On Fine-Tuning Submodular Functions for Data Subset Selection

Megh Manoj Bhalerao

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington
2024

Committee:
Jeffrey Bilmes
Kevin Jamieson
Simon Du

Program Authorized to Offer Degree:
Electrical & Computer Engineering

University of Washington

Abstract

On Fine-Tuning Submodular Functions for Data Subset Selection

Megh Manoj Bhalerao

Chair of the Supervisory Committee:
Jeffrey Bilmes
Department of Electrical & Computer Engineering

We demonstrate that submodular functions, with fine-tuned hyperparameters, serve as extremely effective data subset (i.e., summary) selectors, better than the current state-of-the-art, for training machine learning systems on data subsets. To search and reduce the hyperparameter space, we introduce *meta-summarization* a technique designed to enhance computational efficiency of hyperparameter tuning. Meta-summarization chooses a subset of summaries based on their inter-summary diversity starting from a large set of generated summary candidates. This significantly reduces the summaries to train on relative to training on all of them. This approach enables meta-summarization to find the best performing hyperparameters for a submodular function faster than other hyperparameter search techniques, significantly reducing computation and time. We demonstrate that summaries generated using fine-tuned submodular functions outperform subset selection benchmarks such as DC-Bench (by $\approx 3\%$ absolute) and DeepCore (by $\approx 2\%$ absolute). Fine tuned submodular functions also outperform random and state-of-the-art k-means based subset selection for training a popular ViT-based (vision transformer) architecture, DaViT [20], on ImageNet, thus setting a new state-of-the-art for supervised subset selection.

Acknowledgements

I would like to express my deepest gratitude to everyone who supported me through my journey at the University of Washington.

First, I would like to thank my advisor Prof. Jeff Bilmes for his constant guidance during my tenure at the MELODI Lab. Jeff's immense knowledge in Machine Learning & Submodularity coupled with his hardworking and dynamic nature is truly inspiring. I learnt to systematically think about research problems while working with Jeff. Furthermore, Jeff's attention to detail during our discussions & presentations has taught me to always be precise with my experiments and results. Jeff has taught me to be a good researcher. I would also like to thank Jeff for his submarine software without which the experiments in my thesis would not have been possible.

I am extremely grateful to Prof. Kevin Jamieson and Prof. Simon Du for serving on my masters committee.

I would also like to thank all members of the MELODI Lab: Arnav Das, Gantavya Bhatt, Lilly Kumari, Armagan Er, Harshil Dadlani, Sahil Verma, Wenruo Bai, Shengjie Wang, Tianyi Zhou for their helpful interactions and discussions.

I would like to thank all my family in Seattle & US: Madhura, Yogi, Sharayu, Winnie, Seema & Manas, Asim & Nidhi & Vedant, Parimal & Anagha, Sanju Kaka & Sandhya Kaku, Medha Mavshi & Avinash Kaka, Prashant Mama & Madhavi Mami, and so many more to name a few for making me feel at home. I would also like to thank all my friends and housemates for a fun experience in Seattle. I would also like to thank all my friends from Undergrad & School in the US and India who have been there for me. This acknowledgement page is likely not enough to mention everyone, so apologies for missing out anyone.

Finally, I dedicate this work to my parents and grandparents for their unconditional love and support throughout my life.

Contents

1 Introduction

We study the problem of data subset selection in supervised learning, *i.e.* selecting a representative subset (or summary) of a larger dataset for more efficient model training. This problem is of special importance in today's era of big data where computation costs and carbon emissions from training AI models are becoming an environmental and financial concern [81, 10, 64]. We approach the data subset selection problem from the perspective of maximizing a submodular function (SF) [27] instantiated over the whole dataset with the "right" set of parameters. A submodular function $f_\phi : 2^V \rightarrow \mathbb{R}$ is defined as

$$f_\phi(v|A) \geq f_\phi(v|B) \iff A \subseteq B, \forall v \in V \tag{1}$$

where V is the ground set, $A, B \subseteq V$, and ϕ are parameters of f_ϕ. The notation $f(x|Y)$ denotes the valuation of the element x in the *context* of set Y. It is defined as

