{z}_L})}{\partial P_{\ell,c,j}^{(\mathbf{x}_d)}} P_{\ell,c,j}^{(\mathbf{x}_d)} \right|. \tag{3}$$

where $\mathbf{P}_\ell^{(\mathbf{x}_d)}$ is the output of the partial neural network $\mathcal{N}_{\boldsymbol{z}_1 : \boldsymbol{z}_\ell}$ with $\mathbf{x}_d$ as the input and $[P_{\ell,c,j}^{\mathbf{x}_d}]_{j=1,\ldots,W_\ell \times H_\ell}$ interprets the output of the c-th filter in vectorized form. This function uses the

[2] Popular deep learning libaries do not accelerate sparse matrix operations over dense matrix operations. Thus, pruning network connections cannot be easily capitalized upon with performance improvements. It is also unclear whether moderately sparse matrix operations (i.e., operations on matrices generated by connection pruning) can be significantly accelerated on massively parallel architectures such as GPUs (see Yang et al. (2018) Fig. 7). See Section 5 in Buluç & Gilbert (2008) for challenges in parallel sparse matrix multiplication.

[3] For brevity, we omit parameters $\mathcal{X}, \mathcal{Y}, \mathcal{N}_{\boldsymbol{z}_1 : \boldsymbol{z}_L}, \mathcal{L}$.

first-order Taylor-series approximation of $\mathcal{L}$ to approximate the change in loss if $z_{\ell,c}$ was changed to a constant 0 function. Using the above saliency definition, pruning filter $z_{\ell,c}$ corresponds to collectively zeroing $\mathbf{W}_{\ell,c}$, $b_{\ell,c}$ as well as weight parameters[4] $[\mathbf{W}_{\ell+1,c',\{:,:,c\}}]_{c'=1,\dots,C_{\ell+1}}$ of $\boldsymbol{z}_{\ell+1}$ which utilize the output of $z_{l,c}$. This definition can be extended to elements (e.g. neurons) which output scalars by setting $W_\ell = H_\ell = 1$.

3 Bayesian Early Pruning

3.1 Problem statement.

As has been mentioned before, existing pruning works based on the saliency function are usually done after the training convergence (i.e., (2)) to speed up the testing-time evaluation, which waste considerable time on training these pruned network elements. To resolve this issue, We extend the pruning problem definition (2) along the temporal dimension, allowing network elements to be pruned during the training process consisting of T iterations of SGD. To keep the proposed algorithm to be general, we will use v_t^a instead of the particular $z_{l,c}$ to represent the network element afterwards.

Let $s_t^a \triangleq s(a; \mathcal{X}, \mathcal{Y}, \mathcal{N}_{\boldsymbol{v}_t}, \mathcal{L})$ be a random variable which denotes the saliency of network element v_t^a after t iterations of SGD, $\boldsymbol{s}_t \triangleq [s_t^a]_{a=1,\dots,M}$ for $t = 1, \dots, T$, and $\boldsymbol{s}_{\tau_1:\tau_2} \triangleq [\boldsymbol{s}_t]_{t=\tau_1,\dots,\tau_2}$ be a vector of saliency of all the network elements between iterations τ_1 and τ_2. Our early pruning algorithm is designed with the goal of maximizing the saliency of the *unpruned* elements after iteration T, yet allowing for pruning at each iteration t given some computational budget $B_{t,c}$ for $t = 1, \dots, T$:

$$\rho_T(\boldsymbol{m}_{T-1}, B_{T,c}, B_s) \triangleq \max_{\boldsymbol{m}_T} \boldsymbol{m}_T \cdot \boldsymbol{s}_T \text{ (4a)}$$
$$\text{s.t.} \quad ||\boldsymbol{m}_T||_0 \leq B_s \text{ (4b)} \quad \boldsymbol{m}_T \dot{\leq} \boldsymbol{m}_{T-1} \text{ (4c)} \quad \text{B}_{T,c} \geq 0 \text{ (4d)}$$

$$\rho_t(\boldsymbol{m}_{t-1}, B_{t,c}, B_s) \triangleq \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_{t+1}|\tilde{\mathbf{s}}_{1:t})}\left[\rho_{t+1}(\boldsymbol{m}_t, B_{t,c} - ||\boldsymbol{m}_t||_0, B_s)\right] \text{ (5a)}$$
$$\text{s.t.} \quad \boldsymbol{m}_t \dot{\leq} \boldsymbol{m}_{t-1} \text{ (5b)}$$

