AdaNS: Adaptive Non-Uniform Sampling for Automated Design of Compact DNNs

Mojan Javaheripi, *Student Member, IEEE*, Mohammad Samragh, *Student Member, IEEE*, Tara Javidi, *Senior Member, IEEE*, and Farinaz Koushanfar, *Fellow, IEEE*

***Abstract*—This paper introduces an adaptive sampling methodology for automated compression of Deep Neural Networks (DNNs) for accelerated inference on resource-constrained platforms. Modern DNN compression techniques comprise various hyperparameters that require per-layer customization. Our objective is to locate an *optimal* hyperparameter configuration that leads to lowest model complexity while adhering to a desired inference accuracy. We design a score function that evaluates the aforementioned *optimality*. The optimization problem is then formulated as searching for the maximizers of this score function. To this end, we devise a non-uniform adaptive sampler that aims at reconstructing the band-limited score function. We reduce the total number of required objective function evaluations by realizing a targeted sampler. We propose three adaptive sampling methodologies, i.e., *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*, where new batches of samples are chosen based on the history of previous evaluations. Our algorithms start sampling from a uniform distribution over the entire search-space and iteratively adapt the sampling distribution to achieve highest density around the function maxima. This, in turn, allows for a low-error reconstruction of the objective function around its maximizers. Our extensive evaluations corroborate *AdaNS* effectiveness by outperforming existing rule-based and Reinforcement Learning methods in terms of DNN compression rate and/or inference accuracy.**

***Index Terms*—Compact deep neural networks, adaptive sampling, automated neural network compression, hardware-aware DNN design, multi-objective optimization.**

I. Introduction

With the growing range of applications for Deep Neural Networks (DNNs), the demand for higher accuracy has led to a continuous increase in the complexity of state-of-the-art models. Such high execution cost hinders the deployment of DNNs in real-time applications on commodity hardware. Fortunately, modern neural networks have been shown to incur high redundancies that can be eliminated without compromising inference accuracy. Effective identification and removal of such redundancies has fueled a myriad of research in two interlinked domains: (i) Developing model compression techniques, e.g., pruning [1]–[9] and coding [10]. (ii) Devising automated policies that learn how to configure compression techniques to simultaneously achieve accuracy and compactness [10]–[15]. In this paper, we focus on the latter.

The effectiveness of contemporary compression techniques relies on careful tuning of several hyperparameters across DNN layers, e.g., pruning rates. These hyperparameters directly control the trade-off between accuracy and execution cost on a constrained device. The question to be answered is how to find an optimal hyperparameter configuration that results in a high compression rate while minimally affecting inference accuracy. Existing research in automated compression suggests the use of heuristic methods [2]–[4] or Reinforcement Learning (RL) [12], [13], [16]. To tackle the high-dimensionality of the search space, heuristics and RL-based algorithms specify the hyperparameters one layer at a time. One downside of such approach is the need for many learning episodes to enable identification of the inherent inter-layer correlations. This, in turn, results in a rather slow convergence to the optimal hyperparameter solution.

We propose an alternative approach that *simultaneously* tunes the compression hyperparameters for all DNN layers by encapsulating them as *one* fixed-length vector $\vec{x} \in \mathbb{R}^d$. Each hyperparameter vector translates to a unique compressed DNN by leveraging the transformations suggested in existing DNN compression techniques. In this setting, the search for optimal compression hyperparameters directly translates to optimizing an objective function $f(\vec{x}) : \mathbb{R}^d \to \mathbb{R}$ over $\vec{x}$ where $f(\cdot)$ is an arbitrary measure of quality. We suggest a unified formulation for $f(\cdot)$, called the *score*, which assesses the quality of $\vec{x}$ by combining inference accuracy and a desired execution cost.

One major challenge in maximizing $f(\cdot)$ is that the underlying algebraic model that relates compression hyperparameters to inference accuracy (and possibly the execution cost) is not known. In other words, one can evaluate the pertinent objective function $f(\vec{x})$ for any arbitrary input $\vec{x}$ but does not have access to other information such as the gradient $\frac{\partial f}{\partial x}$. Therefore, numerical optimization methods such as stochastic gradient descent are not directly applicable. Furthermore, since the vectorized search-space grows exponentially with number of DNN layers, brute-force search is not feasible, and more intelligent approaches are required to find the optimal solution.

We resort to empirical evaluations of $f(\cdot)$ and propose *AdaNS*, a non-uniform adaptive sampling methodology that aims at

Manuscript received July 2, 2019; revised November 8, 2019 and April 12, 2020; accepted April 22, 2020. Date of publication May 4, 2020; date of current version August 10, 2020. This work was supported in part by Qualcomm Innovation Fellowship QIF-2019-US and in part by industrial partners of UCSD Center for Machine Integrated Computing and Security (MICS). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Diana Marculescu. *(Corresponding author: Mojan Javaheripi.)*

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: mojan@ucsd.edu; msamragh@ucsd.edu; tjavidi@ucsd.edu; farinaz@ucsd.edu).

Digital Object Identifier 10.1109/JSTSP.2020.2992384

reconstructing the opaque objective function $f(.)$ around its maxima. Different from classic sampling (which accurately reconstructs the function over the entire input space), our sampler's reconstruction objective focuses more on regions of interest, i.e., the maximizers of $f(.)$. To the best of our knowledge, this is the first time that designing compact DNNs, a topic often viewed in computer engineering, has been framed in the context of adaptive sampling, an active area of research in signal processing.

To reduce the total number of required function evaluations while enabling a low-error reconstruction around the maxima, we choose our samples following two incentives: (i) we should sample from likely maximas to reach the optimization goal, and (ii) samples should be drawn from unexplored regions where we have a high reconstruction error (uncertainty). We satisfy the above two properties by realizing targeted sampling. Our proposed algorithm is an iterative process, where a batch of samples (hyperparameter vectors) are obtained and evaluated at each phase. The sampling distribution over the input space for each phase is then refined adaptively based on prior observations before sampling the next batch of hyperparameter vectors. *AdaNS* exploits parallelism to reduce optimization time by concurrent sample evaluations. We devise three adaptive sampling subroutines, namely, *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*, each of which incorporates a different (posterior) sampling distribution.

We study the impact of the sampling strategy on the convergence rate and the maximal returned value. Our empirical evaluations show that *AdaNS-Gaussian* achieves higher values of $f(\cdot)$ with a lower number of function evaluations. To demonstrate *AdaNS* generalizability, we apply it to optimization tasks of various complexity. Our experiments show that *AdaNS* can learn near-optimal hyperparameters in very high-dimensional search-spaces; to the best of knowledge, *AdaNS* is the only black-box optimization method shown to tackle search-space sizes as high as 10^{132}. We show the superiority of *AdaNS* over prior work in RL [12], Bayesian optimization [17], expert-designed architectures [18]–[20], and heuristic methods [1]–[4], [6]–[9], [21]–[24]. We unveil the full potential of *AdaNS* by learning to effectively combine multiple methods: for VGG-16 on ImageNet, *AdaNS* pushes the state-of-the-art FLOPs reduction from $5\times$ to $7.1\times$ with higher accuracy. For compact MobileNets, *AdaNS* obtains on average 1.2% higher top-1 accuracy than the MobileNet Pareto curve. We further show that *AdaNS* is highly scalable and enjoys a linear search speedup with number of distributed computing resources.

Main Contributions:

- We devise three adaptive sampling strategies, i.e., *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*, that search for the optimal hyperparameters $\vec{x} \in \mathbb{R}^d$ for DNN compression. Each strategy iteratively refines the sampling posterior distribution towards generating better samples.
- We suggest a unified formulation for the optimization objective $f(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$, i.e., the *score*. Our customized $f(\cdot)$ simultaneously incorporates accuracy and execution cost to quantitatively asses compressed DNNs.
- We propose a context-aware boundary characterization scheme that prevents sampling from regions that are unlikely to contain the optimal solution.
- We connect *AdaNS* to function reconstruction and sampling theory. Specifically, we devise a non-uniform reconstruction scheme and empirically show that *AdaNS* reduces the reconstruction error around the maxima.

II. Background and Related Work

Automated Policy Making: Designing automated methodologies for achieving compact and accurate neural networks has been the focus of recent work [10]–[12], [25], [26]. In order to achieve high-accuracy and low-complexity neural networks, genetic algorithms have been applied to Neural Architecture Search (NAS) [24] and DNN compression [27]. Reinforcement Learning (RL) [11], [12] is another promising tool for DNN compression. Although effective in finding near-optimal solutions, RL relies on gradient-based training, which can lead to a high computational burden and a slow convergence. Also, RL is not scalable in large continuous action-spaces [23], [28].

In this paper, we develop a multi-objective optimization method based on an adaptive sampling strategy. *AdaNS* is dimensionality-agnostic and can scale well to large search-spaces. This, in turn, allows for simultaneous optimization of all layers' hyperparameters. Our solution offers several benefits: (1) it is inexpensive to implement since it does not involve gradient-based algorithms. (2) Unlike RL algorithms that are inherently sequential, *AdaNS* is highly parallelizable and can offer scalability in distributed settings. (3) *AdaNS* can support both continuous and sparse-valued objective functions. (4) *AdaNS* is the only known method to-date that optimizes *heterogeneous* parameters in search spaces as large as 10^{132}. Prior work either constrain the search-space size, e.g., [12], [17] or are designed specifically for one compression task, e.g., [20].

Sampling: Our approach to DNN compression is loosely related to three classical problems in the literature: Bayesian empirical optimization [17], [29], [30], spectral methods [31], and adaptive sampling theory [32]. Bayesian empirical optimization provides a method for hyperparameter tuning by assuming a prior distribution for the score function and then updating this prior based on new observations. However, this approach depends on the availability of reliable Bayesian priors which, in some regimes, might not be realistic. *AdaNS* does not make any probabilistic assumptions about the objective function and is compatible with arbitrary $f(\cdot)$. Furthermore, Bandit methods such as Bayesian optimization are inherently sequential and difficult to parallelize while our batch implementation allows parallelism in distributed settings.

Spectral methods, when applied to empirical optimization, leverage the structure of the score function together with known results from Fourier analysis and compressed sensing. The main drawback of spectral approaches is their reliance on prior knowledge about the sparse structure of the score function, which cannot be immediately assumed in the context of DNN compression. Adaptive sampling methods are generally well-suited

to scenarios where the structure of the score function is unknown but has local features. In such scenarios, sample spacing can be controlled using the observed values of the variable of interest. Although adaptive sampling is amenable to many fields of research, little is concretely known about its optimal strategies.

III. Problem Formulation

DNN compression, in high-level, is a transformation $\mathcal{T}(M, \vec{x})$ that converts a pretrained model M to a compressed model $\hat{M}_{\vec{x}}$ with lower computational complexity. In this process, the adjustable hyperparameter vector $\vec{x} \in \mathbb{R}^d$ controls the complexity and accuracy of the output model. A desirable compressed network satisfies 2 properties: (i) the generalization capability of $\hat{M}_{\vec{x}}$ should resemble the original network and (ii) the execution cost of $\hat{M}_{\vec{x}}$ on the target hardware platform should be low.

