Pruning during training by network efficacy modeling

Mohit Rajpal[1*], Yehong Zhang[2] and Bryan Kian Hsiang Low[1]

[1]Department of Computer Science, National University of Singapore, Republic of Singapore.
[2]Peng Cheng Laboratory, Shenzhen, People's Republic of China.

*Corresponding author(s). E-mail(s): mohitr@comp.nus.edu.sg;
Contributing authors: zhangyh02@pcl.ac.cn;
lowkh@comp.nus.edu.sg;

Abstract

Deep neural networks (DNNs) are costly to train. Pruning, an approach to alleviate model complexity by zeroing out or pruning DNN elements, has shown promise in reducing training costs for DNNs with little to no efficacy at a given task. This paper presents a novel method to perform *early* pruning of DNN elements (e.g., neurons or convolutional filters) *during the training process* while minimizing losses to model performance. To achieve this, we model the efficacy of DNN elements in a Bayesian manner conditioned upon efficacy data collected during the training and prune DNN elements with low *predictive* efficacy after training completion. Empirical evaluations show that the proposed Bayesian early pruning improves the computational efficiency of DNN training while better preserving model performance compared to other tested pruning approaches.

Keywords: Early pruning, Network efficacy modeling, Network saliency, Multi-output Gaussian process, Foresight pruning

1 Introduction

Deep neural networks (DNNs) are known to be overparameterized (Allen-Zhu, Li, & Liang, 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 performance. *Pruning* (LeCun, Denker, & Solla, 1989) is an approach to alleviate 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* and/or deploy the trained model on resource constrained devices (e.g., mobile phone, camera, etc.). 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 model performance (Han, Pool, Tran, & Dally, 2015; H. Li, Kadav, Durdanovic, Samet, & Graf, 2017; Lin et al., 2019; Molchanov, Tyree, Karras, Aila, & Kautz, 2017).

In particular, the overparameterization of a DNN also leads to considerable training cost 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 early. This problem is further compounded by the development of larger network architectures which are very expensive to train. These observations motivate the need of *early pruning*.

The objective of early pruning is to perform pruning *during training* for reducing training cost while minimizing losses to *test time* model accuracy given a fixed final network sparsity (e.g., determined by the resource constraints of the deployed devices). This necessitates consideration for both the test time model accuracy and training cost as both metrics are highly desirable to users. Related works in pruning during training (Lym et al., 2019) as well as prune and regrow (Bellec, Kappel, Maass, & Legenstein, 2018; Dettmers & Zettlemoyer, 2019; Mostafa & Wang, 2019) approaches do not jointly consider both competing metrics while offering a fixed final network sparsity. Similarly, pruning at initialization (de Jorge et al., 2021; Lee, Ajanthan, & Torr, 2019; Tanaka, Kunin, Yamins, & Ganguli, 2020; C. Wang, Zhang, & Grosse, 2020) offers a method of reducing training cost, which however, overly sacrifices test time model accuracy by pruning *at initialization* when *test time* network element efficacy is not well known. Therefore, previous work does not offer a mechanism to *trade-off* between training cost and test time model accuracy according to user-defined sparsity objectives. See Section 2 for detailed literature review.

To offer a mechanism to trade-off between training cost and test time model accuracy, several problems must be addressed. Firstly, early pruning requires minimizing losses to test time accuracy, but pruning during training when the test time element efficacy is *unknown*. Thus, pruning decisions need to base on the *inferred* test time network element efficacy. *How to infer test time network element efficacy for making accurate early pruning decisions* is an important question that has not been addressed by related work. Secondly, building an inference model requires collecting network efficacy observations during training. A more accurate model requires collecting more observations during training, which increases the DNN training cost. Meanwhile, pruning with few observations incurs a low DNN training cost but sacrifices DNN model performance due to the inaccurate efficacy inference. Given this trade-off, *when should a predicted poorly performing element be pruned?* Finally, as both training cost and test time accuracy are important metrics to end users, *how should these metrics be balanced while addressing the above challenges?* These important questions have not been addressed by related works and are the focus of this work.

To answer these questions, this work considers to model the network element efficacy during training and do early pruning based on the predictive element efficacy and its predictive confidence upon convergence. A network element is pruned when a sufficiently high degree of confidence is reached regarding its poor performance. To naturally trade off between training cost and test time performance, we formulate the early pruning as a constrained optimization problem and propose an efficient algorithm for solving it. The trade-off is achieved by modulating the degree of confidence necessary before a poorly performing element is pruned. Pruning with a high degree of confidence makes fewer mistakes, yet requires collecting more observations which increases the training cost. Conversely, pruning with a low degree of confidence makes more mistakes, yet requires fewer observations, and thus less training cost. The specific contributions of this work include:

- Posing early pruning as a constrained optimization problem such that a fixed final network sparsity can naturally be achieved (Section 4.1);
- Proposing to infer the efficacy of DNN elements using *multi-output Gaussian process* (MOGP) which models the *belief* of element efficacy conditioned upon efficacy measurements collected during training. This approach not only identifies poorly performing elements, but provides a measure of confidence by assigning a probabilistic belief to its prediction (Section 4.2);
- Designing a *Bayesian early pruning* (BEP) algorithm to allow trading off between training cost and test time performance (Section 4.3). To the best of our knowledge, this is the first algorithm that can naturally satisfy a fixed final network sparsity while can dynamically achieve the training time vs. performance trade-off. Existing works either require fixed scheduled pruning strategy or cannot achieve the fixed final network sparsity without tuning parameters.
- Demonstrating strong performance improvements of our BEP algorithm when a large percentage of the network is pruned. This improvement is significant as DNNs continue to grow in size and may require significant pruning to allow training in a short time frame (Section 5).

Pruning typically 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 accommodate this, our algorithm is agnostic, and therefore flexible, to changes in saliency function.

We use BEP to prune neurons and convolutional filters and demonstrate its ability to capture the *training cost vs. performance* trade-off.[1]

2 Related Work

2.1 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. (H. 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, Chen, & Pan, 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, Yao, & Chen, 2016) suggest dynamic network surgery, where pruning is performed on-the-fly during evaluation time. The works of H. Li et al. (2017) and Y. He et al. (2018) propose reinforcement learning for pruning decisions. A comprehensive overview may be found in (Gale, Elsen, & Hooker, 2019).

Knowledge distillation (Hinton, Vinyals, & Dean, 2015; Lu, Guo, & Renals, 2017; Tung & Mori, 2019; Yim, Joo, Bae, & Kim, 2017) aim to transfer the capabilities of a trained network into a smaller network. Weight sharing (Nowlan & Hinton, 1992; Ullrich, Meeds, & Welling, 2017) and low rank matrix factorization (Denton, Zaremba, Bruna, LeCun, & Fergus, 2014; Jaderberg, Vedaldi, & Zisserman, 2014) aim to compress the parameterization of neural networks. Network quantization (Courbariaux, Bengio, & David, 2015; Hubara, Courbariaux, Soudry, El-Yaniv, & Bengio, 2017; Micikevicius et al., 2018) use lower fidelity representation of network elements (e.g. 16-bit) to speed up training and evaluation. Our work is orthogonal to network quantization as we reduce overall training FLOPs.

2.2 Initialization Time or Training Time Pruning

Pruning at initialization builds on the work of Frankle and Carbin (2019). This work shows 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 (C. Wang et al., 2020) propose pruning connection weights prior to the training process through a first order and second order saliency function respectively. This technique is improved upon with IterSnip (de Jorge et al., 2021) and SynFlow (Tanaka et al., 2020).

[1]We didn't consider pruning network connections since it cannot be easily capitalized upon with performance improvements due to the difficulty of accelerating sparse matrix operations with existing deep learning libraries (Buluç & Gilbert, 2008; Yang, Buluç, & Owens, 2018).

PruneFromScratch (Y. Wang et al., 2020) considers pruning at initialization in order to order training cost reduction. In comparison to our approach, above works in initialization time pruning do not offer a mechanism to trade-off between network training time and test time accuracy.

Dynamic sparse reparameterization considers pruning and regrowing parameter weights during the training process (Bellec et al., 2018; Dettmers & Zettlemoyer, 2019; Liu, Xu, Shi, Cheung, & So, 2020; Mostafa & Wang, 2019). Sparse evolutionary training (Mocanu et al., 2018) proposes initializing networks with sparse topology prior to training. Dai, Yin, and Jha (2019) propose a grow and prune approach to learning network architecture and connection layout. Other works (Louizos, Welling, & Kingma, 2018; Narang, Diamos, Sengupta, & Elsen, 2017) propose pruning using heuristics such as L_0 regularization for DNNs and recurrent neural networks. However all the above-mentioned works prune connection weights, their approach cannot be utilized to deliver training time improvements and do not offer a principled approach to capture the training cost vs. performance trade-off.

PruneTrain (Lym et al., 2019) also proposes pruning filters during training to achieve training cost reduction while minimizing degradation to performance. In contrast to our approach, PruneTrain does not allow specification of the desired network size after training. A specified network size is important when training for resource constrained devices such as mobile phones or edge devices which require networks to conform to user specified size limits. It is unclear how to solve the early pruning problem using PruneTrain. We compare with PruneTrain under the early pruning problem definition in Section **??**.

3 Preliminaries of 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 user specified 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 it. 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/important* in minimizing the loss function. Consequently, pruning can be done by maximizing the saliency of the network elements given the user specified 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. The above optimization problem can be efficiently solved by selecting the B most *salient* network elements in $\boldsymbol{v}_T$.

The construction of the saliency function has been discussed in many existing works: Some approaches derived the saliency function from first-order (LeCun et al., 1989; Molchanov et al., 2017) and second-order (Hassibi & Stork, 1992; C. Wang et al., 2020) Taylor series approximations of $\mathcal{L}$. Other common saliency functions include L_1 (H. Li et al., 2017) or L_2 (Wen, Wu, Wang, Chen, & Li, 2016) norm of the network element weights, as well as mean activation (Polyak & Wolf, 2015). In this work, we use a first-order Taylor series approximation saliency function defined for neurons and convolutional filters[2] (Molchanov et al., 2017). Due to the first-order (i.e., gradient based) approximation, this saliency function has minimal memory and computational overhead during DNN training. However, our approach remains flexible to arbitrary choice of saliency function on a plug-n-play basis.

For ease of reference, we summarize notations that will be used frequently in the remaining sections of this paper in Table 1.

4 Bayesian Early Pruning

4.1 Problem Statement

As has been mentioned before, existing pruning works based on the saliency function are typically done after the training convergence (i.e., (2)) to speed up the test-time evaluation on the resource constrained devices, which wastes considerable time on training these network elements that will eventually be pruned. 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.

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, $\boldsymbol{s}_t \triangleq [s_t^a]_{a=1,\ldots,M}$ for $t = 1,\ldots,T$, and $\boldsymbol{s}_{\tau_1:\tau_2} \triangleq [\boldsymbol{s}_t]_{t=\tau_1,\ldots,\tau_2}$ be a vector of saliency of all the network elements between iterations τ_1 and τ_2. Our early pruning algorithm is designed

[2] Implementation details of this saliency function can be found in Appendix A.

Table 1: Summary of important notations.

Notation	Description
M	Total number of network elements in a neural network.
T	Total number of SGD iterations in the training procedure.
s_t^a	Random variable representing saliency of the a-*th* element at iteration t .
$\boldsymbol{s}_t$	A vector of saliency $[s_t^a]_{a=1}^M$ for all the network elements at iteration t .
$\boldsymbol{s}_{\tau_1:\tau_2}$	A matrix of saliency $[\boldsymbol{s}_t]_{t=\tau_1}^{\tau_2}$ between iterations τ_1 and τ_2 .
$\tilde{\mathbf{s}}_{1:t}$	The realization of random vector $\boldsymbol{s}_{1:t}$.
$\boldsymbol{m}_t$	A vector of pruning mask at iteration t .
B_s	Network sparsity budget after training specified by the user.
$B_{t,c}$	Computational budget at iteration t .
$\mu_{t'\|1:t}^a$	Predictive mean of the saliency $s_{t'}^a$ given realized saliency $\tilde{\mathbf{s}}_{1:t}$.
$\sigma_{t'\|1:t}^{aa'}$	Predictive covariance of saliency $s_{t'}^a$ and $s_{t'}^{a'}$ given realized saliency $\tilde{\mathbf{s}}_{1:t}$.

