Pruning Neural Networks via Coresets and Convex Geometry: Towards No Assumptions

Murad Tukan[*†] muradtuk@gmail.com
Loay Mualem[*†] loaymua@gmail.com
Alaa Maalouf[*†] alaamaalouf12@gmail.com

Abstract

Pruning is one of the predominant approaches for compressing deep neural networks (DNNs). Lately, coresets (provable data summarizations) were leveraged for pruning DNNs, adding the advantage of theoretical guarantees on the trade-off between the compression rate and the approximation error. However, coresets in this domain were either data-dependent or generated under restrictive assumptions on both the model's weights and inputs. In real-world scenarios, such assumptions are rarely satisfied, limiting the applicability of coresets. To this end, we suggest a novel and robust framework for computing such coresets under mild assumptions on the model's weights and without any assumption on the training data. The idea is to compute the importance of each neuron in each layer with respect to the output of the following layer. This is achieved by a combination of Löwner ellipsoid and Caratheodory theorem. Our method is simultaneously data-independent, applicable to various networks and datasets (due to the simplified assumptions), and theoretically supported. Experimental results show that our method outperforms existing coreset based neural pruning approaches across a wide range of networks and datasets. For example, our method achieved a 62% compression rate on ResNet50 on ImageNet with 1.09% drop in accuracy.

1 Introduction and Backround

Deep neural networks (DNNs) achieved state-of-the-art (SOTA) performance on a large variety of tasks, e.g., in computer vision [33, 50] and natural language processing (NLP; [87, 20]). However, DNNs usually contain millions or even billions of parameters in order to achieve SOTA performances resulting in large storage requirements and long inference time. This is obstructive when, e.g., dealing with limited hardware or real-time systems such as autonomous cars and text/speech translation. To this end, a large body of research is dedicated to reducing the size and inference costs of DNNs.

Pruning. A dominant approach widely used for reducing the size of DNNs is to utilize a pruning algorithm to remove redundant parameters from the original, over-parameterized network. In general, pruning can be categorized into two main types: (i) Unstructured pruning [31, 6] reduces the number of non-zero parameters by inducing sparsity into weight parameters, which can achieve high compression rates but requires specialized software and/or hardware in order to achieve faster inference times. (ii) Structured pruning [35, 60, 77] modifies the structure of the underlying weight tensors, by removing filters/neurons from each layer, usually resulting in smaller compression rates while directly achieving faster inference times with no specialized software; see section 4

[*]These authors equally contributed to this paper.
[†]Department of Computer Science, University of Haifa

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

1.1 Coresets for Pruning

Notably, many recent papers focused on various types of filter pruning [88, 79] potentially due to the empirical observation that existing filter pruning approaches consistently yield impressive results. However, most pruning methods are based on heuristics, lacking theoretical guarantees on the trade-off between the compression rate and the approximation error. This was the motive for introducing coresets [82, 60] to the world of pruning.

Coresets. In machine learning, we are (usually) given an input set $P \subseteq \mathbb{R}^d$ of n points, its corresponding weights function $w : P \to \mathbb{R}$, a feasible set of queries X, and a loss function $\phi : P \times X \to [0, \infty)$. The tuple (P, w, X, ϕ) is called *query space*, and it defines the optimization problem at hand. For a given problem that is defined by its query space (P, w, X, ϕ), and an error parameter $\varepsilon \in (0, 1)$, an ε-coreset is is a small weighted subset of the input points that approximates the loss of the input set P for every feasible query x, up to a provable bound of $1 + \varepsilon$.

Since coresets approximate the cost of every query, traditional (possibly inefficient) algorithms/solvers can be applied on coresets to obtain an approximation of the optimal solution on the full data, using less time and memory; see Section B.2 in the appendix for more details.

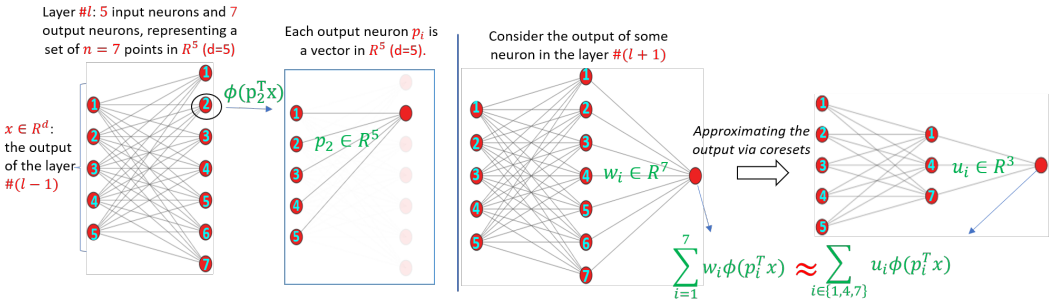

Figure 1: Illustration of our neuron coreset construction on a toy example.

Pruning via coresets. Recently, some inspiring innovative frameworks [82, 60, 5, 7] leveraged the idea of coresets for pruning DNNs. Any layer ℓ can be represented as a set $P = \{p_1, \cdots, p_n\}$ of $n > 1$ points in $\mathbb{R}^d$, where d is the number of input neurons, and n in the number of output neurons, i.e., each point $p \in P$, represents a specific neuron using its d weights (parameters). When the layer receives an input vector $x \in \mathbb{R}^d$, it outputs the vector $(\phi(p_1^T x), \cdots, \phi(p_n^T x))$, where $\phi : \mathbb{R} \to \mathbb{R}$ is an activation function which defines a non-linear mapping. Focusing on a single neuron η in the layer that follows ℓ, defined by its corresponding vector of weights $w = (w_1, \cdots, w_n)$, we set in the context of coresets, $w(p_i) := w_i$ for every $i \in \{1, \cdots, n\}$ - this is just a mapping from p_i to w_i to simplify the writing and reading. Note that the output of this neuron is $\sum_{p \in P} w(p)\phi\left(p^T x\right)$. Assuming that we are given an ε-coreset (C, u) for the query space $\left(P, w, \mathbb{R}^d, \phi\right)$ where $C \subseteq P$, and $u : C \to \mathbb{R}$, we have that (C, u) approximates the output of this specific neuron η for every query using less parameters; see Figure 1. To formalize the stated above, we now define coresets in the context of activation functions.

Definition 1.1 (Coreset for activation functions)**.** Let $\varepsilon \in (0, 1)$, and let $\left(P, w, \mathbb{R}^d, \phi\right)$ be a query space. Then the pair (C, u), is an ε-coreset for $\left(P, w, \mathbb{R}^d, \phi\right)$ if (i) $C \subseteq P$, (ii) $u : C \to [0, \infty)$, and (iii) for every $x \in X$, $\left|1 - \frac{\sum_{q \in C} u(q)\phi\left(q^T x\right)}{\sum_{p \in P} w(p)\phi(p^T x)}\right| \leq \varepsilon$.

Since C is a subset of P, we can remove (assign zero to) all weights from w that corresponds to points not chosen to be in C from P, and replace the weights of the chosen points in C with the new weights vector u; see Figure 1 in [82] for a visual illustration. **To prune neurons,** we refer the reader to Section 2.5 as it is a simple extension. Prior work showed that such approaches successfully result in high compression rates across a wide range of networks and datasets, and even achieves SOTA performance on a verity of them.

The main (strong) advantage of the coreset approach over others was the provided provable theoretical guarantees on the tradeoff between the compression rate and the approximation error, which supports

worse case scenarios. In addition, coresets play an important role in improving the generalization properties of the trained networks [5, 78].

Sensitivity sampling for constructing pruning coresets. To compute such coresets, both [60, 82] utilised the known sensitivity sampling framework [9, 52]. In short the sensitivity of a point $p \in P$ in some query space (P, w, X, ϕ) corresponds to the importance of this point with respect to the query space at hand, and it is defined as $s(p) = \sup_{x \in X} \frac{w(p)\phi(p,x)}{\sum_{q \in P} w(q)\phi(q,x)}$ - where the denominator is not equal to zero. Once we bound these sensitivities, we can sample points (neurons) from P according to sensitivity bounds, and re-weight the sampled points to obtain a coreset. The size of the sample is proportional to the sum of these bounds. See Section B.1 and Theorem B.2 for more details in the Appendix.

1.2 Our contribution

Prior coreset methods for pruning DNNs either (i) imposed restrictive assumptions both on the model's weights and inputs [82], i.e., the input set P representing the neurons, and the query set X which represents the inputs of the layer, are enclosed in a ball in $\mathbb{R}^d$ of radius r_1 and r_2, respectively, or (ii) the methods are data-dependent, i.e., use a mini-batch of the input set to measure the influence of each parameter on the loss function [5, 60].

To this end, in this work, we take coresets a step further into the realm of pruning by introducing a unified framework with provable guarantees for pruning DNNs (weights and neurons/filters) while minimally affecting the generalization error. Our main improvement is that our framework is simultaneously (i) **data-independent**, (ii) **requires a single assumption** on the model's weights, and (iii) **provably guarantees a multiplicative factor approximation**, which is favourable upon additive approximations; see Theorem 2.6. The approach is based on the widely used theory of coresets allowing us to suggest a provable guarantee on the tradeoff between the approximation error and compression rate for each layer.

We conducted experimental results which established new SOTA benchmarks for structured pruning via coresets across a wide range of networks and datasets. We share all of our resulted models [14].

2 Method

In general, the coreset (for pruning) technique hinges upon the insight that any linear layer such as convolutions, can be casted as a matrix multiplication [82]. Hence, we focus in what follows on fully connected (FC) layers, while the details holds for any linear layer. Furthermore, for simplicity, we assume in what follows that the weights of P are all equal to 1 and thus our query space is denote by $\left(P, \mathbb{R}^d, \phi\right)$, and the sensitivity of a point $p \in P$ is simply $s(p) = \sup_{x \in X} \frac{\phi(p,x)}{\sum_{q \in P} \phi(q,x)}$. Note that our proofs are easily extended to the general case where we are given a weight function $w : P \to \mathbb{R}$ as discussed in Section 2.5.