We assume we have access to a scoring oracle, $f(\cdot)$, that assesses $\vec{x}$ based on its corresponding compressed model's accuracy $A(\hat{M}_{\vec{x}})$ and complexity $C(\hat{M}_{\vec{x}})$. Our objective is to empirically optimize this customized score:

$$\max_{\vec{x} \in \mathbb{R}^d} f(A(\hat{M}_{\vec{x}}), C(\hat{M}_{\vec{x}})), \tag{1}$$

For simplicity, we show $f(A(\hat{M}_{\vec{x}}), C(\hat{M}_{\vec{x}}))$ as $f(\vec{x})$ in the rest of the paper. Since full knowledge about $f(\cdot)$ and/or its first derivatives cannot be assumed, often empirical evaluations and optimization is the only viable strategy. Brute-force empirical evaluation of $f(\cdot)$ over $\vec{x} \in \mathbb{R}^d$, in general, is infeasible as the search-space grows exponentially with d. Instead, we propose an empirical zeroth order optimization based on adaptive non-uniform sampling, dubbed *AdaNS*.

In the context of DNN compression, $f(\vec{x})$ can be viewed as a band-limited signal, i.e., the corresponding Fourier transform $F(\omega)$ is contained in a frequency interval $[-B, B]$. As such, one can approximately solve the maximization problem in Eq. (1) by sampling. *AdaNS* iteratively samples the hyperparameter vector space to find an optimized compression configuration $\vec{x}^*$. We consider an adaptive sampler generating $\mathbb{S} = \mathbb{S}_1 \cup \mathbb{S}_2 \cup \cdots \cup \mathbb{S}_T$ where each $\mathbb{S}_t$ represents a fixed-sized set of b samples to be evaluated in iteration t. Let $\vec{x}^*_{\mathbb{S}}$ be the current maximizer of $f(\cdot)$ on the entire set of observed (evaluated) samples $\mathbb{S}$. Our objective is to select $\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_T$ such that $f(\vec{x}^*_{\mathbb{S}})$ is not too far from the actual function maximum f^*. To this end, *AdaNS* adopts a guided search such that samples generated at each iteration are more competent than the previously observed samples, i.e., they result in better DNNs with higher $f(\vec{x})$.

IV. AdaNS Overview

We provide a generic solution to effectively compress a pre-trained DNN while maximally preserving model accuracy. *AdaNS* automation policy acts on a pool of hyperparameters and explores the corresponding search-space using adaptive non-uniform sampling. An overview of *AdaNS* optimization is shown in Fig. 1 and summarized below:

I. First, a pre-processing step characterizes the search-space boundaries within which the optimal solution can reside. These boundaries are specified based on task-enforced constraints on inference accuracy. Using the boundaries, initial hyperparameter vectors $\mathbb{S}_1 = \{\vec{x}_1, \ldots, \vec{x}_b\}$ are sampled (Section IV-C).

II. Each iteration, newly generated samples $\vec{x} \in \mathbb{S}_t$ are translated to their compressed DNNs $\hat{M}_{\vec{x}}$ (Section IV-A). The scores $f(\vec{x})$ are then evaluated in parallel (Section IV-B).

III. Based on the knowledge acquired by new evaluations and previously observed samples, *AdaNS* adaptive sampling subroutine identifies a batch of more competent hyperparameter vectors $\mathbb{S}_{t+1}$. We propose three different subroutines that select the new set of samples, namely, *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian* (Section V).

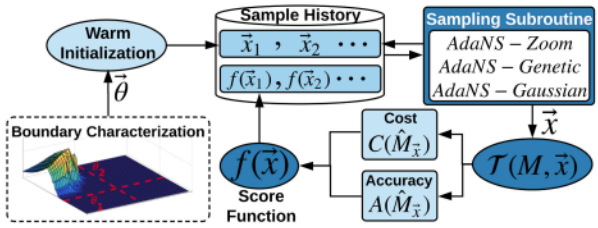

Fig. 1. Overview of *AdaNS* adaptive sampling methodology for hyperparameter customization.

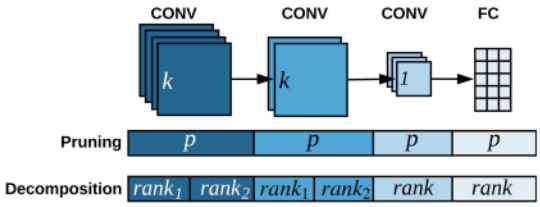

Fig. 2. Vectorized representation of an example 4-layer DNN for *Pruning* and *Decomposition*. Here, CONV and FC denote convolutional and fully-connected layers, respectively.

A. Search-Space Definition

An initial step for the application of *AdaNS* is defining the pertinent search-space for black-box optimization. To this end, we propose a vectorized representation of the hyperparameters for various compression methods. The problem of compression optimization can then be solved by performing a search over this vector-space. We focus on *four* compression tasks, namely, structured [4] and non-structured [33] Pruning, Singular Value Decomposition (SVD) [34], and Tucker-2 approximation [35].

Fig. 2 shows a high-level view of our vectorized compression hyperparameters for a 4-layer neural network. For pruning, we allocate one continuous value $p \in [0, 1]$, per layer, inside $\vec{x}$ to represent the ratio of non-zero values. We apply SVD on weight parameters of fully-connected layers and point-wise convolutions. To represent the integer-valued rank $\in \{1, \ldots, R\}$ for SVD, we allocate one continuous value $rank \in [\frac{1}{R}, 1]$ per decomposed layer to form $\vec{x}$. Tucker-2 is applied on 4-way weight tensors $W \in \mathbb{R}^{k \times k \times c \times f}$ in convolution layers where the compression parameter is a tuple of integer-valued approximation ranks. To form $\vec{x}$, we allocate two normalized and real-valued

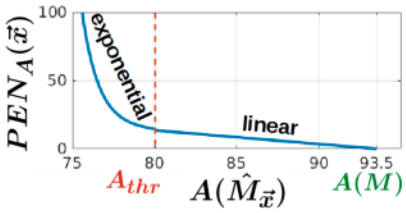

Fig. 3. *AdaNS* accuracy penalty function with $A_{thr} = 80\%$ and baseline accuracy $A(M) = 93.5\%$.

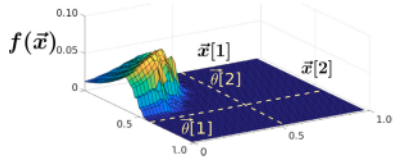

Fig. 4. Search-space for pruning a 2-layer DNN.

hyperparameters $rank_1 \in [\frac{1}{c}, 1]$ and $rank_2 \in [\frac{1}{f}, 1]$ per convolution. We have provided further details on each compression method's hyperparameters and their (unique) interpretation as compressed models, i.e., $\mathcal{T}(M, \vec{x})$, in Appendix A.

B. Scoring Mechanism

We formalize a multi-objective score $f(\vec{x})$ which simultaneously reflects the compressed DNN's accuracy and computational complexity. This score can thus be leveraged to assess the compression quality of each hyperparameter vector $\vec{x}$. To define $f(\cdot)$, let us first represent DNN compression as a constrained optimization as follows:

$$\max_{\vec{x} \in \mathbb{R}^d} \Delta C(M, \vec{x}) \quad s.t. \quad A(\hat{M}_{\vec{x}}) > A_{thr} \tag{2}$$

where $\Delta C(M, \vec{x})$ represents the normalized difference in hardware cost, e.g., FLOPs, between the uncompressed network, M, and the compressed model, $\hat{M}_{\vec{x}}$. Here, A_{thr} is a task-enforced threshold on the post-compression accuracy. Having an accuracy constraint is crucial since the optimization algorithm will converge to a model size of zero otherwise. To solve the constrained optimization problem in Eq. (2), we formulate it as a primal unconstrained optimization using penalty methods [36]:

$$\max_{\vec{x} \in \mathbb{R}^d} \Delta C(M, \vec{x}) - log(PEN_A(\vec{x}))) \tag{3}$$

where the term $log(PEN_A(\vec{x}))$ is the exterior penalty function [37] that enforces a constraint on the compressed model's accuracy, i.e., $A(\hat{M}_{\vec{x}}) > A_{thr}$. The function $PEN_A(\vec{x})$ measures the accuracy degradation as follows:

$$PEN_A(\vec{x}) = \begin{cases} \Delta A(M, \vec{x}) & A(\hat{M}_{\vec{x}}) \geq A_{thr} \\ \Delta A(M, \vec{x}) + e^{[A_{thr} - A(\hat{M}_{\vec{x}})]} & A(\hat{M}_{\vec{x}}) < A_{thr} \end{cases} \tag{4}$$

Fig. 3 visualizes the accuracy penalty. To prevent undesirable drop of accuracy, we greatly diminish the score of individuals that cause lower accuracies than the set constraint, A_{thr}. The log penalty term is estimated as follows:

$$\log(PEN_A(x)) = \begin{cases} log(\Delta A(M, \vec{x})) & A(\hat{M}_{\vec{x}}) \geq A_{thr} \\ A_{thr} - A(\hat{M}_{\vec{x}}) & A(\hat{M}_{\vec{x}}) < A_{thr} \end{cases} \tag{5}$$

For accuracy values satisfying the threshold, this term enforces the accuracy maximization objective. Applying the logarithm smoothens the accuracy variations by damping sudden changes. For accuracy values below A_{thr}, a linear penalty is applied to stop further accuracy loss. To avoid numerical instability, we define the exponential of the primal optimization in Eq. (3) as our score function:

$$f(\vec{x}) = \frac{e^{\Delta C(M, \vec{x})}}{PEN_A(\vec{x})} \tag{6}$$

Maximizing the score function of Eq. (6) is equivalent to maximizing its logarithm value in Eq. (3). To ensure efficiency, inference accuracies are measured on a small held-out portion of the training samples, dubbed *validation* set. *AdaNS* scoring function successfully models the ultimate goal of high compression with minimal accuracy degradation; it is applicable to various compression tasks and can reflect different hardware costs, e.g., power, memory, and runtime.

C. Boundary Characterization for Directed Search

A naïve initialization of samples in the first iteration can result in slow and sub-optimal convergence. We utilize boundary characterization as a pre-processing step to enable a targeted initialization. This approach eliminates unnecessary exploration of outlier subspaces, i.e., regions that are unlikely to contain the optimization solution. Inference accuracy for a compressed DNN $\hat{M}_{\vec{x}}$ is coordinate-wise monotonic with respect to per-layer compression rates: as the compression rates increase, the accuracy drops. As such, we can characterize the boundaries of $\vec{x}[i]$ on a per-layer basis, based on the accuracy threshold without performing any fine-tuning. Fig. 4 visualizes the search-space and the outlier regions for pruning a two-layer model.

The optimal solution to the search problem is a configuration with the highest score. The outlier regions in Fig. 4 are therefore the flat sectors of the space. We find a threshold vector $\vec{\theta}$ where each element constrains a single hyperparameter corresponding to the compressed DNN. In Fig. 4, $\vec{\theta}$ has two elements, each presented by a dashed line. Below we describe how the boundaries $\vec{\theta}[i]$ are obtained given an accuracy threshold A_{thr} for each compression task.