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 $t = 1, \ldots, T$:

$$\rho_T(\boldsymbol{m}_{T-1}, B_{T,c}, B_s) \triangleq \max_{\boldsymbol{m}_T} \ \boldsymbol{m}_T \cdot \boldsymbol{s}_T \quad (3a)$$

$$\text{s.t.} \quad ||\boldsymbol{m}_T||_0 \le B_s \quad (3b) \qquad \boldsymbol{m}_T \dot{\le} \boldsymbol{m}_{T-1} \quad (3c) \qquad \mathrm{B}_{T,c} \ge 0 \quad (3d)$$

$$\rho_t(\boldsymbol{m}_{t-1}, B_{t,c}, B_s) \triangleq \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_{t+1}|\tilde{\mathbf{s}}_{1:t})}\left[\rho_{t+1}(\boldsymbol{m}_t, B_{t,c} - ||\boldsymbol{m}_t||_0, B_s)\right] \quad (4a)$$

$$\text{s.t.} \quad \boldsymbol{m}_t \dot{\le} \boldsymbol{m}_{t-1} \quad (4b)$$

where B_s is the user specified *sparsity* budget of the trained network, $\tilde{\mathbf{s}}_{1:t}$ is a vector of observed values for $\boldsymbol{s}_{1:t}$, $\boldsymbol{m}_0$ is an M-dimensional 1's vector, and $\boldsymbol{m}_t \dot{\le} \boldsymbol{m}_{t-1}$ represents an element-wise comparison between $\boldsymbol{m}_t$ and $\boldsymbol{m}_{t-1}$: $m_t^a \le m_{t-1}^a$ for $a = 1, \ldots, M$. At each iteration t, the saliency $\boldsymbol{s}_t$ is observed and $\boldsymbol{m}_t \in \{0,1\}^M$ in (4a) represents a pruning decision performed to maximize the *expectation* of ρ_{t+1} over the predictive belief of $\boldsymbol{s}_{t+1}$ conditioned upon saliency measurements $\tilde{\mathbf{s}}_{1:t}$ 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 (3c) and (4b) ensure that 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 define a sparsity budget B_s (3b), which specifies the desired network size after training completion. This constraint is important as often training is performed on GPUs for resource constrained devices (e.g., edge devices, or mobile phones) which can only support networks of limited size. We also 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 $B_{T,c} \ge 0$ (3d) to ensure training completion within the specified computational 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, (3a) maximizes the saliency with a pruning mask $\boldsymbol{m}_T$ constrained by a *sparsity* budget B_s (3b). 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.

4.2 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 $p(\boldsymbol{s}_{t+1:T} \mid \tilde{\mathbf{s}}_{1:t})$ of the future saliency in (4a). At the first glance, one may consider to decompose the belief: $p(\boldsymbol{s}_{1:T}) \triangleq \prod_{a=1}^{M} p(\boldsymbol{s}^a_{1:T})$ and model the saliency $\boldsymbol{s}^a_{1:T} \triangleq [s^a_t]_{t=1,\ldots,T}$ 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, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014; C. Wang et al., 2020). Also, modeling 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 *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^a_t of the a-th network element at iteration t is a linear mixture[3] of Q independent latent functions $\{u_q(t)\}_{q=1}^{Q}$: $s^a_t \triangleq \sum_{q=1}^{Q} \gamma^a_q u_q(t)$. 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 resulting distribution over $p(\boldsymbol{s}_{1:T})$ is a multivariate Gaussian distribution with prior zero mean and covariance determined by the mixing covariance: $\text{cov}[s^a_t, s^{a'}_{t'}] = \sum_{q=1}^{Q} \gamma^a_q \gamma^{a'}_q k_q(t, t')$. This explicit covariance between s^a_t and $s^{a'}_{t'}$ helps to exploit the co-evolution and co-adaptation of network elements within the neural networks.

To capture the horizontal asymptote trend of $s^a_1, \ldots, s^a_T$ as visualized in Fig. 1, we turn to a kernel used for modeling decaying exponential curves known as the "exponential kernel" (Swersky, Snoek, & Adams, 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).

Let the *prior* covariance matrix be $\boldsymbol{K}_{\tau_1:\tau_2} \triangleq [\text{cov}[s^a_t, s^{a'}_{t'}]]^{a,a'=1,\ldots,M}_{t,t'=\tau_1,\ldots,\tau_2}$ for any $1 \le \tau_1 \le \tau_2 \le T$. Then, given a vector of observed saliency $\tilde{\mathbf{s}}_{1:t}$, the MOGP

[3]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^a_t and $s^{a'}_{t'}$ can be computed analytically.

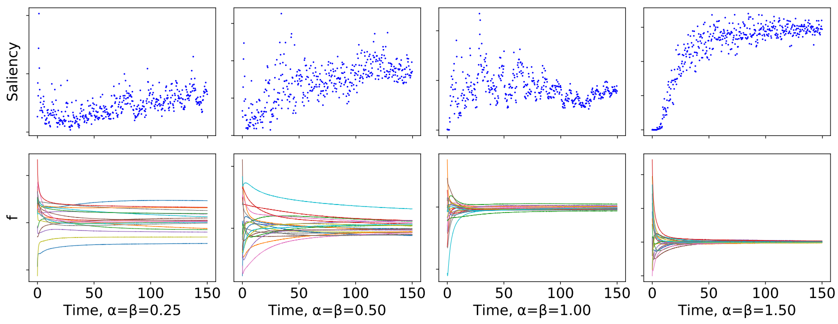

Fig. 1: Above: Saliency of different convolutional filters over 150 epochs of SGD for a convolutional neural network trained on CIFAR-10 dataset. Below: Function samples drawn from GP with the exponential kernel. Both processes follow an exponentially decaying and asymptotic behavior.

regression model can provide a Gaussian predictive distribution $p(\boldsymbol{s}_{t'}|\tilde{\mathrm{s}}_{1:t}) = \mathcal{N}(\boldsymbol{\mu}_{t'|1:t}, \boldsymbol{K}_{t'|1:t})$ for any future saliency $\boldsymbol{s}_{t'}$ with the following *posterior* mean vector and covariance matrix:

$$\boldsymbol{\mu}_{t'|1:t} \triangleq \boldsymbol{K}_{[t't]}\boldsymbol{K}_{1:t}^{-1}\tilde{\mathrm{s}}_{1:t}, \quad \boldsymbol{K}_{t'|1:t} \triangleq \boldsymbol{K}_{t':t'} - \boldsymbol{K}_{[t't]}\boldsymbol{K}_{1:t}^{-1}\boldsymbol{K}_{[t't]}^{\top}$$

where $\boldsymbol{K}_{[t't]} \triangleq [\mathrm{cov}[s_{t'}^{a}, s_{\tau}^{a'}]]_{\tau=1,\ldots,t}^{a,a'=1,\ldots,M}$. Then, the a-th element $\mu_{t'|1:t}^{a}$ of $\boldsymbol{\mu}_{t'|1:t}$ is the predictive mean of the saliency $s_{t'}^{a}$. And the $[a,a']$-th element of $\boldsymbol{K}_{t'|1:t}$ denoted as $\sigma_{t'|1:t}^{aa'}$ is the predictive (co)variance between the saliency $s_{t'}^{a}$ and $s_{t'}^{a'}$.

4.2.1 On the Choice of the "Exponential Kernel"

We justify our choice of the exponential kernel as a modeling mechanism by presenting visualizations of saliency measurements collected during training, and comparing these to samples drawn from the exponential kernel $k_q(t,t') \triangleq \frac{\beta^\alpha}{(t+t'+\beta)^\alpha}$, as shown in Fig. 1. Both the saliency of various convolutional filters and the function samples exhibit exponentially decaying behavior, which makes the exponential kernel a strong fit for modeling saliency evolution over time.

Furthermore, we note that the exponential kernel was used to great effect in Swersky et al. (2014) with respect to modeling loss curves as a function of epochs. Similar to the saliency measurement curves, loss curves also exhibit asymptotic behavior, which provides evidence for the exponential kernel being an apt fit for our saliency modeling task.

4.3 Bayesian Early Pruning (BEP) Algorithm

Solving the above optimizing problem (3) and (4) is difficult due to the interplay between $[\boldsymbol{m}_{t'}]_{t'=t,\ldots,T}$, $[B_{t',c}]_{t'=t,\ldots,T}$, and $\boldsymbol{m}_T \cdot \boldsymbol{s}_T$. To overcome this difficulty, we instead analyze a lower bound of $\rho_t(\cdot)$:

$$\begin{aligned}\rho_t(\boldsymbol{m}_{t-1}, B_{t,c}, B_s) &= \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_{t+1}|\tilde{\mathbf{s}}_{1:t})}\left[\rho_{t+1}(\boldsymbol{m}_t, B_{t,c} - ||\boldsymbol{m}_t||_0, B_s)\right] \\ &\geq \max_{\boldsymbol{m}_t} \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\rho_T(\boldsymbol{m}_t, B_{t,c} - (T-t)||\boldsymbol{m}_t||_0, B_s)].\end{aligned} \tag{5}$$

We prove this lower bound in Appendix B. Substituting this lower bound[4] we define $\hat{\rho}(\cdot)$:

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

This approach allows us to lift (3d) from (3), to which we add a Lagrange multiplier and achieve:

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

for $t = 1, \dots, T-1$ and $\hat{\rho}_T$ is defined as ρ_T without constraint (3d). 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 typically expected prior to Lagrangian reformulation.

To proceed with the analysis, we show the above optimization problem is submodular in $\boldsymbol{m}_t$. In (7), the problem of choosing $\boldsymbol{m}$ from $\{0,1\}^M$ can be considered as selecting a subset A of indexes from $\{1, \dots, M\}$ such that $m_t^a = 1$ for $a \in A$, and $m_t^a = 0$ otherwise. Therefore, $P(\boldsymbol{m}) \triangleq \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}, B_s)]$ can be considered as a set function which we will show to be submodular. To keep notation consistency, we will remain using $P(\boldsymbol{m})$ instead of representing it as a function of the index subset A.

Lemma 1 Let $\boldsymbol{m}'$, $\boldsymbol{m}'' \in \{0,1\}^M$, and $e^{(a)}$ be arbitrary M-dimensional one hot vector with $1 \leq a \leq M$ and $P(\boldsymbol{m}) \triangleq \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}, B_s)]$. We have $P(\boldsymbol{m}' \vee e^{(a)}) - P(\boldsymbol{m}') \geq P(\boldsymbol{m}'' \vee e^{(a)}) - P(\boldsymbol{m}'')$ for any $\boldsymbol{m}' \dot{\leq} \boldsymbol{m}''$, $\boldsymbol{m}' \wedge e^{(a)} = 0^M$, and $\boldsymbol{m}'' \wedge e^{(a)} = 0^M$.

Here, '$\vee$' and '$\wedge$' represent bitwise OR and AND operations, respectively. The bitwise OR operation is used to denote the *inclusion* of $\boldsymbol{e}^{(a)}$ in $\boldsymbol{m}_t$. The proof for Lemma 1 is presented in Appendix C. Greedy approximations for submodular optimization have a running time of $O(||\boldsymbol{m}_{t-1}||_0^2)$, which remains far too slow due to the large number of network elements in DNNs. To overcome this, we exploit the strong tail decay of multivariate gaussian density $p(\boldsymbol{s}_T \mid \tilde{\boldsymbol{s}}_{1:t})$ to deliver an efficient approximation procedure. Our approach relies on the following lemma (its proof is in Appendix D.):

[4]We omit (4b) as it is automatically satisfied due to our lower bound.