$$f(v|X) \triangleq f(v \cup X) - f(X) \tag{2}$$

A constrained submodular maximization problem can be expressed as

$$S^* = \arg\max_{X \in \mathcal{I}} f_\phi(X) \tag{3}$$

where $\mathcal{I}$ a collection of sets which define a given constraint (e.g., independent sets of a matroid). Submodular maximization is an NP-hard problem but the greedy algorithm with respect to function gains gives a constant factor approximation guarantee of $1-1/e$ [73] for cardinality constrained monotone submodular maximization (*i.e.* $\mathcal{I} = \{I : |I| \leq k\}$ where k is the cardinality limit). We can see from Equation 1 that f_ϕ has a diminishing returns property, and thus specific instances of SFs such as the facility location function, monotone-concave composed with modular, etc. naturally model set element diversity [75, 91]. Our summary is a set S obtained after running a submodular maximization algorithm $\mathcal{A}_{max}$ (such as Greedy, Stochastic Greedy [69], Parallel Greedy [16] etc.) on the problem defined in Equation 3. More details can about the algorithms that we use can be found in Section 3.

Using submodular maximization for data subset selection has been considered for over a decade [60, 59]. It is often the case that useful submodular functions are instantiated via a similarity kernel ($\mathcal{K}_\phi$) defined over set elements. Constructing $\mathcal{K}_\phi$ typically entails choosing parameters such as the feature extractor on raw data points and the similarity metric between data points. *Novel to this paper, we demonstrate that the choice of parameters ϕ (which includes adjustments to the corresponding $\mathcal{K}_\phi$ such as parameters of the similarity metric and other optional transforms as well as selection of f_ϕ itself) significantly impacts the quality and diversity of the subset S obtained via submodular maximization when evaluated based on held-out test-set accuracy. The* ***central idea*** *of this paper is to highlight the absolute necessity of fine-tuning the*

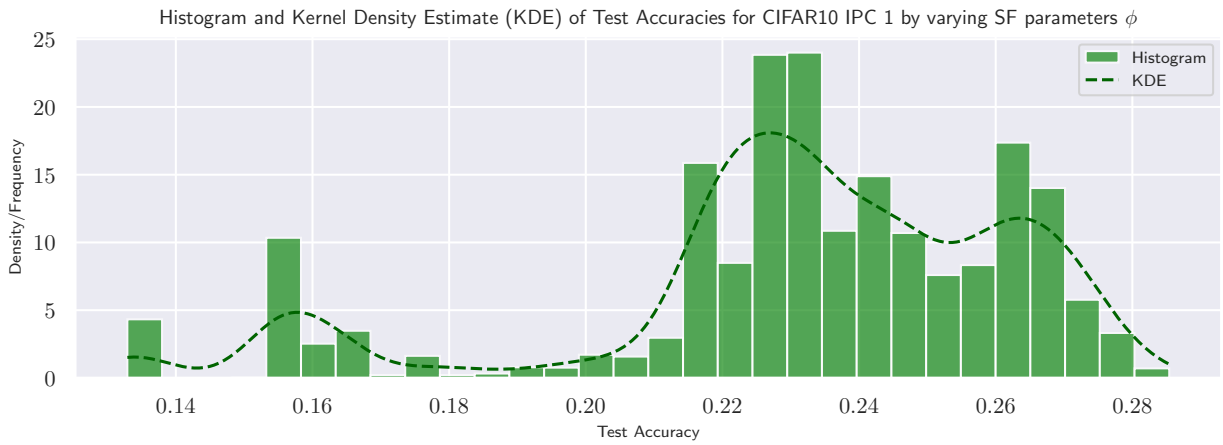


Figure 1: Histogram and kernel density estimate of test accuracies on CIFAR-10 dataset when selecting one Image Per Class (IPC) obtained by varying parameters ϕ. The plot shows test accuracies from 936 different summaries from 936 submodular functions after meta-summarization (Section 2). The mean (stdev) accuracy of uniformly-at-random subsets of the same size is 0.154 (0.28).

parameters ϕ of a submodular function f_ϕ in the context of data subset selection. It is not sufficient to plug-in an arbitrary submodular function for a summarization task, just as it is not sufficient to plug-in an arbitrary untrained neural network for a classification task. Our results go far beyond incremental improvements. Indeed, we show that well-tuned submodular functions serve as state-of-the-art data subset selectors.