Pruning: The threshold vector elements $\vec{\theta}[i] \in [0, 1)$ specify the maximum pruning rate for layer i such that the compressed DNN's accuracy does not violate A_{thr}:

$$\vec{\theta}[i] = max\{p\} \;\; s.t. \;\; \vec{x}[j] = \begin{cases} p & j = i \\ 0 & j \neq i \end{cases} \; A(\hat{M}_{\vec{x}}) > A_{thr}$$

Fig. 5 demonstrates an example of boundary characterization for pruning a VGG network on CIFAR-10. Each curve is obtained by varying the pruning rate for one layer while no other layer is pruned. The collision between the dashed horizontal

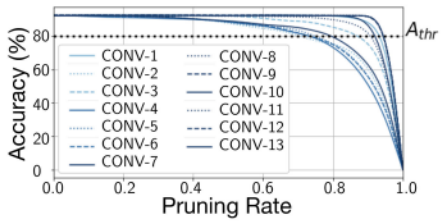

Fig. 5. Boundary characterization for pruning a VGG network trained on CIFAR-10 with $A_{thr} = 80\%$.

line, i.e., the accuracy threshold, and the i^{th} curve provides the threshold boundary θ_i.

Decomposition: For SVD and Tucker-2 decomposition, the threshold vector $\vec{\theta}$ represents per-layer minimum ranks $\vec{\theta}[i] \in (0, 1]$ satisfying A_{thr} where a normalized rank of 1 corresponds to a non-decomposed layer:

$$\vec{\theta}[i] = min\{rank\} \;\; s.t. \;\; x_j = \begin{cases} rank & j = i \\ 1 & j \neq i \end{cases} \; A(\hat{M}_{\vec{x}}) > A_{thr}$$

Note that real-world models have many more layers and the pertinent search-space is of much higher dimensionality than in Fig. 4. For a d-dimensional space, the proposed boundary characterization scheme reduces the effective (continuous) search volume from 1 to $\prod_{i=1}^{d} \vec{\theta}[i]$ for pruning and $\prod_{i=1}^{d}(1 - \vec{\theta}[i])$ for decomposition, therefore, significantly improving search convergence and solution quality.

By filtering out the non-optimal regions, *AdaNS* sampling can find the near-optimal solution within the found margins. After boundary characterization, we randomly draw the initial samples from the space enclosed by the threshold vector $\vec{\theta}$. For pruning, the i^{th} element $\vec{x}[i]$ in each sample vector is drawn from $\mathcal{N}(\vec{\theta}[i]/2, \vec{\theta}[i]/2)$. For decomposition, the i^{th} element $\vec{x}[i]$ is randomly selected from $\mathcal{U}[\vec{\theta}[i], 1]$.

D. Optimization through Adaptive Sampling

In this section, we enclose the mathematical formulation of *AdaNS* adaptive sampling and its connection to optimization. Let us consider a pool of samples $\mathbb{S} = \{\vec{x}_i\}_{i=1}^{N}$ which are generated iteratively and then evaluated via the scoring oracle. Also, let $\vec{x}^*_{\mathbb{S}}$ be the current maximizer for $f(\cdot)$, found across the observed samples. To maximize the probability of finding a near-optimal solution, we seek the following property on $\mathbb{S}$:

$$\max_{\mathbb{S}} \mathcal{P}(f(\vec{x}^*_{\mathbb{S}}) > \alpha_{max} f^*), \quad 0 < \alpha_{max} < 1 \tag{7}$$

where $\mathcal{P}(\cdot)$ denotes probability and α_{max} is a constant value called the *proximity parameter*. As $\alpha_{max} \to 1$, the sampled maxima gets closer to the real function maximizer, i.e., $f(\vec{x}^*_{\mathbb{S}}) \to f^*$. To achieve the objective of Eq. (7), we devise an adaptive sampler for generating $\mathbb{S}$ based on two core ideas:

- We choose our samples ($\mathbb{S}$) sequentially: $\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_T$[1] such that $f(\mathbb{S}_{[1:t]})$ is utilized[2] in choosing $\mathbb{S}_{t+1}$.

[1] $\mathbb{S}_t$ is the set of samples from the t^{th} iteration of *AdaNS*.
[2] We use the notation $f(\mathbb{S})$ as follows: $f(\mathbb{S}) = \{f(\vec{x}) | \vec{x} \in \mathbb{S}\}$

Algorithm 1: Overview of A*daNS* Sampling.

Inputs: Previous samples $\mathbb{S}_{[1:t]}$, fitness oracle $f(\cdot)$, batch size b, target proximity parameter α_{max}
Outputs: $\mathbb{S}_{[1:t+1]}$, $f(\mathbb{S}_{[1:t+1]})$

1: $\tilde{f}^* = \max f(\mathbb{S}_{[1:t]})$
2: $\alpha_t = \max_{\alpha \in [0, \alpha_{max}]}(\alpha) s.t. \; |\{\vec{x} \in \mathbb{S}_{[1:t]} | f(\vec{x}) > \alpha \tilde{f}^*\}| \geq b$
3: $\mathbb{S}_g = \{\vec{x} \in \mathbb{S} | f(\vec{x}) > \alpha_t \tilde{f}^*\}$
4: **procedure** SAMPLING SUBROUTINE
5: Update sampling distribution $\mathcal{G}_{t+1}(\cdot)$ based on $\mathbb{S}_g$
6: Sample $\mathbb{S}_{t+1} = \{\vec{x}_i\}_{i=1}^{b} \sim \mathcal{G}_{t+1}(\cdot)$
7: **return** $\mathbb{S}_{t+1}$
8: **end procedure**
9: $\mathbb{S}_{[1:t+1]} = \mathbb{S}_{[t+1]} \cup \mathbb{S}_{[1:t]}$
10: $f(\mathbb{S}_{[1:t+1]}) = f(\mathbb{S}_{t+1}) \cup f(\mathbb{S}_{[1:t]})$

- We allow $\mathbb{S}_1, \ldots, \mathbb{S}_T$ to be random sets. That is, instead of choosing $\mathbb{S}_{t+1}$ deterministically, we devise a sampling distribution $\mathcal{G}_{t+1}(\cdot)$ such that $\mathbb{S}_{t+1} \sim \mathcal{G}_{t+1}(\cdot)$. More precisely, $\mathbb{S}_{t+1}$ is a random draw from the conditional distribution $\mathcal{G}_{t+1|t}(\cdot | \mathbb{S}_{[1:t]})$.

Algorithm 1 presents a high-level overview of one iteration in *AdaNS* sampling methodology. At each iteration t, the current estimate of the function maxima $\tilde{f}^*$ is obtained based on the previous observations (Line 1). *AdaNS* uses a proximity parameter $\alpha_t \in [0, 1]$ to extract the set of good samples $\mathbb{S}_g$ from the history of previous observations $\mathbb{S}_{[1:t]}$ where $\mathbb{S}_g$ is the set of samples $\vec{x} \in \mathbb{S}_{[1:t]}$ with $f(\vec{x}) > \alpha_t \tilde{f}^*$ (Lines 2 & 3). A large value for the proximity parameter (≈ 1) is ideal since it allows for better identification of the function maximum. However, setting the proximity parameter to a high initial value causes the underlying sampling to become biased towards the initial good samples when the local maxima are not yet found. To mitigate this issue, we adaptively tune α_t throughout iterations, such that $\mathbb{S}_g$ has a diverse set of members, therefore balancing exploration and exploitation in the sampling distributions. At each iteration, α_t is set to the maximum value within $[0, \alpha_{max}]$ that allows $\mathbb{S}_g$ to have at least b members (Line 2). In practice, α_t becomes close to zero in the initial search iterations and gradually increases towards $\alpha_{max} \approx 0.95$ as the *AdaNS* search algorithm proceeds.

Once the set of good samples $\mathbb{S}_g$ is extracted, a sampling subroutine is applied to generate a new set of samples, $\mathbb{S}_{t+1}$. The sampling subroutine utilizes $\mathbb{S}_g$ to update the sampling distribution $\mathcal{G}_{t+1}$ from which the next batch of samples $\mathbb{S}_{t+1}$ are drawn (Lines 5 & 6). The details of *AdaNS* sampling subroutines are enclosed in Section V. The newly generated samples are then evaluated and appended to the sample history. This process is repeated until convergence, or up to a maximum number of iterations (T).

V. AdaNS Adaptive Sampling Routines

Recall that *AdaNS* iteratively establishes a set of evaluated samples, $\mathbb{S}_{[1:T]} = \mathbb{S}_1 \cup \cdots \cup \mathbb{S}_T$, such that the samples generated at each iteration are more competent than the previously

observed samples. In other words:

$$\max f(\mathbb{S}_{[1:t+1]}) \geq \max f(\mathbb{S}_{[1:t]}) \tag{8}$$

This, in turn, ensures that the evaluated samples gradually move closer to the objective function maximizer. The choice of new samples at each iteration in *AdaNS* is based on the adaptive sampling distribution $\mathcal{G}_{t+1}(\cdot)$ utilized in the sampling subroutine (see Algorithm 1). The choice of $\mathcal{G}_{t+1}(\cdot)$ directly affects optimization performance, i.e., the maximal returned value for the objective function as well as the convergence time. To investigate such effects, we design three different sampling subroutines for *AdaNS*, namely, *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*. While each subroutine incorporates a different prior for $\mathcal{G}_{t+1}(\cdot)$, all of the sampling distributions are developed based on the following key insights:

1) In the absence of prior knowledge, uniform sampling provides the most effective exploration of the functional landscape of the optimization objective $f(\cdot)$.
2) Assuming mild regularity conditions on $f(\cdot)$, it is intuitive that increasing sampling rate locally around "good samples" observed so far is more likely to result in additional (future) high-score observations.
3) Assuming mild regularity conditions on $f(\cdot)$, it is intuitive that the line connecting two good samples is likely to be aligned with the optimal (but unknown) gradient ascend.

In the following, we explain how the above insights are leveraged to design the sampling policy for *AdaNS* subroutines.

A. AdaNS-Zoom Sampling Subroutine

Our first sampling subroutine, i.e., *AdaNS-Zoom*, establishes $\mathcal{G}_{t+1}(\cdot)$ using a set of uniform probability distributions as the prior. Uniform sampling requires a large number of samples that maximally cover the optimization space, in order to locate the objective function maximizers. This solution, however, is infeasible for the problem at hand as the number of required samples and evaluations increases exponentially with the number of DNN layers. To alleviate this high sample count, we propose an adaptive methodology that intelligently distributes random samples across the search-space. *AdaNS-Zoom* iteratively (1) divides the space into multiple sub-regions and (2) adaptively tunes the per-region sample density.

Division: This step determines the boundaries that divide up the search-space into the aforementioned sub-regions. Our division algorithm gradually generates a hierarchy of sub-regions based on the information acquired from previously observed samples. We start with the whole search-space considered as one sub-region as shown in Fig. 6(a). Our goal is to use samples to accurately characterize sub-regions that are more likely to contain near-optimal solutions. To increase sampling resolution in such *good* areas, *AdaNS* gradually zooms into high-quality sub-regions by dividing them in half as shown in Fig. 6(b) and 6(d). We quantify the quality of the i^{th} sub-region by its share of good samples w_i:

$$w_i = \frac{|\mathbb{S}_g|_i}{|\mathbb{S}_{[1:t]}|_i} \tag{9}$$

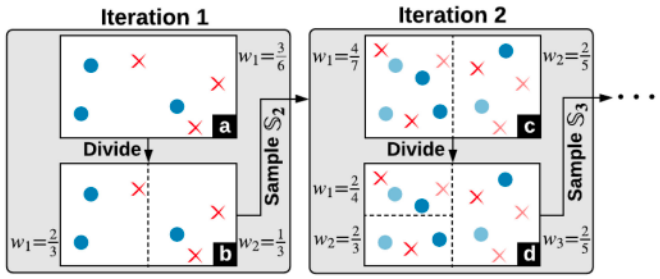

Fig. 6. *AdaNS-Zoom* algorithm. Here, the good and bad samples are shown with blue circles and red crosses, respectively.