Lemma 2 Let $\boldsymbol{e}^{(i)}$ be a M-dimensional one-hot vectors with the i-th element be 1. $\forall\ 1 \leq a, b \leq M, \boldsymbol{m} \in \{0,1\}^M\ s.t. \boldsymbol{m} \wedge (\boldsymbol{e}^{(a)} \vee \boldsymbol{e}^{(b)}) = 0^M$. Given a vector of observed saliency $\tilde{\mathbf{s}}_{1:t}$, if $\mu^a_{T|1:t} \geq \mu^b_{T|1:t}$ and $\mu^a_{T|1:t} \geq 0$, then

$$\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})] \leq \mu^b_{T|1:t}\ \Phi(\nu/\theta) + \theta\ \phi(\nu/\theta)$$

where $\theta \triangleq \sqrt{\sigma^{aa}_{T|1:t} + \sigma^{bb}_{T|1:t} - 2\sigma^{ab}_{T|1:t}}$, $\nu \triangleq \mu^b_{T|1:t} - \mu^a_{T|1:t}$, and Φ and ϕ are standard normal CDF and PDF, respectively.

Due to the strong tail decay[5] of ϕ and Φ, Lemma 2 indicates that, with high probability, opting for $\boldsymbol{m}_t = \boldsymbol{m} \vee e^{(a)}$ is not a much worse choice than $\boldsymbol{m}_t = \boldsymbol{m} \vee e^{(b)}$ given $\mu^a_{T|1:t} \geq \mu^b_{T|1:t}$. This 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 $\{\mu^a_{T|1:t}\}^M_{a=1}$ 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 $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T]$. The remaining excluded elements are then pruned.

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 $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T]$ as a consequence of inclusion of $\boldsymbol{e}^{(a)}$ in $\boldsymbol{m}_t$:

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

In computing $\Delta(\cdot)$, we take the expectation over the distribution $p(\boldsymbol{s}_T \mid \tilde{\mathbf{s}}_{1:t})$, which utilizes both the predictive mean and variance of the network element. Thus, the confidence of the MOGP prediction is considered prior to pruning as mentioned in Section 1. We can now take a Lagrangian approach to make pruning decisions during iteration t by balancing the utility of network element v^a_t against the change of the penalty (i.e., $\lambda_t(T-t)$), as shown 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 $p(\boldsymbol{s}_T \mid \tilde{\boldsymbol{s}}_{1:t})$ 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.

Finally, we show how λ_t offers a probabilistic guarantee in the poor performance of a pruned network element. Using standard Markov inequality, we show the following:

[5]Note as $\mu^a_{T|1:t} \geq \mu^b_{T|1:t}$, $\Phi(\cdot) \leq 0.5$ and experiences tail decay proportional to $\mu^a_{T|1:t} - \mu^b_{T|1:t}$.

Lemma 3 Let $\boldsymbol{e}^{(*)}$ represent a pruned element at time t with the highest predictive mean $\mu^*_{T|1:t} \geq 0$. Given an arbitrary pruned element $\boldsymbol{e}^{(a)}$ at time t, then for all $\delta \in (0, 1)$ the following holds:

$$p\left(\hat{\rho}_T(\boldsymbol{e}^{(a)} \vee \boldsymbol{m}_t, B_s) - \hat{\rho}_T(\boldsymbol{m}_t, B_s) < \frac{\lambda_t}{\delta}(T - t + \epsilon)\right) > 1 - \delta$$

where $\epsilon \triangleq \left[\mu^a_{T|1:t}\Phi(\nu/\theta) + \theta\ \phi(\nu/\theta)\right]/\lambda_t$ with $\theta \triangleq \sqrt{\sigma^{**}_{T|1:t} + \sigma^{aa}_{T|1:t} - 2\sigma^{*a}_{T|1:t}}$, and $\nu \triangleq \mu^a_{T|1:t} - \mu^*_{T|1:t}$.

The proof is in Appendix E. The above lemma shows that λ_t acts as a probabilistic guarantee of the poor performance of a pruned network element. A smaller λ_t offers a higher probability in the poor performance of an element prior to pruning. Consequently, λ_t is inversely correlated with training time where a lower λ_t requires more training time as fewer network elements are pruned. This offers a *trade-off* between training time and performance using the penalty parameter λ_t.

4.4 BEP-LITE

We may further reduce the training cost of BEP by combining with pruning at initialization. This combination is motivated by noticing that initialization pruning techniques (Lee et al., 2019; C. Wang et al., 2020) implicitly utilize the following predictive model of saliency:

$$p(\boldsymbol{s}_T) \triangleq \delta(\boldsymbol{s}_0) \tag{9}$$

where δ represents the Dirac delta function. We observe in validation that this predictive model is effective to identify the *poorest performing elements*.[6] Thus, we may prune at initialization by solving the following optimization problem:

$$\max_{\boldsymbol{m}_0 \in \{0,1\}^M} \sum_{a=1}^{M} \boldsymbol{m}_0 \cdot \boldsymbol{s}_0 \quad \text{s.t.} \quad ||\boldsymbol{m}_0||_0 \leq B_0 \tag{10}$$

where $B_0 \geq B_s$ and is chosen to yield $10\% - 20\%$ training cost overhead over pruning at initialization. Consequently, pruning at initialization is used as a permissive heuristic to determine $\boldsymbol{m}_0$, with the remainder of the pruning decisions made using BEP as in Algorithm 1. This fusion of techniques, termed BEP-LITE, significantly reduces training cost without adversely affecting test performance (Section 5) as BEP is utilized to make difficult pruning decisions by appropriately balancing training cost vs. performance.

4.5 Dynamic penalty scaling

In BEP, each optimization problem $\hat{\rho}_t(\cdot)$ requires a corresponding Lagrange multiplier λ_t. This necessitates several hyperparameters, one for each pruning

[6] See Section 5.1.3 for verification.

Algorithm 1: Bayesian Early Pruning

```
Require: $\mathcal{N}$, $\boldsymbol{v}_1$, $T$, $B_{1,c}$, $B_s$, $\lambda$, (LITE, $B_0$) ▷ DNN $\mathcal{N}$, Lagrangian penalties $\lambda$
1: if LITE then                                                         ▷ See Section 4.4.
2:     $\boldsymbol{s}_0 \leftarrow \text{evaluate}(\mathcal{N}_{\boldsymbol{v}_1})$        ▷ Evaluate saliency at initialization.
3:     $m_0 \leftarrow \arg\max_{\boldsymbol{m}_0 \in \{0,1\}^M} \sum_{a=1}^{M} \boldsymbol{m}_0 \cdot \boldsymbol{s}_0 \quad \text{s.t.} \quad ||\boldsymbol{m}_0||_0 \leq B_0$
4:     $prune(\boldsymbol{v}_1, \boldsymbol{m}_0)$
5: end if
6: $\tilde{\mathbf{s}}_{1:T_0} \leftarrow \text{train}(\mathcal{N}_{\boldsymbol{v}_1}, T_0)$   ▷ Train for $T_0$ iterations to create seed dataset.
7: $B_{T_0,c} \leftarrow B_{1,c} - T_0\, dim(\boldsymbol{v}_1)$        ▷ Track computational effort expenditure.
8: for $k \leftarrow 0, \ldots, \frac{T-T_0}{T_{step}}$; $t \leftarrow T_0 + kT_{step}$ do
9:     $\boldsymbol{\mu}_{T|1:t}, \sigma_{T|1:t} \leftarrow MOGP(\tilde{\mathbf{s}}_{1:t})$          ▷ Train and perform inference.
10:    $\boldsymbol{s}_T \leftarrow argsort(-\boldsymbol{\mu}_{T|1:t})$                  ▷ Sort descending.
11:    $\boldsymbol{m}_t \leftarrow 0^{dim(\boldsymbol{v}_t)}$                       ▷ Initial pruning mask.
12:    for $a \leftarrow \boldsymbol{s}_T^1, \ldots, \boldsymbol{s}_T^{dim(\boldsymbol{v}_t)}$ do        ▷ Consider each network element.
13:        if $B_{t,c} - (T-t)||\boldsymbol{m}_t||_0 > 0$ then
14:            $\boldsymbol{m}_t = \boldsymbol{m}_t \vee \mathbf{e}^{(a)}$
15:        else if $\Delta(a, \boldsymbol{m}_t, \tilde{\mathbf{s}}_{1:t}, B_s) \geq \lambda_t(T-t)$ then   ▷ *Utility vs. penalty.*
16:            $\boldsymbol{m}_t = \boldsymbol{m}_t \vee \mathbf{e}^{(a)}$
17:        else
18:            break
19:        end if
20:    end for
21:    $prune(\boldsymbol{v}_t, \boldsymbol{m}_t)$                          ▷ $dim(\boldsymbol{v}_t)$ is reduced here.
22:    $B_{t+T_{step},c} \leftarrow B_{t,c} - T_{step}||\boldsymbol{m}_t||_0$
23:    $\tilde{\mathbf{s}}_{t+1:t+T_{step}} \leftarrow train(\mathcal{N}_{\boldsymbol{v}_t}, T_{step})$            ▷ Continue training.
24: end for
25: return $\mathcal{N}$
```

iteration. Tuning these hyperparameters is costly, and thus undesirable. Due to this, we propose determining λ_t dynamically using a feedback loop utilizing a singular Lagrange multiplier λ.

A proportional feedback loop can be defined as follows[7]:

$$\lambda_t \triangleq \lambda + K_p \times e(t) \tag{11}$$

where $K_p \geq 0$ is a proportional constant which modulates λ_t according to a signed measure of error $e(\cdot)$ at time t. Note that $\lambda_t \geq \lambda$ as $e(t) \geq 0$, and the opposite occurs if $e(t) \leq 0$, which allows the *error* to serve as *feedback* to determine λ_t. Implicitly, λ_t asserts some control over $e(t+1)$, and thus closing the feedback loop.

[7]This approach is inspired from Proportional-Integral-Derivative (PID) controllers (Bellman, 2015), see Åström, Hägglund, Hang, and Ho (1993) for an introductory survey.

Traditional approaches to determine K_p do not work in our case as λ may vary over several orders of magnitude. Consequently, a natural choice for K_p is λ itself which preserves the same order of magnitude between K_p and λ:

$$\lambda_t = \lambda + \lambda \times e(t) = \lambda(1 + e(t)). \tag{12}$$

Here we make two decisions to adapt the above to our task. First, as λ is likely to be extremely small, we use exponentiation, as opposed to multiplication. Secondly as $\lambda \leq 1$ in practice, we use $1 - e(t)$ as an exponent:

$$\lambda_t = \lambda^\wedge \left[1 - e(t)\right] = \lambda \left[(1/\lambda)^\wedge e(t)\right]. \tag{13}$$

The above derivation is complete with our definition of $e(t)$:

$$e(t) \triangleq (T - t)||\boldsymbol{m}_t||_0 / B_{t,c} - 1. \tag{14}$$

The above determines error by the discrepancy between the anticipated compute required to complete training $(T - t)||\boldsymbol{m}_t||_0$, vs. the remaining budget $B_{t,c}$ with $e(t) = 0$ if the two are equal. This is a natural measure of feedback for λ as we expect the two to be equal if λ is serving well to *early prune* the network.

5 Experiments

This section empirically validates the efficacy of our proposed methods. In particular, we will demonstrate: (a) The effectiveness of our MOGP modelling approach at inferring future saliency measurements; (b) The early pruning performance of BEP compared to related works and the training time vs. performance trade-off; and (c) The robustness of the BEP performance in its hyperparameter tuning.

To avoid the huge cost in validating the above contributions in various settings, we first evaluate our saliency modeling approach as well as our BEP and BEP-LITE algorithms using a small-scale network: a CNN model[8] trained on the CIFAR-10, and CIFAR-100 dataset. The proposed BEP algorithm is compared with several pruning methods applied at initialization stage: (a) *Random*: Random pruning; (b) *SNIP* (Lee et al., 2019); (c) *GraSP* (C. Wang et al., 2020); (d) *PFS*: PruneFromScratch (Y. Wang et al., 2020); and (e) *EagleEye* (B. Li, Wu, Su, & Wang, 2020): A pruning-after-training approach which is applied to the initialization stage for comparison.

