# Data-Independent Structured Pruning of Neural Networks via Coresets

Ben Mussay, Dan Feldman, Samson Zhou, Vladimir Braverman, Margarita Osadchy

***Abstract*—Model compression is crucial for deployment of neural networks on devices with limited computational and memory resources. Many different methods show comparable accuracy of the compressed model and similar compression rates. However, the majority of the compression methods are based on heuristics and offer no worst-case guarantees on the trade-off between the compression rate and the approximation error for an arbitrarily new sample.**

**We propose the first efficient structured pruning algorithm with a provable trade-off between its compression rate and the approximation error for any future test sample. Our method is based on the coreset framework and it approximates the output of a layer of neurons/filters by a coreset of neurons/filters in the previous layer and discards the rest. We apply this framework in a layer-by-layer fashion from the bottom to the top. Unlike previous works, our coreset is data independent, meaning that it provably guarantees the accuracy of the function for any input $x \in \mathbb{R}^d$, including an adversarial one.**

***Index Terms*—Model Compression; Network Pruning; Structured Pruning; Coreset**

## I. Introduction

Neural networks today are the most popular and effective instrument of machine learning with numerous applications in different domains. Since [29] used a model with 60M parameters to win the ImageNet competition in 2012, network architectures have been growing wider and deeper. The vast overparametrization of neural networks offers better convergence [2] and better generalization [45]. The downside of the overparametrization is its high memory and computational costs, which prevent the use of these networks in small devices, e.g., smartphones. Fortunately, it was observed that a trained network could be reduced to smaller sizes without much accuracy loss. Following this observation, many approaches to compress existing models have been proposed (see [15] for a recent review on network sparsification, and [44, 37, 59, 23, 60, 22, 58] for structured pruning).

Network compression is composed of two parts: 1) finding a more compact architecture (reducing the size of the layers or reducing the number of non-zero weights) and 2) updating the network parameters to approximate the original model. Most of the model compression methods search for a smaller architecture and weight approximation simultaneously [22, 60, 40]. A different approach is to decouple architecture search from the network approximation (e.g. [34, 41]). A lot of work has been done in neural architecture search [12]. One can follow this route for finding a smaller architecture [21, 52]. The second step of determining the weights of the small model can be done in several ways:

- Train the small architecture from scratch – the experiments in [41] showed that this could reach accuracy similar to the original large network.
- Retain $K$ most important weights/filters ($K$ is defined by the small architecture) and fine-tune the network (e.g. [34]).
- Find the weights of the small architecture that approximate the output of the original network and then fine-tune the weights (e.g., [23, 42]).

Even though many approaches show comparable results in terms of accuracy of the compressed model and its size, there are other important aspects that a practitioner should consider. For example, if the algorithm is tuned to a specific data (as in [44, 51, 27, 59, 4, 60]), the compressed network would fail to approximate the output of the original network on less typical unseen examples. In contrast, the accuracy of data-independent method (e.g. [34, 22]) is guaranteed for any input. Another important aspect of a compression framework is its construction time. Specifically, one-shot compression [34] is much faster than an iterative approach (e.g. [60]). Additionally, a well approximated model would require shorter fine-tuning time than a less

B. Mussay, D. Feldman, and M. Osadchy are with the Department of Computer Science, University of Haifa, Haifa 31905 Israel. E-mail: bengordoncshaifa@gmail.com, dannyf.post@gmail.com, rita@cs.haifa.ac.il

S. Zhou is with the Department of Computer Science, Carnegie Mellon University, Pittsburgh, IN., USA. E-mail: samsonzhou@gmail.com

V. Braverman is with the Department of Computer Science, Johns Hopkins University, Baltimore, MD., USA. E-mail: vova@cs.jhu.edu

accurate counterpart or training the compressed architecture from scratch. Obviously, the approximation of the original model is a function of the compression rate. However, previous works generally lack strong provable guarantees on the **trade-off between the compression rate and the approximation error**. The absence of worst-case performance analysis can potentially be a glaring problem depending on the application. Finally, there is a choice between model sparsification [31, 9] vs. structured pruning [27, 60, 34, 40]. The former, leads to an irregular network structure, which needs a special treatment to deal with sparse representations, making it hard to achieve actual computational savings. Structured pruning simply reduces the size of the tensors, providing very simple implementation and much higher computational savings. Thus structured pruning is much more practical.

Form the above discussion, we can derive the following requirements for the model compression framework: 1) it should provide provable guarantees on the trade-off between the compression rate and the approximation error, 2) it should be data-independent, 3) it should yield high compression rates, 4) it should be computationally efficient.

To address these goals, we propose an efficient framework with provable guarantees for structured pruning, which is based on the existing theory of coresets [7]. Coresets decrease massive inputs to smaller instances while maintaining a good provable approximation of the original set with respect to a given function. While most of previous work used coreset to reduce large volumes of data, here we apply it to reduce model parameters in neural networks by treating neurons as inputs in a coreset framework. Namely, we reduce the number of neurons in layer $i$ by constructing a coreset of neurons in this layer that provably approximates the output of neurons in layer $i+1$ and discarding the rest. The coreset algorithm is based on importance sampling of the neurons in layer $i$ that provides us with a subset of the neurons and with the new weights connecting these neurons to layer $i+1$. We apply similar concept to derive a filter pruning algorithm for the convolutional layers. The coreset algorithm is applied layer-wise from the bottom to the top of the network, resulting in a unified framework for approximating the entire network.

Our framework satisfies the aforementioned requirements:

**Provable guarantees:** The size of the coreset, and consequently the number of remaining neurons in layer $i$ is provably related to the approximation error of the output for every neuron in layer $i+1$. Thus, we can theoretically derive the trade-off between the compression rate and the approximation error of any layer in the neural network. We implement the direction, in which we construct a coreset of a predefined size (the size is specified by the target small architecture)[1] and provably derive the approximation error, which holds for any input. Most previous work lack such guarantees, and thus may have abnormal behaviour on some of the inputs. It was shown in [26] that pruning via recent methods affects generalization of a network at a class and exemplar level. Full and pruned models have comparable average accuracy, but they diverge considerably in behavior on specific sub-sets of input distribution. Our framework does not suffer such a degradation as it provably holds for *any* input.

**Data-independent compression:** The coreset approximation of neurons provably holds for any input, in particular, it holds for the input layer; thus the proposed framework will provably hold for any input data, termed data-independent compression.

**Fast construction:** Our framework performs one-shot structured pruning that seeks for the best approximation of the original network given the architecture of the small network. Thus it requires much shorter fine-tuning than heuristic-based approximation with no guarantees or training from scratch.

**High Compression Rates:** The proposed theory shows the existence of a small neural coreset. Our empirical results on LeNet-300-100 for MNIST [32] and on fully connected part of VGG16 [50] for CIFAR-10 [28] demonstrate that our framework based on coresets of neurons outperforms sampling-based coresets by improving compression without sacrificing the accuracy. The results on VGG19, ResNet56 [19] for CIFAR-10 and ResNet50 for ImageNet [8] demonstrate that our channel pruning method shows comparable results to the state-of-the-art methods of channel pruning, but is more efficient in terms of model construction.

## II. Related Work

State-of-the-art neural networks are often overparameterized, which causes a significant redundancy of weights. To reduce both computation time and memory requirements of trained networks, many approaches aim at removing this redundancy by model pruning. These works can be categorized based on different criteria, for instance, sparsification (reducing the number of non-zero weights) vs. structured pruning (reducing the width of the layer); data-dependent (trainable) vs. data independent (that works for any input); one-shot pruning

[1]The other direction, in which the size of the coreset is determined given the approximation error is theoretically possible, but is more complicated to implement.

vs. iterative pruning. Below, we discuss previous work following some of these categories.

### A. Sparsification vs. Structured Methods

**Sparsification**, also known as weight pruning, was considered as far back as 1990 [33], but has recently seen more study [9, 14, 15, 17, 48]. One of the most popular approaches is pruning via sparsity. Sparsity can be enforced by $L_1$ regularization to push weights towards zero during training [27]. However, it was observed [18] that after fine-tuning of the pruned network, $L_2$ regularized network outperformed $L_1$, as there is no benefit to pushing values towards zero compared to pruning unimportant (small weight) connections.