2.1 Preliminaries

Notations. For a positive integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. For $c \in \mathbb{R}^d$ and a symmetric positive definite matrix $G \in \mathbb{R}^{d \times d}$, we define $E(G, c) := \left\{ x \in \mathbb{R}^d \middle| (x-c)^T G (x-c) \leq 1 \right\}$ to be the ellipsoid defined by c and G. For an ellipsoid $E(G, c)$, each endpoint of a semi principal axis is called a vertex of $E(G, c)$. We define $\operatorname{rank}(P)$ for any set $P \subseteq \mathbb{R}^d$ to be the dimension of the affine subspace that P lies on. For a set $P \subset \mathbb{R}^d$ the convex hull of P is denoted by $\operatorname{Conv}(P)$. Finally, vectors are treated as column vectors.

2.2 Novelty - Löwner ellipsoid meets Carathéodory

Our method hinges upon a combination of two known tools from convex geometry. The novelty of our approach exploits the following observation. Most activation functions ϕ are continuous non-decreasing functions, which indicate that for every query x and a set of points P, the maximal contribution to $\sum_{p \in P} \phi(p^T x)$ with respect to such activation function is associated to a point on the convex hull of P. By finding a geometrical body B of bounded number of vertices, that is (i)

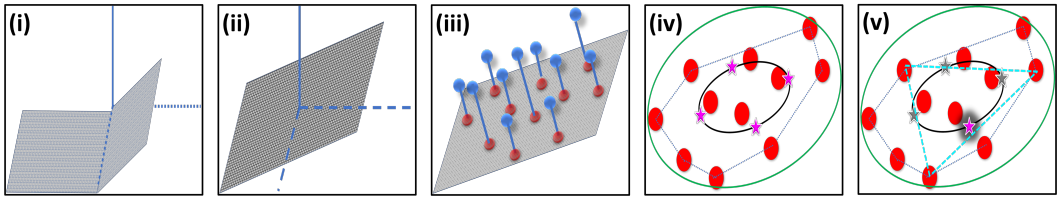

Figure 2: Our novelty in a nut-shell: The very first steps of our technique rely on bounding the ReLu activation function by the ℓ_1-regression loss function, e.g., for $\mathrm{ReLU}\left(p^T x\right)$, where $p = (1, 0)$ in this example (shown in (i)), we first bound it by the ℓ_1-regression loss function (shown in (ii)) using Definition 2.3. Following this step, a set of points P can be projected into a low dimensional subspace of dimension $\mathrm{rank}\,(P)$ using any dimensionality reduction algorithm as presented in (iii), resulting in the set P'. (iv) The convex hull (blue dashed lines) P' (red points) is enclosed by its Löwner ellipsoid (depicted in green). (v) Finally, for each vertex (magenta star) of the enclosed ellipsoid (black ellipsoid), its Carathédory set is found (red points connected by cyan dashed lines).

enclosed in $\mathrm{Conv}\,(P)$ and (ii) with some dilation factor (expanding) B enclose $\mathrm{Conv}\,(P)$, we will be able to represent each point p on the boundary of the convex hull of P as a convex combination of two points p_1, p_2, one of each (p_1) on B and the second (p_2) on its dilated form, which is formalized as the set $\{\alpha(x - c) + c \mid x \in B, \forall \alpha \in [0, 1]\}$, where c here denotes the center of B. For such task, Löwner ellipsoid is leveraged.

Theorem 2.1 (John-Löwner ellipsoid[42]). *Let $L \subseteq \mathbb{R}^d$ be a set of points such that the convex hull of L has a nonempty interior. Then, there exists an ellipsoid $E\,(G, c)$ (also known as the* MVEE*, where $G \in \mathbb{R}^{d \times d}$ is a positive definite matrix and $c \in \mathbb{R}^d$, of minimal volume such that $\frac{1}{d}\left(E\,(G, c) - c\right) + c \subseteq \mathrm{Conv}\,(L) \subseteq E\,(G, c)$. If L is symmetric around the center c, then the dilation factor can be reduced to $\frac{1}{\sqrt{d}}$.*

Algorithm 1: ℓ_∞-CORESET (P, m)

Input : $P \subseteq \mathbb{R}^d$ of n points with rank r
Output : A subset $S \subseteq P$ that satisfies Lemma 2.5

$(Y, z) :=$ affine subspace that P lies on, i.e. $P \subseteq \left\{xYY^T + z \middle| x \in \mathbb{R}^d\right\}$
$P' := \left\{p' := pY^T\right\}$
Let $\mathrm{map} : P' \to P$ a function that maps every $p' \in P'$ to its corresponding point $p \in P$
$S := \emptyset; K := \emptyset$
$E\,(G, c) := \mathrm{MVEE}\,(\mathrm{Conv}\,(P'))$
$V :=$ the vertices of ellipsoid $\frac{1}{r}\left(E\,(G, c) - c\right) + c$
for *every* $v \in V$ **do**
  $K :=$ $K \cup$ CARATHEODORY-SET (v, P')
  `// See Algorithm 4 in the supplementary material.`
end
$S := \{\mathrm{map}\,(q) | q \in K\}$
return S

Algorithm 2: CORESET (P, m)

input : A set $P \subseteq \mathbb{R}^d$ of n points, and a sample size m
output : A weighted set (C, u)

$Q := P, i := 1, C := \emptyset$
while $|Q| \geq 2\mathrm{rank}\,(Q)^2$ **do**
  $S_i := \ell_\infty$-CORESET (Q)
  for *every* $p \in S_i$ **do**
    $s(p) := \frac{2\mathrm{rank}(Q)^{1.5}}{i}$
  end
  $Q := Q \setminus S_i, i := i + 1$
end
for *every* $p \in Q$ **do**
  $s(p) := \frac{2\mathrm{rank}(Q)^{1.5}}{i}$
end
$t := \sum_{p \in P} s(p)$
$C :=$ an i.i.d sample of m points from P, where each $p \in P$ is sampled with probability $\frac{s(p)}{t}$.
$u(p) := \frac{t}{m \cdot s(p)}$ for every $p \in C$
return (C, u)

Afterwards, p_1 and p_2 should be represented by points from P. Each point on B (specifically, p_1) can be represented via a convex combination of $d + 1$ points from P. The same holds for points on the dilated form of B (e.g., p_2) but via a conical combination (linear combination where the weights are non-negative and the sum of weights is not necessarily 1). This problem is solved by invoking Carathéodory theorem.

Theorem 2.2 ([10, 95]). *For any $A \subset \mathbb{R}^d$ and $p \in \mathrm{Conv}(A)$, there exists $m \leq d+1$ points $p_1, \ldots, p_m \in A$ (denoted by a Carathéodory set of p) such that $p \in \mathrm{Conv}(\{p_1, \ldots, p_m\})$.*

Finally, it is known that some functions, including the ReLU function, do not admit an ε-coreset of size $o(n)$ [82, 81]. Thus, we use a generalized form of what is known as the complexity measure of a set of points, which was first introduced in [81] and later leveraged in [76]. This measure is used to determine the complexity of a given set P with respect to ReLU, and the coreset size theoretically.

Definition 2.3 (Regression Complexity Measure). Let $P \subseteq \mathbb{R}^d \times \{1\}$, the regression complexity measure of P is defined as $\mu(P) = \sup_{x \in \mathbb{R}^{d+1}} \frac{\sum_{q \in \{p \in P \mid p^T x \leq 0\}} \left|q^T x\right|}{\sum_{q \in \{p \in P \mid p^T x > 0\}} |q^T x|}$, where the denominator is ≥ 0, and the last entry of every $p \in P$ is 1, reserved for the bias/intercept term.

2.3 Our Pruning Scheme

In what follows, we present our data summarization technique for ReLU on the dot product function. Then in Section 2.5, we discuss that our results can be easily extended to a wide family of activation functions including the Sigmoid function, as recently shown in [76]. First we present Algorithm 1, which serves as a stepping stone towards bounding the sensitivities.

Overview of Algorithm 1. The algorithm receives as input a set $P \subset \mathbb{R}^d$ whose rank is $r \in [d]$ and deterministically finds a subset $S \subseteq P$ which satisfies that for every $j \in [d]$, $X \in \mathbb{R}^{d \times j}$ and $v \in \mathbb{R}^d$, $\frac{\max_{p \in P} \|(p-v)X\|_1}{\max_{p \in S} \|(p-v)X\|_1} \leq r^{1.5}$. To do so, first, we find the affine hyperplane that P lies on, followed by computing the low dimensional representation of P, denoted by P'; see Lines 1–2. Note that if $\mathrm{rank}(P) = d$, then we can either keep P as it is (i.e., $P' := P$), or use dimensionality reduction tricks as detailed in Section 2.5. To compute the output S, we first bound the convex hull of P' by its Löwner ellipsoid $E(G, c)$ in Line 5, followed by computing the dilated ellipsoid of $E(G, c)$, namely, $\frac{1}{r}(E - c) + c$. Let V be the set of vertices of such ellipsoid; see Line 6. Now, for each point $v \in V$, we represent it as a convex combination of $r+1$ points from P' via Theorem 2.2, and store the union of such sets (each of size at most $r+1$) of points into K as done in Lines 7–9. For each point in K, we map it back to $\mathbb{R}^d$ to satisfy Lemma 2.5. To sum up Algorithm 1, we observe that the vertices can be used via canonical combinations with their dilated form to describe every point on the convex hull of the input data (in our end, it would the network's weights). Hence the Carathéodory set of these vertices from the input points lying on the convex hull can be further used to also represent points lying on the convex hull. This is the core idea which enable us in forming our ℓ_∞ coreset for any ℓ_ρ regression problem where $\rho \in (0, \infty)$.

We now discuss Algorithm 2 which is responsible for constructing an ε-coreset with respect to activation functions. Its input is a set $P \subset \mathbb{R}^d$ and a sample size $m \geq 1$.