Motivating Example. Figure 1 shows the histogram and corresponding kernel density estimation [15] of test accuracies obtained by training a model on summaries generated by f_ϕ using different parameters ϕ. We notice from Figure 1 that there is a significant spread of accuracies as we vary the parameters of f_ϕ. For some settings of ϕ, we see that SF-based selection performs worse than even the mean (which is 0.154) of multiple runs of simple uniformly-at-random subset sampling. This further motivates us to tune f_ϕ for subset selection, since a poor choice of parameters ϕ could (catastrophically) fail to give desired performance (the far left tail of the histogram).

Building on this motivation, our approach consists of the following three steps to fine-tune submodular functions: (1) generate summaries using grid-search over the submodular function hyperparameter space Φ ($\phi \in \Phi$); (2) Perform *Meta-Summarization* (again using submodularity) to summarize the generated summaries down to a smaller set of summaries (to bypass the expensive model training on *all* generated summaries). Computing summaries is much faster & cheaper since it consumes only CPU cycles, as opposed to model training on summaries which requires GPU cycles (see Section 2 for meta-summarization details); (3) Train & evaluate models on only the meta-summarized summaries (Algorithm 1 gives details).

How do we differ from previous work? Submodular functions have been widely used in the context of data summarization and subset selection (relevant literature is described in Section 4), but there lacks a comprehensive empirical study that benchmarks their effectiveness in the context of fine-tuned supervised subset selection. The most similar work to ours includes CRIAG [70], MILO [44] and [42]. CRIAG maximizes a facility location (FL) function [68, 98] using a Euclidean similarity measure to select subsets for faster model

training. They use features which are the gradients with respect to the last layer of the model. Although CRAIG uses an FL function, they do not perform submodular fine-tuning or meta-summarization. The purpose of our work is not to propose a new similarity measure, or feature extractor, but rather to show that submodular functions should be tuned before applying them to select subsets. Thus, any new similarity measure of feature extractor can be seamlessly integrated into our general framework.

MILO is a recent work which uses submodular functions for data subset selection. Although the "tuning" in MILO paper's title refers to standard deep-model architecture hyperparameter tuning (rather than the submodular fine-tuning we refer to in the present work), MILO does perform a bit of submodular tuning but it is limited to only an RBF kernel width tuning under a fixed cosine similarity metric. MILO also does not perform further similarity transforms & submodular fine-tuning such as our similarity matrix nearest neighbor KNN adjustment and a gravity-based transform (see Appendix A for further details). Moreover, MILO does not use any meta-summarization and greedy order training (Section 2 for details), and hence there is a possibility that they train on redundant summaries. This is unlike meta-summarization which is specifically optimized to train models on a diverse set of distinct summaries. Additionally, MILO selects new a new subset every R epochs, whereas we select a single static subset at the beginning of model training. We note, however, that our approach can be easily extended to selecting dynamic subsets every R epochs which we leave to future work.

The [42] paper uses submodular functions such facility location and disparity min [43] (although disparity min is not submodular) for supervised subset selection. Their setting, however, is limited to the **simple non-parametric** K-NN classifier. Moreover, they do not perform meta-summarization.

The papers [95, 96, 62, 84, 60, 59, 9] use submodular selection for **speech** and **natural language** data, where datasets & models are smaller (< 10000 examples) and low dimensional, and they do not perform any fine-tuning, whereas we use much larger and higher dimensional vision datasets (CIFAR-10 [52], CIFAR-100, TinyImagenet [71], and Imagenet-1K [79, 19]).

[95] performs submodular function based selection, but their approach is to relate submodular functions to traditional models such as Naive Bayes and K-Nearest Neighbor classifiers, whereas we perform subset selection to train modern neural networks such as ResNets [31], ConvNets [85], and a vision transformer (ViT) [20, 21]. Furthermore, the papers [96, 57, 95] for speech and active learning do not involve a meta-summarization stage, which is novel in our approach.