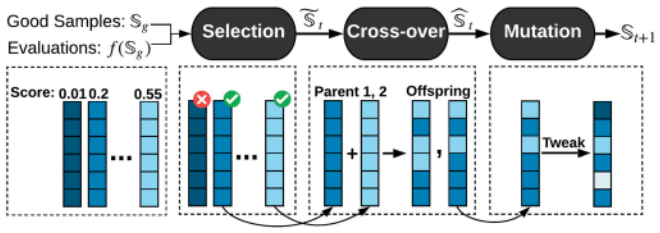

Fig. 7. Overview of *AdaNS-Genetic* sampling subroutine.

where $|\cdot|_i$ counts the total number of samples contained in sub-region i throughout all iterations. We then choose the sub-region with maximum w_i in each iteration to be divided in half along its longest dimension.

Sampling: In the first iteration of the algorithm, we uniformly sample the entire search-space since no prior knowledge is available. Throughout next iterations, after each division takes place, new w_is are computed to reflect the portion of good samples lying in the new sub-regions. To generate a new sample, we randomly select a sub-region with w_i representing the (relative) chance of the i^{th} region being selected. We then draw a sample from a uniform distribution over the selected sub-region. By repeating this process b times, we obtain the next batch of samples $\mathbb{S}_{t+1} = \{\vec{x}_i\}_{i=1}^{b}$ in Algorithm 1, Line 6.

B. AdaNS-Genetic Sampling Subroutine

To generate higher quality samples based on previous observations, our second subroutine utilizes a genetic algorithm. Genetic algorithms are metaheuristic approaches inspired by natural evolution and the notion of "survival of the fittest." These methods can be leveraged as powerful tools to explore large search-spaces while enjoying high scalability and significantly low training overhead [23], [38], [39].

Our prior work on utilizing genetic algorithms for DNN compression has shown impressive results [40]. In this paper, we aim to utilize similar genetic operations and investigate *AdaNS-Genetic* from a sampling point of view. At iteration t, the genetic algorithm evolves the previously evaluated good samples $\mathbb{S}_g$ into a new, more competent batch of samples. This is achieved by performing a set of bio-inspired operations, i.e., selection, crossover, and mutation, shown in Fig. 7. Below we delineate the details of each aforementioned operation.

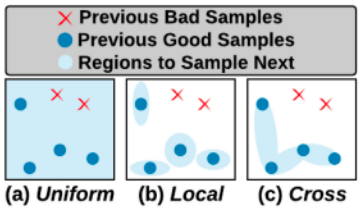

Fig. 8. $2D$ illustration of *AdaNS-Gaussian* sampling strategies.

Selection: The selection step chooses high-quality samples $\widetilde{\mathbb{S}}_t = \{\widetilde{x}_1 \ldots, \widetilde{x}_b\}$ from $\mathbb{S}_g$ that will be utilized in generating the next batch of samples. This is done by performing a non-uniform sampling (without replacement) from $\mathbb{S}_g$, where the probability of selecting each sample $\vec{x}_i^g$ is proportional to its score $f_i = f(\vec{x}_i^g)$. We normalize the scores as follows:

$$f_i \leftarrow \frac{f_i - f_{min}}{\sum_{i=1}^{K}(f_i - f_{min})}, \tag{10}$$

Here, f_{min} is the minimum score within $f(\mathbb{S}_g)$. Subtraction of the minimum score ensures that the probability of selecting the lowest-quality sample is zero and it is always eliminated. Such score-based selection is inherently random and allows for exploration. As the same time, due to the score-proportionate selection, high-quality samples are more likely to appear in the selection. This approach enables *AdaNS-Genetic* sampling to achieve a balance between exploration/exploitation.

Crossover: Given the selected set $\widetilde{\mathbb{S}}_t = \{\widetilde{x}_1 \ldots \widetilde{x}_b\}$, crossover generates two offspring by randomly swapping the vector elements in pairs of adjacent samples $\{\widetilde{x}_{2k-1}, \widetilde{x}_{2k}\}_{k=1}^{\frac{b}{2}}$. We use two parameters to control the degree of crossover operation: p_{cross} determines the probability of applying crossover between two samples, and p_{swap} is the per-element swapping probability. The intuition behind crossover is to allow high-quality hyperparameter configurations to exchange learned patterns and enable knowledge transfer across samples.

Mutation: Mutation randomly tweaks the elements in each sample vector in the crossover-ed set $\widehat{\mathbb{S}}_t$. Similar to crossover, we define two control parameters: p_{mutate} is the probability that the sample gets mutated and p_{tweak} determines the per-element tweaking probability. Mutation allows for the exploration of neighborhoods around the selected points. By choosing a small p_{mutate} and p_{tweak}, we ensure that the desirable qualities of strong samples are preserved. Each element of a sample $\vec{x}_i \in \mathbb{R}^d$ is mutated by adding a random value drawn from a zero-mean Normal distribution $\mathcal{N}(0,\ 0.2)$. We then clip the values to ensure they remain in the valid range, i.e., [0,1].

C. AdaNS-Gaussian Sampling Subroutine

Our last sampling subroutine utilizes a combination of Gaussian and uniform distributions to model the adaptive sampling distribution $\mathcal{G}(\cdot)$ based on the previously observed good samples $\mathbb{S}_g$. Specifically, we draw the new set of samples from a combination of three sampling distributions that collectively form $\mathcal{G}(\cdot)$ as illustrated in Fig. 8; here, previously observed good samples are shown in dark blue, red crosses denote observed low-score (bad) samples, and the sampling density is marked with light blue. We design each of *AdaNS-Gaussian* sampling distributions to address one of the insights mentioned at the begining of Section V:

1) We draw a portion of new samples uniformly from $\mathbb{X} \subset [0, 1]^d$ (Fig. 8(a)), dubbed *Uniform* samples. This policy explores the whole space without prior knowledge.
2) We draw another portion of samples, dubbed *Local*, from the vicinity of good samples $\mathbb{S}_g$ (Fig. 8(b)). The underlying PDF is a mixture of Gaussians, with centers located at $\mathbb{S}_g$.
3) We draw the last portion of samples, dubbed *Cross*, from a mixture of Gaussians centered in the mid-points of good samples $\mathbb{S}_g$ (Fig. 8(c)). This policy has a high sampling density along the line connecting two good samples.

The intuitions behind *AdaNS-Gaussian* sampling strategy rest upon a line of work in sampling theory [41]–[43] that prove Gaussian Kernels with adaptive variances can reconstruct smooth non-linear functions. Inspired by that, we vary the Gaussian variance spatially according to local information about the approximand [41]; the parameters of the Gaussian Mixture Models (GMMs) are chosen to maximize the likelihood of previously successful samples. Below we elaborate on the above sampling policies.

Uniform Samples: To allow *AdaNS-Gaussian* to explore unseen regions, we select a portion of samples uniformly at random from the entire search-space. This prevents the search from becoming too localized upon observing several good samples in a small region.

Local Samples: To effectively explore the vicinity of good samples, we use a GMM for sampling [44]. Formally, the sampling PDF is:

$$\mathcal{P}(\vec{x}) = \sum_{i=1}^{K} w_i\, \mathcal{N}(\vec{x}_i^g, \Sigma), \qquad \vec{x}_i^g \in \mathbb{S}_g \tag{11}$$

where $\mathcal{N}(\vec{x}, \Sigma)$ is a multi-variate Gaussian distribution [45] with mean vector $\vec{x} \in \mathbb{R}^d$ and diagonal covariance matrix represented with $\Sigma \in \mathbb{R}^d \times \mathbb{R}^d$. Here, w_i is a weight parameter that adjusts the probability of choosing the i^{th} multi-variate Gaussian. We set the weights proportional to the value of the objective function, i.e., $w_i \propto f(\vec{x}_i^g)$. The standard deviation Σ is determined such that the Gaussians $\mathcal{N}(\cdot, \cdot)$ cover the so-far observed span of $\mathbb{S}_g$:

$$\begin{cases} \sigma_{jj}^2 = (\max\limits_{\vec{x} \in \mathbb{S}_g} \vec{x}[j] - \min\limits_{\vec{x} \in \mathbb{S}_g} \vec{x}[j])^2 & \forall j \in \{1 \ldots d\} \\ \sigma_{ij}^2 = 0 & \forall j \neq i \end{cases} \tag{12}$$

The above choices for the weights and the covariance matrix allow for an adaptive sampling scheme. First, by incorporating the scores into the weights ($w_i \propto f(\vec{x}_i^g)$), regions around high-score samples are given a higher priority to be explored. Second, the standard deviation in Eq. (12) adaptively configures the sampling range in all dimensions: if members of $\mathbb{S}_g$ agree on dimension j, the corresponding σ_{jj}^2 will be small and the generated samples will be very similar in the j^{th} dimension; conversely, if members of $\mathbb{S}_g$ disagree on dimension j, the corresponding σ_{jj}^2 will be large and the generated samples will

be scattered in the j^{th} dimension. This enables *AdaNS-Gaussian* to automatically tune exploration and exploitation.

Cross Samples: The line connecting two good samples may represent the direction of gradient ascent for the objective function $f(\cdot)$. To explore such regions, we draw a portion of samples from the mid points of current good samples $\mathbb{S}_g$. To generate one sample, we pick a pair of good samples $\{\vec{x}_1, \vec{x}_2\} \in \mathbb{S}_g$, and draw a sample from the multivariate Gaussian $\mathcal{N}(\vec{\mu}_{1,2}, \Sigma_{1,2})$. The mean value is $\vec{\mu}_{1,2} = \frac{\vec{x}_1+\vec{x}_2}{2}$ and the diagonal covariance matrix is set as:

$$\begin{cases} \sigma^2_{jj} = (\vec{x}_1[j] - \vec{x}_2[j])^2/4 & \forall j \in \{1 \dots d\} \\ \sigma^2_{ij} = 0 & \forall j \neq i \end{cases} \tag{13}$$

This approach searches for samples in the confined space between pairs of good samples. Multiple *Cross* samples are generated by repeating the above process. We explore several strategies for selecting $\{\vec{x}_1, \vec{x}_2\}$:

❶ Select both $\vec{x}_1$ and $\vec{x}_2$ randomly from $\mathbb{S}_g$.
❷ Sort all possible pairs $\{\vec{x}_1, \vec{x}_2\} \in \mathbb{S}_g$ based on their sum of scores $f(\vec{x}_1) + f(\vec{x}_2)$ and select pairs from the sorted list.
❸ Sort individual members $\vec{x} \in \mathbb{S}_g$ based on their scores $f(\vec{x})$, sequentially select $\vec{x}_1$ from the sorted list and $\vec{x}_2$ as the sample with maximum Euclidean distance to $\vec{x}_1$.
❹ Choose $\vec{x}_1$ as in case ❸ but select $\vec{x}_2$ randomly.

Our experiments show that the fourth strategy renders the best performance on average. Therefore throughout the rest of the paper, we use the latter method for pair selection.