Overview of Algorithm 2. First set $Q := P$. At each iteration i, the algorithm obtains a subset $S_i \subseteq Q$ as an output to a call to ℓ_∞-CORESET(Q) as stated in Line 3 of Algorithm 1 such that for every $x \in \mathbb{R}^d$, it holds that $\frac{\max_{p \in P} \left|p^T x\right|}{\max_{p \in S_i} |p^T x|} \leq r^{1.5}$ with r being the rank of Q. The sensitivity of each point in S_i is bounded from above by $\frac{2d^{1.5}}{i}$ as stated in Lines 4–6. The idea behind these bounds lies in our proof of Theorem 2.6. The set S_i is removed from Q, and this procedure is repeated with respect to Q until the size of Q is small enough. The obtained sensitivities are the ones needed for computing the pruning coresets. Finally, we utilize the sensitivity sampling framework of [9] to obtain the desired coreset; see Lines 13–14.

2.4 Analysis

In this section, we prove the correctness of our algorithms. The following lemma shows that for each point p that is inside some convex hull S, its ℓ_1 distance to any affine subspace is always bounded from above by ℓ_1 distance from the same affine subspace, of some other point $q \in S$.

Lemma 2.4. *Let $d, \ell, m \geq 1$ be integers. Let $p \in \mathbb{R}^d$ and $A \subseteq \mathbb{R}^d$ be a set of m points with $p \in \mathrm{Conv}(A)$ so that there exists $\alpha : A \to [0,1]$ such that $\sum_{q \in A} \alpha(q) = 1$ and $\sum_{q \in A} \alpha(q) \cdot q = p$. Then for every $Y \in \mathbb{R}^{d \times \ell}$ and $v \in \mathbb{R}^\ell$, $\|(p-v)Y\|_1 \leq \max_{q \in A} \|(q-v)Y\|_1$.*

The following states the provable guarantees of Algorithm 1.

Lemma 2.5 (ℓ_∞-coreset for ℓ_1-regression)**.** *Let $P \subseteq \mathbb{R}^d$ be a set of points, and r be the rank of P. Let $j \in [d-1]$ and let S be the output of a call to ℓ_∞-*CORESET(P)*. Then (i) $|S| \in O\left(r^2\right)$, and (ii) for every $X \in \mathbb{R}^{d\times j}$ and $v \in \mathbb{R}^d$, $\frac{\max_{q\in P}\|(q-v)X\|_1}{\max_{q\in S}\|(q-v)X\|_1} \in \left[1, 2r^{1.5}\right]$.*

The following theorem states our main result.

Theorem 2.6 (ReLU ε-coreset)**.** *Let $P \subseteq \mathbb{R}^d$, $\varepsilon, \delta \in (0,1)$, $r = \operatorname{rank}(P)$, $\mu(P)$ be as in Definition 2.3, and let $m \in O\left(\frac{\mu(P) r^{3.5}\log n}{\varepsilon^2}\left(d\left(\log\left(\mu(P)\, r \log n\right)\right) + \log\left(\frac{1}{\delta}\right)\right)\right)$. Let (C, w) be the output of a call to* CORESET(P, m)*; see Algorithm 2. Then, with probability at least $1-\delta$, (C, w) is an ε-coreset for $\left(P, \mathbb{R}^d, \operatorname{ReLU}(\cdot)\right)$; see Definition 1.1.*

Time analysis. Letting r be the rank of P, the time complexity of Algorithm 1 can be dissected to two main parts: (i) Computing the Löwner ellipsoid in $O(nr^2 \log n)$ time using the method proposed in [98] and (ii) computing the Carathédory set in $O(nr + r^4 \log n)$ time via [70]. Since V can contain up to $O(r^2)$ points, the overall time for Algorithm 1 is $O(nr^2 \log n + nr^3 + r^6 \log n)$. As for Algorithm 2, it takes $O(n^2 r^2 \log n + nr^4 \log n)$. Indeed, as explained in Section 2.5, a dimensionality reduction algorithm may be applied to improve the run time (reducing the r^6 factor). Furthermore, the run time of our algorithm can be improved, using the merge-and-reduce tree from the literature of coresets to reduce the n^2 terms to $n \log n$, i.e., the running time can be reduced to $O(n \log^2(n) r^2 + nr^4)$. For a data-independent provable method, this running time is reasonable.

Our advantages over previous results. Our coreset supports different activation functions without the need to change the sensitivity that much. Specifically, it will only be multiplied by some scalar, unlike previous coresets where different losses impose drastically different sensitivities/leverage scores and algorithms. This is since our coreset unlike other coresets is in its essence a framework of coresets for different ℓ_ρ losses, as it can be used as is for different ℓ_ρ losses and yet still attain ϵ-approximation. In addition, when the rank of the input points is small, then our method outperforms previous methods.If the input data is of full rank, previous methods obtain faster coresets construction.

On the boundness of the regression complexity measure. First of all, there exists an example where the complexity measure is unbounded, e.g., consider a set of points distributed evenly on a unit ball. In this case, you can always find a point where a hyperplane separating it from the rest of the points can be found such that the one half-space of this hyperplane contains only this point while the other half-space contains the rest of the points. This leads to an infinite complexity measure. Such an example is also mentioned in [82], when assessing the hardness of generating multiplicative-approximation coresets for ReLU functions.

Theoretically, the complexity measure is influenced by how free can the bias term be (the last entry of x); see Definition 2.3. This term is the only thing that can ensure that one point can be separated from the rest in the sense of finding a separating hyperplane, leading to an infinite complexity measure. Bounding on this term, leads to bounded complexity measure from a theoretical point of view.

In the context of model pruning, from the perspective of the complexity measure, the model's weights are the input denoted by a matrix $P \in \mathbb{R}^{n\times(d+1)}$, while the query is now $\mathbb{R}^d \times \{1\}$. Thus the complexity measure is now $\mu(P) := \sup_{x\in\mathbb{R}^d\times\{1\}} \frac{\sum_{q\in\{p\in P \mid p^T x \le 0\}} \left|q^T x\right|}{\sum_{q\in\{p\in P \mid p^T x > 0\}} |q^T x|}$. With this in mind, we observe that the complexity measure is now an instance of the complexity measure used in [76]. The complexity measure now relies entirely on the structure of the model's weights, where the goal is to find the largest ratio between the sum of the absolute of the values inside the rectified neurons prior to applying the rectification, and the sum values of non-rectified neurons. To bound this measure, we can use a variant of the algorithm described in the proof of Theorem 3 in [81].

2.5 Extensions

Our suggested scheme can be extended to support many other variants of the pruning problem.

Various activation functions. Our result can be extended to a family of activation functions called "Nice hinge functions"; see Definition D.1. Let (P, X, ϕ) be a query space, where ϕ is a "Nice hinge functions". To bound the sensitivity of a point p, we first bound the nominator of $s(p)$ by proving that $\forall x \in X : \phi(p^T x) < \left|p^T x\right|$. For bounding the denominator from below, recently [76] proved that $\forall x \in X : \sum_{p\in P} \operatorname{ReLU}\left(p^T x\right) \lessapprox \sum_{p\in P} \phi(p^T x)$; see full detail in Section D.3.

Weighted Input. In the context of deep learning, the output of each neuron is multiplied with a scalar which brings the necessity of having the ability to deal with weighted set of points. Algorithm 2 can be extended easily to the case as generously detailed in Section D.1 in the Appendix.

Dimensionality reduction. All coreset-based pruning methods rely heavily on the dimensionality of the model's layers, as well as our method. To ensure sufficient pruning ratio, we apply either PCA, TSNE, MDS, or the JL transform on the weights of each layer prior to generating its coreset.

From weight to neuron pruning. Most coreset-based pruning methods, e.g., [5, 82], first provide a scheme for (provable) weight pruning, which is then used as a stepping stone towards pruning neurons as follows. [5] first suggested coreset-based neuron pruning via the use of a generated controled set of queries to evaluate the importance of weights. Any neuron that has a maximal activation value lower or equal to zero, will be pruned from the network as its impact on the rest of the neurons is minimal. On the other hand [82] altered the definition of sensitivity such that it takes into account the sensitivity of a neuron in a layer ℓ with respect to all the neurons in the layer $\ell + 1$, which basically means that the sensitivity of each neuron is taken be the maximal sensitivity over every weight function (neuron in the next layer) defined by the layer. Hence, we follow the same logic for such method; see Section D.2 in the supplementary material.

From neuron to filter pruning. Convolutions can be expressed as matrix multiplications, which enables our method to prune filters from any model as done by [60, 82, 61].

3 Experimental Results

In this section, we study various widely used network architectures and benchmark data-sets. Following [82], to test the robustness of our methods on each of the neuron and filter pruning tasks independently, two sets of experiments are conducted. The first focuses on pruning neurons (Section 3.1) whereas the second focuses on pruning filters (Section 3.2), both via our coreset method.

The setting. In all experiments we report the *Pruning ratio* – the percentage of the parameters that were removed from the original mode. Here, *PR* stands for pruning ratio, *FR* stands for floating-point reduction ratio and *Err* – the percentage of misclassified test instances of our method compared to coreset-based pruning methods and more. *Baseline Err* is the error of the original uncompressed network, while *Pruned Err* is the classification error of the compressed model. In our experiment we compress and fine-tune the network once, no iterative pruning was applied, thus, the compared methods also satisfy this setting. Each experiment was conducted 5 times, in the tables, we report for our method the best error achieved and we highlight in parentheses next to it the average error and standard deviation across the 5 trails. In all of our experiments, the models are fine-tuned till convergence (after pruning). Implementation details are given in Section E in the Appendix.

Software/Hardware. Our algorithms were implemented in Python 3.6 [108] using Numpy [83], and Pytorch [84]. Tests were performed on NVIDIA DGX A100 servers with 8 NVIDIA A100 GPUs each, fast InfiniBand interconnect and supporting infrastructure.

Baselines. Our results are compared to (i) PFP [60], (ii) FT [58], (iii) SoftNet [34], (iv) ThiNet [68], and (v) PvC [82], (vi) Soft Pruning [34], (vii) CCP [85], (viii) FPGM [36], (ix) ThiNet-70, (x) ThiNet-50 [68], (xi) Pruning via Coresets (PvC) [82], (xii) Pruning from Scratch (PfS) [111], and (xiii) Rethinking the value of network pruning (Rethink) [65].

3.1 Neuron Pruning

We test our method on VGG16 [92] using CIFAR10 [49], and LeNet300-100 using MNIST [55].