Our contributions can be summarized as: (1) We introduce a *principled* hyperparameter fine-tuning strategy for submodular functions used for data-subset selection which involves a novel meta-summarization procedure. Meta-summarization saves enormous computation by training on only a diverse and representative set of summaries generated by the grid-search hyperparameter sweep; (2) we empirically demonstrate that the greedy order of training on meta-summarized summaries finds the maximum and minimum accuracies of the hyperparameter sweep much faster than training on summaries in a random order (an example can be seen in Figure 3); (3) we set a new state-of-the-art for subset selection on computer vision datasets using

submodular functions. We beat the DC-Bench baseline by an average of $\approx 3\%$ (absolute) and the DeepCore benchmark by an average of $\approx 2\%$ (absolute) **across all datasets and subset sizes**; (4) our source code is highly modular and new datasets, feature extractors, submodular functions, hyperparameters, and subset selection benchmarks can be integrated for further evaluation. Our open-source code along with instructions to reproduce results can be found in the supplimentary material. We explain our approach in further detail in the next section.

2 Tuning Submodular Functions

Summary Generation. Commonly used strategies for hyper-parameter search include grid search [76], sequential search [33], Bayesian optimization [26], random search [4] etc. We use grid search to generate summaries corresponding to each configuration of our hyperparameter space since: (1) it is trivially parallelizable unlike sequential and Bayesian search (which can be parallelized although using more involved algorithms); and (2) random, sequential, and Bayesian search techniques require computation of the final objective of hyperparameter optimization for each configuration. In our case, the final objective is the test accuracy obtained after training a model on a data subset (*i.e.,* a summary), the computation of which is expensive (since it requires model training on a GPU) relative to summary generation. This is in contrast to our approach which first uses grid search to quickly generate all summaries (generating a summary is computationally inexpensive since submodular maximization requires only CPU cycles and can easily be parallelized) and then we train only on a few representative summaries out of all the generated summaries (obtained via meta-summarization, explained in Section 2). All summaries are generated using the Submaraine command line interface [8]. Let us call the set of all summaries generated by grid search as $\mathcal{S}$ (a meta groundset). Our goal is to show that it is critical to fine-tune parameters of an SF for the purpose of subset selection, and propose an efficient meta-summarization based strategy for tuning. Alternative hyperparameter tuning strategies are orthogonal to our work and can be seamlessly combined into our framework. We defer the study of fine-tuning SFs using other hyperparameter optimization algorithms to a future work.

Meta Summarization. Model training on all summaries in $\mathcal{S}$ is computationally expensive. Moreover, it is possible that different hyperparameter configurations result in the same summary indices (i.e., it is possible for $\arg\max f_{\phi_1} = \arg\max f_{\phi_2}$ with $\phi_1 \neq \phi_2$). To avoid such redundant trainings, to avoid similar trainings (where $\arg\max f_{\phi_1} \approx \arg\max f_{\phi_2}$), and to avoid unnecessary training on all summaries generated from the hyperparameter sweep (as would grid search do), we summarize $\mathcal{S}$ down to a representative set $\mathcal{Q}$. We use a submodular function instantiated using a Jaccard similarity measure [34] between elements in $\mathcal{S}$. The Jaccard matrix is computed using the Submarine CLI [8] by passing in $\mathcal{S}$ being the input. The Jaccard