Sparsification methods showed high reduction in non-zero weights, e.g., the compression rate of AlexNet can reach $\times 35$ reduction with the combination of pruning, quantization, and Huffman coding [17]. The main drawback in weight pruning is that it leads to an irregular network structure, which needs a special treatment to deal with sparse representations, making it hard to achieve actual computational savings.

**Structured Pruning** (e.g., [27, 40, 60, 34, 37, 39, 35]) simply reduces the size of the tensors. In the case of CNNs it reduces the number of filters in a layer. These methods are preferable over sparsification, as the resulting models are very easy to use and they save an actual inference time.

The method in [34] measures the importance of channels by calculating the sum of absolute values of weights. Other channel pruning methods either impose channel-wise sparsity in training, followed by pruning channels with small scaling factors, and fine-tuning (e.g, [40, 55, 37]) or perform channel pruning by minimizing the reconstruction error of feature maps between the pruned and pre-trained model (e.g., [23].) The approach in [37, 60] seeks the best trade-off between loss minimization and filter selection. The method in [38] first prunes low saliency filters across all layers and then dynamically updates the filter saliency and recovers the mistakenly pruned filter. An attempt of global pruning was done in [39]; it employs GAN framework to make the output of the pruned and the baseline networks to follow the same distribution.

Our method belongs to structured pruning, and thus the resulting pruned networks provide large savings in computations and are easy to implement.

### B. Data-Dependent vs. Data-Independent

**Data-dependent** pruning approaches [40, 42, 43, 4, 60] utilize training/validation data to determine the filters to be pruned. The method in [27] first identifies weak neurons by analyzing their activation on a large validation dataset. Then those weak neurons are pruned and the network is retrained. The process is repeated several times. The method in [60] evaluates the contribution of filters to the discriminative power of the network by utilizing training/validation data. The method in [46] captures the dependency between channels in each layer using Taylor expansion and then exploits these dependencies to determine which channels to prune. The optimization of the channel selection problem requires evaluating Hessian on training samples. Knowledge Distillation [25] is another approach to model compression that uses data to fit the outputs of the small student network to the original teacher network.

**Data-independent** methods [34, 20, 56, 22, 35] compress neural network based on weights' characteristics. The work in [22] observes that keeping only the filters with the largest norm (as was done in [34, 20, 56]) does not provide a good approximation of the network and the distribution of norms in each layer. To resolve this problem, [22] suggests to search for filters with the most replaceable contribution. These are found by the means of Geometric Median.

To avoid using real data, [5, 57] replaced real data with a synthetic one in knowledge distillation approach. Since the synthetic data fits the distribution of the data set, pruned network could still suffer from the same instability problems as data-dependent approaches.

The relationship between an input feature map and its corresponding 2D kernels was studied in [35]. Based on this relationship, [35] proposed kernel sparsity and entropy (KSE) indicator for quantifying the importance of the input feature maps. KSE can handle every layer in parallel in a data-free manner. Kernel clustering is then used to quantize the kernels for CNN compression.

Data-independent pruning increases the robustness of the compressed network to future samples, unlike data-dependent pruning, which utilizes the training data. Our method is data-independent and it is theoretically guaranteed to maintain the same level of approximation on the out-of-the-distribution samples.

### C. Three Criteria: Accuracy, Compression, Efficient Construction

It was pointed out in [49], that network compression should be evaluated based on three criteria: accuracy, inference efficiency, and construction efficiency. Accuracy has been the main target in all previous work. As we discussed in the previous section, inference efficiency is not equivalent to the compression rate: Structured methods that provide the same compression rate as the

sparsification methods yield much higher computational savings in inference time.

The last criterion – construction efficiency, includes both the complexity of the pruning algorithm (for example one-shot methods are much less expensive than the iterative methods) and the amount of training or fine-tuning required. Most previous work have neglected this criterion. Furthermore, [41] observed that fine-tuning a pruned model gives a comparable performance with training that model with randomly initialized weights, concluding that pruning algorithms that assume a predefined target network architecture could be replaced with a direct training of the target architecture from scratch. We show that this suggestion has a profound drawback, which is the efficiency of producing the small network. Fine-tuning the approximated model is much faster than training the same architectures from random initialization. We show this empirically on several architectures in Section IV-C. Thus both architectural search and network approximation play an important role in network compression.

### D. Coresets

Our compression algorithm is based on a data summarization approach known as coresets. Over the past decade, coreset constructions have been recognized for high achievements in data reduction in a variety of applications, including $k$-means, SVD, regression, low-rank approximation, PageRank, convex hull, and SVM; see details in [47]. Many of the non-deterministic coreset based methods rely on the sensitivity framework, in which elements of the input are sampled according to their sensitivity [7, 53], which is used as a measure of their importance. The sampled elements are usually reweighted afterwards.

### E. Coreset-based Model Compression

Similar to our work, the approaches in [4, 36] use corests for model compression. However, [4] performs sparsification, while we offer structured pruning. Both [4, 36] compute the importance of weight/filter, using a validation set. The coreset is chosen for the specific distribution (of data) so consequently, the compressed model is data-dependent. In our construction, the input of the neural network is assumed to be an arbitrary vector in $\mathbb{R}^d$ and the sensitivity of a neuron is computed for every input in $\mathbb{R}^d$. This means that we create a data-independent coreset; its size is independent of the properties of the specific data at hand, and the compression provably approximates any future test sample.

In [11], $k$-means coresets were suggested to compress layers by adding a sparsity constraint. The weighting of the filters in the coreset was obtained based on their activation magnitudes over the training set. The compression pipeline also included a pre-processing step that followed a simple heuristic that eliminates filters based on the mean of their activation norms over the training set. This construction is obviously data-dependent and it uses coresets as an alternative mechanism for low-rank approximation of filters.

## III. Method

We propose an algorithm for compressing layer $i$ and we apply it to all layers from the bottom to the top of the network. We start with a necessary background on coresets (Section III-A). Next (Section III-B), we show how to approximate a single neuron, providing a theoretical analysis of the proposed construction. We then generalize this result to a fully connected layer (Section III-C). Finally, we extend our theoretical results to a convolution layer (Section III-D).

### A. Background

For any $\alpha > 0$, let $\mathbb{B}_\alpha(0)$ denote the ball of points in $\mathbb{R}^d$ with distance at most $\alpha$ from the origin.

**Definition 1** (Weighted set)**.** *Let $P \subset \mathbb{R}^d$ be a finite set, and $w$ be a function that maps every $p \in P$ to a* weight *$w(p) > 0$. The pair $(P, w)$ is called a* weighted set*.*

A coreset in this paper is applied to a query space, which consists of an input weighted set, an objective function and a class of models (queries) as follows.

**Definition 2** (Query space)**.** *Let $P' = (P, w)$ be a weighted set called the* input set*. Let $X \subseteq \mathbb{R}^d$ be a set of queries, and $f : P \times X \to [0, \infty)$ be a* loss function*. The tuple $(P, w, X, f)$ is called a* query space*.*

Given a set of points $P$ and a set of queries $X$, a coreset of $P$ is a weighted set of points that provides a good approximation to $P$ for any query $x \in X$. We state the definition of coresets with multiplicative guarantees below, though we shall also reference coresets with additive guarantees.

**Definition 3** ($\varepsilon$-coreset, multiplicative guarantee)**.** *Let $(P, w, X, f)$ be a query space, and $\varepsilon \in (0, 1)$ be an error parameter. An $\varepsilon$-coreset of $(P, w, X, f)$ is a weighted set $(C, u)$ such that for every $x \in X$*

$$\left| \sum_{p \in P} w(p) f(p, x) - \sum_{q \in C} u(q) f(q, x) \right| \leq \varepsilon \sum_{p \in P} w(p) f(p, x)$$

The size of our coresets depends on two parameters: the complexity of the activation function, which is defined below, and the sum of a supremum that is defined later. We now recall the well-known definition of VC dimension [54] using the variant from [13].