where B_s is a *sparsity* budget, $\tilde{\mathbf{s}}_{1:t}$ is a vector of observed values for $\boldsymbol{s}_{1:t}$, $\boldsymbol{m}_0$ is an M-dimensional 1's vector, and $\boldsymbol{m}_t \dot{\leq} \boldsymbol{m}_{t-1}$ represents an element-wise comparison between $\boldsymbol{m}_t$ and $\boldsymbol{m}_{t-1}$: $m_t^a \leq m_{t-1}^a$ for $a = 1, \dots, M$. At each iteration t, the saliency $\boldsymbol{s}_t$ is observed and $\boldsymbol{m}_t \in \{0, 1\}^M$ in ρ_t represents a pruning decision performed to maximize the *expectectation* of ρ_{t+1} conditioned upon saliency measurements $\boldsymbol{s}_{1:t}$ collected up to and including iteration t. This recursive structure terminates with base case ρ_T where the saliency of the *unpruned* elements is maximized after T iterations of training.

In the above early pruning formulation, constraints (4c) and (5b) ensure pruning is performed in a practical manner whereby once a network element is pruned, it can no longer be recovered in a later training iteration. We constrain a total computational effort budget $B_{t,c}$ which is reduced per training iteration t by the number of unpruned network elements $||\boldsymbol{m}_t||_0$. We constrain $B_{T,c} \geq 0$ (4d) to ensure training completion within the specified budget. Here we assume that a more sparse pruning mask $\boldsymbol{m}_t$ corresponds to lower computational effort during training iteration t due to updating fewer network elements. Finally, (4a) maximizes the saliency with a pruning mask $\boldsymbol{m}_T$ constrained by a *sparsity* budget B_s (4b). Our early pruning formulation balances the saliency of network elements after convergence against the total computational effort to train such network (i.e., $\boldsymbol{m}_T \cdot \boldsymbol{s}_T$ vs. $\sum_{t=1}^T ||\boldsymbol{m}_t||_0$). This appropriately captures the balancing act of training-time early pruning whereby the computational effort is saved by *early pruning* network elements while preserving the saliency of the remaining network elements after convergence.

3.2 Modeling the Saliency with Multi-Output Gaussian Process

To solve the above early pruning problem, we need to model the belief $p(\boldsymbol{s}_{1:T})$ of the saliency for computing the predictive belief $p(\boldsymbol{s}_{t+1:T}|\tilde{\mathbf{s}}_{1:t})$ of the future saliency in (5a). At the first glance, one may consider to decompose the belief: $p(\boldsymbol{s}_{1:T}) \triangleq \prod_{a=1}^M p(\boldsymbol{s}_{1:T}^a)$ and model the saliency $\boldsymbol{s}_{1:T}^a \triangleq [s_t^a]_{t=1,\dots,T}$ of each network element independently. Such independent models, however, ignore the *co-adaptation* and *co-evolution* of the network elements which have been shown to be a common occurrence in DNN (Hinton et al., 2012; Srivastava et al., 2014; Wang et al., 2020). Also, modeling

[4]Here we use $\{\}$ to distinguish indexing into a tensor from indexing into the sequence of tensors $[\mathbf{W}_{\ell+1,c'}]$.

the correlations between the saliency of different network elements *explicitly* is non-trivial since considerable feature engineering is needed for representing diverse network elements such as neurons, connections, or convolutional filters.