Algorithm 1 Meta Summarization Procedure

Input: $\mathcal{S}$ (all summaries from hyperparameter sweep), $n = |\mathcal{S}|$ (number of summaries),
J (Jaccard similarity matrix between sets in $\mathcal{S}$).
Output: $\mathcal{Q}$ (summaries we want to train on, *i.e.* $\mathcal{Q} \subset \mathcal{S}$), C (conditional curvature array).
$\mathcal{Q} \leftarrow \emptyset$ ▷ Initialize meta summaries to null set
Instantiate f_{ms} following Equation 5
$g_{max} = \max_{i \subset \mathcal{S} \setminus \mathcal{Q}}(f_{ms}(i|\mathcal{Q}))$
while $g_{max} > 0$ **do** ▷ Loop while gains are > 0 since at 0 subsequent summaries are fully redundant.
    $i_{max} = \arg\max_{i \subset \mathcal{S} \setminus \mathcal{Q}}(f_{ms}(i|\mathcal{Q}))$ ▷ Get element index corresponding to maximum gain value
    $\mathcal{Q} \leftarrow \mathcal{Q}.\text{append}(i_{max})$ ▷ Append to preserve the order.
    cc $\leftarrow$ current conditional curvature
    $C \leftarrow C.\text{append}(\text{cc})$
    $g_{max} = \max_{i \subset \mathcal{S} \setminus \mathcal{Q}}(f_{ms}(i|\mathcal{Q}))$
After getting $\mathcal{Q}$, use conditional curvature array C to decide when to stop training (see Figure 2).

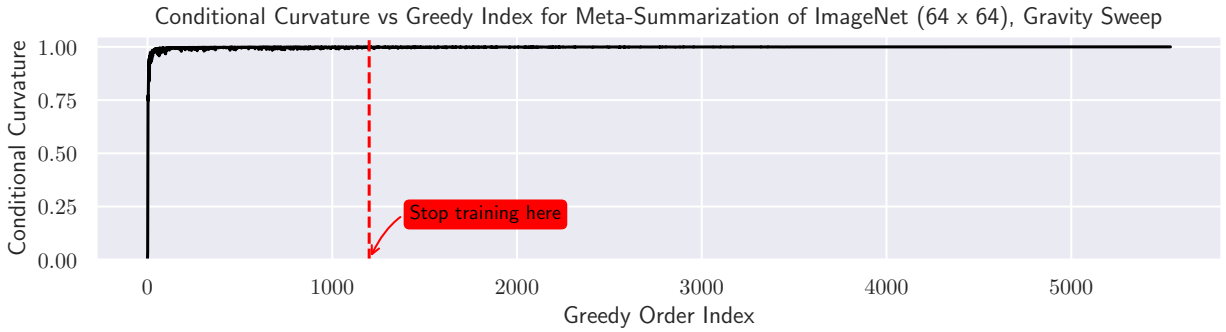


Figure 2: Meta-summarization conditional curvature for an ImageNet (64×64) sweep. We observe the conditional curvature plot and note that after index ≈ 1200, the curvature is *almost* unity. We thus selecting summaries at this point, since thereafter summaries are quite similar to the earlier ones.

similarity (intersection over union) between two sets X and Y is defined as:

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}. \tag{4}$$

We instantiate a meta-summarization (ms) FL function using this Jaccard matrix as follows:

$$f_{\text{ms}}(\mathcal{X}) = \sum_{Y \in \mathcal{S}} \max_{X \in \mathcal{X}} J(X, Y) \tag{5}$$

To obtain a representative set of summaries of $\mathcal{S}$, *i.e.,* $\mathcal{Q}$, we first need to specify the number of summaries that we are "willing" to train on, *i.e.* $|\mathcal{Q}|$. We make this decision based on the order provided by greedy algorithm as it maximizes the submodular function and based on a quantity we call the conditional curvature. Given a greedy order of summaries $q_1, q_2, \ldots$ define $Q_i = \{q_1, q_2, \ldots, q_i\}$. Then the conditional curvature is defined as $\text{cc}_i = 1 - f_{\text{ms}}(q_i|Q_{i-1})/f_{\text{ms}}(q_i)$. We note that this quantity is inspired by standard submodular curvature [39, 92] but is geared towards an ordering. The conditional curvature is used exactly as implemented in Submarine [8]. Note that cc_i is not monotone, but it does tend to reach its maximum value of unity (1) once the meta summarization has exhausted all diversity in the order. Hence we choose a threshold once we find an i such that $\text{cc}_i \geq \tau$ where τ is a threshold. This is shown in Figure 2 (for ImageNet (64×64)).