**Definition 4** (VC-dimension [13])**.** *Let $(P, w, X, f)$ be a query space. For every $x \in \mathbb{R}^d$, and $r \geq 0$ we define*

$$\text{range}_{P,f}(x, r) := \{p \in P \mid f(p, x) \leq r\}$$

*and*

$$\text{ranges}(P, X, f) := \left\{C \cap \text{range}_{P,f}(x, r) \mid C \subseteq P, x \in X, r \geq 0\right\}.$$

*For a set* ranges *of subsets of $\mathbb{R}^d$, the VC-dimension of $(\mathbb{R}^d, \text{ranges})$ is the size $|C|$ of the largest subset $C \subseteq \mathbb{R}^d$ such that*

$$|\{C \cap \text{range} \mid \text{range} \in \text{ranges}\}| = 2^{|C|}.$$

*The* VC-dimension of the query space $(P, X, f)$ *is the VC-dimension of* $(P, \text{ranges}(P, X, f))$.

The VC-dimension of all the query spaces that correspond to the activation functions in Table I is $O(d)$ [3].

The following theorem bounds the size of the coreset for a given query space and explains how to construct it. Unlike previous papers such as [13], we consider additive error and not multiplicative error.

**Theorem 5** (Coreset Construction [7])**.** *Let $d$ be the VC-dimension of a query space $(P, w, X, f)$. Suppose $s : P \to [0, \infty)$ such that $s(p) \geq w(p) \sup_{x \in X} f(p, x)$. Let $t = \sum_{p \in P} s(p)$, and $\varepsilon, \delta \in (0, 1)$. Let $c \geq 1$ be a sufficiently large constant that can be determined from the proof, and let $C$ be a sample (multi-set) of*

$$m \geq \frac{ct}{\varepsilon^2}\left(d \log t + \log\left(\frac{1}{\delta}\right)\right)$$

*i.i.d. points from $P$, where for every $p \in P$ and $q \in C$ we have $\text{pr}(p = q) = s(p)/t$. Then, with probability at least $1 - \delta$,*

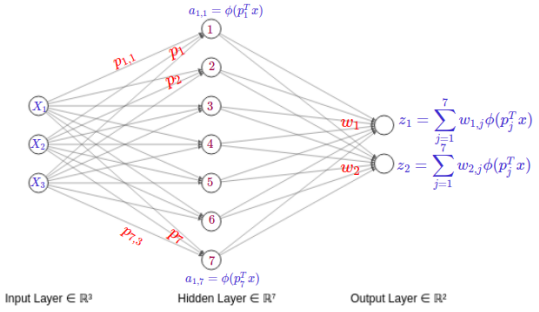
$$\forall x \in X : \left|\sum_{p \in P} w(p) f(p, x) - \sum_{q \in C} \frac{w(q)}{m \text{pr}(q)} \cdot f(q, x)\right| \leq \varepsilon.$$

### B. Data-Independent Coreset for a Single Neuron

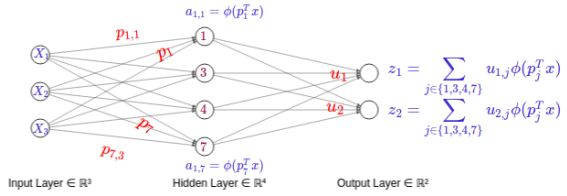
Let $a_j^i = \phi(p_j^T x)$ be the $j$th neuron in layer $i$, in which $p_j$ denotes its weights and $x$ denotes an arbitrary input in $\mathbb{R}^d$ (see Figure 1, top). We first consider an approximation of a single neuron in layer $i + 1$. The linear part of this neuron is $z = \sum_{j=1}^{|P|} w(p_j)\phi(p_j^T x)$. We would like to approximate $z$ by $\tilde{z} = \sum_{l \in J^*} u(p_l)\phi(p_l^T x)$ where $J^* \subset \{1, ..., |P|\}$ is a small subset, and we want this approximation to be bounded by a multiplicative factor that holds for any $x \in \mathbb{R}^d$. Unfortunately, our result in Theorem 6 shows that this idealized goal is impossible. However, we show in Theorem 7 and Corollary 8 that we can construct a small coreset $C$, such that $|z - \tilde{z}| \leq \varepsilon$ for any input $x \in \mathbb{R}^d$.

| Activation Function | Definition |
|---|---|
| ReLU | $\max(x, 0)$ |
| $\sigma$ | $\frac{1}{1+e^{-x}}$ |
| binary | $\begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| $\zeta$ | $\ln(1 + e^x)$ |
| soft-clipping | $\frac{1}{\alpha} \log \frac{1+e^{\alpha x}}{1+e^{\alpha(x-1)}}$ |
| Gaussian | $e^{-x}$ |

*TABLE I: Examples of activation functions $\phi$ for which we can construct a coreset of size $O(\frac{\alpha\beta}{\varepsilon^2})$ that approximates $\frac{1}{|P|}\sum_{p \in P} \phi(p^T x)$ with $\varepsilon$-additive error.*




(a)




(b)

*Fig. 1: Illustration of our neuron coreset construction on a toy example: (a) a full network, (b) the compressed network. Observe that a few neurons in layer 1 (the hidden layer) of (b) have been pruned. Both neurons in the second layer in (b) choose the same coreset comprising neurons $\{1, 3, 4, 7\}$ from layer 1, but with different weights. The compressed network has pruned neurons $\{2, 5, 6\}$ from layer 1.*

Algorithm 1 summarizes the coreset construction for a single neuron with an activation function $\phi$, (our results for common neural activation functions are summarized in Table I). To approximate a neuron in layer $i+1$, it samples $m$ (the size of the coreset) neurons in previous layer $i$ based on their sensitivity (formulated in Corollary 8) and updates their weights to maintain the approximation error (Theorem 5). The sensitivity of a neuron is a function of its weights norm, which is similar to the heuristic approaches, but we prove that by sampling $m$ neurons in layer $i$ based on their sensitivity, we bound the approximation error of the neuron in layer $i+1$.

**Algorithm 1:** CORESET$(P, w, m, \phi, \beta)$

**Input:** A weighted set $(P, w)$, an integer (sample size) $m \geq 1$, an (activation) function $\phi : \mathbb{R} \to [0, \infty)$, an upper bound $\beta > 0$.

**Output:** A weighted set $(C, u)$; see Theorem 7 and Corollary 8.

**for** *every* $p \in P$ **do**
  $\mathrm{pr}(p) := \dfrac{w(p)\phi(\beta \|p\|)}{\sum_{q \in P} w(q)\phi(\beta \|q\|)}$
  $u(p) := 0$
$C \leftarrow \emptyset$
**for** $m$ *iterations* **do**
  Sample a point $q$ from $P$ such that $p \in P$ is chosen with probability $\mathrm{pr}(p)$.
  $C := C \cup \{q\}$
  $u(q) := u(q) + \dfrac{w(q)}{m \cdot \mathrm{pr}(q)}$
**return** $(C, u)$

*1) Main Theoretical Results:* Most of the coresets provide a $(1+\varepsilon)$-multiplicative factor approximation for every query that is applied on the input set. The bound on the coreset size is independent or at least sub-linear in the original number $n$ of points, for any given input set. Unfortunately, the following theorem proves that it is impossible to compute small coresets for many common activation functions including ReLU. This holds even if there are constraints on both the length of the input set and the test set of samples.