VI. Reconstruction

In this section, we provide an empirical analysis to verify the effectiveness of the designed sampling distributions $\mathcal{G}(\cdot)$ in finding near-optimal solutions. Towards this goal, we devise a methodology that reconstructs the opaque objective function, based on previously evaluated samples. Specifically, at each iteration t, we create an estimate $\widetilde{f}(\cdot)$ for the hidden function $f(\cdot)$ using $f(\mathbb{S}_{[1:t]})$. Upon obtaining the new batch of (unseen) samples $\mathbb{S}_{t+1}$ and new measurements $f(\mathbb{S}_{t+1})$, we compute the (normalized) estimation errors as:

$$e(\vec{x}) = \frac{f(\vec{x}) - \widetilde{f}(\vec{x})}{f(\vec{x})}, \qquad \vec{x} \in \mathbb{S}_{t+1} \tag{14}$$

We then compute the mean mean absolute error e_{avg} over all samples. e_{avg} measures how much the obtained value of the objective function for new samples deviates from our estimation based on reconstruction. A high error implies that the sampling subroutine is exploring the space rather than using knowledge from $\mathbb{S}_{[1:t]}$ to find better samples. Conversely, a small e_{avg} shows that the adaptive sampler is exploiting previous good samples to generate $\mathbb{S}_{t+1}$. Fig. 9 presents the task of pruning a VGG network trained on CIFAR-10 with *AdaNS-Gaussian*. Below, we summarize our observations:

1) The growth in the average score among good samples $f(\mathbb{S}_g)$ shows that *AdaNS* adaptive sampler iteratively sample values closer to the function maximizer.

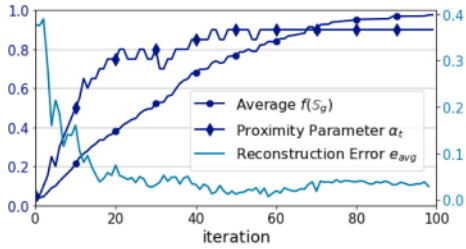

Fig. 9. Per-iteration analysis of *AdaNS* sampling.

2) The growth in the proximity parameter α_t together with property 1 suggests that the probability of finding the near-optimal solution is increasing as desired in Eq. (7).
3) The drop in average reconstruction error e_{avg} together with property 1 demonstrates the ability of *AdaNS* in reconstructing the objective function around its potential maximizers. Specifically, the history of prior samples across *AdaNS* iterations are sufficient to estimate the value of the objective function at the future (unseen) samples that are closer to the function maximizers.

Establishing $\widetilde{f}$***:*** We consider a GMM prior for $\widetilde{f}(\cdot)$:

$$\widetilde{f}(\vec{x}) = \sum_{\vec{x}_i \in \mathbb{S}_{[1:t]}} w_i \, \text{Gauss}(\vec{x}, \vec{x}_i, \Sigma_i) \tag{15}$$

where $Gauss(\vec{x}, \vec{y}, \Sigma)$ denotes the value of a multi-dimensional Gaussian kernel with mean $\vec{y}$ and covariance matrix Σ measured at point $\vec{x}$. Here, w_i are scalar weights, $\vec{x}_i \in \mathbb{S}_{[1:t]}$ are the centers of the GMM model located at all previously evaluated samples, and $\Sigma_i \in \mathbb{R}^{d \times d}$ are diagonal covariance matrices. For the i^{th} component of the GMM centered at $\vec{x}_i$, we find the closest sample $\vec{x}_j$ with minimum Euclidean distance to $\vec{x}_i$. We then use the element-wise absolute difference, $\Delta\vec{x}_i = |\vec{x}_i - \vec{x}_j|$ to compute the covariance matrix Σ_i:

$$\begin{cases} \sigma^2_{mm} = \beta(\Delta\vec{x}_i[m])^2 & \forall m \in \{1 \dots d\} \\ \sigma^2_{mn} = 0 & \forall m \neq n \end{cases} \tag{16}$$

The scalar $\beta = \frac{d}{8log(2)}$ normalizes the diagonal elements such that the multivariate Gaussian component is diminished by a factor of 2 in the mid-point of $\vec{x}_i, \vec{x}_j$. Given the Gaussian means $\vec{x}_i$ and covariance matrices Σ_i, the weights w_i can be determined by minimizing the error between real function evaluations $f(\mathbb{S}_{[1:t]})$ and the estimates $\widetilde{f}(\mathbb{S}_{[1:t]})$:

$$w_1 \dots w_K = \underset{w_1 \dots w_K}{\text{argmin}} \sum_{x \in \mathbb{S}_{[1:t-1]}} |f(\vec{x}) - \widetilde{f}(\vec{x})|^2 \tag{17}$$

which is solved by Least-Square Optimization [46].

VII. Experiments

We provide extensive evaluations on CIFAR-10 and ImageNet benchmarks and compare with prior works in RL, Bayesian Optimization, and several heuristics. The evaluated network architectures include AlexNet, VGG, ResNet family, and MobileNets, implemented in PyTorch. We randomly select 1,000 images from the training data to use as validation set for score computation. We optimize hyperparameters for non-structured

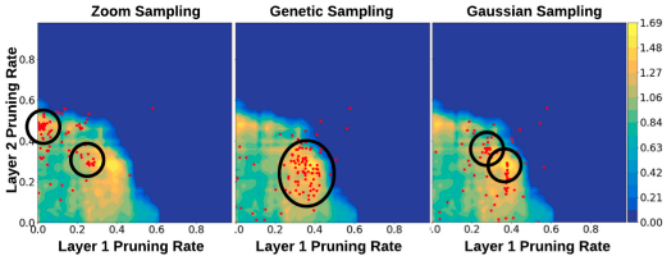

Fig. 10. Comparison between *AdaNS* sampling subroutines for pruning a 2-layer DNN. The total number of samples is the same in each experiment. *AdaNS-Gaussian* achieves better exploration/exploitation tradeoff as it identifies the global maximum and concentrates the sampling around it.

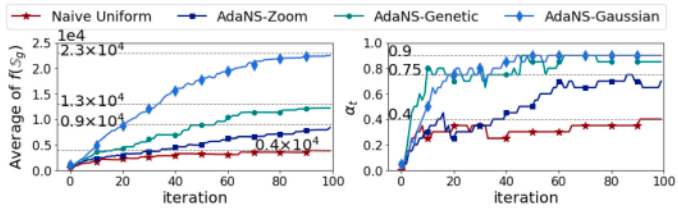

Fig. 11. Convergence analysis of various sampling strategies across algorithm iterations. (left): mean score achieved by good samples. (right): Proximity parameter value.

(P_n) and structured (P_s) pruning, SVD and Tucker decomposition (D), and combination of multiple methods ($D + P_s$). In our experiments, sample portions in *AdaNS-Gaussian* are assigned to be 45% *Local*, 45% *Cross*, and 10% *Uniform*. For *AdaNS-Genetic*, the crossover and mutation parameters are set to $p_{mutate} = 0.2, p_{tweak} = 0.05, p_{cross} = 0.8, p_{swap} = 0.2$ following [39], [40]. Additional implementation details and our experimental setup for training and fine-tuning of benchmarked DNNs are included in Appendix B.

A. Effect of Sampling Strategy on Convergence

In this section, we investigate the impact of the sampling strategy on convergence and optimization end result. We first visualize the samples from our three subroutines for an example 2-layer network. Next, we move to pruning a real-world DNN benchmark and compare the convergence behavior of *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*.

2-layer Example Network: The hyperparameter space for this example is $\vec{x} \in [0, 1]^2$ where each element of $\vec{x}$ represents the pruning rate for one layer. For this small example network, we executed a brute-force grid search to extract the heatmap of the pertinent objective function $f(\vec{x})$ as shown in Fig. 10. Here, the blue and yellow colors denote minimum and maximum score function values, respectively, and red dots represent the samples. As seen, the *AdaNS-Zoom* subroutine (left) becomes concentrated on two of the local maxima regions (shown by black circles) but misses the global maximum located at (0.4,0.3). *AdaNS-Genetic* (middle) achieves more diversity than *AdaNS-Zoom*, and finds the neighborhood of the actual maximum; however, it does not achieve concentrated sampling. Finally, *AdaNS-Gaussian* (right) identifies the global maximum and concentrates the sampling around it.

VGG Benchmark: We use *AdaNS* for the task of structured pruning to compress a VGG network on CIFAR-10 dataset. We also provide the performance of a naïve method that uniformly samples the entire search-space. Fig. 11-(left) presents the evolution of averageed good scores $f(\mathbb{S}_g)$ at each iteration, with $\mathbb{S}_g$ denoting the set of good samples. As seen, the naïve sampling fails to find the maxima in the high-dimensional optimization space at hand. This means unless carefully designed, a given naïve sampling strategy that arbitrarily changes the underlying hyperparameters fails to sample from correct regions of the space. However, our careful design of the sampling strategies can significantly change the performance by successfully arriving at the optima. *AdaNS-Gaussian* shows the largest growth in score suggesting that it is more successful in generating more competent samples based on prior observations. This is due to the Gaussian kernels in *AdaNS-Gaussian* which enable maximal exploration of sub-regions that potentially contain near-optimal solutions. *AdaNS-Zoom* and *AdaNS-Genetic* perform a more localized search and thus demonstrate slower convergence.

Fig. 11-(right) shows how the proximity parameter α_t evolves over time. Initially, α_t is tuned to a small value to allow exploration of the entire search-space. As the search proceeds, the value of α_t is increased to direct the sampling towards the identified good samples. As seen, α_t rises quickly to $\alpha_{max} \approx 0.9$ for *AdaNS-Gaussian* and *AdaNS-Genetic* while it has a slower growth in *AdaNS-Zoom*. This suggests that *AdaNS-Gaussian* and *AdaNS-Genetic* can quickly learn to generate samples that are equally good or better than previously seen samples.

We further dissect the maximal score function in terms of accuracy and FLOPs in Table I. The obtained accuracy and execution cost from each method confirms the importance of the sampling strategy on the final optimization result.

TABLE I
ACCURACY AND FLOPS OF THE FINAL COMPRESSED MODEL USING VARIOUS SAMPLING METHODOLOGIES. SAMPLE SIZE IS $b = 50$ AND WE LET EACH SAMPLING ALGORITHM RUN FOR 100 ITERATIONS ON A VGG NETWORK TRAINED ON CIFAR-10. A_{thr} IS SET TO 70%

Sampling Strategy	Accuracy* (%)	FLOPs (%)
Naïve Uniform	73.5	40.1
AdaNS-Zoom	70.4	37.4
AdaNS-Genetic	70.6	35.3
AdaNS-Gaussian	73.5	34.3

* The reported accuracy is before fine-tuning.

B. Quantitative Results on CIFAR-10

We apply *AdaNS-Gaussian* and *AdaNS-Genetic* to pre-trained CIFAR-10 architectures and compare our results with prior art in Table II. We set the number of samples to 100 for ResNet-56 and ResNet-50, 200 for ResNet-110, and 50 for VGG. A_{thr} is set to 90% for ResNet-X and 65% for VGG. For all networks, we let *AdaNS* sampling run for 50 iterations.