Training on Summaries in Greedy Order. We train the model, starting from scratch (random initial parameters) in each case, on candidate summaries presented in an order provided by the greedy submodular maximization algorithm run on the meta-summarization function f_{ms} mentioned above. as seen in Algorithm 1. We leverage the insight that submodular function naturally models diversity, and that greedy algorithm run up to cardinality constraint k also provides summarizes with cardinality constraint k' for all $k' \leq k$. This means that each next choice by greedy should be distinct from previous choices, and so it is likely that the greedy order will touch as many different kinds summaries as quickly as possible as it proceeds through its steps. For this reason, the meta-summarization's greedy order should allow us to gauge the performance of that particular hyper-parameter space as early as possible. This is because that greedy chooses set "elements" (which are summary subsets in the meta-summarization case) that are least similar to previously chosen summaries. Thus, we expect to see many different summaries in any prefix of the greedy order. We show from Figure 3 that the greedy order finds both the best **and** worst performing summaries (as measured by test accuracy) in the hyperparameter sweep (for CIFAR-10) faster than a large ensemble of random orders (i.e., 1000 random permutations in this case). This motivates us to train our model on the greedy order ($\mathcal{Q}$) from meta-summarization in order to find a good summary as quickly as possible, rather than simply training on summaries randomly or via a grid search (or Bayesian reasoning, etc.).

The utility of meta-summarization & training on greedy order, therefore, is two-fold. First, it gives us a diverse set of summaries to train on as early as possible. The user (*i.e.,* a machine learning researcher) has the ability to specify the number of summaries they wish to train on. They can make this choice based on computational constrains or based on how quickly the gains of the function f_{ms} decreases, as mentioned earlier in Section 2. Secondly, the greedy order training helps us identify the best and worst performing hyperparameter configurations early on in the training, hence helping us judge the effectiveness of a particular hyperparameter sweep early on in the training. Obviously, we do not care about the worst performing hyperparameter configuration and we only care about the worst performing hyperparameter configuration. However, the worst performing hyperparameter configuration (and a diverse set of hyperparameter configuration performances) comes as a byproduct of testing summaries in the greedy (diverse first) order. Our contention is that, while the present work is a form of *using submodularity for fine-tuning submodular hyperparameters*, our submodular-order meta-hyperparameter search could be used in any situation where hyperparameter tuning (or, say, neural architecture search or AutoML) is needed, but we leave this latter case to future work.

3 Experiments & Results

We evaluate our approach on standard computer vision datasets namely CIFAR-10 [52], CIFAR-100 [52], TinyImageNet [54], and ImageNet [19]. It must be noted all results are reported by first selecting a subset, and training a model from scratch (random initialization) on **only** the selected subset. This different from