Discussion. Table 1 present the results of LeNet300-100 and VGG16. Observe that in both architectures, our method outperformed the competing methods under the same compression scenarios. For example, we pruned roughly 90% of the parameters of the LeNet-300-100 model while improving the accuracy of the original model. We witness a similar phenomena on the VGG16 model, where we pruned roughly 90% of the parameters of the dense layers resulting in accuracy improvement. This confirms the insights in [5] that coresets help in improving the generalization properties of DNNs.

Table 1: Pruning of LeNet-300-100 architecture on the MNIST dataset and of VGG-16 on the CIFAR-10 dataset. Here, we report the compression with respect to the fully connected layers only.

Model	Method	Baseline Err. (%)	Pruned Err. (%)	PR (%)
LeNet-300-100	PFP	1.59	2.00	84.32
	FT	1.59	1.94	81.68
	SoftNet	1.59	2.00	81.69
	ThiNet	1.59	12.17	75.01
	PvC	2.16	2.03	90
	Our method (90)	**2.07**	**1.98** (**2.02** $\pm$ **0.04**)	**90**
	Our method (92.6)	**2.07**	**2.64** (**2.74** $\pm$ **0.1**)	**92.6**
	Our method (94.6)	**2.07**	**3.51** (**3.58** $\pm$ **0.07**)	**94.6**
VGG-16	PvC	8.95	8.16	75
	Our method	**5.9**	**5.9** (**6.2** $\pm$ **0.3**)	**90**

3.2 Filter pruning

We compressed the convolutional layers of (i) ResNet50 [33] on ILSVRC-2012 [18], (ii) ResNet56 [33], (iii) VGG19 [92] on CIFAR10 and (iv) VGG16 [92] on CIFAR10.

Table 2: Filter pruning results on different neural networks with respect to the CIFAR10 dataset.

Model	Method	Baseline Err. (%)	Pruned Err. (%)	PR (%)	FR (%)
VGG-19	PfS	6.4	6.29	52	NA
	Rethink	6.5	6.22	80	NA
	Structured Pruning	6.33	6.20	88	NA
	PvC	6.33	6.02	88	NA
	Our method (88)	**6.33**	**5.85** (**6.03** $\pm$ **0.18**)	**88**	NA
	Our method (91.28)	**6.33**	**6.23** (**6.35** $\pm$ **0.12**)	**91.28**	NA
VGG-16	ThiNet	7.11	9.24	63.95	64.02
	FT	7.11	8.22	80.09	80.14
	SoftNet	7.11	7.92	63.95	63.91
	PFP (94)	7.11	7.61	94.32	85.03
	PFP (87)	7.11	7.17	87.06	70.32
	PFP (80)	7.11	7.06	80.02	59.21
	Our method (95.32)	**7.11**	**7.31** (**7.55** $\pm$ **0.24**)	**95.32**	**85.09**
	Our method (87)	**7.11**	**6.63** (**6.76** $\pm$ **0.13**)	**87**	**68.2**
	Our method (79.53)	**7.11**	**6.3** (**6.38** $\pm$ **0.08**)	**79.53**	**59.14**
ResNet56	ThiNet	7.05	8.433	49.23	49.74
	Channel Pruning	7.2	8.2	N/A	50
	AMC	7.2	8.1	64.78	50
	CCP	6.5	6.42	57	52.6
	PvC	6.21	7.0	55	N/A
	Our method	**6.61**	**7.26** (**7.56** $\pm$ **0.3**)	**63.95**	**50**

Table 3: Filter pruning of ResNet50 on ImageNet (ILSVRC-2012).

Method	Baseline Err. (%)	Pruned Err. (%)	PR (%)	FR (%)
PFP	23.87	24.79	44.04	30.05
Soft Pruning	23.85	25.39	49.35	41.80
CCP	23.85	24.5	56	48.8
FPGM	23.85	25.17	62	53.5
ThiNet-70	27.72	26.97	33.72	36.78
ThiNet-50	27.72	28.0	51.5	55.82
PvC	23.78	25.11	62	N/A
Our method	**23.78**	**24.87** (**25.07** $\pm$ **0.2**)	**62**	**61.5**
Our method (18.01)	**23.78**	**24.1** (**24.2** $\pm$ **0.1**)	**18.01**	**10.82**

Filter pruning of DNNs on CIFAR10 and ImageNet (ILSVRC-2012). For Cifar10, we used PyTorch implementations of VGG19 and VGG16. We compressed both models using our approach by different compression rates as shown in Table 2 with a comparison to other methods. For ImageNet, we compressed the baseline model of ResNet50 [33] roughly by 62% in terms of number of parameters. Table 3 provides comparison between our method and other baselines.

Discussion. As can be seen in Tables 1, 2, and 3, our method either outperforms the competing methods or achieves comparable results. As for the coreset methods, our algorithm achieves better result than all of them in this setting, e.g., we compressed 62% of ResNet50 trained on ImageNet while incurring 1.09% drop in accuracy, improving the recent coreset result of PvC [82] for the same compression ratio, while PFP [60] compressed 44.04% to achieve comparable results.

4 Related work

DNNs can be compressed before training [97, 110, 57], during training [118, 115, 51], or after training [93]. Furthermore, such procedures may also be repeated iteratively [88]. As previously noted pruning can be categorized into structured and unstructured pruning.

Unstructured pruning. Weight pruning [56] techniques aim to reduce the number of weights in a layer while approximately preserving its output. Approaches of this type include the works of [54, 21, 39, 1, 62], where the desired sparsity is embedded as a constraint or via a regularizer into the training pipeline, and those of [31, 88, 30], where weights with absolute values below a threshold are removed. The approaches of [5, 6] use a mini-batch of data points to approximate the influence of each parameter on the loss function. Other data-informed techniques include [28, 63, 80, 79, 116]. A thorough overview of recent pruning approaches is given by [27, 8]. However, unlike our approach, weight-based pruning approaches generate sparse models instead of smaller ones thus requiring specialized hardware and sparse linear algebra libraries in order to speed up inference.

Structured pruning. Pruning entire neurons and filters directly shrinks the network leading to smaller storage requirements and improved inference-time performance on any hardware [59, 67]. Lately, these approaches were investigated in many papers [64, 59, 11, 36, 22, 46, 114, 113]. Usually, filters are pruned by assigning an importance score to each neuron/filter, either solely weight-based [37, 35] or data-informed [116, 60], and removing those with a score below a threshold. The procedure can be embedded into an iterative pruning scheme [88] that requires potentially expensive retrain cycles.

Tensor decomposition. Some of the work in DNN compression entails decomposing the layer into multiple smaller ones, e.g., via low-rank tensor decomposition [19, 41, 74, 48, 96, 40, 2, 105, 117, 53, 61]. Other approaches to tensor decomposition include weight sharing, random projections, and feature hashing [112, 3, 91, 12, 13, 107]. However, such techniques usually require expensive approximation algorithms or use heuristics since tensor decomposition is generally NP-hard.

Coresets. In the recent years, coresets got increasing attention, and where leveraged to compress the input datasets of many machine learning algorithms, improving there performance, e.g., regression [72, 38, 81, 47, 103], decision trees [44], matrix approximation [26, 70, 25, 89, 73], data discretization [75], clustering [24, 29, 66, 4, 45, 90, 106], ℓ_z-regression [16, 17, 94], *SVM* [32, 101, 99, 100, 102], deep learning models [69, 5, 60] and even for path planning in the field of robotics [104]. For extensive surveys on coresets, we refer the reader to [23, 86, 43, 71].

5 Conclusions and Future Work

In this paper, we provided a coreset-based pruning technique that hinges upon a combination of tools from convex geometry, while achieving SOTA results with respect to coreset-based structured pruning approaches on a variety of networks. Our main improvement is that our coreset is (training) data-independent and assumes a single assumption on the models weights.

Future work includes (i) suggesting a coreset based budget allocation framework, to determine the (optimal) per layer prune ratio while achieving an overall desired compression rate, (ii) extending our coreset technique to other layers such as attention layers, and (iii) bridging the gap between coreset based pruning approaches and tensor-decomposition methods, as both techniques are theoretically supported by bounding the approximation error given specific compression rate, we can leverage these

bounds to formulate the compression problem as an optimization problem which iterates between the two approaches to search for the local minimum.

References

[1] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3180–3189, 2017.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, pages 254–263, 2018.

[4] Olivier Bachem, Mario Lucic, and Silvio Lattanzi. One-shot coresets: The case of k-clustering. In *International conference on artificial intelligence and statistics*, pages 784–792. PMLR, 2018.

[5] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

[6] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *arXiv preprint arXiv:1910.05422*, 2019.

[7] Gantavya Bhatt and Jeff Bilmes. Tighter m-DPP Coreset Sample Complexity Bounds. In *ICML 2021 Workshop: SubSetML: Subset Selection in Machine Learning: From Theory to Practice*, Virtual, July 2021.

[8] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020*, pages 129–146. 2020.

[9] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

[10] Constantin Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.

[11] Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[12] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015.

[13] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks. *CoRR*, abs/1506.04449, 2015.

[14] Code. All resulted pruned models presented in this paper, 2022. The authors commit to publish upon acceptance of this paper or reviewer request.

[15] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

[16] Michael B Cohen and Richard Peng. Lp row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 183–192, 2015.

[17] Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W Mahoney. Sampling algorithms and coresets for \ell_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[19] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[21] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[22] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

[23] Dan Feldman. Core-sets: An updated survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, https://arxiv.org/abs/2011.09384*, 10(1):e1335, 2020.

[24] Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable training of mixture models via coresets. In *Advances in neural information processing systems*, pages 2142–2150, 2011.

[25] Dan Feldman, Morteza Monemizadeh, Christian Sohler, and David P Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 630–649. Society for Industrial and Applied Mathematics, 2010.

[26] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453. SIAM, 2013.

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

[28] Noah Gamboa, Kais Kudrolli, Anand Dhoot, and Ardavan Pedram. Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators. *arXiv preprint arXiv:2001.03253*, 2020.

[29] Lei Gu. A coreset-based semi-supverised clustering using one-class support vector machines. In *Control Engineering and Communication Technology (ICCECT), 2012 International Conference on*, pages 52–55. IEEE, 2012.