Non-structured Pruning (P_n in Table II): We perform non-structured pruning on ResNet-50 and report the ratio of non-zero model parameters. A_{thr}, is set to 93% and we do not perform any fine-tuning on the compressed model. As shown, *AdaNS-Gaussian* and *AdaNS-Genetic* achieve higher accuracy with

TABLE II
COMPARISON WITH CONTEMPORARY COMPRESSION METHODS BASED ON THE NON-ZERO PARAMETER RATIO/FLOPS

CIFAR-10

	Model Policy	Top1 (%)	Cost (%)
ResNet-50	Baseline	93.7	100
	AMC [12]	93.5	40.0
	Rethinking [21]	93.4	20.0
	Genetic (P_n)	**93.6**	**30.0**
	Gaussian (P_n)	**94.0**	**23.1**
ResNet-56	Baseline	93.6	100
	Rethinking [21]	93.1	72.4
	CP [4]	91.8	50.0
	AMC [12]	91.9	50.0
	SFP [3]	93.3	47.4
	PocketFlow [6]	92.8	40.0
	Genetic (P_s)	**93.2**	**44.0**
	Gaussian (P_s)	**93.1**	**40.6**
	Gaussian (D)	**93.5**	**59.1**
	Gaussian (D+P_s)	**93.2**	**36.9**
ResNet-110	Baseline	94.0	100
	Rethinking [21]	93.6	61.4
	Filter Pruning [5]	93.3	61.4
	AMC [12]	93.8	59.2
	Genetic (P_s)	**93.6**	**41.2**
	Gaussian (P_s)	**93.9**	**33.9**
	Gaussian (D)	**93.9**	**55.3**
	Gaussian (D+P_s)	**92.6**	**21.9**
VGG	Baseline	93.6	100
	Rethinking [21]	93.7	65.8
	ThiNet [7]	93.4	39.4
	NRE [2]	93.4	32.4
	Genetic (P_s)	**93.3**	**32.8**
	Gaussian (P_s)	**93.2**	**29.6**
	Gaussian (D)	**93.5**	**25.5**
	Gaussian (D+P_s)	**93.1**	**14.5**

ImageNet

	Model Policy	Top1 (%)	Top5 (%)	Cost (%)
AlexNet	Baseline	60.7	80.0	100
	CAC [17]	54.8	-	5.0
	Genetic (P_n)	**56.1**	**78.2**	**8.5**
	Gaussian (P_n)	**55.1**	**78.3**	**7.0**
ResNet-50	Baseline	75.1	93.0	100
	SFP [3]	62.1	84.6	58.2
	SSS [24]	71.8	90.8	57.0
	Rethinking [21]	75.0	-	50.0
	CP [4]	-	90.8	50.0
	GDP [1]	71.9	90.7	48.7
	ThiNet [7]	71.0	90.0	44.0
	Rethinking [21]	71.6	-	30.0
	Genetic (P_s)	**73.2**	**91.4**	**41.9**
	Gaussian (P_s)	**72.6**	**91.1**	**29.1**
	Gaussian (D)	**74.3**	**92.1**	**44.5**
	Gaussian (D+P_s)	**72.1**	**90.9**	**23.7**
VGG-16	Baseline	71.1	90.0	100
	GDP [1]	67.5	87.9	24.5
	RNP [9]	-	86.3	20.0
	SPP [8]	-	87.6	20.0
	AMC [12]	-	88.2	20.0
	Rethinking [21]	71.0	-	20.0
	Genetic (P_s)	-	**88.1**	**20.0**
	Gaussian (P_s)	**68.8**	**88.3**	**19.6**
	Gaussian (D)	**70.6**	**90.1**	**31.0**
	Gaussian (D+P_s)	**68.4**	**88.5**	**14.1**

$1.7\times$ and $1.3\times$ lower parameters compared to state-of-the-art Reinforcement Learning method, AMC [12]. Note that lower FLOPs and comparable accuracy of [21] are due to training the model from scratch whereas *AdaNS* and [12] do not fine-tune.

Structured Pruning ***(P_s in Table II):*** Comparisons are based on the number of operations per inference, i.e., FLOPs, relative to the uncompressed baseline. With similar accuracy, *AdaNS-Gaussian* achieves $1.5\times$ lower FLOPs than prior art (on average).

Decomposition and Pruning ***($D + P_s$ in Table II):*** To unveil the full optimization potential of our adaptive sampling methodology, we allow *AdaNS* to learn and combine multiple compression techniques, namely, structured pruning, SVD, and Tucker. The $D + P_s$ experiments are conducted by first decomposing the network and then applying pruning. As shown in Table II, *AdaNS* pushes the limits of compression by $2.4\times$ on average with less than 1% drop in accuracy compared to state-of-the-art works. We also report FLOPs reduction by the combination of SVD and Tucker decomposition methods (shown by D in Table II).

C. Quantitative Results on ImageNet

Table II summarizes *AdaNS* results on ImageNet. Number of samples is 20 for AlexNet, 100 for ResNet-50, and 50 for VGG-16. We run *AdaNS* for 50 iterations with A_{thr} of 10% for all models. The final accuracy is improved by fine-tuning.

Non-structured Pruning ***(P_n in Table II):*** We perform non-structured pruning on AlexNet and report the ratio of non-zero model parameters. As seen, *AdaNS-Gaussian* achieves higher accuracy with 2% higher parameter size compared to a Bayesian Optimization approach, i.e., CAC [17].

Structured Pruning ***(P_s in Table II):*** On ResNet-50, *AdaNS-Genetic* and *AdaNS-Gaussian* compress the models to $1.2\times$ and $1.6\times$ less FLOPs on average while achieving higher top-5 accuracy compared to prior works. On VGG-16, *AdaNS* outperforms all heuristic methods and gives competing results with [21] and [12]. Note that [21] does not propose a hyper-parameter optimization algorithm and merely focuses on the training already-compressed DNNs. As Such, their approach is orthogonal to *AdaNS* and can be combined with our method to further improve final accuracy.

Decomposition and Pruning ***($D + P_s$ in Tab. II):*** Using a combination of decomposition and structured pruning, *AdaNS* achieves $2.0\times$ lower FLOPs than related work (on average) with slightly higher accuracy on ResNet-50. On VGG-16, *AdaNS* pushes the state-of-the-art FLOPs reduction from $5.0\times$ to $7.1\times$ with higher accuracy.

D. Compressing Compact Networks

To further demonstrate the effectiveness of *AdaNS* optimization, we apply compression to MobileNet architectures trained on ImageNet dataset. These networks are specifically designed for embedded applications with strict efficiency constraints. As such, MobileNets inherently have very low complexity/redundancy which renders their compression quite challenging. We apply pruning to MobileNetV1 and MobileNetV2 with a sample size of $b = 50$, and let the adaptive sampling run for 100 iterations.

We compare the compression rate and accuracy achieved by *AdaNS* with the FLOPs-accuracy Pareto curve of the original MobileNet architectures [18], [19]. We further compare *AdaNS* with the state-of-the-art AutoML approaches [12], [16] and a compression-aware training methodology, US-Nets [20], [47]. Table III encloses the results of applying structured pruning to MobileNetV1 and MobileNetV2. We benchmark several target FLOPs and compare them with prior work with similar computational complexities. On average, *AdaNS* achieves 1.2% better accuracy than the MobileNetV1 Pareto curve. Compared to US-Nets, *AdaNS* achieves an average of 1.0% higher accuracy. Under $\sim 50\%$ FLOPs, *AdaNS* achieves 1.3% higher accuracy than NetAdapt. Compared to AMC, *AdaNS* achieves lower FLOPs with comparable accuracy (-0.1%). On MobileNetV2, for a 30% FLOPs reduction, *AdaNS* achieves lower FLOPs and higher accuracy than US-Nets and higher accuracy with the same FLOPs compared to AMC and the MobileNetV2 Pareto curve.

Measured Speedup: We present measured speedups of *AdaNS* compressed MobileNets on an embedded CPU (ARM Cortex-A57) and GPU (NVIDIA Pascal) in Table IV. Measurements are averaged on 100 runs using a batch size of 32. *AdaNS*

TABLE III
PRUNING OF MOBILENETV1&V2 ON IMAGENET

	Policy	Top1 (%)	Top5 (%)	FLOPs
MobileNetV1	Baseline (1×)	70.6	89.5	569 M
	MobileNetV1 (0.75×) [18]	68.4	88.2	325 M
	US-Nets [47]	69.5	-	325 M
	AdaNS-Gaussian	**70.5**	89.3	**323 M**
	US-Nets [47]	68.8	-	287 M
	AMC [12]	70.5	89.1	285 M
	NetAdapt [16]	69.1	-	284 M
	AdaNS-Gaussian	**70.4**	89.1	**283 M**
	US-Nets [47]	66.8	-	217 M
	AdaNS-Gaussian	**67.9**	88.1	**210 M**
	MobileNetV1 (0.5×) [18]	63.7	-	149 M
	US-Nets [47]	63.5	-	136 M
	AdaNS-Gaussian	**64.1**	85.4	**136 M**
MobileNetV2	Baseline (1×)	71.6	90.3	313 M
	MobileNetV2 (0.75×) [19]	69.8	88.3	220 M
	US-Nets [47]	70.0	-	222 M
	AMC [12]	-	89.3	220 M
	AdaNS-Gaussian	**70.1**	**89.5**	**220 M**

TABLE IV
AdaNS COMPRESSED MOBILENETS SPEEDUP ON EMBEDDED CPU AND GPU FOR STRUCTURED PRUNING ON IMAGENET

Model	Theoretical Speedup	Real Speedup	
		Cortex-A57 (CPU)	Pascal (GPU)
MobileNetV1	1.7×	1.6×	1.3×
	2×	1.7×	1.4×
	2.7×	2.5×	1.7×
	4×	3.4×	1.9×
MobileNetV2	1.4×	1.4×	1.4×

successfully models the hardware cost to achieve real speedups on par with theory.

E. Search Overhead and Scalability

The core computational load in *AdaNS* algorithm corresponds to the evaluation of a batch of b samples. For each sample $\vec{x}_i$ in the batch, the evaluation phase comprises transforming the sample to its corresponding compressed DNN $\hat{M}_{\vec{x}_i}$, measuring the inference accuracy on the validation data, and emulating the execution cost. Since samples in a batch are independent, the evaluation step can be well-parallelized on multiple GPU devices to achieve faster search convergence. Aside from evaluation, each iteration includes updating the sampling strategy and generating a new batch of samples. These steps, however, incur negligible runtime compared to the evaluation stage.

Table V summarizes the runtime of *AdaNS* algorithm for several benchmarks and datasets. Runtimes are measured on a machine with an Intel Xeon E5 CPU and four NVIDIA Titan Xp GPUs. The results show high scalability: runtime drops almost linearly with the number of GPUs. The state-of-the-art RL algorithm reports ∼1 hour to compress CIFAR-10 architectures [12]. For their most complex benchmark, i.e., ResNet-56, *AdaNS* achieves a search time of only ∼12 minutes on a single GPU and ∼3 minutes on four GPUs.

TABLE V
SEARCH RUNTIME OF *AdaNS-GAUSSIAN* FOR PRUNING ON VARIOUS BENCHMARKS. HERE, b DENOTES THE NUMBER OF SAMPLES PER ITERATION AND N_{iters} IS THE NUMBER OF SEARCH ITERATIONS

Dataset	Arch.	b	N_{iters}	Search Time (minutes)			
				1 GPU	2 GPU	3 GPU	4 GPU
ImageNet	AlexNet	50	50	10	5	3	3
	VGG-16	50	50	112	57	38	28
	ResNet-50	100	50	145	73	49	36
	MobileNetV1	50	100	97	48	32	24
	MobileNetV2	50	100	116	57	38	25
CIFAR-10	VGG	50	50	3	2	1	1
	ResNet-50	100	50	35	19	13	11
	ResNet-56	100	50	12	7	5	3
	ResNet-110	200	50	55	30	22	16

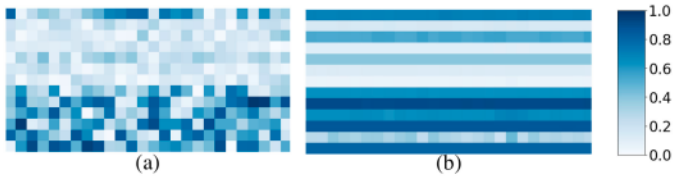

Fig. 12. (a) Set of randomly initialized samples at the first iteration. (b) Set of good samples $\mathbb{S}_g$ upon convergence.