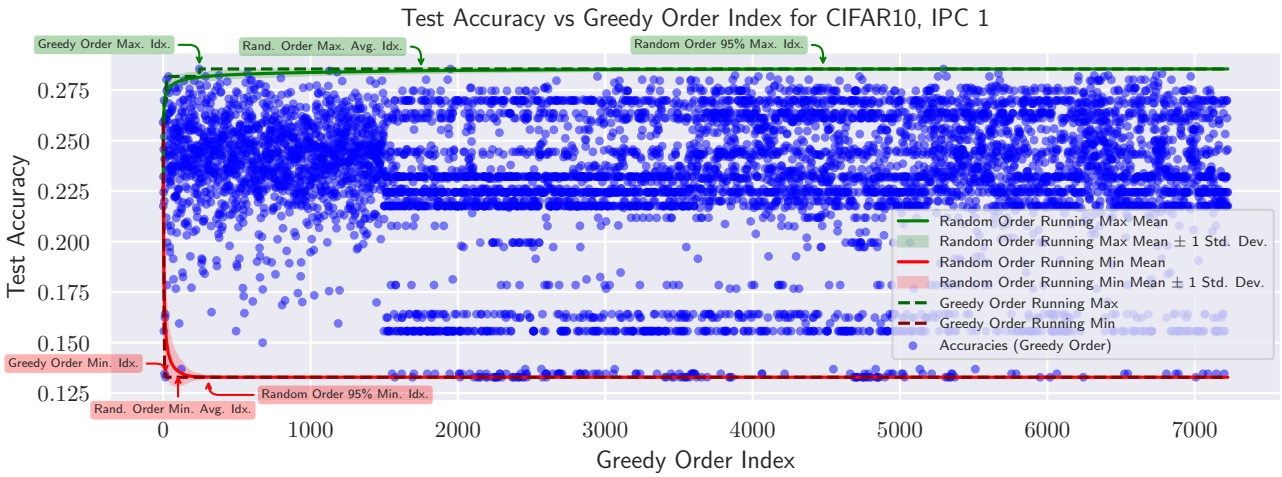


Figure 3: Running maximum and minimum test accuracy while training model on summaries in greedy order obtained from the meta-summarization stage. This figure shows that greedy order finds the best test accuracy (top left green box) faster than the average best index out of 1000 random permutations (top middle green box). Also, the greedy order finds the best significantly earlier than 95% confidence (top right green box), which is where 95% of the 1000 random permutations have found the best accuracy. The blue dots in this figure show the accuracies plotted in greedy order. Incidentally, we also note that since the greedy order is a form of diverse-first order, it also finds the worst performing accuracy (bottom left red box) and a diversity of different performing accuracies faster than both the average worst accuracy of the random orders (bottom middle red box) as well as the 95% confident worse performing accuracy (bottom right red box). The above behavior is the same across many different data sets, machine learning models, and summary sizes as shown in Section F.

the adaptive (or active) subset selection setting [70, 77, 90] which select new subsets every R epochs based partially on a partially trained model. Our approach can be easily extended to the adaptive setting, but this is a digression from the main goal of the paper which is to show that fine-tuned submodular functions are strong subset selectors.

Which Submodular Functions do we use? We use the Facility Location (FL) function which has shown to be effective in previous work such as [95, 70]. The FL function over a ground set V of training samples and evaluated on a set $X \subseteq V$ is defined as:

$$f_{fl}^{V}(X) = \sum_{v \in V} \max_{x \in X} S(v, x) \tag{6}$$

where S is matrix of similarity values between set elements of the ground set, *i.e.* $v \in V$. The FL function has additional advantages. This includes that it is monotone non-decreasing, i.e. $\forall A, B \subseteq V$ then $A \subseteq B \iff f_{fl}(A) \leq f_{fl}(B)$. Cardinality constrained monotone submodular maximization gives a $1 - 1/e$ constant factor approximation guarantee which is a better guarantee than maximization of non-monotone submodular functions such as graph-cut [24, 100]. Also, the query complexity (*i.e.,* complexity of each function evaluation) of FL is $\mathcal{O}(n^2)$, as opposed to, say, a determinantal point process (DPP) which involves an $\mathcal{O}(n^3)$ determinant evaluation. To perform submodular maximization, we use the Minoux accelerated greedy algorithm [67] which is a faster version of the standard greedy algorithm [73] with the same $1 - 1/e$ approximation guarantee. We choose all subsets to be class balanced in order for a fair comparison with

baselines [29, 17]. We utilize a partition matroid rank constraint (Appendix C for more details) to ensure class balance which, using the greedy procedure, still offers a $1/2$ guarantee in the worst case but in practice it is significantly better than this (simple small-scale empirical experiments show that its performance is upwards of 98% of an exhaustive search).

Tuned Submodularity Outperforms Coreset Baselines on DC-Bench. We compare tuned submodularity against DC-Bench [17] which is primarily a dataset distillation (DD) [80] benchmark, but it includes subset selection baselines as well such as Herding [97], Forgetting [89], and (what we call, see below) Nearest Neighbor K-Means based selection (NNKMeans). As is standard, a 3-layer ConvNet [53, 85, 87] is trained on subsets of CIFAR-10 and CIFAR-100 while a 4-layer version of the same architecture is used for TinyImagent and Imagenet, following the standard setting in DC-Bench [17] for fair comparison of results. We use the low resolution (64×64) ImageNet dataset following the dataset distillation literature (DD use the low resolution ImageNet for computational reasons). We also include an additional baseline where the subset is selected using a single "off-the-shelf" submodular function, with a dense Euclidean similarity matrix without additional transforms. We call this *Untuned Submodular*, since we do not perform any fine-tuning to obtain parameters of this SF, but rather heuristically choose the set of parameters for the FL (this can be thought of as a lower bound to our approach of fine-tuned submodular). The subset size is defined in terms of Images Per Class (IPC), and hence all the subsets are class balanced. Figure 5 shows tuned submodularity (green) outperforms the state-of-the-art NNKMeans across **all** datasets and subset sizes by an average of 2.5%. We see that fine-tuned submodularity significantly improves over untuned submodular. This provides further evidence that fine-tuning is necessary to obtain the best performance when using a submodular function for subset selection. Interestingly, we also see that *untuned submodular* (yellow) outperforms NNKMeans across most datasets and subset sizes.