[30] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[31] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[32] Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI*, pages 836–841, 2007.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[35] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2234–2240. AAAI Press, 2018.

[36] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

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

[38] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. *Advances in Neural Information Processing Systems*, 29:4080–4088, 2016.

[39] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[40] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

[41] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

[42] Fritz John. Extremum problems with inequalities as subsidiary conditions. In *Traces and emergence of nonlinear programming*, pages 197–215. Springer, 2014.

[43] Ibrahim Jubran, Alaa Maalouf, and Dan Feldman. Introduction to coresets: Accurate coresets. *arXiv preprint arXiv:1910.08707*, 2019.

[44] Ibrahim Jubran, Ernesto Evgeniy Sanches Shayda, Ilan Newman, and Dan Feldman. Coresets for decision trees of signals. *Advances in Neural Information Processing Systems*, 34, 2021.

[45] Ibrahim Jubran, Murad Tukan, Alaa Maalouf, and Dan Feldman. Sets clustering. In *International Conference on Machine Learning*, pages 4994–5005. PMLR, 2020.

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

[47] Zohar Karnin and Edo Liberty. Discrepancy, coresets, and sketches in machine learning. In *Conference on Learning Theory*, pages 1975–1993. PMLR, 2019.

[48] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[49] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[51] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. *arXiv preprint arXiv:2002.03231*, 2020.

[52] Michael Langberg and Leonard J Schulman. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 598–607. SIAM, 2010.

[53] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR (Poster)*, 2015.

[54] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2554–2564. IEEE, 2016.

[55] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[56] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[57] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

[58] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[59] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019.

[60] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[61] Lucas Liebenwein, Alaa Maalouf, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. *Advances in Neural Information Processing Systems*, 34:5328–5344, 2021.

[62] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188, 2017.

[63] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

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

[65] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[66] Mario Lucic, Olivier Bachem, and Andreas Krause. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1–9, Cadiz, Spain, 09–11 May 2016. PMLR.

[67] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

[68] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[69] Alaa Maalouf, Gilad Eini, Ben Mussay, Dan Feldman, and Margarita Osadchy. A unified approach to coreset learning. *arXiv preprint arXiv:2111.03044*, 2021.

[70] Alaa Maalouf, Ibrahim Jubran, and Dan Feldman. Fast and accurate least-mean-squares solvers. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8307–8318, 2019.

[71] Alaa Maalouf, Ibrahim Jubran, and Dan Feldman. Introduction to coresets: Approximated mean. *arXiv preprint arXiv:2111.03046*, 2021.

[72] Alaa Maalouf, Ibrahim Jubran, and Danny Feldman. Fast and accurate least-mean-squares solvers for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[73] Alaa Maalouf, Ibrahim Jubran, Murad Tukan, and Dan Feldman. Coresets for the average case error for finite query sets. *Sensors*, 21(19):6689, 2021.

[74] Alaa Maalouf, Harry Lang, Daniela Rus, and Dan Feldman. Deep learning meets projective clustering. In *International Conference on Learning Representations*, 2020.

[75] Alaa Maalouf, Murad Tukan, Eric Price, Daniel M Kane, and Dan Feldman. Coresets for data discretization and sine wave fitting. In *International Conference on Artificial Intelligence and Statistics*, pages 10622–10639. PMLR, 2022.

[76] Tung Mai, Anup B Rao, and Cameron Musco. Coresets for classification–simplified and strengthened. *arXiv preprint arXiv:2106.04254*, 2021.

[77] Zelda Mariet and Suvrit Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

[78] Baharan Mirzasoleiman, Kaidi Cao, and Jure Leskovec. Coresets for robust training of deep neural networks against noisy labels. *Advances in Neural Information Processing Systems*, 33, 2020.

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

[80] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[81] Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David P Woodruff. On coresets for logistic regression. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6562–6571, 2018.

[82] Ben Mussay, Dan Feldman, Samson Zhou, Vladimir Braverman, and Margarita Osadchy. Data-independent structured pruning of neural networks via coresets. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[83] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[84] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

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

[86] Jeff M Phillips. Coresets and sketches. *arXiv preprint arXiv:1601.00617*, 2016.

[87] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[88] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[89] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152. IEEE, 2006.

[90] Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms*, pages 232–251. Springer, 2019.

[91] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov):2615–2637, 2009.

[92] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[93] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. *arXiv preprint arXiv:2004.14340*, 2020.

[94] Christian Sohler and David P Woodruff. Subspace embeddings for the l1-norm with applications. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 755–764, 2011.

[95] Ernst Steinitz. Bedingt konvergente reihen und konvexe systeme. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1913(143):128–176, 1913.

[96] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[97] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

[98] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[99] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

[100] Ivor W Tsang, James Tin-Yau Kwok, and Pak-Ming Cheung. Very large svm training using core vector machines. In *AISTATS*, 2005.

[101] IW-H Tsang, JT-Y Kwok, and Jacek M Zurada. Generalized core vector machines. *IEEE Transactions on Neural Networks*, 17(5):1126–1140, 2006.

[102] Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines. *Theoretical Computer Science*, 2021.

[103] Murad Tukan, Alaa Maalouf, and Dan Feldman. Coresets for near-convex functions. *Advances in Neural Information Processing Systems*, 33:997–1009, 2020.

[104] Murad Tukan, Alaa Maalouf, Dan Feldman, and Roi Poranne. Obstacle aware sampling for path planning. *arXiv preprint arXiv:2203.04075*, 2022.

[105] Murad Tukan, Alaa Maalouf, Matan Weksler, and Dan Feldman. No fine-tuning, no cry: Robust svd for compressing deep networks. *Sensors*, 21(16):5599, 2021.

[106] Murad Tukan, Xuan Wu, Samson Zhou, Vladimir Braverman, and Dan Feldman. New coresets for projective clustering and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 5391–5415. PMLR, 2022.

[107] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[108] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[109] Kasturi Varadarajan and Xin Xiao. A near-linear algorithm for projective clustering integer points. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1329–1342. SIAM, 2012.

[110] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

[111] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12273–12280, 2020.

[112] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

[113] Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018.

[114] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*, pages 10820–10830. PMLR, 2020.

[115] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[116] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[117] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[118] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 2
   (b) Did you include complete proofs of all theoretical results? [Yes] See Sections 2.4 and C
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Computing the Carathéodory set

Overview of Algorithm 4. First, a convex combination of v with respect to P is formulated as a linear programming problem. This is done by reformulating the input set of points P as a matrix denoted as $A \in \mathbb{R}^{(d+1)\times n}$ (see Line 1). We then formulate the goal vector $b \in \mathbb{R}^{d+1}$ to be v concatenated with an entry of 1 which serves to make sure that the solution to

$$\begin{aligned} \underset{x\in\mathbb{R}^{n+1}}{\text{minimize}} \quad & \mathbf{1}_{n+1}^T x \\ \text{subject to} \quad & Ax = b, \\ & x_i \in [0,1] \quad \forall i \in [d] \end{aligned}$$

satisfies that $\sum_{i\in[n]} x_i P_i = v$ and $\sum_{i\in[n]} x_i = 1$. Solving this problem takes roughly $O^*\left(n^{\omega+o(1)}\right)$ where ω is the matrix multiplication exponent as elaborated in [15]; see Lines 2–3. We observe that x from Line 3 might be dense, i.e., the number of non-zero entries exceeds $d+1$. To ensure that we have at max $d+1$ non-zero entries, we use Algorithm 1 of [70] which aims to find a set of $d+1$ points, where their weighted average is the desired v given the initial weight vector x; see Line 4.

Algorithm 3: CARATHEODORY-SET (v, P)

Input : A point $v \in \mathbb{R}^d$ and a set $P \subseteq \mathbb{R}^d$ of n points
Output : A subset $C \subseteq P$ of at max $d+1$ points such that $p \in \text{Conv}(C)$

1 $A := \left[\begin{bmatrix} P_1 \\ 1 \end{bmatrix}, \begin{bmatrix} P_2 \\ 1 \end{bmatrix}, \ldots, \begin{bmatrix} P_n \\ 1 \end{bmatrix}\right]$ `/* P_i here denotes the ith point in P */`
2 $b := \begin{bmatrix} v \\ 1 \end{bmatrix}$
3 $x := \underset{\substack{x\in[0,\infty)^d \\ Ax=b}}{\arg\min} \mathbf{1}_d^T x$ `// 1_d denotes a d dimensional vector of 1s`
4 $C :=$ FAST-CARATHEODORY-SET $\left(P, x, d^2+2\right)$ `/* See Algorithm 1 of [70] */`
5 **return** C

B Coreset-Related Technical Details

Definition B.1 (VC-dimension [9])**.** For a query space $(P, w, \mathbb{R}^d, f)$ and $r \in [0,\infty)$, we define

$$\text{ranges}(x, r) = \{p \in P \mid w(p)f(p, x) \leq r\},$$

for every $x \in \mathbb{R}^d$ and $r \geq 0$. The dimension of $(P, w, \mathbb{R}^d, f)$ is the size $|S|$ of the largest subset $S \subset P$ such that

$$\left|\left\{S \cap \text{ranges}(x, r) \mid x \in \mathbb{R}^d, r \geq 0\right\}\right| = 2^{|S|},$$

where $|A|$ denotes the number of points in A for every $A \subseteq \mathbb{R}^d$.

B.1 Sensitivity Sampling Missing Details

We want to use the sensitivity sampling framework to compute a coreset for a set of points P in $\mathbb{R}^d$.

First, we need to bound the sensitivity of each point $p \in P$. The sensitivity pf a point $p \in P$ is defined as $s(p) = \sup_{x\in X} \frac{\phi(p,x)}{\sum_{q\in P}\phi(q,x)}$ where the denominator is not zero.

Hence, for every $p \in P$, we wish to compute a number $s'(p)$, such that $s'(p) \geq s(p)$. Once the bound $s'(p)$ on the sensitivity $s(p)$ of each point p is computed, we define $T = \sum_{p\in P} s'(p)$ as the total sensitivity. Now, to obtain a coreset, we can sample points according to the distribution $s'(p)/T$, i.e., we sample $m > 0$ points from P, where at each sample, the point $p \in P$ is sampled i.i.d with probability $s'(p)/T$. We also re-weight the sampled points to obtain a coreset.