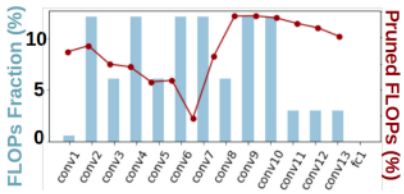

Fig. 13. Original per-layer FLOPs versus *AdaNS* pruning pattern.

F. Analysis and Discussion

In this section, we look into *AdaNS* generated samples and provide discussions. For brevity, we only focus on *AdaNS-Gaussian* subroutine that achieves superior results compared to *AdaNS-Zoom* and *AdaNS-Genetic*. We consider the VGG architecture trained on CIFAR-10 and compress it with structured pruning for $A_{thr} = 60\%$. The initial samples obtained from our directed initialization method are shown in Fig. 12(a), where each column corresponds to a hyperparameter vector and each row represents a certain DNN layer. After applying *AdaNS-Gaussian* for 50 iterations, the set of good samples in Fig. 12(b) are learned; the columns (members of $\mathbb{S}_g$) strongly resemble one another and have similarly high scores upon convergence. *AdaNS* successfully learns expert-designed rules: first and last layers of the network (first and last rows in Fig. 12(b)) are given high densities to maintain the inference accuracy. *AdaNS* performs *whole-network* compression by encoding all DNN layers' hyperparameters in each sample. As such, our algorithm can learn which configuration of hyperparameters least affects model accuracy and most reduces the overall FLOPs. To show this capability, we present the per-layer FLOPs from one of the obtained good samples of Fig. 12(b) in Fig. 13. For each layer, the bars show the percentage of total FLOPs in the original model; the curve shows the percentage of pruned FLOPs in the

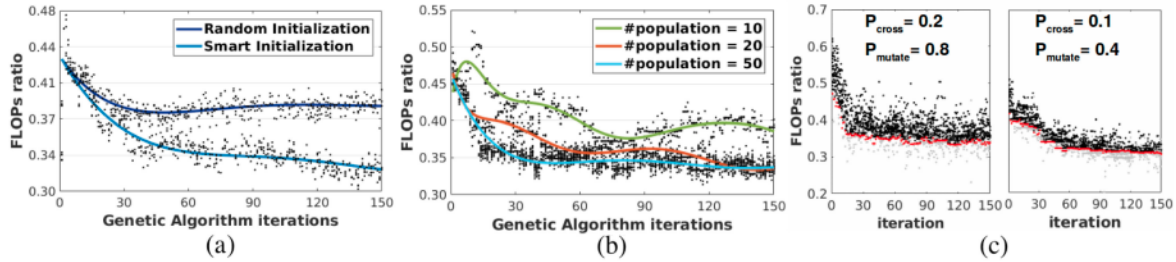

Fig. 14. Ablation studies for VGG on CIFAR-10:(a) Effect of initialization method. (b) Effect of sample count. We show the trend lines as well as a fraction of samples (black dots) across *AdaNS* iterations. (c) Effect of mutation and cross-over probabilities for *AdaNS-Genetic*. Here, samples shown in grey have lower accuracy than $A_{thr} = 60\%$ while samples shown in black meet the threshold. Red dots correspond to the highest-quality sample in that iteration.

compressed network. As seen, the optimal pruning rates (red curve) show arbitrary patterns that are very hard to identify for human experts. *AdaNS* automatically extracts such patterns by an adaptive search.

G. Ablation Study

This section studies the effect of various *AdaNS* parameters on algorithm convergence and end result. For brevity, we only focus on structured pruning for VGG on CIFAR-10.

Effect of Initialization: Fig. 14(a) shows the evolution of FLOPs ratio when running *AdaNS-Genetic* algorithm with two initialization policies: one with uniformly random samples and one with our proposed initialization scheme discussed in Section IV-C. As seen, naive initialization harms the convergence rate and final FLOPs.

Effect of Sample Count: Fig. 14(b) presents the effect of number of evaluated samples per iteration of *AdaNS-Genetic* on convergence. A higher number of samples results in a smoother convergence and lower final FLOPs, due to the higher capacity for exploration/exploitation. This effect saturates for a large enough sample set. We further observed that the per-iteration sample count should be proportional to the individual length.

Effect of Accuracy Threshold: The accuracy threshold A_{thr} in Eq. (4) determines model accuracy after compression. In our experiments, we observed a monotonic correlation between A_{thr} and final accuracy after fine-tuning. This property eliminates the need for fine-tuning each compressed DNN configuration in between algorithm iterations which significantly improves *AdaNS* search efficiency and timing overhead. Note that a smaller A_{thr} generally results in a lower hardware cost.

Mutation and Crossover Parameters: These parameters affect *AdaNS-Genetic* sampling convergence. We conduct two experiments, one with $(P_{mutate}, P_{cross}) = (0.8, 0.2)$ and the other with $(P_{mutate}, P_{cross}) = (0.4, 0.1)$ and compare the convergence in Fig. 14(c). Higher (P_{mutate}, P_{cross}) allows more exploring, leading to faster convergence while smaller probabilities result in a more stable evolution. As seen, both settings converge to similar final FLOPs.

Effect of Pair Selection for Cross Samples: Fig. 15 compares the convergence behavior of the four proposed pair selection strategies for *Cross* samples (Section V-C). As seen, strategy ❹ achieves the highest final score with lowest variations (shown with the error bars). Strategy ❷ has a fast but premature convergence due to lack of exploration. In addition, strategy ❷ has a high variation. This is due to the high dependency of strategy ❷ on the configuration of good samples, which differs across runs. Strategy ❶ and ❸ offer a smooth convergence with low variance but their final score is lower than strategy ❹.

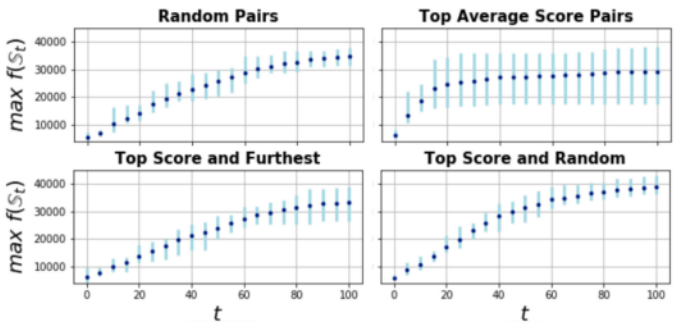

Fig. 15. Convergence curves for *Cross* samples pair selection (Section V-C). Graphs are generated over 10 runs.

VIII. Conclusion

This paper proposes *AdaNS*, and adaptive sampling methodology that can automatically tune hyperparameters for DNN compression. We first formulate DNN compression as searching a vector space that spans possible hyperparameters. Next, we define a generic score function to quantify the "goodness" of each hyperparameter vector by combining inference accuracy and execution cost. This approach allows us to address DNN compression as optimizing the unknown score function by sampling from its input space. To find the maximizer of the score function, we develop an iterative sampling strategy, along with three adaptive sampling subroutines: *AdaNS-Zoom*, *AdaNS-Genetic*, and *AdaNS-Gaussian*. We evaluate these sampling strategies and show that *AdaNS-Gaussian* can achieve superior search convergence. We examine *AdaNS* on structured and non-structured pruning of deep neural networks and show that outperforms the majority of human-designed stat-of-the-art network pruning algorithms.

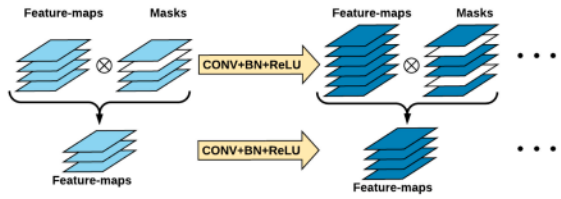

Fig. 16. Structured pruning. Top: pruning is implemented using masks (white plains show pruned feature-maps). Bottom: masks and pruned channels are removed after optimization.

APPENDIX A

This section serves as a supplement for Section IV-A. Recall the global encoding scheme $\mathcal{T}(M, \vec{x})$ that translates a vector of compression hyperparameters $\vec{x}$ to its corresponding compressed architecture $\hat{M}_{\vec{x}}$. For each compression method studied in the paper, we provide details on composing the vector of compression hyperparameters $\vec{x}$. We further introduce the transformation $\mathcal{T}(M, \vec{x})$ that uniquely determines the compressed model based on the provided hyperparameters $\vec{x}$.

Pruning: Pruning reduces the model size by setting a percentage of low priority parameters/channels to zero [4], [5], [33]. We allocate one continuous value $p \in [0, 1]$, per layer, to represent the ratio of non-zero values. We consider two contemporary DNN pruning methods, namely, structured and non-structured pruning. Structured pruning aims at removing a portion of feature-map channels while non-structured pruning removes a subset of DNN layer weights.

Having defined the vector of hyperparameters $\vec{x}$ for pruning, we implement the transformation $\mathcal{T}(M, \vec{x})$ following common practice in prior art. For structured pruning, we use the sum of absolute gradients of model loss with respect to ReLU feature-map channels for pruning priority [22]. For a ReLU layer with c feature-map channels and a pruning rate p, the $\lfloor p \times c \rfloor$ channels with lowest priorities (lower absolute gradients) are removed. For non-structured pruning, we use the absolute value of weights to prioritize them [33]. For a weight tensor $W \in \mathbb{R}^{k \times k \times c \times f}$ and pruning rate p, the $\lfloor p \times k \times k \times c \times f \rfloor$ elements with lowest absolute values are pruned.

Fig. 16 illustrates our implementation of structured pruning for convolutional layers; feature-map channels are multiplied by binary masks to implement pruning. Pruning rate can be altered at each layer by setting proper values (0 or 1) at the corresponding masks. Non-structured pruning is different than structured pruning in that the binary masks are applied on weights rather than output feature-maps. Usage of masks enables simulating the functional behavior of the pruned network at search time, without need for re-compiling the DNN graph per hyperparameter configuration. The graph is only modified once at test time where the channels with 0-valued masks are removed.

SVD: We apply SVD on weight parameters of fully-connected layers ($W \in \mathbb{R}^{c \times f}$) and point-wise convolutions ($W \in \mathbb{R}^{1 \times 1 \times c \times f}$). For SVD, the compression parameter is the decomposition rank which takes an integer value in $\{1, \ldots, R\}$ where $R = min(c, f)$. We allocate one continuous-valued $rank \in [\frac{1}{R}, 1]$ per layer to form the hyperparameter vector $\vec{x}$. Each element $\vec{x}[i]$ is interpreted as a decomposition rank equal to $\lfloor \vec{x}[i] \times R_i \rfloor$. For SVD, the transformation $\mathcal{T}(M, \vec{x})$ decomposes the weight kernels in different DNN layers with their corresponding ranks in $\vec{x}$ as shown in Fig. 17.