To resolve such issues, we use the *multi-output Gaussian process* (MOGP) to jointly model the belief $p(\boldsymbol{s}_{1:T})$ of all saliency measurements. To be specific, we assume that the saliency s_t^a of the a-th network element at iteration t is a linear mixture[5] of Q independent latent functions $\{u_q(t)\}_{q=1}^Q$: $s_t^a \triangleq \mu_t^a + \sum_{q=1}^Q \gamma_q^a u_q(t)$ where $\mu_t^a \triangleq \mathbb{E}[s_t^a]$. As shown in (Álvarez & Lawrence, 2011), if each $u_q(t)$ is an independent GP with *prior* zero mean and covariance $k_q(t,t')$, then the joint belief $p(\boldsymbol{s}_{1:T})$ follows a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{1:T}, \boldsymbol{K}_{1:T})$ where the *prior* mean vector $\boldsymbol{\mu}_{\tau_1:\tau_2} \triangleq [\mu_t^a]_{t=\tau_1,\dots,\tau_2}^{a=1,\dots,M}$ and covariance matrix $\boldsymbol{K}_{\tau_1:\tau_2} \triangleq [\text{cov}[s_t^a, s_{t'}^{a'}]]_{t,t'=\tau_1,\dots,\tau_2}^{a,a'=1,\dots,M}$ for any $1 \leq \tau_1 \leq \tau_2 \leq T$. In particular, $\text{cov}[s_t^a, s_{t'}^{a'}] = \sum_{q=1}^Q \gamma_q^a \gamma_q^{a'} k_q(t,t')$ due to above linear mixture assumption, which helps to exploit the co-evolution and co-adaptation of network elements within neural networks. To capture the horizontal asymptote trend of $s_1^a, \dots, s_T^a$ as visualized in Appendix A.1, we turn to a kernel used for modeling decaying exponential curves known as the "exponential kernel" (Swersky et al., 2014) and set $k_q(t,t') \triangleq \frac{\beta_q{}^{\alpha_q}}{(t+t'+\beta_q)^{\alpha_q}}$ where α_q and β_q are hyperparameters of MOGP and can be learned via maximum likelihood estimation (Álvarez & Lawrence, 2011). Then, given a vector of observed saliency $\tilde{\mathbf{s}}_{1:t}$, the MOGP regression model can provide a Gaussian predictive distribution for any future saliency $\boldsymbol{s}_{t'}$. Thus, the predictive mean $\mu_{t'|1:t}^a \triangleq \mathbb{E}[s_{t'}^a \mid \tilde{\mathbf{s}}_{1:t}]$ of the saliency $s_{t'}^a$ and the predictive (co)variance $\sigma_{t'|1:t}^{aa'} \triangleq \text{cov}[s_{t'}^a, s_{t'}^{a'} \mid \tilde{\mathbf{s}}_{1:t}]$ between the saliency $s_{t'}^a$ and $s_{t'}^{a'}$ can be computed analytically, as detailed in Appendix A.1.

3.3 Early pruning algorithm.

Solving the above optimizing problem (4) and (5) is difficult due to the interplay between $[\boldsymbol{m}_{t'}]_{t'=t,\dots,T}$, $[B_{t',c}]_{t'=t,\dots,T}$, and $\boldsymbol{m}_T \cdot \boldsymbol{s}_T$. Instead, we consider a simplification of the above problem by only considering solutions of the form $\boldsymbol{m}_{T-1} = \boldsymbol{m}_{T-2} = \ldots = \boldsymbol{m}_t$ which yields[6]:

$$\hat{\rho}_t(\boldsymbol{m}_{t-1}, B_{t,c}, B_s) \triangleq \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})} \left[\rho_T(\boldsymbol{m}_t, B_{t,c} - (T-t)||\boldsymbol{m}_t||_0, B_s)\right] \tag{6}$$

This approach allows us to lift (4d) from (4), to which we add a Lagrange multiplier and achieve:

$$\hat{\rho}_t(\boldsymbol{m}_{t-1}, B_{t,c}, B_s) \triangleq \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})} \left[\hat{\rho}_T(\boldsymbol{m}_t, B_s)\right] + \lambda\left(B_{t,c} - (T-t)||\boldsymbol{m}_t||_0\right) \tag{7}$$

for $t = 1, \dots, T-1$ and $\hat{\rho}_T$ is defined as ρ_T without constraint (4d). Consequently, such a $\hat{\rho}_T$ can be solved in a greedy manner as in (2). Afterwards, we will omit $B_{t,c}$ as a parameter of $\hat{\rho}_T$ as it no longer constrains the solution of $\hat{\rho}_T$. Note that the presence of an *additive* penalty in a *maximization* problem is due to the constraint $B_{T,c} \geq 0 \Leftrightarrow -B_{T,c} \leq 0$ which is the typically expected prior to Lagrangian reformulation. The above optimization problem remains NP-hard as $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}_t, B_s)]$ is submodular in $\boldsymbol{m}_t$ (see Appendix B). Although greedy approximations exist for submodular optimization, their running time of $O(||\boldsymbol{m}_{t-1}||_0^2)$ remains far too slow due to the large number of network elements in DNNs. Fortunately, it can be significantly simplified by exploiting the following lemma (its proof is in Appendix C.):

Lemma 1. *Let $\boldsymbol{e}^{(i)}$ be an M-dimensional one-hot vectors with the i-th element be* 1. $\forall\ 1 \leq a, b \leq M$; $\boldsymbol{m} \in \{0,1\}^M$ *s.t.* $\boldsymbol{m} \wedge (\boldsymbol{e}^{(a)} \vee \boldsymbol{e}^{(b)}) = 0$. *Given a vector of observed saliency $\tilde{\mathbf{s}}_{1:t}$, if $\mu_{T|1:t}^a \geq \mu_{T|1:t}^b$ and $\mu_{T|1:t}^a \geq 0$, then $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\rho_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\rho_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})\right] \leq \mu_{T|1:t}^b \Phi\big(\frac{\mu_{T|1:t}^b - \mu_{T|1:t}^a}{\theta}\big) + \theta\phi\big(\frac{\mu_{T|1:t}^b - \mu_{T|1:t}^a}{\theta}\big)$ where Φ and ϕ are standard normal CDF and PDF respectively, and $\theta \triangleq \sqrt{\sigma_{T|1:t}^{aa} + \sigma_{T|1:t}^{bb} - 2\sigma_{T|1:t}^{ab}}$. In particular, if $\frac{\mu_{T|1:t}^a - \mu_{T|1:t}^b}{\theta} \gg 1$, then $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\rho_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\rho_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})\right] \leq \epsilon$.*