Then, we apply BEP and BEP-LITE to prune ResNet-50 on the ImageNet dataset and compare against related works. For our ResNet-50 we compare against (a) *IterSnip* (de Jorge et al., 2021); and (b): *SynFlow* (Tanaka et al., 2020) as well as previously mentioned related works. Our ResNet-50 validation shows that BEP is able to train networks with higher accuracy when compared to related work. Furthermore, utilizing the BEP-LITE heuristic, these networks

[8]Code is available at https://github.com/rstudio/keras/blob/master/vignettes/examples/cifar10_cnn.R

Table 2: Comparing log likelihood (standard error) of test data for independent GPs (GP) vs. MOGP with n latent functions (n-MOGP) on different size of collected saliency measurements from CIFAR-10 and CIFAR-100 training. The log likelihood are given as a multiple of -10^4 (lower is better). MOGP outperforms GP, particularly on the small dataset. Using additional latent functions improves MOGP modeling with diminishing returns. The large dataset is easier to model due to an overabundance of data, thus MOGP may show limited improvement due to task simplicity (e.g., see Lyr 3, Large dataset). Results are averaged over 20 runs. Extremely large values are due to the GP model being unable to fit the data.

CIFAR-10									
	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	1.19(0.5)	1.08(0.06)	1.07(1.07)e5	0.96(0.04)	0.93(0.03)	2.47(0.04)	0.49(0.01)	0.48(0.01)	1.33(0.02)
4-MOGP	1.15(0.05)	0.89(0.06)	2.44(0.05)	0.91(0.02)	0.80(0.03)	2.20(0.03)	0.38(0.02)	0.39(0.02)	1.25(0.02)
8-MOGP	1.09(0.04)	0.86(0.05)	2.38(0.04)	0.84(0.03)	0.78(0.03)	2.16(0.03)	0.32(0.01)	0.35(0.02)	1.20(0.02)
18-MOGP	0.97(0.04)	**0.80**(0.05)	2.33(0.04)	0.89(0.03)	0.76(0.03)	2.13(0.03)	0.31(0.01)	0.35(0.02)	1.20(0.02)
32-MOGP	**0.96**(0.06)	0.81(0.06)	**2.32**(0.04)	**0.79**(0.03)	**0.74**(0.03)	**2.13**(0.03)	**0.31**(0.01)	**0.34**(0.02)	**1.20**(0.02)

CIFAR-100									
	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(0.06)	5.7(5.7)e4	5.6(5.6)e4	0.64(0.04)	0.70(0.04)	**2.13(0.05)**	3.4(3.4)e3	0.31(0.02)	1.06(0.02)
4-MOGP	0.79(0.05)	0.98(0.12)	3.13(0.10)	0.44(0.04)	0.60(0.10)	2.29(0.06)	0.12(0.01)	0.24(0.03)	1.07(0.03)
8-MOGP	0.65(0.05)	0.89(0.11)	3.00(0.09)	0.38(0.04)	0.60(0.10)	2.20(0.06)	0.10(0.01)	0.18(0.01)	1.02(0.03)
18-MOGP	**0.62**(0.05)	**0.84**(0.11)	2.93(0.10)	0.36(0.03)	**0.56**(0.10)	2.22(0.07)	0.09(0.01)	0.18(0.01)	1.01(0.03)
32-MOGP	0.65(0.05)	0.85(0.09)	**2.89**(0.10)	**0.36**(0.03)	0.59(0.10)	2.16(0.06)	**0.09**(0.02)	**0.18**(0.01)	**1.00**(0.03)

can be trained with only a small amount of training cost overhead when compared to pruning at initialization.

In this work, we use training FLOPs to measure training cost. Due to the continued growth in training cost of DNNs, we focus on the task of pruning a large percentage of the DNN. Due to the cubic time complexity of MOGPs, we used a variational approximation (Hensman, Matthews, & Ghahramani, 2015). In all of our models, we used 60 variational inducing points per latent function. The GPflow library (Matthews et al., 2017) is used to build our MOGP models.

5.1 Small-Scale Experiments

5.1.1 Saliency Modeling Evaluation

A key assertion in our approach is the importance of capturing co-adaptation and co-evolution effects in network elements. To verify that our MOGP approach captures these effects, we compare MOGP vs. GP belief modeling with GP assumes independence in saliency measurements across network elements (i.e., $p(\boldsymbol{s}_{1:T}) \triangleq \prod_{a=1}^{M} p(\boldsymbol{s}^a_{1:T})$). To validate this assertion, we collect saliency measurements of convolutional filters and neurons (network elements) by instrumenting the training process of our 5-layer CNN on the CIFAR-10/CIFAR-100 dataset. During early pruning, the MOGP infers the future saliency of network elements given a dataset of saliency measurements up to the present. Pruning

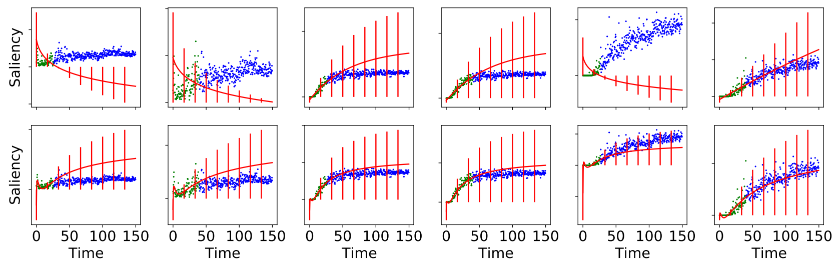

Fig. 2: Visualization of qualitative differences between GP and MOGP prediction. Top: GP. Bottom: 18-MOGP. Dataset is separated into training (green) set of observations and future saliency forms the validation (blue) set. Posterior belief of the saliency is visualized as predictive mean (red line), and 95% confidence interval (error bar).In many cases (e.g., top left graph), GP is unable to predict the long term trends of the data due to irregular element saliency observations. However, MOGP is able to overcome data irregularities by utilizing correlations between saliency of the network elements.

decisions are then made on the inferred saliency of network elements. To verify the robustness of our inferring approach we validate the MOGP approach on inferring the future saliency given past saliency measurements.[9]

We train the belief models with small ($t = [0, 26]$ epochs), medium ($t = [0, 40]$ epochs), and large ($t = [0, 75]$ epochs) training dataset of saliency measurements. For GPs, a separate model was trained per network element (convolutional filter, or neuron). For MOGP, all network elements in a single layer shared one MOGP model. We measure these models' ability in inferring the future (unobserved) saliency measurements using log likelihood and present the results in Table 2 for CIFAR-10 and CIFAR-100. As can be seen, our MOGP approach better captures the saliency of network elements than a GP approach. The log likelihood of MOGP trained using large dataset is much smaller than that trained using small dataset, which implies the necessity of collecting more saliency measurements before performing pruning during training and shows evidence of the trade-off between training cost vs. performance.

We visualize the qualitative differences between GP and MOGP prediction in Fig. 2. We observe that MOGP is able to capture the long term trend of saliency curves with significantly less data than GP. In many cases, GP is unable to predict the long term trends of the data due to irregular element saliency observations. However, MOGP is able to overcome data irregularities by utilizing correlations between saliency of the network elements.

5.1.2 Dynamic penalty scaling

We applied the early pruning algorithm on the aforementioned architecture, and training regimen. We investigated the behavior of the penalty parameter,

[9]Complete experimental setups are in Appendix F.1.

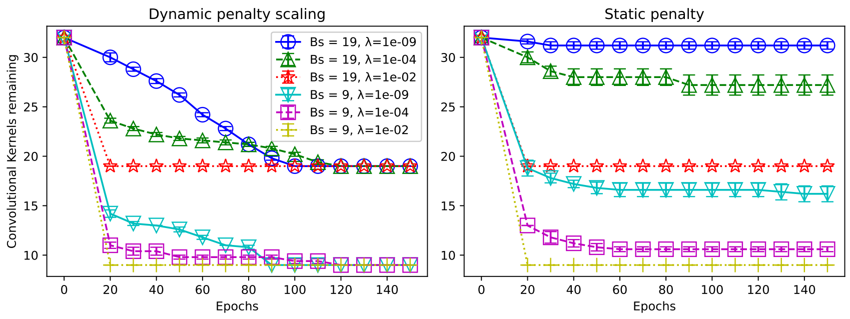

Fig. 3: Comparing dynamic penalty scaling vs. static on pruning a 32-convolutional filter layer in a CNN. Dynamic penalty scaling encourages gradual pruning across a wide variety of settings of λ.

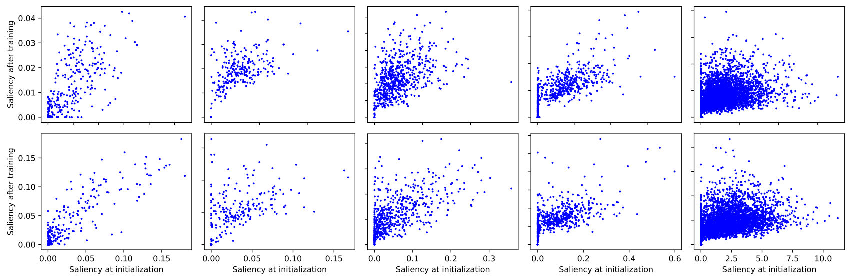

Fig. 4: Correlation between saliency of network elements at initialization, and saliency of network elements after training. Top: CIFAR-10, Bottom: CIFAR-100. Left-to-right: Layers 1 through 5 of the convolutional neural network.

λ. We observed that a singular penalty parameter does not perform gradual pruning of network elements. To rectify this issue, we used a feedback loop to determine the penalty at iteration t, λ_t dynamically. Dynamic penalty scaling[10] uses feedback from earlier pruning iterations to increase or decrease the iteration penalty at time t: $\lambda_t = \lambda\,[(1/\lambda)\hat{}\,((T-t)||\boldsymbol{m}_t||_0/B_{t,c}-1)]$. The dynamic penalty is increased if the anticipated compute required to complete training, $(T-t)||\boldsymbol{m}_t||_0$ begins to exceed the amount of compute budget remaining, $B_{t,c}$. In such case, a higher penalty is needed to satisfy the computational budget constraint as per (7). We compare dynamic penalty scaling, and penalty without scaling in Fig. 3 using $T_0 = 20$ epochs, $T_{step} = 10$ epochs for the first convolutional layer of our CNN. Dynamic penalty scaling encourages gradual pruning across a wide variety of settings of λ. We use dynamic penalty scaling in our validation.

Table 3: Performance (standard error) of the tested algorithms with varying inference FLOPs (percentage of pruned FLOPs) for CIFAR-10 and CIFAR-100. The BEP is tested with varying penalty: $\lambda = 1e{-}2$, $1e{-}4$, and $1e{-}7$. Unpruned baseline train and inference FLOPs are 1.9e13 and 2.5M, respectively.