**Theorem 6** (No coreset for multiplicative error)**.** *Let $\phi : \mathbb{R} \to [0, \infty)$ such that $\phi(b) > 0$ if and only if $b > 0$. Let $\alpha, \beta > 0$, $\varepsilon \in (0, 1)$ and $n \geq 1$ be an integer. Then there is a set $P \subseteq \mathbb{B}_\alpha(0)$ of $n$ points such that if a weighted set $(C, u)$ satisfies $C \subseteq P$ and*

$$\forall x \in \mathbb{B}_\beta(0) : \quad (1)$$
$$\left| \sum_{p \in P} \phi(p^T x) - \sum_{q \in C} u(q)\phi(q^T x) \right| \leq \varepsilon \sum_{p \in P} \phi(p^T x),$$

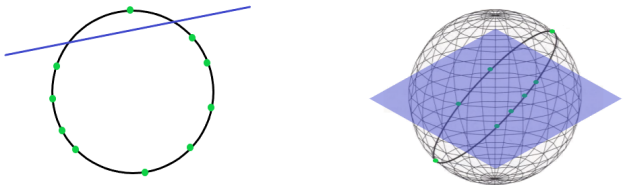


*Fig. 2:* ***(left)*** *Any point on a circle can be separated from the other points via a line.* ***(right)*** *The same holds for a circle which is the intersection of a $d$-dimensional sphere and a hyperplane; see Theorem 6.*

*then $C = P$.*

*Proof.* Consider the points on $\mathbb{B}_\alpha(0)$ whose norm is $\alpha$ and last coordinate is $\alpha/2$. This is a $(d-1)$-dimensional sphere $S$ that is centered at $(0, \cdots, 0, \alpha/2)$. For every point $p$ on this sphere there is a hyperplane that passes through the origin and separates $p$ from the rest of the points in $S$. Formally, by applying the Hyperplane separation theorem, there is an arbitrarily short vector $x_p$ (which is orthogonal to this hyperplane) such that $x_p^T p > 0$, but $x_p^T q < 0$ for every $q \in S \setminus \{p\}$; see Fig. 2. By the definition of $\phi$, we also have $\phi(x_p^T p) > 0$, but $\phi(x_p^T q) = 0$ for every $q \in S \setminus \{p\}$.

Let $P$ be an arbitrary set of $n$ points in $S$, and $C \subset P$. Hence exists a point $p \in P \setminus C$. By the previous paragraph,

$$\left| \sum_{q \in P} \phi(x_p^T q) - \sum_{q \in C} u(q)\phi(x_p^T q) \right| = \left| \phi(x_p^T p) - 0 \right|$$
$$= \phi(x_p^T p) = \sum_{q \in P} \phi(x_p^T q) > \varepsilon \sum_{q \in P} \phi(x_p^T q).$$

Therefore $C$ does not satisfy equation 1 in Theorem 6. □

The following theorem motivates the usage of additive $\varepsilon$-error instead of multiplicative $(1+\varepsilon)$ error. Fortunately, in this case there is a bound on the coreset's size for appropriate sampling distributions.

**Theorem 7.** *Let $\alpha, \beta > 0$ and $(P, w, \mathbb{B}_\beta(0), f)$ be a query space of VC-dimension $d$ such that $P \subseteq \mathbb{B}_\alpha(0)$, the weights $w$ are non-negative, $f(p, x) = \phi(p^T x)$ and $\phi : \mathbb{R} \to [0, \infty)$ is a non-decreasing function. Let $\varepsilon, \delta \in (0, 1)$ and*

$$m \geq \frac{ct}{\varepsilon^2}\left(d \log t + \log\left(\frac{1}{\delta}\right)\right)$$

*where*

$$t = \phi(\alpha\beta) \sum_{p \in P} w(p)$$

*and $c$ is a sufficiently large constant that can be determined from the proof.*

*Let $(C, u)$ be the output of a call to* CORESET$(P, w, m, \phi, \beta)$*; see Algorithm 1. Then, $|C| \leq m$ and, with probability at least $1 - \delta$,*

$$\left| \sum_{p \in P} w(p)\phi(p^T x) - \sum_{p \in C} u(p)\phi(p^T x) \right| \leq \varepsilon.$$

*Proof.* We want to apply Algorithm 1, and to this end we need to prove a bound that is independent of $x$ on the supremum $s$, the total supremum $t$, and the VC-dimension of the query space.

**Bound on $f(p, x)$:** Put $p \in P$ and $x \in \mathbb{B}_\beta(0)$. Hence,

$$f(p, x) = \phi(p^T x) \leq \phi(\|p\| \, \|x\|) \tag{2}$$
$$\leq \phi(\|p\| \, \beta) \tag{3}$$
$$\leq \phi(\alpha\beta), \tag{4}$$

where equation 2 holds by the Cauchy-Schwarz inequality and since $\phi$ is non-decreasing, equation 3 holds since $x \in \mathbb{B}_\beta(0)$, and equation 4 holds since $p \in \mathbb{B}_\alpha(0)$.

**Bound on the total sup $t$:** Using our bound on $f(p, x)$,

$$t = \sum_{p \in P} s(p) = \sum_{p \in P} w(p)\phi(\|p\| \, \beta) \leq \phi(\alpha\beta) \sum_{p \in P} w(p),$$

where the last inequality is by equation 4.

**Bound on the VC-dimension:** of the query space $(P, w, \mathbb{B}_\beta(0), f)$ is $O(d)$ as proved e.g. in [3].

**Putting all together:** By applying Theorem 5 with $X = \mathbb{B}_\beta(0)$, we obtain that, with probability at least $1 - \delta$,

$$\forall x \in \mathbb{B}_\beta(0) : \left| \sum_{p \in P} w(p) f(p, x) - \sum_{q \in C} u(q) f(q, x) \right| \leq \varepsilon.$$

Assume that the last equality indeed holds. Hence,

$$\forall x \in \mathbb{B}_\beta(0) : \left| \sum_{p \in P} w(p)\phi(p^T x) - \sum_{q \in C} u(q)\phi(q^T x) \right| \leq \varepsilon.$$

□

As weights of a neural network can take positive and negative values, and the activation functions $\phi : \mathbb{R} \to \mathbb{R}$ may return negative values, we generalize our result to include negative weights and any monotonic (non-decreasing or non-increasing) bounded activation function in the following corollary.

**Corollary 8.** *Let $(P, w, \mathbb{B}_\beta(0), f)$ be a general query space, of VC-dimension $O(d)$ such that $f(p, x) = \phi(p^T x)$ for some monotonic function $\phi : \mathbb{R} \to \mathbb{R}$ and $P \subseteq \mathbb{B}_\alpha(0)$. Let*

$$s(p) = \sup_{x \in X} |w(p)\phi(p^T x)|$$

*for every $p \in P$. Let $c \geq 1$ be a sufficiently large constant that can be determined from the proof, $t = \sum_{p \in P} s(p)$, and*

$$m \geq \frac{ct}{\varepsilon^2} \left( d \log t + \log\left(\frac{1}{\delta}\right) \right).$$

*Let $(C, u)$ be the output of a call to* CORESET$(P, w, m, \phi, \beta)$*; see Algorithm 1. Then, $|C| \leq m$ and, with probability at least $1 - \delta$,*

$$\forall x \in \mathbb{B}_\beta(0) : \left| \sum_{p \in P} w(p)\phi(p^T x) - \sum_{p \in C} u(p)\phi(p^T x) \right| \leq \varepsilon.$$

*Proof.* We assume that $\phi$ is a non-decreasing function. Otherwise, we apply the proof below for the non-decreasing function $\phi^* = -\phi$ and corresponding weight $w^*(p) = -w(p)$ for every $p \in P$. The correctness follows from $w(p)\phi(p^T x) = w^*(p)\phi^*(p^T x)$ for every $p \in P$.