As the bound $s'(p)$ (on $s(p)$) is tighter, the total sensitivity T gets smaller, and then the coreset size (required number of sampled points) gets smaller, and vice versa.

Theorem B.2 (Restatement of Theorem 5.5 in [9]). *Let $P \subseteq \mathbb{R}^d$ be a set of n points, $w : P \to [0, \infty)$ be a weight function , and let $f : P \times \mathbb{R}^d \to [0, \infty)$ be a loss function. For every $p \in P$ define the* sensitivity *of p as*

$$\sup_{x \in \mathbb{R}^d} \frac{w(p)f(p,x)}{\sum_{q \in P} w(q)f(q,x)},$$

where the sup is over every $x \in \mathbb{R}^d$ such that the denominator is non-zero. Let $s : P \to [0, 1]$ be a function such that $s(p)$ is an upper bound on the sensitivity of p. Let $t = \sum_{p \in P} s(p)$ and d' be the VC dimension *of the quadruple $\left(P, w, \mathbb{R}^d, f\right)$; see Definition B.1. Let $c \geq 1$ be a sufficiently large constant, $\varepsilon, \delta \in (0, 1)$, and let S be a random sample of $|S| \geq \frac{ct}{\varepsilon^2}\left(d' \log t + \log \frac{1}{\delta}\right)$ i.i.d points from P, such that every $p \in P$ is sampled with probability $\frac{s(p)}{t}$. Let $v(p) = \frac{tw(p)}{s(p)|S|}$ for every $p \in S$. Then, with probability at least $1 - \delta$, (S, v) is an ε-coreset for $\left(P, w, \mathbb{R}^d, f\right)$.*

B.2 From Coresets to Approximating the Optimal Solution

In optimization problems (or machine learning in general), the goal is usually to find a query that minimizes (or maximizes) some cost function. In the context of coresets, the goal is to find a small weighted subset such that for a given cost function, the cost of applying any solution (hypotheses/query) on the coreset approximates the cost of applying the same solution on the whole data. Since a coreset approximates the cost of every query, we do note that in many cases, coresets are applied for approximating the optimal solution. Specifically, solving the desired optimization problem on the whole data can be a hard problem when the time needed for such a solution is either polynomial or exponential in the size of the whole data, or when the required memory is too high. In this case, coresets can be leveraged, by computing the the optimal solution of fitting an ε-coreset and applying it on the original data. If the computed coresets gives worst-case $(1 + \varepsilon)$-approximation error, then we provably $(1 + 4\varepsilon)$-approximation towards the optimal cost of solving the optimization on the whole data (the proof is very easy, it is done by applying the triangle inequality few times). In other words, we can solve the problem on the coreset to obtain a solution x^*, and then apply x^* to the whole data giving a good approximation for solving the problem from the beginning on the whole data.

C Proofs of Technical Results

C.1 Proof of Lemma 2.4

Lemma C.1. *Let $d, \ell, m \geq 1$ be integers. Let $p \in \mathbb{R}^d$ and $A \subseteq \mathbb{R}^d$ be a set of m points with $p \in \mathrm{Conv}(A)$ so that there exists $\alpha : A \to [0, 1]$ such that $\sum_{q \in A} \alpha(q) = 1$ and $\sum_{q \in A} \alpha(q) \cdot q = p$. Then for every $Y \in \mathbb{R}^{d \times \ell}$ and $v \in \mathbb{R}^\ell$, $\left\|(p - v)Y\right\|_1 \leq \max_{q \in A} \left\|(q - v)Y\right\|_1$.*

Proof. Since we can write p as the convex combination of points $q \in A$ with weight $\alpha(q)$, we have

$$\left\|p^T Y - v\right\|_1 = \left\| \left(\sum_{q \in A} \alpha(q) q^T \right) Y - v \right\|_1 .$$

Moreover, we have $\sum_{q \in A} \alpha(q) = 1$, so we can decompose v into

$$\left\|p^T Y - v\right\|_1 = \left\| \sum_{q \in A} \alpha(q) \left(q^T Y - v\right) \right\|_1 .$$

By triangle inequality (or Jensen's inequality),

$$\left\|p^T Y - v\right\|_1 \leq \sum_{q \in A} \alpha(q) \left\|q^T Y - v\right\|_1 \leq \max_{q \in A} \left\|q^T Y - v\right\|_1 .$$

□

C.2 Proof of Lemma 2.5

Lemma C.2 (ℓ_∞-coreset for ℓ_1-regression)**.** *Let $P \subseteq \mathbb{R}^d$ be a set of points, and r be the rank of P. Let $j \in [d-1]$ and let S be the output of a call to ℓ_∞-CORESET(P). Then (i) $|S| \in O\left(r^2\right)$, and (ii) for every $X \in \mathbb{R}^{d\times j}$ and $v \in \mathbb{R}^d$, $\frac{\max_{q\in P}\|(q-v)X\|_1}{\max_{q\in S}\|(q-v)X\|_1} \in \left[1, 2r^{1.5}\right]$.*

Proof. Let Y, z, P', K, S and map be defined as in Algorithm 1. Since P lies on a r-dimensional affine subspace, it holds that for every $p \in P$, $p = (p-z)YY^T + z$. Note that $Y \in \mathbb{R}^{d\times r}$ is an orthogonal matrix (i.e., Y^TY is the identity matrix in $\mathbb{R}^{r\times r}$) and $z \in \mathbb{R}^d$ denotes the translation of the affine subspace that P lies on.

Claim (i). $E(G,c)$ is the Löwner ellipsoid of P', which has $2r$ vertices. Since V is the set of vertices of the shrunk form of $E(G,c)$ that is contained in $\mathrm{Conv}\,(P')$, each point from V can be represented as convex combination of $r+1$ points from P' by Carathéodory's Theorem. Then the number of points in K is at most $2r\,(r+1)$, i.e., $|K| \in O\left(r^2\right)$. Thus, $|S| \in O\left(r^2\right)$ due to the fact that it can be constructed from K through the use of map.

Claim (ii). First put $X \in \mathbb{R}^{d\times j}$ and $v \in \mathbb{R}^d$, and let $p \in \arg\sup_{q\in P} \|(p-v)\,X\|_1$. Since $S \subseteq P$, it holds that $\frac{\max_{q\in P}\|(q-v)X\|_1}{\max_{q\in S}\|(q-v)X\|_1} \geq 1$. Let $a := zYY^T + z - v$, we have

$$\|(p-v)\,X\|_1 = \left\|\left((p-z)\,YY^T + z - v\right)X\right\|_1^T = \left\|\left(pYY^T + a\right)X\right\|_1 = \left\|\left(p'Y^T + a\right)X\right\|_1, \tag{1}$$

where the first equality holds since $\mathrm{rank}(P) = r$, the second holds by definition of a, and the last equality holds by the construction of $p' = pY$ at Line 2 of Algorithm 1.

Note that since V is the set of vertices of $\frac{1}{r}\left(E(G,c) - c\right) + c$, by the definition of the Löwner ellipsoid,

$$V \subseteq \mathrm{Conv}\,(V) \subseteq \frac{1}{r}(E(G,c)-c) + c \subseteq \mathrm{Conv}\,(P') \subseteq E(G,c) \subseteq \mathrm{Conv}\left(r^{1.5}\,(V-c) + c\right).$$

Since $\mathrm{Conv}\,(P')$ enclose $\mathrm{Conv}\,(V)$, and is enclosed by $\mathrm{Conv}\left(r^{1.5}\,(V-c)+c\right)$, then there exists a point $q \in \mathrm{Conv}\,(V)$ and $\gamma \in [0,1]$ such that $p'Y^T = \gamma qY^T + (1-\gamma)\left(r^{1.5}\,(q-c)+c\right)Y^T$, where by definition it holds that $r^{1.5}\,(q-c)+c \in \mathrm{Conv}\left(r^{1.5}\,(V-c)+c\right)$, and $p' = pY$.

By invoking Lemma 2.4, we obtain that

$$\left\|\left(p'Y^T + a\right)X\right\|_1 \leq \max\left\{\left\|\left(qY^T + a\right)X\right\|_1, \left\|\left(r^{1.5}\,(q-c) + c + a\right)Y^TX\right\|_1\right\}. \tag{2}$$

We note that

$$\left\|\left(qY^T + a\right)X\right\|_1 \leq \max_{\tilde{q}\in V}\left\|\left(\tilde{q}Y^T + a\right)X\right\|_1 \leq \max_{\tilde{q}\in K}\left\|\left(\tilde{q}Y^T + a\right)X\right\|_1, \tag{3}$$

where the first inequality follows from plugging $p := q$ and $A := V$ into Lemma 2.4, and the second inequality holds similarly since every point V lies in $\mathrm{Conv}\,(K)$. By invoking triangle inequality, we obtain that

$$\begin{aligned}
&\left\|\left(r^{1.5}\left((q-c)+c\right)Y^T + a\right)X\right\|_1 = \left\|\left(r^{1.5}qY^T + \left(1-r^{1.5}\right)cY^T + a\right)X\right\|_1 \\
&= \left\|\left(r^{1.5}qY^T + r^{1.5}a + \left(1-r^{1.5}\right)cY^T + \left(1-r^{1.5}\right)a\right)X\right\|_1 \\
&\leq r^{1.5}\left\|\left(qY^T + a\right)X\right\|_1 + \left(r^{1.5}-1\right)\left\|\left(cY^T + a\right)X\right\|_1,
\end{aligned} \tag{4}$$

where the first equality follows by a simple rearrangement, and the second holds since $r^{1.5}a + \left(1-r^{1.5}\right)a = a$. Observe that $c \in \mathrm{Conv}\,(V)$. Hence, by Lemma 2.4,

$$\left\|\left(cY^T + a\right)X\right\|_1 \leq \max_{\tilde{q}\in K}\left\|\left(\tilde{q}Y^T + a\right)X\right\|_1. \tag{5}$$