Tucker-2: Tucker decomposition is a generalized Higher Order SVD (HOSVD) for arbitrary-shaped tensors. We apply this method on 4-way weight tensors in convolutional layers, $W \in \mathbb{R}^{k \times k \times c \times f}$. We focus on Tucker-2 which only decomposes the tensor along c and f directions, i.e., output and input channels. For Tucker-2 decomposition, the compression parameter for each convolution layer is a tuple of integer-valued approximation ranks (r_1, r_2), where $r_1 \in \{1, \ldots, c\}$ and $r_2 \in \{1, \ldots, f\}$. To implement Tucker-2, we allocate two normalized and real-valued hyperparameters $rank_1 \in [\frac{1}{c}, 1]$ and $rank_2 \in [\frac{1}{f}, 1]$ per layer. We map these continuous values to valid integer ranks by using a similar approach as in SVD. The transformation $\mathcal{T}(M, \vec{x})$ then decomposes DNN layer weight kernels with their corresponding ranks in $\vec{x}$ following the practice shown in Fig. 17.

APPENDIX B

This section serves as a supplement for Section VII which provides implementation details and our experimental setup for training and fine-tuning of benchmarked DNNs.

Data Augmentation: For the CIFAR-10 dataset, we use standard data augmentation routines popular in prior work [48], [49]. The samples are normalized using per-channel mean and standard deviation. At training time, random horizontal mirroring, shifting, and slight rotation are also applied. For ImageNet, we use the augmentation scheme proposed in [50], [51] to pre-process input samples. During training, we resize the shorted edge of the image to 256 pixels. A 224×224 crop is then randomly sampled from the image. We also perform per-channel normalization as well as horizontal mirroring [52].

Training and Fine-tuning: We train the networks from scratch following the parameter setup and training schedule adopted by the original papers: [48] for ResNet-X on CIFAR-10 and ResNet-50 on ImageNet, [50] for VGG-16 on ImageNet, [52] for AlexNet on ImageNet, and [18], [19] for MobileNets on ImageNet. For CIFAR-10, we use a VGG-variant as implemented in [2]. We compress the models using *AdaNS* and fine-tune them for 20 and 60 epochs with batch sizes 64 and 128 on CIFAR-10 and ImageNet, respectively. SGD with momentum of 0.9 and Adam optimizer are used for fine-tuning CIFAR-10 and ImageNet, respectively. The fine-tuning learning rate is initialized to $1e-3$ and reduced by a factor of 10, 20 after 5, 15 epochs on ImageNet and after 20, 40 epochs on CIFAR-10. For MobileNet benchmarks the fine-tuning learning rate is initialized to $1e-4$ and reduced by a factor of 10 after 5, 10, and 15 epochs. Experiments are run on a machine with an Intel Xeon-E5 CPU and 4 Nvidia Titan Xp GPUs.

Implementation Details: Among the compression techniques studied in this paper, structured pruning needs especial attention. The pruned network's functionality is simulated using mask layers as discussed in Appendix A. For regular network architectures such as VGG and AlexNet, the mask layers utilized for structured pruning are placed after ReLU activations.

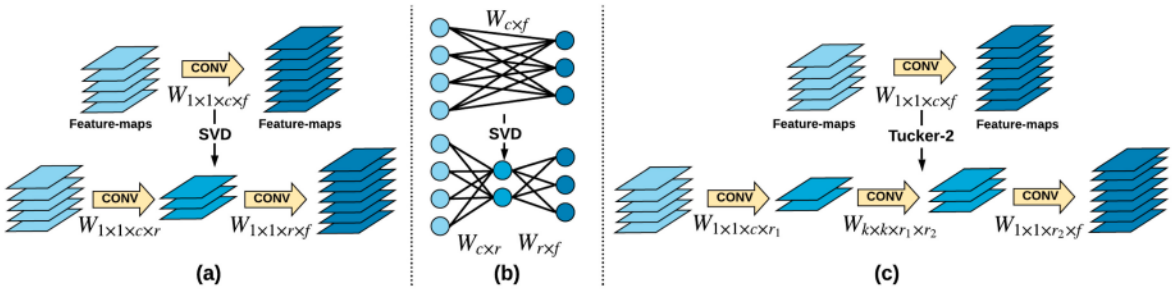

Fig. 17. Illustration of weight decomposition for common neural network layers. (a) Applying SVD to a point-wise convolution layer. (b) Applying SVD to a fully-connected layer. (c) Applying Tucker-2 on a convolution layer.

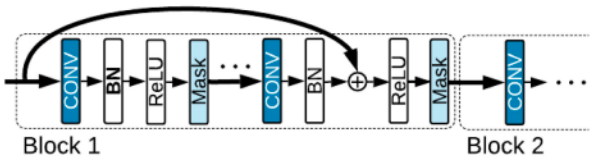

Fig. 18. Mask layers in ResNet residual blocks.

For residual blocks in ResNet architectures, we put the corresponding mask layer after adding the residual connection, as shown in Fig. 18. This approach accommodates different residual connections; in the case where the residual connection is performed using a convolution layer,[3] the succeeding mask also applies compression to the convolution inside the residual connection. For MobileNets, we place the masks after depth-wise separable convolutions, with no mask applied after the preceding point-wise convolution layer. Since there is a one to one mapping of channels in the depth-wise layer, pruning the output feature-maps of the depth-wise layer is equivalent to pruning the preceding point-wise convolution as well.

REFERENCES

[1] S. Lin *et al.*, "Accelerating convolutional networks via global & dynamic filter pruning," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2425–2432.
[2] C. Jiang *et al.*, "Efficient DNN neuron pruning by minimizing layer-wise nonlinear reconstruction error," in *Proc. Int. 27th Joint Conf. Artif. Intell.*, 2018, pp. 2298–2304.
[3] Y. He *et al.*, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proc. 27th Int. Joint Conf. Artificial Intell.*, 2018, pp. 2234–2240.
[4] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 1389–1397.
[5] H. Li *et al.*, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Represent.*, 2017.
[6] J. Wu *et al.*, "PocketFlow: An automated framework for compressing and accelerating deep neural networks," *NIPS Workshop CDNNRIA*, 2018.
[7] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 5058–5066.
[8] H. Wang *et al.*, "Structured probabilistic pruning for convolutional neural network acceleration," 2017, *arXiv:1709.06994*.
[9] J. Lin *et al.*, "Runtime neural pruning," in *Proc. Advances Neural Inf. Process. Syst*, 2017, pp. 2181–2191.
[10] M. Samragh *et al.*, "CodeX: Bit-flexible encoding for streaming-based FPGA acceleration of DNNs," 2019, *arXiv:1901.05582*.

[3]This is sometimes necessary to adjust the number of channels.

[11] K. Wang *et al.*, "HAQ: Hardware-aware automated quantization," in *Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2019, pp. 8612–8620.
[12] Y. He *et al.*, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 784–800.
[13] A. Elthakeb *et al.*, "ReLeQ: An automatic reinforcement learning approach for deep quantization of neural networks," in *Proc. NeurIPS ML Syst. Workshop*, 2018.
[14] M. Samragh *et al.*, "AutoRank: Automated rank selection for effective neural network customization," in *Proc. ML-for-Syst. Workshop 46th Int. Symp. Comput. Architecture*, 2019.
[15] M. Javaheripi *et al.*, "Peeking into the black box: A tutorial on automated design optimization and parameter search," *IEEE Solid-State Circuits Mag.*, vol. 11, no. 4, pp. 23–28, Fall 2019.
[16] T.-J. Yang *et al.*, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 285–300.
[17] C. Chen *et al.*, "Constraint-aware deep neural network compression," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 400–415.
[18] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[19] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 4510–4520.
[20] J. Yu, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.
[21] Z. Liu *et al.*, "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Representations*, 2018.
[22] P. Molchanov *et al.*, "Pruning convolutional neural networks for resource efficient transfer learning," in *Proc. Int. Conf. Learn. Represent.*, 2017.
[23] T. Salimans *et al.*, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.
[24] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 304–320.
[25] M. Javaheripi *et al.*, "SWNet: Small-world neural networks and rapid convergence," 2019, *arXiv:1904.04862*.
[26] S. Hussain, M. Javaheripi, P. Neekhara, R. Kastner, and F. Koushanfar, "FastWave: Accelerating autoregressive convolutional neural networks on FPGA," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2019, pp. 1–8.
[27] Y. Hu *et al.*, "A novel channel pruning method for deep neural network compression," 2018, *arXiv:1805.11394*.
[28] L. F. Yang and M. Wang, "Reinforcement leaning in feature space: Matrix bandit, kernels, and regret bound," 2019, *arXiv:1905.10389*.
[29] N. Srinivas *et al.*, "Information-theoretic regret bounds for gaussian process optimization in the bandit setting," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 3250–3265, May 2012.
[30] S. Shekhar *et al.*, "Gaussian process bandits with adaptive discretization," *Electron. J. Statist.*, vol. 12, no. 2, pp. 3829–3874, 2018.
[31] E. Hazan *et al.*, "Hyperparameter optimization: A spectral approach," in *Proc. Int. Conf. Learn. Represent.*, 2018.
[32] J. Haupt *et al.*, "Distilled sensing: Adaptive sampling for sparse detection and estimation," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 6222–6235, Sep. 2011.
[33] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Proc. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[34] H. Zhou *et al.*, "Less is more: Towards compact CNNs," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 662–677.

[35] Y.-D. Kim *et al.*, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.
[36] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. Boca Raton, FL, USA: CRC Press, 1997.
[37] E. K. Chong and S. H. Zak, *An Introduction to Optimization*, vol. 76. Hoboken, NJ, USA: Wiley, 2013.
[38] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.
[39] L. Xie and A. Yuille, "Genetic CNN," in *IEEE Int. Conf. Comput. Vision*, 2017, pp. 1379–1388.
[40] M. Javaheripi *et al.*, "GeneCAI: Genetic evolution for acquiring compact AI," 2020, *arXiv:2004.04249*.
[41] T. Hangelbroek and A. Ron, "Nonlinear approximation using Gaussian kernels," *J. Functional Anal.*, vol. 259, no. 1, pp. 203–219, 2010.
[42] K. Hamm, "Nonuniform sampling and recovery of bandlimited functions in higher dimensions," *J. Math. Anal. Appl.*, vol. 450, no. 2, pp. 1459–1478, 2017.
[43] Y. Ying and D.-X. Zhou, "Learnability of Gaussians with flexible variances," *J. Mach. Learn. Res.*, vol. 8, pp. 249–276, 2007.
[44] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. 7th Int. Conf. Pattern Recognit.*, 2004, pp. 28–31.
[45] E. Lukacs, "A characterization of the normal distribution," *Ann. Math.. Statist.*, vol. 13, no. 1, pp. 91–93, 1942.
[46] R. T. Birge, "The calculation of errors by the method of least squares," *Phys. Rev.*, vol. 40, no. 2, p. 207, 1932.
[47] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proc. Int. Conf. Comput. Vision*, Oct. 2019, pp. 1803–1811.
[48] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.
[49] G. Huang *et al.*, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 646–661.
[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015.
[51] K. He *et al.*, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 630–645.
[52] A. Krizhevsky *et al.*,"Imagenet classification with deep convolutional neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.