Indeed, put $x \in \mathbb{B}_\beta(0)$, and $\phi$ non-decreasing. Hence,

$$\begin{aligned}
&\left| \sum_{p \in P} w(p)\phi(p^T x) - \sum_{p \in C} u(p)\phi(p^T x) \right| \\
&\leq \left| \sum_{\substack{p \in P \\ w(p)\phi(p^T x) \geq 0}} w(p)\phi(p^T x) - \sum_{\substack{p \in C \\ u(p)\phi(p^T x) \geq 0}} u(p)\phi(p^T x) \right| \\
&+ \left| \sum_{\substack{p \in P \\ w(p)\phi(p^T x) < 0}} w(p)\phi(p^T x) - \sum_{\substack{p \in C \\ u(p)\phi(p^T x) < 0}} u(p)\phi(p^T x) \right| \\
&\leq \left| \sum_{\substack{p \in P \\ w(p)\phi(p^T x) \geq 0}} w(p)\phi(p^T x) - \sum_{\substack{p \in C \\ u(p)\phi(p^T x) \geq 0}} u(p)\phi(p^T x) \right| \\
&+ \left| \sum_{\substack{p \in P \\ w(p)\phi(p^T x) < 0}} |w(p)\phi(p^T x)| - \sum_{\substack{p \in C \\ u(p)\phi(p^T x) < 0}} |u(p)\phi(p^T x)| \right|
\end{aligned}$$

The first inequality in the proof is obtained by separating each sum into points with positive and negative weights and applying Cauchy-Schwarz inequality. The second inequality is obtained by bounding points with positive and negative weights separately using Theorem 7. □

### C. From Coreset per Neuron to Coreset per Layer

Applying Algorithm 1 to each neuron in a layer $i + 1$ could result in the situation that a neuron in layer $i$ is selected to the coreset of some neurons in layer $i + 1$, but not to others. In this situation, it cannot be removed. To perform neuron pruning, every neuron in layer $i + 1$

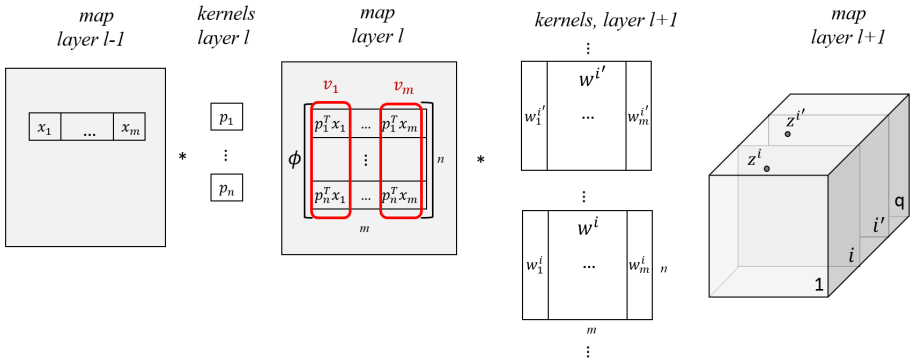



*Fig. 3: Illustration for the construction of channel pruning coreset. The filters in layer $l$ are one-dimensional for simplicity. The same construction applies for any tensor.*

should select the same neurons for its coreset, maybe with different weights. Thus, we wish to compute a single coreset for multiple weighted sets that are different only by their weight function. Each such a set represents a neuron in level $i+1$, which includes $k$ neurons. Algorithm 2 and Corollary 9 show how to compute a single coreset for multiple weighted sets. Figure 1 provides an illustration of the layer pruning on a toy example. The pruning can then be extended to the entire network using a layer-by-layer approach.

**Algorithm 2:** CORESET PER LAYER$(P, w_1, \cdots, w_k, m, \phi, \beta)$

**Input:** Weighted sets $(P, w_1), \cdots, (P, w_k)$, an integer (sample size) $m \geq 1$, an (activation) function $\phi : \mathbb{R} \to [0, \infty)$, an upper bound $\beta > 0$.

**Output:** A weighted set $(C, u)$; see Theorem 7.

**for** *every* $p \in P$ **do**
  $\text{pr}(p) := \dfrac{\max_{i\in[k]} w_i(p)\phi(\beta \|p\|)}{\sum_{q\in P} \max_{i\in[k]} w_i(q)\phi(\beta \|q\|)}$
  $u(p) := 0$
$C \leftarrow \emptyset$
**for** $m$ *iterations* **do**
  Sample a point $q$ from $P$ such that $p \in P$ is chosen with probability $\text{pr}(p)$.
  $C := C \cup \{q\}$
  $\forall i \in [k] : u_i(q) := u_i(q) + \dfrac{w_i(q)}{m \cdot \text{pr}(q)}$
**return** $(C, u_1, \cdots, u_k)$

**Corollary 9** (Coreset per Layer)**.** *Let $(P, w_1, \mathbb{B}_\beta(0), f), \ldots, (P, w_k, \mathbb{B}_\beta(0), f)$ be $k$ query spaces, each of VC-dimension $O(d)$ such that $f(p, x) = \phi(p^T x)$ for some non-decreasing $\phi : \mathbb{R} \to [0, \infty)$ and $P \subseteq \mathbb{B}_\alpha(0)$. Let*

$$s(p) = \max_{i\in[k]} \sup_{x\in X} w_i(p)\phi(p^T x)$$

*for every $p \in P$. Let $c \geq 1$ be a sufficiently large constant that can be determined from the proof, $t = \sum_{p\in P} s(p)$*

$$m \geq \frac{ct}{\varepsilon^2}\left(d \log t + \log\left(\frac{1}{\delta}\right)\right).$$

*Let $(C, u_1, \cdots, u_k)$ be the output of a call to* CORESET$(P, w_1, \cdots, w_k, m, \phi, \beta)$*; see Algorithm 2. Then, $|C| \leq m$ and, with probability at least $1 - \delta$,*

$$\forall i \in [k], x \in \mathbb{B}_\beta(0) : \left|\sum_{p\in P} w_i(p)\phi(p^T x) - \sum_{p\in C} u_i(p)\phi(p^T x)\right| \leq \varepsilon.$$

The proof follows directly from the observation in Theorem 5 that $s(p) \geq w(p) \sup_{x\in X} f(p, x)$.

One can theoretically bound the approximation error of the entire network by aggregating the error over its layers similarly to [36]. This can be done for fully connected networks (Algorithm 2) and for convolutional networks (discussed in Section III-D).

### *D. From Neural Pruning to Channel Punning in CNNs*

Let $x_1, \ldots, x_m \in \mathbb{R}^d$ denote $m$ adjacent segments in the map of layer $l-1$ (see illustration in Figure 3). Let $p_1, \ldots, p_n \in \mathbb{R}^d$ be a set of kernels in convolutional layer $l$ and let $w^1, \ldots, w^q \in \mathbb{R}^{m\times n}$ be a set of kernels in convolutional layer $l+1$. Let $v_j := [p_1^T x_j, \ldots, p_n^T x_j]$ be a linear part of the map in layer $l$ that corresponds to the input segment $x_j$ (see Figure 3 for illustration) and let

$v$ denote a concatenation of $v_1, \ldots, v_m$. Consequently, $\phi(v)$ is $m \times n$ patch in the map of layer $l$, which is obtained by applying $n$ kernels of layer $l$ to input $x_1, \ldots, x_m$.

We start by approximating the linear part of element $z^i$ in channel $i$ of the map in layer $l+1$ using a coreset of kernels in layer $l$. Under the above notations,

$$z^i = \langle w^i, \phi(v) \rangle = \sum_{j=1}^{m} (w_j^i)^T \phi(v_j) \tag{5}$$
$$= \sum_{j=1}^{m} \left( \sum_{k=1}^{n} w_{jk}^i \phi(p_k^T x_j) \right),$$

where $w^i$ is the corresponding kernel in layer $l+1$. Theorem 7 and Corollary 8 show how to approximate each of $\sum_{k=1}^{n} w_{jk}^i \phi(p_k^T x_j)$, but these approximations should use the same subset of kernels $\{p_k\}$. To this end, we define $m$ weighted sets $(P, w_1^i), \ldots, (P, w_m^i)$, where $P = \{p_1, \ldots, p_n\}$ and compute a single coreset for them using Algorithm 2. We note that here each weighted set operates on a different query $x_j$. Since coreset holds for any query, we can still use Algorithm 2 to find a single coreset of $P$ to approximate $z^i$.

Next, we consider another element $\tilde{z}^i$ in the same channel $i$ of the linear part of the map in layer $l+1$. Similarly to Eq. 5,

$$\tilde{z}^i = \sum_{j=1}^{m} \left( \sum_{k=1}^{n} w_{jk}^i \phi(p_k^T \tilde{x}_j) \right),$$

where $\tilde{x}_j$, $j = 1, \ldots, m$ is a different part of the map in layer $l-1$. $\tilde{z}^i$ differs from $z^i$ only in the query, thus we can define the same weighted sets for $z^i$ and $\tilde{z}^i$. Same argument applies to all elements in the channel, thus same coreset will hold for approximating every element in channel $i$ of the map of layer $l+1$.