By the construction of S, it holds that for every $p \in S$, $pY \in K$. Thus, combining (2), (3), (4) and (5) yields $\frac{1}{2r^{1.5}}\left\|\left(p'Y^T + a\right)X\right\|_1 \leq \max_{\tilde{q}\in K}\left\|\left(\tilde{q}Y^T + a\right)X\right\|_1 = \max_{\tilde{q}\in S}\left\|\left(\tilde{q}YY^T + a\right)X\right\|_1 = \max_{\tilde{q}\in S}\|(\tilde{q}-v)\,X\|_1$ where the last equality holds by (1). This concludes Lemma 2.5. □

C.3 Proof of Theorem 2.6

Proof. For space constraints let ϕ denote the ReLU function. To obtain a coreset, we first need to bound the sensitivity of each $p \in P$. Put $p \in P$ and let $x \in \arg\sup_{x' \in \mathbb{R}^d} \frac{\phi(p^T x')}{\sum_{q \in P} \phi(q^T x')}$ where the supremum is over every $x' \in \mathbb{R}^d$ such that the denominator is not zero. Observe that

$$\frac{\sum_{q \in P} \left|q^T x\right|}{\sum_{q \in P} \phi\left(q^T x\right)} = \frac{\sum_{q \in P} \phi\left(q^T x\right) + \sum_{q \in P} \phi\left(-q^T x\right)}{\sum_{q \in P} \phi\left(q^T x\right)} = 1 + \frac{\sum_{q \in P} \phi\left(-q^T x\right)}{\sum_{q \in P} \phi\left(q^T x\right)} \leq 1 + \mu\left(P\right),$$

where the last inequality follows from Definition 2.3. Thus

$$\sum_{q \in P} \phi\left(q^T x\right) \geq \frac{1}{1 + \mu\left(P\right)} \sum_{q \in P} \left|q^T x\right|. \tag{6}$$

Let $\beta = (1 + \mu(P))$. We next observe that $\frac{\phi(p^T x)}{\sum_{q \in P} \phi(q^T x)} \leq \frac{|p^T x|}{\sum_{q \in P} \phi(q^T x)} \leq \beta \frac{|p^T x|}{\sum_{q \in P} |q^T x|}$, where the first inequality holds by properties of ϕ, and the second is by (6). Hence the sensitivity of p is bounded by

$$s(p) = \frac{\phi\left(p^T x\right)}{\sum_{q \in P} \phi\left(q^T x\right)} \leq \beta \frac{\left|p^T x\right|}{\sum_{q \in P} \left|q^T x\right|}. \tag{7}$$

Let i be the iteration counter from Algorithm 2 as defined in Line 1, used in Line 5 and incremented in Line 7. The idea follows that of [109] where points being discarded from P at lower levels (smaller i's) have higher sensitivity. This notion also resembles that of the "Onion sampling" of [45]. Now, assume that $p \in S_i$ at iteration i of the while loop (Line 3 of Algorithm 2). In this case, observe that by plugging $P := Q \setminus \bigcup_{\hat{i}=1}^{i-1} S_{\hat{i}}$, $j = 1$ and $v = -b\frac{x}{\|x\|_2}$ into Lemma 2.5, we obtain a subset $S_i \subseteq Q$ such that

$$\max_{q \in P} \left|q^T x\right| \leq \max_{q \in S} 2r^{1.5} \left|q^T x\right|. \tag{8}$$

Thus for every $q \in S_i$,

$$\frac{s(p)}{(1 + \mu\left(P\right))} \leq \frac{\left|p^T x\right|}{\sum_{q \in P} \left|q^T x\right|} \leq \frac{\left|p^T x\right|}{\sum_{\hat{i}=1}^{i} \max_{q \in S_{\hat{i}}} \left|q^T x\right|} \leq 2r^{1.5} \frac{\left|p^T x\right|}{\sum_{\hat{i}=1}^{i} \left|p^T x\right|} = \frac{2r^{1.5}}{i},$$

where the first inequality is by (7), the second inequality follows from the observation that $\left\{\arg\max_{q \in S_{\hat{i}}} \left|q^T x\right|\right\}_{\hat{i}=1}^{i} \subseteq P$ and the last inequality holds by (8). Hence, we have obtained a bound on the sensitivity of each point $p \in P$. As for the total sensitivity, we observe that $t = \sum_{p \in P} s(p) \in O\left((1 + \mu\left(P\right)) r^{3.5} \log n\right)$. Theorem B.2 states that to obtain an ε-coreset with probability at least $1 - \delta$, the sample size m must be $O\left(\frac{\mu(P) r^{3.5} \log n}{\varepsilon^2} \left(d\left(\log\left(\mu\left(P\right) r \log n\right)\right) + \log\left(\frac{1}{\delta}\right)\right)\right)$. □

D Extension

D.1 Handling Weighted Sets of Points

Similarly to [5, 60], we split the input data P into two sets $P_+, P_- \subseteq P$ such that $P_+ = \{p \in P | w(p) \geq 0\}$ while $P_- = \{p \in P | w(p) < 0\}$. Following this step we call Algorithm 4 for each of the two sets with corresponding weights and corresponding sample sizes. To account for proper sample sizes, we split our theoretical bound of the required sample size for generating ε-coreset into two terms for both P_+ and P_- respectively, i.e., we formulate $m = m_+ + m_-$ where $m_+ = \frac{|P_-|}{|P|} m$ (similarly for m_-). Hence, we obtain an ε-coreset for each of the query spaces $\left(P_+, w, \mathbb{R}^d, \phi\right)$ and $\left(P_-, w, \mathbb{R}^d, \phi\right)$.

D.2 From Weight to Neuron Pruning

Most coreset-based pruning methods, e.g., [5, 82], first provide a scheme for (provable) weight pruning, which is then used as a stepping stone towards pruning neurons as follows. To prune neurons from a layer, post to computing the coreset-based weight pruning for each neuron, ideally we would have that at certain layer, for all neurons, the generated coreset contains the same set of neurons from previous layers, which in this case we can remove the neurons which are not in the coreset. However, such scenario is almost implausible. To deal with such problem, we discuss two ways to do so. The first method to deal with such problem is inspired by the technique used in [82] which alters the definition of sensitivity such that it takes into account the sensitivity of a neuron in a layer ℓ with respect to all the neurons in the layer $\ell + 1$, basically the sensitivity of each neuron is taken be the maximal sensitivity over every weight function (neuron in the next layer) defined by the layer. Hence, we follow the same logic for such method, more details .

Algorithm 4: GENERALIZED-CORESET (P, w, m)

input : A set $P \subseteq \mathbb{R}^d$ of n points, a weight function $w(p) : P \to [0, \infty)$ and a sample size m
output : A weighted set (C, u)

$Q := P$, $i := 1$, $C := \emptyset$
while $|Q| \geq 2\text{rank}\,(Q)^2$ **do**
  $Q' := \{w(q)q | q \in Q\}$
  $\text{map}_w : Q' \to Q$ a map that maps from Q' to Q
  $S_i := \ell_\infty$-CORESET (Q')
  for *every* $p \in S_i$ **do**
    $s\left(\text{map}_w\left(p\right)\right) := \frac{2r^{1.5}}{i}$
  end
  $Q := Q \setminus \{\text{map}_w\left(q\right) | q \in S_i\}$, $i := i + 1$
end
for *every* $p \in Q$ **do**
  $s(p) := \frac{2r^{1.5}}{i}$
end
$t := \sum_{p \in P} s(p)$
$C :=$ an i.i.d sample of m points from P, where each $p \in P$ is sampled with probability $\frac{s(p)}{t}$.
$u(p) := \frac{tw(p)}{m \cdot s(p)}$ for every $p \in C$
return (C, u)

D.3 Other Activation Functions

[76] recently showed that there exists a family of functions $\mathcal{F}$ called "Nice hinge functions" such that for any query $x \in \mathbb{R}^d$ and a set of points $P \subseteq \mathbb{R}^d$, for any $\phi \in \mathcal{F}$, it holds that $\frac{\sum_{p \in P} \phi\left(p^T x\right)}{\sum_{q \in P} \text{ReLU}(q^T x)}$ is bounded from below. Formally speaking, below is the definition of a "nice hinge function".

Definition D.1 (Restatement of Definition 7 of [76])**.** We call $f : \mathbb{R} \to [0, \infty)$ an (L, a_1, a_2)-nice hinge function if for a fixed constant L, a_1 and a_2,

(i) f is L-Lipschitz,

(ii) $|f(z) - \text{ReLU}\,(z)| \leq a_1$ for all z, and

(iii) $f(z) \geq a_2$ for all $z \geq 0$.

As noted by [76], the hinge and log losses are $(1, 1, 1)$-nice and $(1, \ln 2, \ln 2)$-nice hinge functions respectively. Similarly, it is easy to show that the activation function $\phi(x) = \ln\left(1 + e^x\right)$ is

$(1, \ln 2, \ln 2)$-nice hinge functions. Following the same steps applied by [76], we obtain that

$$\sum_{p \in P} \phi\left(p^T x\right) \geq \min\left\{\frac{a_2}{2a_1}, \frac{1}{2}\right\} \frac{\sum_{q \in P} \left|q^T x\right|}{\mu\left(P\right) + 1},$$

where $\phi\left(\cdot\right)$ is a (L, a_1, a_2) where a_2 is assumed to be positive.

Unlike the ReLU activation function, to support for other activation functions, we need to restrict our query space to contain queries such that $\forall p \in P : \phi\left(p^T x\right) \leq r\left|p^T x\right|$ where r denotes the rank of P. Let X' denote the set of all such queries.

Hence, under this additional assumption, we obtain that for every $p \in P$ and $x \in X'$

$$\frac{\phi\left(p^T x\right)}{\sum_{q \in P} \phi\left(q^T x\right)} \leq \frac{\left(1 + \mu\left(P\right)\right) \phi\left(p^T x\right)}{\min\left\{\frac{a_2}{2a_1}, \frac{1}{2}\right\} \sum_{q \in P} \left|q^T x\right|} \leq \frac{\left(1 + \mu\left(P\right)\right) r \left|p^T x\right|}{\min\left\{\frac{a_2}{2a_1}, \frac{1}{2}\right\} \sum_{q \in P} \left|q^T x\right|}.$$

Following the same steps done at the proof of Theorem 2.6, we can generate an ε-coreset with respect to (P, ϕ, X').