	CIFAR-10							
	$223K$ Inference FLOPs (91%)		$95K$ Inference FLOPs (96.2%)		$24K$ Inference FLOPs (99.0%)		$6K$ Inference FLOPs (99.8%)	
	Val Acc	Train FLOPs	Val Acc	Train FLOPs	Val Acc	Train FLOPs	Val Acc	Train FLOPs
Random	72.3(0.6)%	1.7e12	66.5(0.7)%	7.1e11	41.4(7.9)%	1.8e11	14.5(4.5)%	4.6e10
SNIP	75.4(4.7)%	1.7e12	67.7(0.7)%	7.1e11	50.8(0.8)%	1.8e11	29.4(4.9)%	4.6e10
GraSP	74.6(0.6)%	1.7e12	66.5(0.9)%	7.1e11	50.7(0.6)%	1.8e11	32.9(1.0)%	4.6e10
PFS	71.3(2.0)%	1.7e12	59.9(5.8)%	7.1e11	43.3(2.5)%	1.8e11	31.7(1.7)%	4.6e10
EagleEye	60.9(7.5)%	1.7e12	44.6(11.3)%	7.1e11	47.8(7.8)%	1.8e11	28.7(4.5)%	4.6e10
BEP $1e-2$	75.9(0.3)%	4.0e12(9.5e10)	69.7(0.4)%	3.1e12(2.1e9)	54.8(1.0)%	2.6e12(4.1e9)	18.9(5.4)%	2.5e12(2.2e10)
BEP $1e-4$	75.4(1.7)%	4.3e12(3.7e10)	70.5(3.2)%	3.3e12(3.3e10)	55.7(0.9)%	2.6e12(8.4e9)	**36.1**(**1.1**)%	2.5e12(2.0e9)
BEP $1e-7$	**76.0**(**0.1**)%	4.5e12(1.4e11)	**70.6**(**0.2**)%	3.4e12(6.6e10)	**56.2**(**0.4**)%	2.7e12(1.4e10)	30.4(5.1)%	2.5e12(2.2e9)

	CIFAR-100							
	$251K$ Inference FLOPs (89.4%)		$113K$ Inference FLOPs (95.7%)		$33K$ Inference FLOPs (98.8%)		$10K$ Inference FLOPs (99.6%)	
	Val Acc	Train FLOPs	Val Acc	Train FLOPs	Val Acc	Train FLOPs	Val Acc	Train FLOPs
Random	27.8(6.7)%	1.9e12	27.1(0.7)%	8.5e12	3.7(2.7)%	2.5e11	1.0(0.0)%	8.0e10
SNIP	22.9(9.0)%	1.9e12	15.7(6.1)%	8.5e12	9.0(3.7)%	2.5e11	2.2(1.2)%	8.0e10
GraSP	28.4(7.0)%	1.9e12	22.6(5.4)%	8.5e12	13.9(3.2)%	2.5e11	1.0(0.0)%	8.0e10
PFS	37.3(0.9)%	1.9e12	26.9(4.0)%	8.5e12	19.3(2.4)%	2.5e11	8.5(0.7)%	8.0e10
EagleEye	19.8(12.0)%	1.9e12	20.2(7.1)%	8.5e12	12.6(2.6)%	2.5e11	4.7(2.2)%	8.0e10
BEP $1e-2$	40.6(0.2)%	4.2e12(4.8e9)	32.2(0.6)%	3.3e12(2.2e9)	19.1(0.5)%	2.8e12(4.3e8)	7.1(1.6)%	2.7e12(1.3e9)
BEP $1e-4$	**41.3**(**0.3**)%	4.6e12(3.7e10)	**32.4**(**0.3**)%	3.5e12(2.3e10)	**19.7**(**0.8**)%	2.9e12(6.5e10)	**8.5**(**0.8**)%	2.7e12(4.7e10)
BEP $1e-7$	40.6(0.2)%	4.8e12(1.0e11)	33.0(0.5)%	3.5e12(5.9e10)	19.5(0.5)%	2.9e12(1.2e10)	6.6(1.5)%	2.7e12(5.2e9)

5.1.3 BEP-LITE heuristic

For BEP-LITE we utilize the following predictive model of saliency

$$p(\boldsymbol{s}_T) \triangleq \delta(\boldsymbol{s}_0) \tag{15}$$

where δ represents the Dirac delta function. To verify the effectiveness of this model as a *permissive heuristic* for BEP, we plot the relation between saliency at initialization, and after training.

Using the same experimental setup as Section 5.1.1, we plot the saliency measurement collected at initialization, and after training completion. This is presented in Figure 4.

As can be seen, saliency at initialization well correlates with saliency after training, hence demonstrating the validity of our heuristic. Following this observation, we utilize the above predictive model as a permissive heuristic applied at initialization to speed up the BEP algorithm. Due to the relatively better performance offered by GraSP (C. Wang et al., 2020), we use it as the saliency measurement at initialization.

5.1.4 BEP on CIFAR-10/CIFAR-100 dataset

We apply the tested algorithms to prune a portion of filters/neurons of each layer[11] and evaluate their performance with various degrees of pruning percentage.

[10]Further details can be found in Appendix 4.5.

[11]We do pruning *per layer* instead of across the whole network since the saliency measurement has been known to not work well in comparing network element efficacy across layers (see Appendix A.1 and A.2 of (Molchanov et al., 2017) and Section 3 of (C. Wang et al., 2020)). Developing novel saliency functions which overcome this shortcoming is outside the scope of this work.

Table 4: Ablation study showing validation accuracy (standard error) with varying early pruning hyperparameters: MOGP variational inducing points (Ind. pnts.), MOGP latent functions (Lat. func.), and T_{step}. Default setting for hyperparameters are 60, 1.0×, and 10 respectively. Outside of the highest sparsity setting ($6K$ Inf. FLOPs), the validation accuracy of DNN is robust to changes of all hyperparameters, with mild degradation observed in the extremal settings.

		CIFAR-10			CIFAR-100		
		$95K$ Inf.	$24K$ Inf.	$6K$ Inf.	$95K$ Inf.	$24K$ Inf.	$6K$ Inf.
Ind. pnts.	1	70.2(0.3)%	55.6(0.3)%	32.1(0.5)%	31.7(0.6)%	19.1(0.2)%	6.6(1.2)%
	2	70.4(0.2)%	55.8(0.3)%	35.4(0.2)%	33.3(0.4)%	18.8(0.2)%	8.0(0.2)%
	5	70.5(0.2)%	56.3(0.2)%	35.9(0.2)%	32.9(0.2)%	19.3(0.3)%	4.7(1.3)%
	10	70.4(0.5)%	56.1(0.4)%	30.6(4.7)%	32.6(0.4)%	19.1(0.6)%	7.4(1.5)%
	26	69.6(0.4)%	56.8(0.2)%	29.7(4.9)%	31.7(0.6)%	19.2(0.5)%	8.3(0.7)%
	40	70.9(0.1)%	55.6(0.6)%	30.5(5.1)%	32.3(0.7)%	19.6(0.3)%	6.6(1.4)%
	60	70.5(3.2)%	55.7(0.9)%	36.1(1.1)%	32.4(0.3)%	19.7(0.8)%	8.5(0.8)%
	90	70.4(0.3)%	55.1(0.7)%	35.5(1.9)%	32.6(0.4)%	18.5(0.6)%	8.7(0.3)%
Lat. func.	0.10×	70.1(0.3)%	55.3(0.7)%	30.9(4.7)%	31.8(0.4)%	20.2(0.2)%	5.9(1.8)%
	0.15×	70.6(0.2)%	55.1(0.4)%	25.9(5.8)%	32.1(0.3)%	20.2(0.2)%	6.5(1.3)%
	0.20×	69.8(0.1)%	56.0(0.3)%	20.4(5.7)%	33.3(0.2)%	19.3(0.5)%	4.7(1.3)%
	0.25×	70.4(0.4)%	55.6(0.8)%	35.8(0.2)%	32.6(0.3)%	16.3(3.8)%	7.4(1.8)%
	0.50×	70.0(0.2)%	56.9(0.4)%	34.5(0.6)%	32.1(0.5)%	18.9(0.7)%	7.0(1.5)%
	1.0×	70.5(3.2)%	55.7(0.9)%	36.1(1.1)%	32.4(0.3)%	19.7(0.8)%	8.5(0.8)%
	2.0×	69.8(0.3)%	55.7(0.7)%	34.8(0.5)%	32.0(0.4)%	20.8(0.2)%	7.7(0.4)%
T_{step}	2	69.2(0.5)%	54.7(0.6)%	29.4(5.0)%	32.1(0.2)%	20.0(0.3)%	4.3(1.5)%
	5	70.3(0.2)%	55.6(0.5)%	31.6(5.4)%	32.7(0.4)%	19.4(0.4)%	5.2(1.8)%
	10	70.5(3.2)%	55.7(0.9)%	36.1(1.1)%	32.4(0.3)%	19.7(0.4)%	8.5(0.8)%
	20	70.3(0.2)%	56.2(0.1)%	29.8(5.0)%	32.8(0.5)%	19.6(0.4)%	6.8(1.5)%
	40	70.8(0.3)%	55.5(0.6)%	34.6(0.5)%	32.8(0.2)%	18.9(0.4)%	8.3(0.5)%
	80	72.0(0.3)%	58.4(1.8)%	39.2(6.6)%	33.9(0.5)%	22.3(0.7)%	9.5(0.8)%
	100	74.3(0.4)%	62.4(0.6)%	36.7(6.4)%	37.1(0.2)%	24.6(0.4)%	9.4(2.0)%

As shown in Table 3, our approach better preserves performance at equivalent pruning percentage. A lower penalty λ yields higher performing results but larger training FLOPs, which shows that λ in BEP serves well at balancing performance vs. computational cost. A clearer superiority of BEP in validation accuracy can be observed when the pruning percentage is large (i.e., right column of Table 3). Although PruneFromScratch (Y. Wang et al., 2020) demonstrates comparable performance to BEP in some cases, we show in more complex experiments (Section 5.2) a significant performance gap emerges. Although BEP incurs larger training FLOPs than other tested algorithms, we can further reduce the training cost via BEP-LITE as will be shown in Section 5.2. EagleEye achieves much lower validation accuracy than other tested algorithms, which implies that an *after training* pruning technique typically doesn't work well when applied to the initialization stage for reducing training cost.

Table 5: BEP and BEP-LITE vs. related work for ResNet-50 on ImageNet dataset. We vary the inference FLOPs (percentage of pruned FLOPs) of the model after training. 'Model' refers to all inclusive overhead of the pruning algorithm. PFS algorithm failed to prune to the desired sparsity in the unlisted scenarios. Unpruned baseline train and inference FLOPs are 9.3e17 and 7.3e9, respectively.

	ResNet-50											
	9.7e8 Inf. FLOPs (86.7%)			4.4e8 Inf. FLOPs (94.0%)			4.2e8 Inf. FLOPs (94.3%)			1.7e8 Inf. FLOPs (97.7%)		
	Acc	Train FLOPs	Model	Acc	Train FLOPs	Model	Acc	Train FLOPs	Model	Acc	Train FLOPs	Model
Random	69.2%	1.2e17	0.0h	51.6%	5.7e16	0.0h	35.7%	5.4e16	0.0h	34.3%	2.3e16	0.0h
SNIP	69.3%	1.2e17	0.7h	50.9%	5.7e16	0.7h	35.1%	5.4e16	0.7h	31.8%	2.3e16	0.7h
GraSP	69.3%	1.2e17	2.7h	52.2%	5.7e16	2.7h	36.7%	5.4e16	2.7h	34.1%	2.3e16	2.7h
IterSNIP	0.4%	1.2e17	1.4h	0.4%	5.7e16	1.4h	0.5%	5.4e16	1.4h	0.4%	2.3e16	1.4h
SynFlow	32.3%	1.2e17	1.2h	0.3%	5.7e16	1.2h	0.1%	5.4e16	1.2h	0.1%	2.3e16	1.2h
PFS	60.3%	1.2e17	1.6h	-	-	-	-	-	-	-	-	-
EagleEye	26.1%	1.2e17	18h	36.6%	5.7e16	18h	24.1%	5.4e16	18h	26.6%	2.3e16	18h
BEP-LITE 1e − 4	69.7%	1.4e17	2.6h	**53.7**%	6.6e16	2.9h	39.5%	6.2e16	2.8h	**37.0**%	2.6e16	2.4h
BEP 1e − 4	**70.0**%	2.2e17	1.9h	53.5%	1.7e17	2.4h	**40.0**%	1.6e17	2.3h	36.1%	1.3e17	1.6h
	ResNet-50 (Compare with PruneTrain)											
	1.4e9 Inf. FLOPs (80.7%)			5.4e8 Inf. FLOPs (92.6%)			1.3e8 Inf. FLOPs (98.2%)			3.0e7 Inf. FLOPs (99.6%)		
PruneTrain	69.2%	2.9e17	0.0h	60.6%	2.0e17	0.0h	40.6%	7.2e16	0.0h	8.3%	5.8e16	0.0h
BEP-LITE 1e − 4	71.4%	1.4e17	2.4h	66.3%	6.6e16	2.9h	**53.8**%	6.2e16	2.6h	**20.6**%	2.6e16	1.9h
BEP 1e − 4	**71.6**%	2.4e17	2.8h	**66.8**%	1.7e17	3.0h	53.6%	1.6e17	1.9h	20.6%	1.3e17	1.7h

5.1.5 Ablation Study

The objective of BEP is to reduce the cost for DNN training. As such, hyperparameter tuning of BEP on a per DNN architecture basis is not feasible due to its expensiveness. Thus, we check the robustness of BEP and MOGP hyperparameters in this section, which demonstrates that tuning is not necessary on a per DNN basis.