Finally, we consider an element in a different channel $i'$ of the map in layer $l+1$. Its linear part is defined as follows,

$$z^{i'} = \sum_{k=1}^{n} w_{jk}^{i'} \phi(p_k^T x_j)$$

and its approximation must choose the same subset of kernels $\{p_k\}$ as the approximation of elements in channel $i$. This case is similar to approximating multiple neurons proposed in Section III-C. Following the derivation for dense layer pruning, we define weighted sets for each channel $i = 1, \ldots, q$ as $\{(P, w_1^i), \ldots, (P, w_m^i)\}$ and apply Algorithm 2 to the union of the weighted sets associated with all channels in the map: $\bigcup_{i=1}^{q} \{(P, w_1^i), \ldots, (P, w_m^i)\}$.

## IV. Experiments

Our first set of experiments focuses on compression of dense layers. We start with the analysis of the proposed *Coreset for Neuron* algorithm and then compare the performance of the *Coreset per Layer* algorithm with other coreset-based methods. We then continue with testing our coreset-based framework for dense layers on benchmark architectures and data sets. We conclude this part of the experiments with an ablation analysis that evaluates the contribution of different parts of our framework.

The second set of experiments focuses on channel pruning. We test our coreset-based channel pruning on the benchmark architectures and data sets and compare it to previous methods that use heuristics for model approximation. Then, we compare the speed of the fine-tuning that updates the model approximated using our coreset-based channel pruning with training the same architecture from scratch and with other structured pruning methods. This experiment demonstrates the affect of accurate approximation on the construction time. Finally, we test our theoretical approximation guarantees on inputs from adversarial distribution in comparison to a heuristic method for structured pruning that lacks these guarantees.

In all experiments $\beta$, which is an upper bound on the norm of queries is set to one, since input images are normalized.

### A. Coreset per Neuron

*1) Approximation error vs. Coreset Size:* We analyzed the empirical trade-off between the approximation error of ReLU neuron and the size of its coreset, constructed by Algorithm 1 and Corollary 8. We compared our coreset to *uniform sampling*, which also implements Algorithm 1, but sets the probability of a point to $1/n$ ($n$ is the size of the full set), and to *percentile*, which deterministically retains the inputs with the highest norms (note that in percentile the points are not weighted). We ran three tests, varying the distribution of weights. In the first and second tests (Figure 4, (a) and (b)) the weights were drawn from the Gaussian and Uniform distributions respectively. The total number of neurons was set to 1000. We selected subsets of neurons of increasing sizes from 50 to 1000 with a step of 50. In the third test (Figure 4, (c)) we used the trained weights from the first layer of Lenet-300-100 including 300 neurons (LeNet-300-100 network comprises two fully connected hidden layers with 300 and 100 neurons correspondingly and ReLU activations, trained on MNIST [32] train set). We varied the coreset

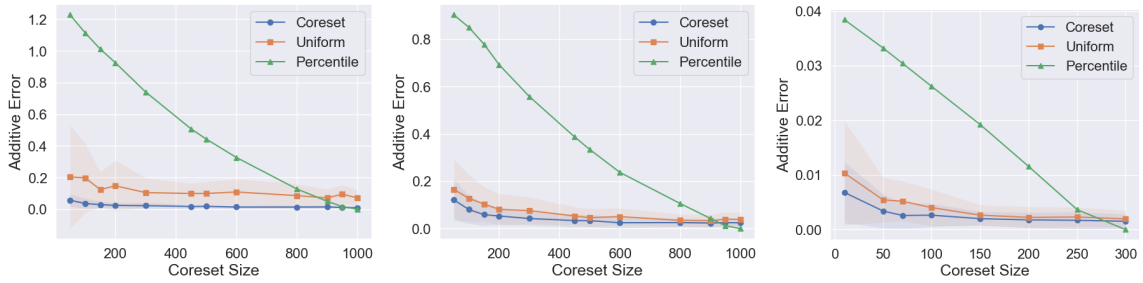



*Fig. 4: Approximation error of a single neuron on MNIST dataset across different coreset sizes. The weights of the points in (a) are drawn from the Gaussian distribution, in (b) from the Uniform distribution and in (c) we used the trained weights from LeNet-300-100. Our coreset, computed by Algorithm 1 and Corollary 8, outperforms other reduction methods.*

size from 50 to 300 with a step 50. To evaluate the approximation error, we used images from MNIST test set as queries. Each point in the plot was computed by 1) running the full network and the compressed network (with corresponding compression level) on each image $x$ in the test set, 2) computing additive approximation error $\left|\sum_{p\in P} w(p)\phi(p^T x) - \sum_{p\in C} u(p)\phi(p^T x)\right|$, 3) averaging the resulting error over the test set. In all three tests, our coresets outperformed the tested methods across all coreset sizes.

*2) Comparison with Other Coreset-Based Methods:* We compared the average approximation error vs. compression rates of our neural pruning coreset with several other well-known algorithms listed blow.

- **Baselines:** uniform sampling, percentile (which deterministically retains the inputs with the highest norms), and Singular Value Decomposition (SVD);
- **Schemes for matrix sparsification:** based on L1 and L2 norms and their combination [10, 1, 30];
- **Sensitivity sampling:** CoreNet and CoreNet++ [4].

The tests were performed on LeNet-200-105 architecture, which includes two fully connected hidden layers with 200 and 105 neurons correspondingly and ReLU activations, trained on MNIST [32]. We computed the average error of the tested algorithms over the samples of the MNIST test set after performing each test ten times. We measured the corresponding average approximation error as defined in [4]:

$$error_{\mathcal{P}_{test}} = \frac{1}{\mathcal{P}_{test}} \sum_{x\in\mathcal{P}_test} \|\phi_{\hat{\theta}}(x) - \phi_{\theta}(x)\|_1,$$

where $\phi_{\hat{\theta}}(x)$ and $\phi_{\theta}(x)$ are the outputs of the approximated and the original networks respectively. The results are summarized in Figure 5. As expected, all algorithms perform better with lower compression, but our algorithm outperforms other methods, especially for high compression rates.

| Network | Error(%) | # Parameters | Compression Ratio |
|---|---|---|---|
| LeNet-300-100 | 2.16 | 267K | |
| LeNet-300-100 Pruned | **2.03** | **26K** | **90%** |
| VGG16 | 8.95 | 1.4M | |
| VGG16 Pruned | **8.16** | **350K** | **75%** |

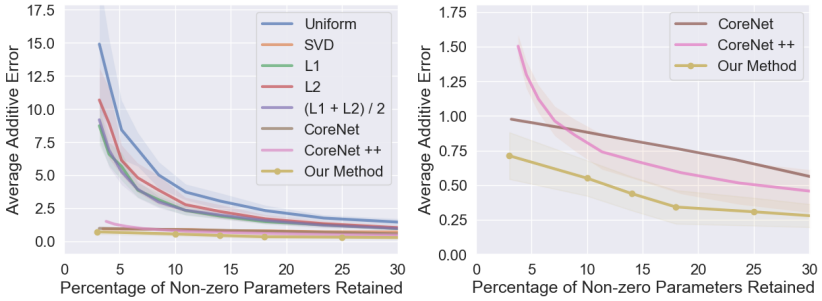
*TABLE II: Empirical evaluations of our coresets on existing architectures for MNIST and CIFAR-10. Note the improvement of accuracy in both cases.*

### *B. Neural Pruning via Coreset per Layer*

We tested the proposed framework for neural pruning via coresets on dense layers of two popular models: LeNet-300-100 (fully-connected network described above) on MNIST [32], and VGG16 [50] on CIFAR-10 [28]. In addition to the convolutional and pooling layers, VGG16 [50] includes 3 dense layers – the first two with 4096 neurons and the last with 1000 neurons (we applied our algorithm for neural pruning to the dense layers). In both networks, we first applied neural pruning using Coreset per Layer (Algorithm 2), and then fine-tuned the remaining weights until convergence.