Here, '$\vee$' and '$\wedge$' represent bitwise OR and AND operations, respectively. The bitwise OR operation is used to denote the *inclusion* of $\boldsymbol{e}^{(a)}$ or $\boldsymbol{e}^{(b)}$ in $\boldsymbol{m}_t$. This lemma shows that for any pruning mask $\boldsymbol{m}$, $\mu_{T|1:t}^a \geq \mu_{T|1:t}^b$ is sufficient evidence for opting for $\boldsymbol{m}_t = \boldsymbol{m} \vee e^{(a)}$ as opposed to $\boldsymbol{m}_t = \boldsymbol{m} \vee e^{(b)}$.

[5] Among the various types of MOGPs (see Álvarez & Lawrence (2011) for a detailed review.), we choose this linear model such that the correlations between s_t^a and $s_{t'}^{a'}$ can be computed analytically.

[6] We omit (5b) as it is automatically satisfied due to our simplification.

Lemma 1 admits the following approach to optimize $\hat{\rho}_t$: starting with $\boldsymbol{m}_t = 0^M$, we consider the inclusion of network elements in $\boldsymbol{m}_t$ by the *descending* order of $\{\mu^a_{T|1:t}\}_{a=1}^M$ which can be computed analytically using MOGP. A network element denoted by $\boldsymbol{e}^{(a)}$ is included in $\boldsymbol{m}_t$ if it improves the objective in (6). The algorithm terminates once the highest not-yet-included element does not improve the objective function as a consequence of the penalty term outweighing the improvement in $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\rho_T]$. The remaining excluded elements are then pruned. Due to Lemma 1, this approach is close to the optimum of $\hat{\rho}_t$ as we considered network elements by *descending* order of $\mu^a_{T|1:t}$.

Following the algorithm sketch above, we define the utility of network element v^a_t with respect to candidate pruning mask $\boldsymbol{m}_t \dot{\leq} \boldsymbol{m}_{t-1}$ which measures the improvement in $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\rho_T]$ as a consequence of inclusion of $\boldsymbol{e}^{(a)}$ in $\boldsymbol{m}_t$:

$$\Delta(a, \boldsymbol{m}_t, \tilde{\mathbf{s}}_{1:t}, B_s) \triangleq \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\rho_T(\boldsymbol{e}^{(a)} \vee \boldsymbol{m}_t, B_s) - \rho_T(\boldsymbol{m}_t, B_s)\right]. \tag{8}$$

We can now take a Lagrangian approach to pruning decisions during iteration t by balancing the utility of network element v^a_t against the change of the penalty (i.e., $\lambda(T-t)$) in Algorithm 1. Due to the relatively expensive cost of performing early pruning, we chose to early prune every T_{step} iterations of SGD. Typically T_{step} was chosen to correspond to 10-20 epochs of training. To compute $\Delta(\cdot)$ we sampled from $p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})$ and used a greedy selection algorithm per sample as in (2). During implementation, we also enforced an additional hard constraint $||\boldsymbol{m}_t||_0 \geq B_s$ which we believe is desirable for practicality reasons. We used a fixed value of $B_{1,c} = ||\boldsymbol{m}_0||_0 T_0 + B_s(T - T_0)$ in all our experiments.