E Implementation Details

First observe that the *map* function in Line 3 of Algorithm 1 is hard to implement if all we have is Y and P' (see Lines 1–2) due to the fact that when the rank of P is not d, then Y becomes a singular matrix. A way around such problem (practically speaking yet also theoretically sound) is to reformulate P and P' as matrices, where our ℓ_∞-coreset will now be regarded as a set of indices of the rows selected as the desired coreset. **Note** that while we relied on an "accurate" measure of the rank of points in Algorithm 1, in our experiments, we used the rank function from *Numpy*, and still produced favorable results. Furthermore, our algorithms work also when the input has full rank. In addition, we can still obtain an ε-coreset when using approximated algorithms for the rank computation problem, where the error associated with our coreset may increase. In this case, we can increase our coreset size to reduce our approximation error to be the original desired error.

F Complexity Measure - Clarification

First, Note that while the complexity measure was first defined for construction of coresets with respect to the logistic regression problem, it also has been used for the ReLU regression problem (minimizing the sum of ReLU losses) [76].

The complexity is expected to be small other than in some cases [81, 76]. We also operate under the same assumption, i.e., the complexity measure is reasonably small.

Specifically speaking, when given a set P (expressing our neurons) containing points in $\mathbb{R}^3$ (for example), such that point in P lie on a 2-dimensional affine subspace parallel to the xy-plane, notice that in our setting, the complexity measure is defined as the maximal value of an optimization problem involving our vectors and a set of queries $X := \mathbb{R}^2 \times \{1\}$.

The existing of a hyperplane whose normal in X such that one point from P can be separated from the rest of the points in P, leads to large complexity measure.

In Figure 3, a clear separation can be made between one point and the rest leading to two sets of points: The first set contains one single point that has a positive dot product with the normal x to the separating hyperplane, while the other set contains the remaining point each with negative dot product with x. This leads to a large complexity measure, and as the separating hyperplane gets closer and closer to the set containing the single point, the complexity measure increases, as it can tend to infinity.

On the other hand, when one can not separate a single point or minimal set of points from the rest of the data, we expect the complexity measure to be small; see Figure 4.

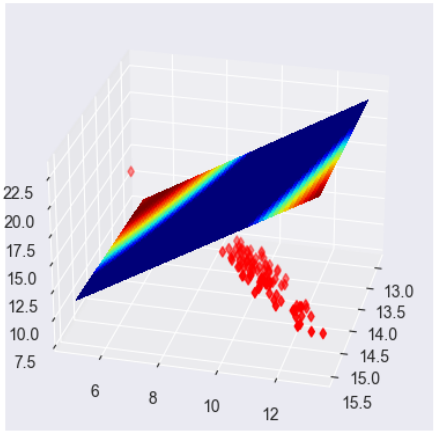

Figure 3: Linearly separable data leading to sufficiently large complexity measure.

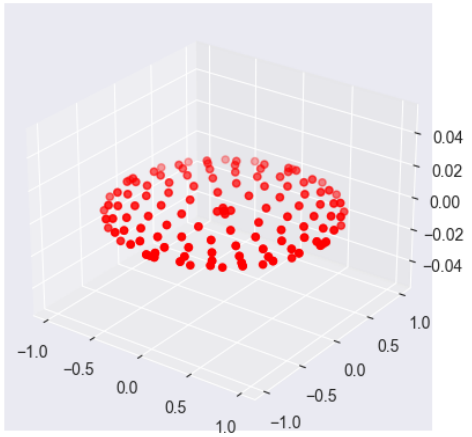

Figure 4: Non-linearly separable data leading to sufficiently small complexity measure.

Notice that in Figure 4, the input data is centered around the origin, which means that the data is not linearly separable. Thus leading to small complexity measure, since x represents the normal to a hyperplane emerging from the origin.

In fact, during our experiments, the complexity measure in the case of LeNet300-100 was around 15 when the input data (matrix representing the neurons) had 300 rows (points).

It is common in coreset literature from a practical point of view, sample sizes that are smaller than the bound on the coreset size are being used. This is due to the fact that such bounds are pessimistic in nature.

This motivated the choice of not incorporating the complexity measure in our sample size nor the sensitivity sampling since such a term will be eliminated when computing the sampling probability; see Theorem B.2. Our experiments confirmed such an observation, i.e., our coreset lead to favorable results when the complexity measure was not incorporated in our computations, or when the sample size was much smaller than the bound on the coreset size.

The appendix in the supplementary material has been modified in light of this.

G Additional Experiments

In all of our experiments, our hyper-parameters were drawn from [60].

G.1 The effect of fine-tuning

In this experiment, we aim to show the effect of fine-tuning on our compressed model. Specifically, Figure 5 shows VGG19's network accuracy over fine-tuning, where we start better than previous methods, followed by a slow incline in accuracy until we outperform previous models (around epoch 18).

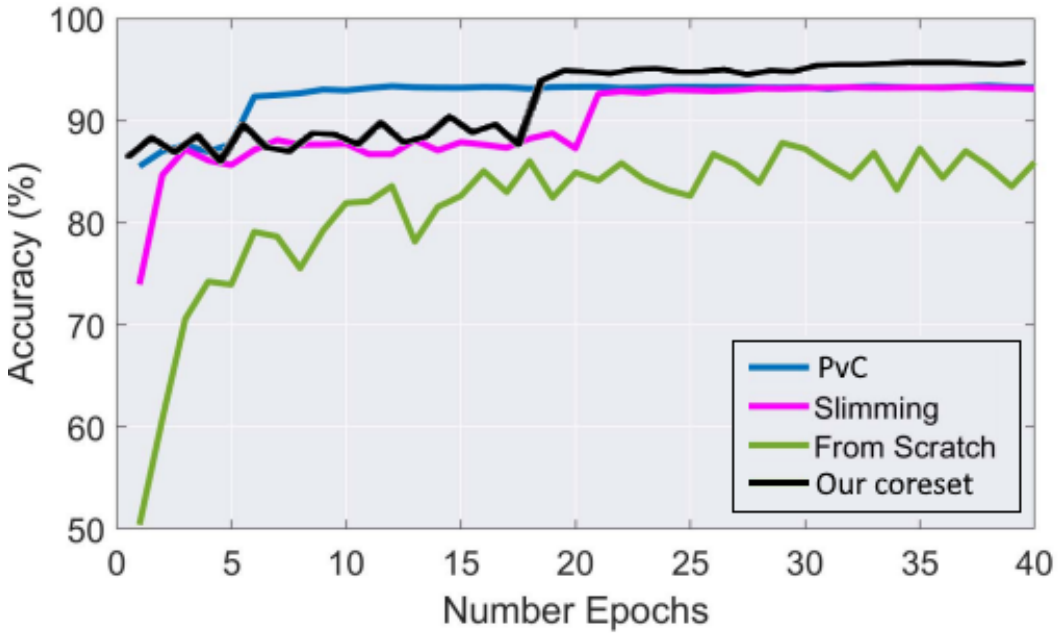

Figure 5: Accuracy of our proposed framework in comparison to previous methods and training the pruned network from scratch. The results above reflect the accuracy of VGG19 on CIFAR10.

G.2 Sensitivity based distribution

At Figure 6, we plot the sensitivity distribution of our sampling method in comparison to the sampling probabilities achieved by [82]. Our advantage lies in the observation that our induced probability distribution entails longer tails, i.e., important point are scarce.

G.3 Comparison with PvC

In this experiment, we aim to show the efficacy of our approach against that of [82]. We considered a single neuron in LeNet-300-100 where we computed the average additive error of the cost of the coreset from the cost of taking all the samples (neurons from previous layer), over set of 1000 queries. As shown in Figure 7, for very small coreset sizes, PvC [82] attains smaller error, however as we increase the sample size, our coreset outperforms that of [82].

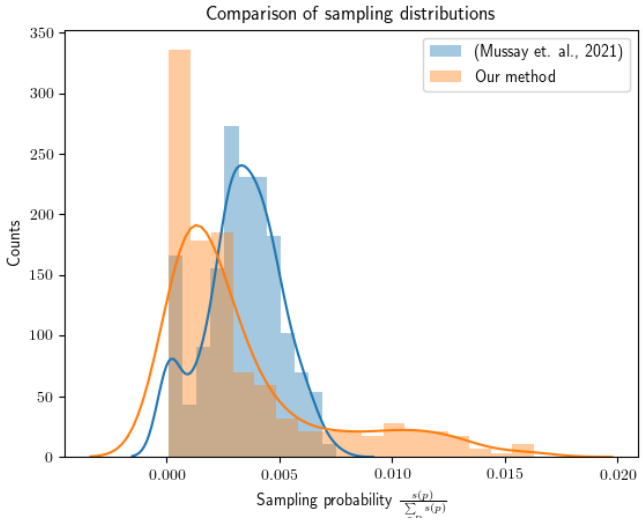

Figure 6: A comparison against [82] with respect to the distribution of the sampling probabilities of weights of a single neuron at some layer of LeNet-300-100. Here the x-axis denotes the sampling probability of points, while the y-axis presents the number of points with certain probability.

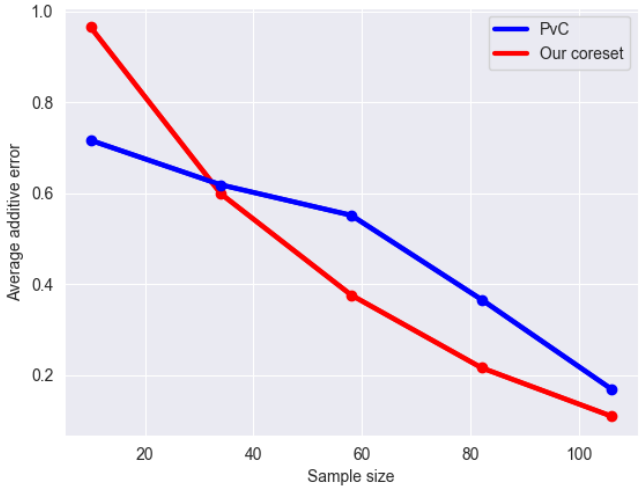

Figure 7: Average additive error of our coreset and that of [82] on LeNet-300-100.