Our method was able to prune roughly $90\%$ of the parameters of LeNet-300-100 network without any accuracy loss – in fact, it slightly improved the classification accuracy. After compressing the dense layers of VGG16 network by roughly $75\%$, its accuracy also showed slight improvement. We summarize our findings in Table II.

*1) Ablation Analysis:* The proposed compression framework includes for every layer a selection of neu-




(a) (b)

*Fig. 5: Average accuracy of various algorithms on LeNet-200-105 on MNIST dataset across different sparsity rates. Plot (a) shows the results of all tested methods. Plot (b) focuses on the three top methods. Our method, which constructs neural coresets by applying the Algorithm 2 and Corollary 8 in a layer-by-layer fashion, outperforms other coreset-based algorithms.*

rons using Algorithm 2 followed by fine-tuning. We performed the following ablation analysis to evaluate the contribution of different parts of our framework on LeNet-300-100 trained on MNIST. First, we removed the fine-tuning, to test the improvement due to Algorithm 2 over the uniform sampling. Figure 6, (a) shows the classification accuracy without fine-tuning as a function of the compression rate. Figure 6, (b) shows that fine-tuning improves both methods, but the advantage of the coreset is still apparent across almost all compression rates and it increases at the higher compression rates. Note that the model selected by the coreset can be fine-tuned to 98% classification accuracy for any compression rate, while the model chosen uniformly cannot maintain the same accuracy for high compression rates.

These results demonstrate that our coreset algorithm provides better approximation than the uniform sampling. Moreover, it requires significantly less fine-tuning: fine-tuning until convergence of the uniform sampling took close to 2 epochs, while fine-tuning of our method required about half of that time.

### C. Channel Pruning

We ran our experiments on three benchmark models: VGG [50] and ResNet56 [19] on CIFAR-10 [28], and ResNet50 [19] on ILSVRC2012 [8]. In all our experiments, we applied the proposed coreset framework to approximate the original network using the small target architecture. Specifically, we compressed the convolutional layers of the original network using the coreset for channel pruning (detailed in Section III-D) to the size, defined by the target small architecture. The results below show that channel pruning via coresets is comparable with the prior work in terms of the size and accuracy of the final model, but the compressed network provides better approximation of the original network, consequently it requires less fine-tuning, and this approximation holds for any input.

*1) Compressing VGG19:* We used Pytorch implementation[2] of VGG19 network for CIFAR10 from [40] with about 20M parameters as our baseline model. The target architecture of the small network[3] corresponds to 70% compression ratio and to the reduction of the parameters by roughly $88\%$. The results are summarized in Table III.

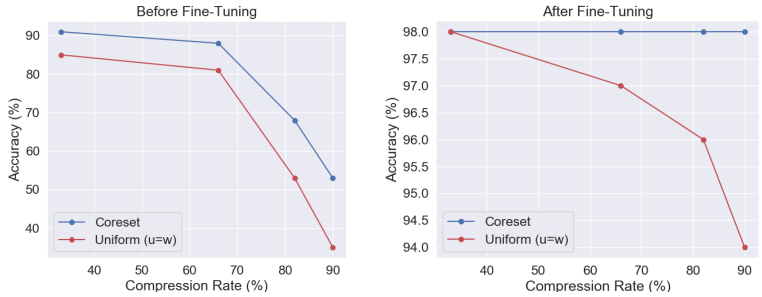
The approximation of the baseline VGG19 using coresets before fine-tuning provided 12% improvement of accuracy compared to [40]. Consequently, our approximation needs less fine-tuning time to reach the original (or even improved) accuracy (see Section IV-D).

| Pruning Method | Baseline Error(%) | Small Model Error(%) | Compression Ratio |
|---|---|---|---|
| Unstructured Pruning [18] | 6.5 | 6.48 | 80% |
| Structured Pruning [40] | 6.33 | 6.20 | 70% |
| Pruning via Coresets | 6.33 | 6.02 | 70% |

*TABLE III: Pruning of VGG19 for CIFAR-10*

*2) Compressing ResNets:* We tested our method on two models: ResNet56 for CIFAR-10 and ResNet50 on ILSVRC-2012. The baseline ResNet56 [19] includes 56 convolutional layers with batch-normalization and dense layer with 64 neurons, in total about 860K parameters. The baseline ResNet50 [19] includes 50 convolutional layers with batch-normalization and dense layer with

[2]https://github.com/Eric-mingjie/network-slimming
[3]https://github.com/foolwood/pytorch-slimming




(a) (b)

*Fig. 6: Average accuracy over 5 runs of the proposed framework and the uniform baseline on LeNet-300-100 on MNIST dataset across different compression rates. Plot (a) shows the results before fine-tuning, plot (b)-after fine-tuning. The fine-tuning was done until convergence. Fine-tuning of the uniform sampling almost doubles in time compared to the fine-tuning of the coreset.*

2048 neurons, in total about 26M parameters. We applied the same 40% compression to every convolutional layer using the coreset algorithms for channel pruning in both networks. We then fine-tuned the compressed network. We decreased the number of parameters in ResNet56 by roughly 55% and in ResNet50 by roughly 62%. We compare our results for ResNet56 with previous methods in Table IV and for ResNet50 in Table V. Our algorithm shows comparable accuracy and compression ratio with previous methods, but it has the advantage of simple and efficient construction and theoretical guarantees for any input.

| Pruning Method | Baseline Error(%) | Small Model Error(%) | Compression Ratio |
|---|---|---|---|
| Channel Pruning [24] | 7.2 | 8.2 | 39% |
| AMC [21] | 7.2 | 8.1 | 40% |
| Soft Pruning [20] | 6.41 | 6.65 | 40% |
| CCP [46] | 6.5 | 6.42 | 40% |
| Pruning via Coresets | 6.21 | 7.0 | 40% |

*TABLE IV: Channel Pruning of ResNet56 (with 860K parameters) on CIFAR-10*

### *D. The value of approximation in channel pruning*

Recently, [41, 48] challenged some common beliefs about fine-tuning of compressed models and how it should be done. The claim in [41] was that when using a predefined architecture in structured compression, training the small architecture from random initialization is the best solution. Thus the only question one need to ask in network compression is what is the compressed architecture. Even though fully trained network could

| Pruning Method | Baseline Top-1 Err.(%) | Small Model Top-1 Err.(%) | Compression Ratio |
|---|---|---|---|
| Soft Pruning [20] | 23.85 | 25.39 | 30% |
| CCP [46] | 23.85 | 24.5 | 35% |
| FPGM [22] | 23.85 | 25.17 | 40% |
| ThiNet-70 [42] | 27.72 | 26.97 | 30% |
| ThiNet-50 [42] | 27.72 | 28.0 | 50% |
| Pruning via Coresets | 23.87 | 25.11 | 40% |

*TABLE V: Channel Pruning of ResNet50 (with 26M parameters) on ILSVRC-2012*

match the accuracy of a fine-tuned network, there is an advantage of the approximated network in terms of construction time: fine-tuning a well-approximated network is significantly faster than training the same architecture from scratch. The plots showing network accuracy over fine-tuning (Figure 7) support our claim for all tested architectures.

Knowledge distillation methods are comparable with most existing compression techniques, but the training process of knowledge distillation is particularly slow (as detailed in [6]) and often takes nearly as long as training from scratch [25]. Our coreset construction is done by a simple algorithm that takes less than a minute for a very large VGG19 network and requires only very short fine-tuning(Figure 7). Thus our approach is much more efficient than methods based on knowledge distillation.