Algorithm 1 Bayesian Early Pruning

Require: $\mathcal{N}$, $\boldsymbol{v}_1$, T_0, T_{step}, T, $B_{1,c}$, B_s, λ ▷ DNN $\mathcal{N}$, Lagrangian penalty λ
$\boldsymbol{S}_{1:T_0} \leftarrow$ train$(\mathcal{N}_{\boldsymbol{v}_1}, T_0)$ ▷ Train for T_0 iterations to create seed dataset.
$B_{T_0,c} \leftarrow B_{1,c} - T_0\, dim(\boldsymbol{v}_1)$ ▷ Track computational effort expenditure.
for $k \leftarrow 0, \ldots, \frac{T-T_0}{T_{step}}$; $t \leftarrow T_0 + kT_{step}$ **do** ▷ Early prune every T_{step} iterations from T_0.
  $\boldsymbol{s}_T \leftarrow argsort(-\boldsymbol{\mu}_{T|1:t})$ ▷ Sort descending.
  $\boldsymbol{m}_t \leftarrow 0^{dim(\boldsymbol{v}_t)}$ ▷ Initial pruning mask.
  for $a \leftarrow \boldsymbol{s}^1_T, \ldots, \boldsymbol{s}^{dim(\boldsymbol{v}_t)}_T$ **do** ▷ Consider each network element.
    if $B_{t,c} - (T-t)||\boldsymbol{m}_t||_0 > 0$ **then** ▷ Remaining $B_{t,c}$ budget can support training $\boldsymbol{v}^a_t$.
      $\boldsymbol{m}_t = \boldsymbol{m}_t \vee \boldsymbol{e}^{(a)}$
    else if $\Delta(a, \boldsymbol{m}_t, \tilde{\mathbf{s}}_{1:t}, B_s) \geq \lambda(T-t)$ **then** ▷ Balance utility against change of penalty.
      $\boldsymbol{m}_t = \boldsymbol{m}_t \vee \boldsymbol{e}^{(a)}$
    else
      break
  $prune(\boldsymbol{v}_t, \boldsymbol{m}_t)$ ▷ $dim(\boldsymbol{v}_t)$ is reduced here.
  $B_{t+T_{step},c} \leftarrow B_{t,c} - T_{step}||\boldsymbol{m}_t||_0$
  $\boldsymbol{S}_{t+1:t+T_{step}} \leftarrow train(\mathcal{N}_{\boldsymbol{v}_t}, T_{step})$ ▷ Continue training with pruned network.
return $\mathcal{N}$

4 Experiments and Discussion

We evaluate our modeling approach as well as our BEP algorithm on the CIFAR-10, CIFAR-100 (Krizhevsky, 2009), and ImageNet (Deng et al., 2009) datasets. For CIFAR-10/CIFAR-100 we used a benchmark Convolutional Neural Network (CNN) with 4 convolutional layers, and 1 dense layer.[7] For ImageNet we validated on the ResNet-50 architecture (He et al., 2016a).

Due to the cubic time complexity of MOGPs, we used a variational approximation (Hensman et al., 2015). In all of our models, we used 60 variational inducing points per latent function. We used GPFlow library (Matthews et al., 2017) to build our models.

[7] Code available at `https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py`

Table 1: Comparing log likelihood of test data for independent GPs (GP) vs. MOGP with n latent functions (n-MOGP) on collected saliency measurements from CIFAR-100 training. Measurements are given as a multiple of 10^4. MOGP outperforms GP, particularly on the small dataset. Results are averaged over 20 runs. Extremely large values are due to the GP model being unable to fit the data.

	Small dataset			Medium dataset			Large dataset		
	Lyr 1	Lyr 2	Lyr 3	Lyr 1	Lyr 2	Lyr 3	Lyr 1	Lyr 2	Lyr 3
GP	-0.75	$-5.7e4$	-5.6e4	-0.64	-0.70	$\mathbf{-2.13}$	$-3.4e3$	-0.31	-1.06
4-MOGP	-0.79	-0.98	-3.13	-0.44	-0.60	-2.29	-0.12	-0.24	-1.07
8-MOGP	-0.65	-0.89	-3.00	-0.38	-0.60	-2.20	-0.10	-0.18	-1.02
18-MOGP	$\mathbf{-0.62}$	$\mathbf{-0.84}$	-2.93	-0.36	$\mathbf{-0.56}$	-2.22	-0.09	-0.18	-1.01
32-MOGP	-0.65	-0.85	$\mathbf{-2.89}$	$\mathbf{-0.36}$	-0.59	-2.16	$\mathbf{-0.09}$	$\mathbf{-0.18}$	$\mathbf{-1.00}$

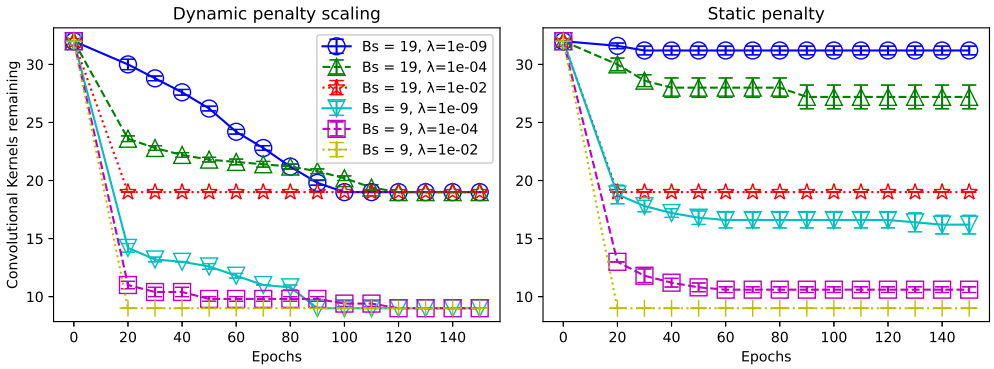

Figure 1: Comparing dynamic penalty scaling vs. static on pruning a 32-convolutional filter layer in a CNN. Dynamic penalty scaling encourages gradual pruning across a wide variety of settings of λ.