We vary the number of MOGP variational inducing points, MOGP latent functions as well as T_{step}, and test the performance of BEP 1e − 4 on CIFAR-10/CIFAR-100 at $95K$, $24K$, and $6K$ inference FLOPs as shown in Table 4. We observe that in general, the validation accuracy of the pruned DNN is robust to the changes of all hyperparameters. Degradation is observed in extremal hyperparameter settings. Reducing the inducing points, and latent functions has a strong effect on the effectiveness of the algorithm in the extremal setting (e.g., 6K Inf. FLOPS and minimal inducing points or latent functions). However, this can be easily avoided in practice. The hyperparameter robustness in our approach demonstrates the feasibility of applying BEP to "never-before-seen" network architectures and datasets without additional hyperparameter tuning.

5.2 ResNet Early Pruning

We train ResNet-50 with BEP and other tested algorithm for 100 epochs on 4× Nvidia Geforce GTX 1080Ti GPUs. More experimental details can be found in Appendix F.1. We used $\lambda = 1\text{e}{-4}$ as it showed strong performance in our smaller scale experiments. As can be observed in Table 5, the proposed methods achieve higher validation accuracy than other tested algorithms, with BEP-LITE showing only a modest 15% increase in training FLOPs over pruning at initialization. BEP-LITE achieves a 85% training cost reduction over BEP for 1.7e8 inference FLOPs while achieving superior validation accuracy. The

Table 6: Timing evaluation. Overhead time consists of disk I/O, and image decoding. Unpruned baseline is 47h wall-clock and 31h GPU time.

		ResNet-50 (Timing)									
		86.7%	94.0%	94.3%	97.7%			86.7%	94.0%	94.3%	97.7%
BEP-LITE	GPU	12.2h	6.9h	6.4h	3.8h	BEP	GPU	14.6h	9.2h	9.2h	7.1h
	Wall-clock	27.8h	22.6h	22.4h	19.4h		Wall-clock	30.2h	25.2h	24.9h	22.7
	Overhead	15.6h	16.0h	15.8h	15.6h		Overhead	15.6h	15.9h	15.8h	15.6h
	Model	2.6h	2.9h	2.8h	2.4h		Model	1.9h	2.4h	2.3h	1.6h

modeling and pruning overhead of our algorithm is not larger than other tested algorithms. PruneFromScratch (Y. Wang et al., 2020) shows severe degradation when compared to BEP in the 86.7% pruned FLOPs experiment, and fails to prune altogether in higher sparsity settings. IterSnip (de Jorge et al., 2021) and SynFlow (Tanaka et al., 2020) are unable to prune effectively at high pruning ratios, with severe degradation observed in all tested scenarios. EagleEye continues showing poor performance, which demonstrates the inability of *pruning-after-training* techniques to be applied to the early pruning problem. We note that PruneTrain does not provide a mechanism to constrain the trained network size (See constraint (3b). To compare with PruneTrain, we train ResNet-50 under a varying penalty parameter (see (Lym et al., 2019) for further details). After training is completed for these networks, we train equivalent inference cost networks using BEP. In particular, the validation performance of the trained network outshines the competing approaches at larger pruning ratios. This improvement is crucial as DNNs continue to grow in size and require considerable pruning to allow training and inference on commodity hardware.

5.3 Training-time improvements and discussion

Our approach delivers training time improvements in Wall-clock time. In Table 6 we show the GPU, wall-clock, overhead, and model time for BEP and BEP-LITE on the ResNet-50 pruning tasks. GPU training time speedup is correlated with the size of the model after training completion. BEP-LITE delivers improved performance in wall-clock and GPU time. This improvement is delivered with no significant loss of performance after training when compared to BEP.

The measured wall-clock time is significantly higher than GPU time due to the disk I/O and image decoding overhead of training. The GPU time reduction is well correlated with the amount of pruning, with higher pruning yielding shorter GPU time. However, due to the constant training overhead, these improvements do not perfectly translate to wall-clock time improvements. Despite this, BEP and BEP-LITE are able to deliver significant improvements in wall-clock training time compared to the unpruned baseline of 47h. In particular, with 86.7% pruned flops, BEP-LITE shows a 40% improvement in wall-clock time with only a 5.5% drop in accuracy.

The training overhead can be significantly reduced in many ways to deliver further wall-clock time improvements. Disk I/O can be reduced by utilizing faster disks, or disk arrays for higher throughput. Image decoding overhead can be alleviated by storing predecoded files in bitmap form. These approaches can further reduce the wall-clock time of the training process. Thus our approach delivers significant, practical improvements in GPU time reduction and wall-clock time reduction. The wall-clock time reduction can be further improved with minimal effort.

6 Conclusion

This paper presents a novel efficient algorithm to perform pruning of DNN elements such as neurons, or convolutional layers *during the training process*. 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 2) 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. In particular, BEP shows strong improvement in inference performance with minimal cost overhead when a significant portion of the DNN is pruned. Moreover, the proposed BEP is robust to changes in hyperparameters (see Table 4), which demonstrates its applicability to "never-before-seen" network architectures and datasets without further hyperparameter tuning. Our approach also remains flexible to changes in saliency function, and appropriately *balances* the training cost vs. performance trade-off in DNN pruning.

References

Allen-Zhu, Z., Li, Y., Liang, Y. (2019). Learning and generalization in overparameterized neural networks, going beyond two layers. *Proc. NeurIPS* (pp. 6155–6166).

Álvarez, M.A., & Lawrence, N.D. (2011). Computationally efficient convolved multiple output Gaussian processes. *JMLR*, *12*(1), 1459–1500.

Åström, K.J., Hägglund, T., Hang, C.C., Ho, W.K. (1993). Automatic tuning and adaptation for PID controllers - A survey. *Control Engineering Practice*, *1*(4), 699–714.

Bellec, G., Kappel, D., Maass, W., Legenstein, R.A. (2018). Deep rewiring: Training very sparse deep networks. *Proc. ICLR.*

Bellman, R.E. (2015). *Adaptive control processes: A guided tour.* Princeton, New Jersey: Princeton University Press.

Buluç, A., & Gilbert, J.R. (2008). Challenges and advances in parallel sparse matrix-matrix multiplication. *Proc. ICCP* (pp. 503–510).

Courbariaux, M., Bengio, Y., David, J. (2015). *BinaryConnect: Training deep neural networks with binary weights during propagations* (arXiv:1511.00363).

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

de Jorge, P., Sanyal, A., Behl, H.S., Torr, P.H.S., Rogez, G., Dokania, P.K. (2021). Progressive skeletonization: Trimming more fat from a network at initialization. *Proc. ICLR.*

Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. *Proc. NeurIPS* (pp. 1269–1277).

Dettmers, T., & Zettlemoyer, L. (2019). *Sparse networks from scratch: Faster training without losing performance* (arXiv:1907.04840).

Dong, X., Chen, S., Pan, S.J. (2017). Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Proc. NeurIPS* (pp. 4857–4867).

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *Proc. ICLR.*

Gale, T., Elsen, E., Hooker, S. (2019). *The state of sparsity in deep neural networks* (arXiv:1902.09574).

Guo, Y., Yao, A., Chen, Y. (2016). Dynamic network surgery for efficient DNNs. *Proc. NeurIPS* (pp. 1379–1387).

Han, S., Pool, J., Tran, J., Dally, W. (2015). Learning both weights and connections for efficient neural networks. *Proc. NeurIPS* (pp. 1135–1143).

Hassibi, B., & Stork, D.G. (1992). Second order derivatives for network pruning: Optimal brain surgeon. *Proc. NeurIPS* (pp. 164–171).

He, K., Zhang, X., Ren, S., Sun, J. (2016a). Deep residual learning for image recognition. *Proc. CVPR* (pp. 770–778).

He, K., Zhang, X., Ren, S., Sun, J. (2016b). Identity mappings in deep residual networks. *Proc. ECCV* (pp. 4432–4440).

He, Y., Lin, J., Liu, Z., Wang, H., Li, L., Han, S. (2018). AMC: AutoML for model compression and acceleration on mobile devices. *Proc. ECCV* (pp. 784–800).

Hensman, J., Matthews, A., Ghahramani, Z. (2015). Scalable variational Gaussian process classification. *Proc. AISTATS* (pp. 351–360).

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors* (arXiv:1207.0580).

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *JMLR*, *18*(1), 6869–6898.

Jaderberg, M., Vedaldi, A., Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *Proc. BMVC.*

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

Kingma, D.P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Proc. ICLR.*

LeCun, Y., Denker, J.S., Solla, S.A. (1989). Optimal brain damage. *Proc. NeurIPS* (pp. 598–605).

Lee, N., Ajanthan, T., Torr, P.H.S. (2019). SNIP: Single-shot network pruning based on connection sensitivity. *Proc. ICLR.*

Li, B., Wu, B., Su, J., Wang, G. (2020). EagleEye: Fast sub-net evaluation for efficient neural network pruning. *Proc. ECCV* (pp. 639–654).

Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P. (2017). Pruning filters for efficient convnets. *Proc. ICLR.*

Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., … Doermann, D. (2019). Towards optimal structured cnn pruning via generative adversarial learning. *Proc. CVPR* (pp. 2790–2799).

Liu, J., Xu, Z., Shi, R., Cheung, R.C.C., So, H.K. (2020). Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *Proc. ICLR.*

Louizos, C., Welling, M., Kingma, D.P. (2018). Learning sparse neural networks through l_0 regularization. *Proc. ICLR.*

Lu, L., Guo, M., Renals, S. (2017). Knowledge distillation for small-footprint highway networks. *Proc. ICASSP* (pp. 4820–4824).

Lym, S., Choukse, E., Zangeneh, S., Wen, W., Sanghavi, S., Erez, M. (2019). PruneTrain: Fast neural network training by dynamic sparse model reconfiguration. *Proc. SC* (pp. 1–13).

Matthews, A., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., … Hensman, J. (2017). GPflow: A Gaussian process library using tensorflow. *JMLR*, *18*(1), 1-6.

Micikevicius, P., Narang, S., Alben, J., Diamos, G.F., Elsen, E., García, D., … Wu, H. (2018). Mixed precision training. *Proc. ICLR.*

Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A. (2018). Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature*, *9*(1), 1–12.

Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J. (2017). Pruning convolutional neural networks for resource efficient inference. *Proc. ICLR.*

Mostafa, H., & Wang, X. (2019). Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *Proc. ICML* (pp. 4646–4655).

Mozer, M., & Smolensky, P. (1988). Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Proc. NeurIPS* (pp. 107–115).

Nadarajah, S., & Kotz, S. (2008). Exact distribution of the max/min of two Gaussian random variables. *Trans. VLSI*, *16*(2), 210–212.

Narang, S., Diamos, G., Sengupta, S., Elsen, E. (2017). Exploring sparsity in recurrent neural networks. *Proc. ICLR.*

Nowlan, S.J., & Hinton, G.E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, *4*(4), 473–493.