Furthermore, our algorithm offers better approximation of the original network than methods based on heuristic and thus it require less fine-tuning time. We compare the fine-tuning progress of our coreset-based pruning with two representative approaches: 1) one-shot pruning followed by fine-tuning [40] and 2) iterative

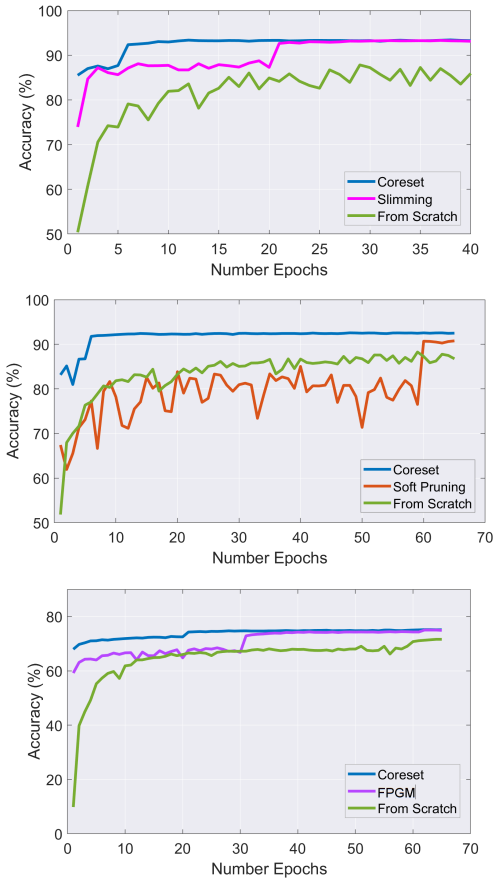



*Fig. 7: Accuracy of the proposed framework in comparison to previous methods and to training the pruned network from scratch. Plot (a) shows the results for VGG19 with CIFAR-10 in comparison to [40], plot (b) shows the results for ResNet56 with CIFAR-10 in comparison to [20] and plot (c) shows the results for ResNet50 with ImageNet in comparison to [22].*

pruning and fine-tuning [20, 22]. The results of Network Slimming [40] on VGG19, Soft Pruning [20] on ResNet56 and FPGM [22] on ResNet50 are shown in Figure 7a-c respectively in comparison to our method and to training the small architecture from scratch. We observe that fine-tuning these methods is much slower than ours in all cases.

### *E. Generalization of the Pruned Network*

Our theory bounds the approximation error of each layer in the small network w.r.t. to the corresponding layer in the original network for any input. In the following experiment we test the generalization of this approximation to inputs from adversarial distribution. We added adversarial noise produced by FGSM method [16] with increasing magnitude to all images from the test set of CIFAR-10. The noise was produced for each tested network (in a white-box attack setting). We tested the original VGG19 network, the 70% pruned network via coresets and 70% pruned network form [40]. All these networks are prone to adversarial examples, since no protection has been used. To cause slow degradation in accuracy of the original VGG19, we increased the magnitude of the adversarial noise in very small steps. This way, we shifted the distribution of inputs gradually from the original one. Figure 8 shows that our coreset-based pruning is closer to the original network and it maintains the same degradation level (where there is a degradation), which corresponds to the approximation error bound. The heuristic method from [40], shows the same performance for the original distribution of inputs, but it degrades much faster with each distribution shift and finally drops to the chance-level.

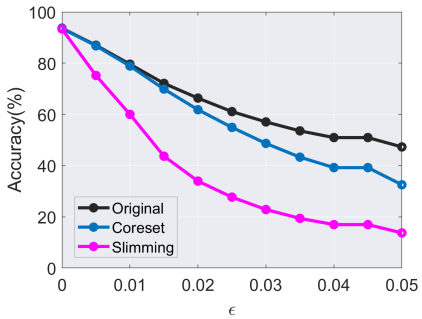



*Fig. 8: Comparison of our coreset-based pruning of VGG19 for CIFAR-10 with a heuristic pruning [40] on adversarial examples with increasing magnitude of noise.*

## V. Conclusion

We proposed the first structured pruning algorithm with provable trade-off between the compression rate to the approximation error for any future test sample. We base our compression algorithm on coreset framework and construct coresets for most common activation functions. Our tests on ReLU dense networks and popular CNN architectures show high compression rates with comparable accuracy, and our theory guarantees the worst case accuracy vs. compression trade-off for any future test sample, even an adversarial one. The construction time of the proposed framework including fine-tuning is very fast, rendering our method very attractive to practitioners. In future work, we plan to derive coreset of a full network and extend it to other architectures.

## REFERENCES


[1] D. Achlioptas, Z. Karnin, and E. Liberty. Matrix entry-wise sampling: Simple is best, 2013.

[2] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *ICML*, pages 242–252, 2019.

[3] M. Anthony and P. L. Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[4] C. Baykal, L. Liebenwein, I. Gilitschenski, D. Feldman, and D. Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *CoRR*, abs/1804.05345, 2018.

[5] K. Bhardwaj, N. Suda, and R. Marculescu. Dream distillation: A data-independent model compression framework. *CoRR*, abs/1905.07072, 2019.

[6] C. Blakeney, X. Li, Y. Yan, and Z. Zong. Parallel blockwise knowledge distillation for deep neural network compression. *CoRR*, abs/2012.03096, 2020.

[7] V. Braverman, D. Feldman, and H. Lang. New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889, 2016.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. Imagenet: A large-scale hierarchical image database. *CVPR*, pages 248–255, 2009.

[9] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*, pages 4857–4867, 2017.

[10] P. Drineas and A. Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111(8):385–389, 2011.

[11] A. Dubey, M. Chatterjee, and N. Ahuja. Coreset-based neural network compression. In *ECCV*, pages 469–486, 2018.

[12] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019.

[13] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578, 2011.

[14] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018.

[15] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[17] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[18] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28*, pages 1135–1143, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2015.

[20] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018.

[21] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

[22] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. *CVPR*, pages 4335–4344, 2018.

[23] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406, 2017.

[24] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *ICCV*, pages 1398–1406, 2017.

[25] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[26] S. Hooker, A. Courville, Y. Dauphin, and A. Frome. What does a pruned deep neural network forgets? *Bridging AI and Cognitive Science ICLR Workshop*, 2020.

[27] H. Hu, R. Peng, Y. Tai, and C. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250, 2016.

[28] A. Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] A. Kundu and P. Drineas. A note on randomized element-wise matrix sparsification. *CoRR*, abs/1404.0320, 2014.

[31] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, pages 2554–2564, 2016.

[32] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[34] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016.

[35] Y. Li, S. Lin, B. Zhang, J. Liu, D. S. Doermann, Y. Wu, F. Huang, and R. Ji. Exploiting kernel sparsity and entropy for interpretable CNN compression. In *CVPR*, pages 2800–2809, 2019.

[36] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

[37] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(2):574–588, 2020.

[38] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In J. Lang, editor, *IJCAI*, pages 2425–2432, 2018.

[39] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. S. Doermann. Towards optimal structured CNN pruning via generative adversarial learning. In *CVPR*, pages 2790–2799, 2019.

[40] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. *ICCV*, pages 2755–2763, 2017.

[41] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[42] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *ICCV*, pages 5068–5076, 2017.

[43] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *ArXiv*, abs/1611.06440, 2016.

[44] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989.

[45] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *CoRR*, abs/1805.12076, 2018.

[46] H. Peng, J. Wu, S. Chen, and J. Huang. Collaborative channel pruning for deep networks. In *ICML*, pages 5113–5122, 2019.

[47] J. M. Phillips. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

[48] A. Renda, J. Frankle, and M. Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *ICLR*, 2020.

[49] A. Renda, J. Frankle, and M. Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *ICLR*, 2020.

[50] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[51] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[52] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pages 2820–2828, 2019.

[53] E. Tolochinsky and D. Feldman. Coresets for monotonic functions with applications to deep learning. *CoRR*, abs/1802.07382, 2018.

[54] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[55] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29*, pages 2074–2082, 2016.

[56] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018.

[57] H. Yin, P. Molchanov, J. M. Alvarez, Z. Li, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *CVPR*, pages 8712–8721. IEEE, 2020.

[58] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *ArXiv*, abs/1909.08174, 2019.

[59] R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis. NISP: pruning networks using neuron importance score propagation. In *CVPR 2018*, pages 9194–9203, 2018.

[60] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo,

Q. Wu, J. Huang, and J.-H. Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018.