4.1 Modeling Evaluation

We compared our MOGP modeling approach against GP modeling. A dataset of saliency measurements was constructed by instrumenting the training process of our 5-layer CNN on the CIFAR-10/CIFAR-100 dataset. Keras (Chollet, 2015) was used to train this model over 150 epoch.[8]

We trained MOGP models with small dataset ($t = [0, 26]$ epochs), medium dataset ($t = [0, 40]$ epochs), and large dataset ($t = [0, 75]$ epochs). Separate models were trained for each layers saliency measurements.[9] We measured the log likelihood of the remainder of the saliency evaluations using these trained models. We present the performance of the models in Table 1 for CIFAR-100.[10] Our MOGP approach better captures the saliency of network elements than a GP approach. Furthermore, using additional latent functions improves MOGP modeling with diminishing returns.

4.2 Small-scale experiments

We applied the early pruning algorithm on the aforementioned architecture, and training regimen. We investigated the behavior of the penalty parameter, λ. We observed that the penalty parameter was difficult to tune properly, either being too aggressive at pruning, or too passive. To rectify this issue, we used dynamic penalty scaling: $\lambda_{dyn} = \lambda\left[(1/\lambda)^{\wedge}\left((T-t)||\boldsymbol{m}_t||_0/B_{t,c} - 1\right)\right]$. This increases the penalty if the anticipated compute required to complete training, $(T-t)||\boldsymbol{m}_t||_0$ begins to exceed the amount of compute budget remaining, $B_{t,c}$. We compare dynamic penalty scaling, and penalty without scaling in Fig. 1 using $T_0 = 20$ epochs, $T_{step} = 10$ epochs for the first convolutional layer of our CNN. Going forward, we use λ_{dyn} as the penalty parameter in our experiments.

[8]Complete experimental setup details found in Appendix C.3.

[9]In our observations, training larger models on saliency measurements from multiple layers yielded no improvement in log-likelihood.

[10]For CIFAR-10 see Appendix C.1.

Table 2: Performance against percentage of neurons/filters pruned per layer with varying λ for tested algorithms.

	CIFAR-10				CIFAR-100			
	70%	80%	90%	95%	70%	80%	90%	95%
DSR	74.1%	65.3%	52.8%	22.1%	37.9%	29.7%	17.5%	4.4%
SNIP	75.4%	67.7%	50.8%	29.4%	22.9%	15.7%	9.9%	2.2%
GraSP	74.6%	66.5%	50.7%	32.9%	28.4%	22.6%	13.9%	0.9%
BEP 1e−2	75.9%	69.7%	54.8%	18.9%	40.6%	32.2%	19.1%	7.1%
BEP 1e−4	75.4%	70.5%	55.7%	**36.1%**	**41.3%**	32.4%	**19.7%**	**8.5%**
BEP 1e−7	**76.0%**	**70.6%**	**56.2%**	30.4%	40.6%	**33.0%**	19.5%	6.6%

Table 3: BEP vs. related work on ResNet-50 on ImageNet dataset. We vary percentage of residual unit sequence channels and filters pruned. Time refers to wall time including any pruning algorithm execution. Benchmark wall-clock time for ResNet-50 is 55h on our hardware.

	Seq 30%, Lyr 60%			Seq 40%, Lyr 90%		
	Top-1	Top-5	Time	Top-1	Top-5	Time
SNIP	72.0%	90.6%	28.6h	62.0%	83.8%	16.5h
GraSP	72.1%	90.6%	29.8h	61.6%	83.6%	19.2h
BEP 1e−1	72.2%	90.6%	34.5h	62.0%	83.8%	24.3h
BEP 1e−4	**72.5%**	**91.0%**	37.0h	**62.3%**	**84.5%**	23.8h

We compare our work with *SNIP* (Lee et al., 2019), *GraSP* (Wang et al., 2020), and momentum-based *dynamic sparse remaparameterization* (DSR) (Dettmers & Zettlemoyer, 2019). To compare against DSR, we instantiate a smaller network of the size BEP yields after training has completed as it is a prune and regrow method. The SNIP and GraSP approaches are extended to neurons/filters by averaging the weight parameter saliencies similarly to (3). We experimented with various degrees of sparsity, using BEP to prune a portion of filters/neurons of each layer.[11] We present the results in Table 2. Our approach better preserves performance at equivalent sparsity. A lower penalty yields higher performing results showing λ serves well at balancing performance vs. computational budget.