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

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

Swersky, K., Snoek, J., Adams, R.P. (2014). *Freeze-thaw Bayesian optimization* (arXiv:1406.3896).

Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. *Proc. NeurIPS.*

Tung, F., & Mori, G. (2019). Similarity-preserving knowledge distillation. *Proc. ICCV* (pp. 1365–1374).

Ullrich, K., Meeds, E., Welling, M. (2017). Soft weight-sharing for neural network compression. *Proc. ICLR.*

Wang, C., Zhang, G., Grosse, R.B. (2020). Picking winning tickets before training by preserving gradient flow. *Proc. ICLR.*

Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., Hu, X. (2020). Pruning from scratch. *Proc. AAAI* (pp. 12273–12280).

Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H. (2016). Learning structured sparsity in deep neural networks. *Proc. NeurIPS* (pp. 2074–2082).

Yang, C., Buluç, A., Owens, J.D. (2018). Design principles for sparse matrix multiplication on the GPU. *Proc. Euro-Par* (pp. 672–687).

Yim, J., Joo, D., Bae, J., Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *Proc. CVPR* (pp. 7130–7138).

Appendix A Saliency function

In this work, we use a first order Taylor-series saliency function proposed by Molchanov et al. (2017). Our design (Section 4) remains 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 $[z_{\ell,c}]_{\ell=1,\dots,L}^{c=1,\dots,C_\ell}$. 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 $\boldsymbol{z}_{\ell-1} \triangleq [z_{\ell-1,c'}]_{c'=1,\dots,C_{\ell-1}}$ 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 $\ell = 1,\dots,L$. Let $\mathcal{N}_{\boldsymbol{z}_\ell:\boldsymbol{z}_{\ell'}} \triangleq \boldsymbol{z}_{\ell'} \circ,\dots,\circ \boldsymbol{z}_\ell$ denote a *partial* neural network of layers $[\ell,\dots,\ell']_{1\le\ell\le\ell'\le L}$. The Taylor-series saliency function on the convolutional filter $z_{\ell,c}$ denoted as $s([\ell,c])$ is defined[12]:

$$s([\ell,c]) \triangleq \frac{1}{D}\sum_{d=1}^{D}\left|\frac{1}{W_\ell \times H_\ell}\sum_{j=1}^{W_\ell\times H_\ell}\frac{\partial\mathcal{L}(\mathbf{P}_\ell^{(\mathbf{x}_d)}, y_d;\, \mathcal{N}_{\boldsymbol{z}_{\ell+1}:\boldsymbol{z}_L})}{\partial P_{\ell,c,j}^{(\mathbf{x}_d)}} P_{\ell,c,j}^{(\mathbf{x}_d)}\right|. \tag{A1}$$

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 $[P_{\ell,c,j}^{\mathbf{x}_d}]_{j=1,\dots,W_\ell\times H_\ell}$ interprets the output of the c-th filter in vectorized form. This function uses the 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[13] $[\mathbf{W}_{\ell+1,c',\{:,:,c\}}]_{c'=1,\dots,C_{\ell+1}}$ 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 $W_\ell = H_\ell = 1$.

Appendix B Proof of Pruning Lower Bound

We state Lemma 4 asserting the lower bound in (5).

Lemma 4 Let $\boldsymbol{m}_t \in \{0,1\}^M$ then the following holds true:

$$\begin{aligned}&\max_{\boldsymbol{m}_t}\mathbb{E}_{p(\boldsymbol{s}_{t+1}|\tilde{\mathbf{s}}_{1:t})}\left[\rho_{t+1}(\boldsymbol{m}_t, B_{t,c} - ||\boldsymbol{m}_t||_0, B_s)\right]\\ &\ge \max_{\boldsymbol{m}_t}\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\rho_T(\boldsymbol{m}_t, B_{t,c} - (T-t)||\boldsymbol{m}_t||_0, B_s)].\end{aligned} \tag{B2}$$

Proof To prove the above, we show a solution to the latter that can be transformed into an equivalent feasible solution to the former. Let

$$\boldsymbol{m}_t^* \triangleq \max_{\boldsymbol{m}_t}\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\rho_T(\boldsymbol{m}_t, B_{t,c} - (T-t)||\boldsymbol{m}_t||_0, B_s)].$$

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

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

Accordingly, we define a feasible solution for the former optimization problem:

$$\boldsymbol{m}^*_{t+1} = \boldsymbol{m}^*_{t+2} = \ldots = \boldsymbol{m}^*_T = \boldsymbol{m}^*_t.$$

Let the above serve as solutions to $\rho_{t+1}, \rho_{t+2}, \ldots, \rho_T$ satisfies the constraint of ρ_t in the former optimization problem:

$$\begin{aligned}&\rho_t(\boldsymbol{m}^*_t, B_{t,c} - ||\boldsymbol{m}^*_t||_0, B_s)\\ &= \rho_{t+1}(\boldsymbol{m}^*_t, B_{t,c} - 2||\boldsymbol{m}^*_t||_0, B_s)\\ &\qquad\vdots\\ &= \rho_T(\boldsymbol{m}^*_t, B_{t,c} - (T-t)||\boldsymbol{m}^*_t||_0, B_s)\end{aligned}$$

which completes the proof as the maximization of the former optimization can only be greater or equal to a feasible solution. □

Appendix C Proof of Lemma 1

We restate Lemma 1 for clarity.

Lemma 1 Let $\boldsymbol{m}'$, $\boldsymbol{m}'' \in \{0,1\}^M$, and $e^{(a)}$ be arbitrary M-dimensional one hot vector with $1 \leq a \leq M$ and $P(\boldsymbol{m}) \triangleq \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}, B_s)]$. We have $P(\boldsymbol{m}' \vee e^{(a)}) - P(\boldsymbol{m}') \geq P(\boldsymbol{m}'' \vee e^{(a)}) - P(\boldsymbol{m}'')$ for any $\boldsymbol{m}' \dot{\leq} \boldsymbol{m}''$, $\boldsymbol{m}' \wedge e^{(a)} = 0^M$, and $\boldsymbol{m}'' \wedge e^{(a)} = 0^M$.

Proof According to (3),

$$\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}, B_s)] = \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\max_{\boldsymbol{m}_T}\left[\boldsymbol{m}_T \cdot \tilde{\boldsymbol{s}_T},\ \text{s.t.}\ ||\boldsymbol{m}_T||_0 \leq B_s, \boldsymbol{m}_T \dot{\leq} \boldsymbol{m}\right]\right]$$

Let $\alpha(\boldsymbol{m}) \triangleq \arg\max_{\boldsymbol{m}_T}\left[\boldsymbol{m}_T \cdot \tilde{\boldsymbol{s}_T},\ \text{s.t.}\ ||\boldsymbol{m}_T||_0 \leq B_s, \boldsymbol{m}_T \dot{\leq} \boldsymbol{m}\right]$ return the optimized mask $\boldsymbol{m}_T$ given any $\boldsymbol{m}$, $\Lambda_{\boldsymbol{m}} \triangleq \min(\alpha(\boldsymbol{m}) \odot \boldsymbol{s}_T)$ be the minimal saliency of the network elements selected at iteration T for $P(\boldsymbol{m})$. Then, we have

$$\begin{aligned}P(\boldsymbol{m} \vee e^{(a)}) &= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m} \vee e^{(a)}, B_s)\right]\\ &= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m}, B_s) - \Lambda_{\boldsymbol{m}} + \max(s^a_T, \Lambda_{\boldsymbol{m}})\right]\end{aligned}$$

The second equality is due to the fact that the network element v^a_T would only replace the lowest included element in $\boldsymbol{m}_T$ in order to maximize the objective. Then,

$$\begin{aligned}&P(\boldsymbol{m} \vee e^{(a)}) - P(\boldsymbol{m})\\ &= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m}, B_s) - \Lambda_{\boldsymbol{m}} + \max(s^a_T, \Lambda_{\boldsymbol{m}})\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m}, B_s)\right]\\ &= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[-\Lambda_{\boldsymbol{m}} + \max(s^a_T, \Lambda_{\boldsymbol{m}})\right]\\ &= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\boldsymbol{s}}_{1:t})}\left[\max(s^a_T - \Lambda_{\boldsymbol{m}}, 0)\right] \qquad \text{(C3)}\end{aligned}$$

Given $\boldsymbol{m}' \dot{\leq} \boldsymbol{m}''$, we have $\Lambda_{m'} \leq \Lambda_{m''}$ since $\boldsymbol{m}_T \dot{\leq} \boldsymbol{m}$ in $\alpha(\boldsymbol{m}')$ is a tighter constraint than that in $\alpha(\boldsymbol{m}'')$. Consequently, we can get $s^a_t - \Lambda_{m'} \geq s^a_t - \Lambda_{m''}$, and thus

$$\left[P(\boldsymbol{m}' \vee e^{(a)}) - P(\boldsymbol{m}')\right] \geq \left[P(\boldsymbol{m}'' \vee e^{(a)}) - P(\boldsymbol{m}'')\right].$$

□

Appendix D Proof of Lemma 2

We restate Lemma 2 for clarity.

Lemma 2 Let $\boldsymbol{e}^{(i)}$ be a 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.* $\boldsymbol{m} \wedge (\boldsymbol{e}^{(a)} \vee \boldsymbol{e}^{(b)}) = 0^M$. Given a vector of observed saliency $\tilde{\mathbf{s}}_{1:t}$, if $\mu^a_{T|1:t} \geq \mu^b_{T|1:t}$ and $\mu^a_{T|1:t} \geq 0$, then

$$\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})] \leq \mu^b_{T|1:t}\ \Phi(\nu/\theta) + \theta\ \phi(\nu/\theta)$$

where $\theta \triangleq \sqrt{\sigma^{aa}_{T|1:t} + \sigma^{bb}_{T|1:t} - 2\sigma^{ab}_{T|1:t}}$, $\nu \triangleq \mu^b_{T|1:t} - \mu^a_{T|1:t}$, and Φ and ϕ are standard normal CDF and PDF, respectively.

To prove this Lemma, we prove the following first:

Lemma 5 $\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})\right] \leq \mathbb{E}[\max(s^b_T - s^a_T, 0)]$.

Proof Due to (C3), we have

$$\begin{aligned}
&\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(b)})\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\hat{\rho}_T(\boldsymbol{m} \vee \boldsymbol{e}^{(a)})\right] \\
&= P(\boldsymbol{m} \vee e^{(b)}) - P(\boldsymbol{m}) - (P(\boldsymbol{m} \vee e^{(a)}) - P(\boldsymbol{m})) \\
&= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\max(s^b_T - \Lambda_{\boldsymbol{m}}, 0)\right] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\max(s^a_T - \Lambda_{\boldsymbol{m}}, 0)\right] \\
&= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\max(s^b_T - \Lambda_{\boldsymbol{m}}, 0) - \max(s^a_T - \Lambda_{\boldsymbol{m}}, 0)\right] && \text{(D4)} \\
&= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\max(s^b_T - s^a_T, \Lambda_{\boldsymbol{m}} - s^a_T) - \max(0, \Lambda_{\boldsymbol{m}} - s^a_T)\right] && \text{(D5)} \\
&\leq \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}\left[\max(s^b_T - s^a_T, 0)\right] && \text{(D6)}
\end{aligned}$$

The equality (D5) is achieved by adding $\Lambda_{\boldsymbol{m}} - s^a_T$ in each term of the two max functions in (D4). The inequality (D6) can be proved by considering the following two cases:

If $\Lambda_{\boldsymbol{m}} - s^a_T \geq 0$, then

$$\begin{aligned}
&\max(s^b_T - s^a_T, \Lambda_{\boldsymbol{m}} - s^a_T) - \max(0, \Lambda_{\boldsymbol{m}} - s^a_T) \\
&= \max(s^b_T - s^a_T, \Lambda_{\boldsymbol{m}} - s^a_T) - (\Lambda_{\boldsymbol{m}} - s^a_T) \\
&= \max(s^b_T - s^a_T - (\Lambda_{\boldsymbol{m}} - s^a_T), 0) \\
&\leq \max(s^b_T - s^a_T, 0)\ .
\end{aligned}$$

If $\Lambda_{\boldsymbol{m}} - s^a_T < 0$, then

$$\begin{aligned}
&\max(s^b_T - s^a_T, \Lambda_{\boldsymbol{m}} - s^a_T) - \max(0, \Lambda_{\boldsymbol{m}} - s^a_T) \\
&= \max(s^b_T - s^a_T, \Lambda_{\boldsymbol{m}} - s^a_T) \\
&\leq \max(s^b_T - s^a_T, 0)\ .
\end{aligned}$$

□

Next we utilize a well known bound regarding the maximum of two Gaussian random variables (Nadarajah & Kotz, 2008), which we restate:

Lemma 6 Let s^a, s^b be Gaussian random variables with means μ^a, μ^b and standard deviations σ^a, σ^b, then $\mathbb{E}[\max(s^a, s^b)] \leq \mu^a \Phi(\frac{\mu^b - \mu^a}{\theta}) + \mu^b \Phi(\frac{\mu^b - \mu^a}{\theta}) + \theta\phi(\frac{\mu^b - \mu^a}{\theta})$ where $\theta \triangleq \sqrt{[\sigma^b]^2 + [\sigma^a]^2 - 2\text{cov}(s^b, s^a)}$ and Φ, ϕ are standard normal CDF and PDF respectively.

Then,

$$\begin{aligned}
&\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\max(s^b_T - s^a_T, 0)] \\
&= \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\max(s^b_T, s^a_T)] - \mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[s^a_T] \\
&\leq (\mu^b_{T|1:t} + \mu^a_{T|1:t})\Phi(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta}) + \theta\phi(\frac{\mu^b_{T|1:t}\mu^a_{T|1:t}}{\theta}) - \mu^a_{T|1:t} \\
&= \mu^b_{T|1:t}\Phi(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta}) + \theta\phi(\frac{\mu^b_{T|1:t}\mu^a_{T|1:t}}{\theta}) + \mu^a_{T|1:t}\left(\Phi\left(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta}\right) - 1\right) \\
&\leq \mu^b_{T|1:t}\Phi(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta}) + \theta\phi(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta})
\end{aligned}$$

The first inequality follows from Lemma 6. The second inequality is due to $\Phi(\frac{\mu^b_{T|1:t} - \mu^a_{T|1:t}}{\theta}) \leq 1$ and $\mu^a_{T|1:t} \geq 0$.

Appendix E Proof of Lemma 3

We restate Lemma 3 for clarity.

Lemma 3 Let $\boldsymbol{e}^{(*)}$ represent a pruned element at time t with the highest predictive mean $\mu^*_{T|1:t} \geq 0$. Given an arbitrary pruned element $\boldsymbol{e}^{(a)}$ at time t, then for all $\delta \in (0, 1)$ the following holds:

$$p\left(\hat{\rho}_T(\boldsymbol{e}^{(a)} \vee \boldsymbol{m}_t, B_s) - \hat{\rho}_T(\boldsymbol{m}_t, B_s) < \frac{\lambda_t}{\delta}(T - t + \epsilon)\right) > 1 - \delta$$

where $\epsilon \triangleq \left[\mu^a_{T|1:t}\Phi(\nu/\theta) + \theta\ \phi(\nu/\theta)\right]/\lambda_t$ with $\theta \triangleq \sqrt{\sigma^{**}_{T|1:t} + \sigma^{aa}_{T|1:t} - 2\sigma^{*a}_{T|1:t}}$, and $\nu \triangleq \mu^a_{T|1:t} - \mu^*_{T|1:t}$.

Proof The proof follows as a consequence of Lemma 2 and Markov inequality. By definition of $e^{(*)}$ being a *pruned* element with the highest $\mu^*_{T|1:t}$ according to Algorithm 1 Line 15:

$$\Delta(*, \boldsymbol{m}_t, \tilde{\mathbf{s}}_{1:t}, B_s) \leq \lambda_t(T - t).$$

By substituting the definition of Δ:

$$\mathbb{E}_{p(\boldsymbol{s}_T|\tilde{\mathbf{s}}_{1:t})}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(*)}) - \hat{\rho}_T(\boldsymbol{m}_t)] \leq \lambda_t(T - t). \tag{E7}$$

Consequently, as $\mu^*_{T|1:t} \geq \mu^a_{T|1:t}$, we can apply Lemma 2 and achieve:

$$\begin{aligned}
&\mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(a)}) - \hat{\rho}_T(\boldsymbol{m}_t)] \\
=&\mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(*)})] - \mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t)] + \mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(a)})] - \mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(*)})] \\
=&\mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(*)}) - \hat{\rho}_T(\boldsymbol{m}_t)] + \mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(a)})] - \mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(*)})] \\
\leq&\lambda_t(T-t) + \lambda_t\epsilon \\
=&\lambda_t(T-t+\epsilon)
\end{aligned}$$

where the inequality is due to (E7) and Lemma 2. The proof is complete by applying Markov's inequality:

$$\begin{aligned}
&p\left(\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(a)}) - \hat{\rho}_T(\boldsymbol{m}_t) \geq \frac{\lambda_t}{\delta}(T-t+\epsilon)\right) \\
\leq&\frac{\mathbb{E}[\hat{\rho}_T(\boldsymbol{m}_t \vee \boldsymbol{e}^{(a)}) - \hat{\rho}_T(\boldsymbol{m}_t)]}{\lambda_t(T-t+\epsilon)/\delta} \leq \frac{\lambda_t(T-t+\epsilon)}{\lambda_t(T-t+\epsilon)/\delta} = \delta\ .
\end{aligned}$$

Observing the negation of the above yields the desired result. □

Appendix F Experimental details

F.1 Experimental details

To train our CIFAR-10 and CIFAR-100 models we used an Adam optimizer (Kingma & Ba, 2015) with an initial learning rate of 0.001. The learning rate used an exponential decay of $k = 0.985$, and a batch size of 32 was used. Training was paused three times evenly spaced per epoch. During this pause, we collected saliency measurements using 40% of the training dataset. This instrumentation subset was randomly select from the training dataset at initialization, and remained constant throughout the training procedure. We performed data preprocessing of saliency evaluations into a standardized $[0, 10]$ range.[14] We used (A1) to measure saliency of neurons/convolutional filters. For the convolutional layers we used 12 latent MOGP functions. For the dense layer we used 4 latent MOGP functions.

For our ResNet-50 model we used an SGD with Momentum optimizer with an initial learning rate of 0.1. The learning rate was divided by ten at $t = [30, 60, 80]$ epochs. We collected saliency data every 5 iterations of SGD, and averaged them into buckets corresponding to 625 iterations of SGD to form our dataset. We used a minimum of 10 latent functions per MOGP, however this was dynamically increased if the model couldn't fit the data up to a maximum of 15.

We sampled 10K points from our MOGP model to estimate $\Delta(\cdot)$ for CIFAR-10/CIFAR-100. For ResNet we sampled 15K points. We repeated experiments 5 times for reporting accuracy on CIFAR-10/CIFAR-100.

[14] Generally, saliency evaluations are relatively small (≤ 0.01), which leads to poor fitting models or positive log-likelihood. Precise details of our data preprocessing is in Appendix F.3.

F.2 Pruning on ResNet

ResNet architecture is composed of a sequence of residual units: $Z_\ell \triangleq \mathcal{F}(\mathbf{P}_{\ell-1}) + \mathbf{P}_{\ell-1}$, where $\mathbf{P}_{\ell-1}$ is the output of the previous residual unit $Z_{\ell-1}$ and '+' denotes elementwise addition. Internally, $\mathcal{F}$ is typically implemented as three stacked convolutional layers: $\mathcal{F}(\mathbf{P}_{\ell-1}) \triangleq [z_{\ell_3} \circ z_{\ell_2} \circ z_{\ell_1}](\mathbf{P}_{\ell-1})$ where z_{ℓ_1}, z_{ℓ_2}, z_{ℓ_3} are convolutional layers. Within this setting we consider convolutional filter pruning. Although z_{ℓ_1}, z_{ℓ_2} may be pruned using the procedure described earlier. Pruning z_{ℓ_3} requires a different procedure. Due to the direct addition of $\mathbf{P}_{\ell-1}$ to $\mathcal{F}(\mathbf{P}_{\ell-1})$, the output dimensions of $Z_{\ell-1}$ and z_{ℓ_3} must match exactly. Thus a ResNet architecture consists of sequences of residual units of length B with matching input/output dimensions: $\zeta \triangleq [Z_\ell]_{\ell=1,\ldots,B}$, s.t. $dim(\mathbf{P}_1) = dim(\mathbf{P}_2) = \ldots = dim(\mathbf{P}_B)$. We propose *group pruning* of layers $[z_{\ell_3}]_{\ell=1,\ldots,B}$ where filters are removed from all z_{ℓ_3} in a residual unit sequence in tandem. We define $\boldsymbol{s}([\zeta, c]) \triangleq \sum_{\ell=1}^{B} s([\ell_3, c])$, where $s(\cdot)$ is defined for convolutional layers as in (A1). To prune the channel c from ζ, we prune it from each layer in $[z_{\ell 3}]_{\ell=1,\ldots,B}$. Typically we pruned sequence channels less aggressively than convolutional filters as these channels feed into several convolutional layers.

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.

F.3 Data preprocessing

Our chief goal in this work is to speed up training of large-scale DNNs such as ResNet (K. He, Zhang, Ren, & Sun, 2016a, 2016b) on the ImageNet dataset. Pruning ResNet requires a careful definition of network element saliency to allow pruning of all layers. ResNet contains long sequences of *residual units* with matching number of input/output channels. The inputs of residual units are connected with *shortcut connections* (i.e., through addition) to the output of the residual unit. Due to shortcut connections, this structure requires that within a sequence of residual units, the number of inputs/output channels of all residual units must match exactly. This requires *group pruning* of residual unit channels for 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.

We followed the same data preprocessing procedure for both our small scale and ImageNet experiments. To standardize the saliency measurements for a training dataset $\tilde{\boldsymbol{s}}_{1:t}$ in our modeling experiments we clip them between 0 and an upper bound computed as follows: $ub \triangleq percentile(\tilde{\boldsymbol{s}}_{1:t}, 95) \times 1.3$. This procedure removes outliers. We used 1.3 as a multiplier, as this upper bound is used to transform test dataset as well, which may have higher saliency evaluations.

After clipping the training data, we perform a trend check for each element v^a by fitting a Linear Regression model to the data $\tilde{\boldsymbol{s}}^a_{1:t}$. For $\tilde{\boldsymbol{s}}^a_{1:t}$ with an increasing trend (i.e., the linear regression model has positive slope) we perform the transformation $\tilde{\boldsymbol{s}}^a_{1:t} = ub - \tilde{\boldsymbol{s}}^a_{1:t}$. The reasoning behind this is that the exponential kernel strongly prefers *decaying* curves. After this preprocessing, we scale up the saliency measurements to a $[0, 10]$ range: $\tilde{\boldsymbol{s}}_{1:t} = \tilde{\boldsymbol{s}}_{1:t} \times 10$. We found that without scaling to larger values, log-likelihood of our models demonstrated extremely high positive values due to small values of unscaled saliency measurements.

We transform the test data in our modeling experiments $\tilde{\boldsymbol{s}}_{t+1:T}$ with the same procedure using the same ub and per-element v^a regression models as computed by the training data. We measure log-likelihood after this transformation for both the test dataset in our small scale experiments.

During the BEP Algorithm, the same steps are followed, however we inverse the trend check transformation ($\tilde{\boldsymbol{s}}^a_{1:t} = ub - \tilde{\boldsymbol{s}}^a_{1:t}$) on the predicted MOGP distribution of $\boldsymbol{s}_T$ prior to sampling for estimation of $\Delta(\cdot)$.