4.3 Speeding up ResNet Training on ImageNet

Our chief goal in this work is to speed up training of large-scale DNNs such as ResNet (He et al., 2016a;b) on the ImageNet dataset. Although many convolutional layers of ResNet may be pruned as described by (3), the structure of ResNet requires careful definition of saliency to allow pruning of all layers. ResNet contains long sequences of residual units with matching number of input/output filters. The inputs of residual units are connected with *shortcut connections* (i.e., through addition) to the output of the residual unit.[12] This structure requires that within a sequence of residual units, the number of inputs/outputs to the units must match exactly. Thus we *group prune* channels of residual units within a sequence of residual units, where group pruning an output channel of a residual unit sequence requires pruning it from the inputs/outputs of all residual units within the sequence.[13]

We trained Resnet-50 with BEP as well as SNIP and GraSP.[14] We group pruned less aggressively as residual unit channels feed into a large number of residual units, thus making aggressive pruning likely to degrade performance. We ran BEP iterations at $t = [15, 20, 25, 35, 45, 55, 75]$ epochs. We trained for 100 epochs on $4\times$ Nvidia Geforce GTX 1080 Ti GPUs. More experimental details found in Appendix C.3. We present our results in Table 3. We achieve higher performance than related

[11] In our observations saliency measurements don't well capture network element efficacy when comparing across layers. Thus pruning whole networks using network element saliency yields poor performing networks with bottlenecks. This limitation of saliency functions is well known (See Molchanov et al. (2017) Appendix A.1 and A.2; Wang et al. (2020) Section 3 last paragraph). Development of saliency functions which overcome this shortcoming while remaining performant is a difficult open problem outside the scope of this work.

[12] Precise details of the ResNet architecture may be found in He et al. (2016a) Section 3.

[13] We formally define saliency on residual unit sequences in Appendix C.4

[14] We omit comparison to DSR due to differing underlying deep learning library which makes walltime comparisons inaccurate.

techniques, albeit at longer wall time. Our approach captures the *training time vs. performance* tradeoff present in DNNs, unlike competing approaches.

5 Related Work

Pruning and related techniques. Initial works in DNN pruning center around saliency based pruning after training including Skeletonization (Mozer & Smolensky, 1988), Optimal Brain Damage and followup work (Hassibi & Stork, 1992; LeCun et al., 1989) as well as sensitivity based pruning (Karnin, 1990). In recent years, saliency functions been adapted to pruning neurons or convolutional filters. Li et al. (2017) define a saliency function on convolutional *filters* by using the L_1 norm. Molchanov et al. (2017) propose using a first-order Taylor-series approximation on the objective function as a saliency measure. Dong et al. (2017) propose layer wise pruning of weight parameters using a Hessian based saliency measure. Several variants of pruning after training exist. Han et al. (2015) propose *iterative pruning* where pruning is performed in stages alternating with fine tune training. Guo et al. (2016) suggest dynamic network surgery, where pruning is performed on-the-fly during evaluation time. Li et al. (2017) and He et al. propose reinforcement learning for pruning decisions. A comprehensive overview may be found in Gale et al. (2019).

Knowledge distillation (Hinton et al., 2015; Lu et al., 2017; Tung & Mori, 2019; Yim et al., 2017) aim to transfer the capabilities of a trained network into a smaller network. Weight sharing (Nowlan & Hinton, 1992; Ullrich et al., 2017) and low rank matrix factorization (Denton et al., 2014; Jaderberg et al., 2014) aim to compress the parameterization of neural networks. Network quantization (Courbariaux et al., 2015; Hubara et al., 2017; Micikevicius et al., 2018) use lower fidelity representation of network elements (e.g. 16-bit) to speed up training and evaluation. Although speedup during training is achievable through network quantization, this technique requires hardware support and only provides coarse granularity in trading off computational effort vs. performance. Current GPUs only extend native support to 16-bit floating point operations. Furthermore, our approach is orthogonal to quantization allowing the techniques to be combined for further speedup.

Initialization time or runtime pruning. Frankle & Carbin (2019) show that a randomly initialized DNN contains a small subnetwork, which if trained by itself, yields equivalent performance to the original network. SNIP (Lee et al., 2019) and GraSP (Wang et al., 2020) propose pruning connection weights prior to the training process through a first order and second order saliency function respectively. Sparse Evolutionary Training (Mocanu et al., 2018) propose initializing networks with sparse topology prior to training. Narang et al. (2017) consider connection weight pruning during training for recurrent neural networks using a heuristic approach.

Dynamic sparse reparameterization considers pruning and regrowing parameter weights during the training process (Bellec et al., 2018; Dettmers & Zettlemoyer, 2019; Mostafa & Wang, 2019). Dai et al. (2019) propose a grow and prune approach to learning network architecture and connection layout. We differ from existing work as our focus is on speeding up neural network training, meanwhile other works in runtime pruning aim to achieve sparse network layouts. To the best of our knowledge, except for small speedups presented in (Dettmers & Zettlemoyer, 2019), the mentioned works do not demonstrate practical speedup during training time using popular deep learning libraries run on modern GPUs.

6 Conclusion

This paper presents a novel efficient algorithm to perform pruning of DNN elements such as neurons, or convolutional layers during the training. To achieve *early* pruning before the training converges while preserving the performance of the DNN upon convergence, a Bayesian model (i.e., MOGP) is used to predict the saliency of DNN elements in the future (unseen) training iterations by exploiting the exponentially decaying behavior of the saliency and the correlations between saliency of different network elements. Then, we exploit a property (Lemma 1) of the objective function and propose an efficient Bayesian early pruning algorithm. Empirical evaluations on benchmark datasets show that our algorithm performs favorably to related works for pruning convolutional filters and neurons. Our approach remains flexible to changes in saliency function, and appropriately *balances* the training time vs. performance tradeoff in training DNNs. We are able to train an early pruned ResNet-50 model achieving a 48.6% speedup (37h vs. 55h) while maintaining a validation accuracy of 72.5%.

REFERENCES

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Proc. NeurIPS*, pp. 6155–6166, 2019.

Mauricio A. Álvarez and Neil D. Lawrence. Computationally efficient convolved multiple output Gaussian processes. *JMLR*, 12(1):1459–1500, 2011.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert A. Legenstein. Deep rewiring: Training very sparse deep networks. In *Proc. ICLR*, 2018.

Aydin Buluç and John R. Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *Proc. ICCP*, pp. 503–510, 2008.

François Chollet, 2015. URL `https://github.com/fchollet/keras`.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. arXiv:1511.00363, 2015.

Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Trans. Computers*, 68(10):1487–1497, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*, pp. 248–255, 2009.

Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proc. NeurIPS*, pp. 1269–1277, 2014.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. arXiv:1907.04840, 2019.

Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Proc. NeurIPS*, pp. 4857–4867, 2017.

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

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

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *Proc. NeurIPS*, pp. 1379–1387, 2016.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural networks. In *Proc. NeurIPS*, pp. 1135–1143, 2015.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Proc. NeurIPS*, pp. 164–171, 1992.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proc. ECCV*, pp. 4432–4440, 2016b.

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 *Proc. ECCV*, pp. 815–832.

James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. In *Proc. AISTATS*, pp. 351–360, 2015.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. arXiv:1503.02531, 2015.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *JMLR*, 18 (1):6869–6898, 2017.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. BMVC*, 2014.

Ehud D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Networks*, 1(2):239–242, 1990.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Univ. Toronto, 2009.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Proc. NeurIPS*, pp. 598–605, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. In *Proc. ICLR*, 2019.

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

Liang Lu, Michelle Guo, and Steve Renals. Knowledge distillation for small-footprint highway networks. In *Proc. ICASSP*, pp. 4820–4824, 2017.

Alexander Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using tensorflow. *JMLR*, 18(1):1–6, 2017.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *Proc. ICLR*, 2018.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature*, 9(1):1–12, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proc. ICLR*, 2017.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proc. ICML*, pp. 4646–4655, 2019.

Michael Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Proc. NeurIPS*, pp. 107–115, 1988.

Saralees Nadarajah and Samuel Kotz. Exact distribution of the max/min of two gaussian random variables. *Trans. VLSI*, 16(2):210–212, 2008.

Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *Proc. ICLR*, 2017.

Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

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

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

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw Bayesian optimization. arXiv:1406.3896, 2014.

Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *Proc. ICCV*, pp. 1365–1374, 2019.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *Proc. ICLR*, 2017.

Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. Picking winning tickets before training by preserving gradient flow. In *Proc. ICLR*, 2020.

Carl Yang, Aydin Buluç, and John D. Owens. Design principles for sparse matrix multiplication on the GPU. In *Proc. Euro-Par*, pp. 672–687, 2018.

Junho Yim, Donggyu Joo, Ji-Hoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proc. CVPR*, pp. 7130–7138, 2017.