Aside - a note on NNKMeans. DC-Bench proposes what we call Nearest Neighbor K-Means based selection [17] in the year 2022 which they call "K-Center". However, the term "K-Center" is used for a various objective for subset selection [50, 82, 23, 99] dating at least as far back as 2006. Hence, to avoid the discrepancy in terminology, we call the DC-Bench baseline "NNKMeans".

Figure 4 shows the comparison of Tuned Submodularity vs. representative & more computationally expensive works on dataset distillation such as Distribution Matching (DM) [102], Differentiable Siamese Augmentation [101], and Dataset Condensation [103]. Dataset distillation [93] inherently has an advantage over subset selection because it allows synthesizing new data points over simply selecting representative points from the whole dataset. This makes dataset distillation more competitive at much smaller sizes due to the flexibility of combining aspects from various images into a single newly synthesized image. We see from Figure 4 that distillation outperforms tuned submodularity and NNKMeans at smaller subset sizes. As subset size increases, however, fine-tuned submodularity even beats DD (and **always** beats NNKMeans). We do not include the state of the art Matching Training Trajectories [13] distillation baseline in Figure 4, since both it is extremely computationally expensive and also since our goal is not to outperform dataset

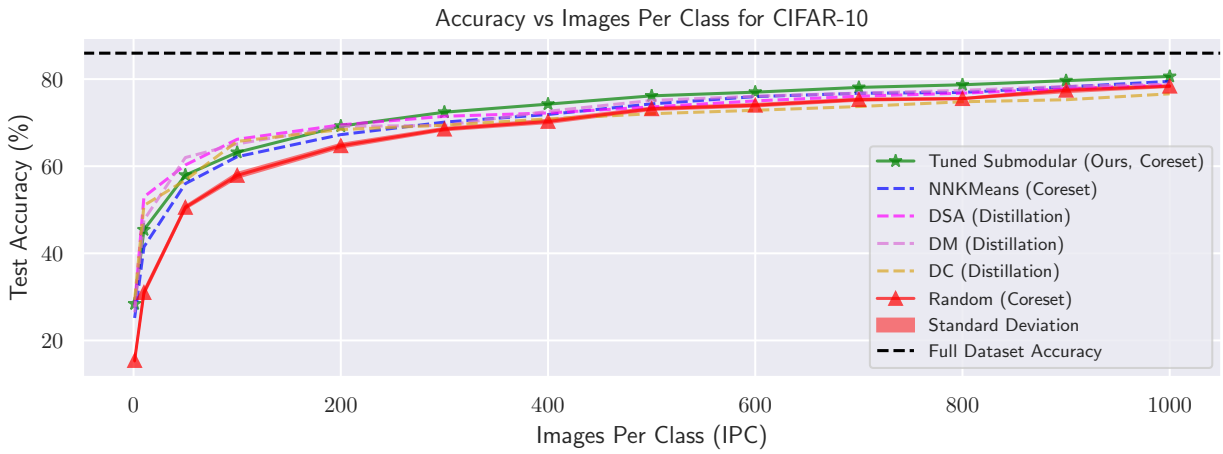


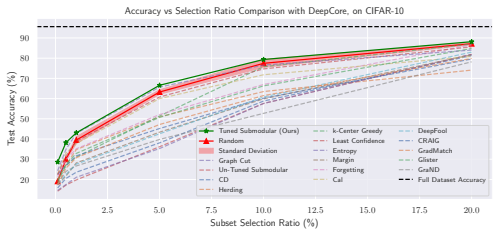
Figure 4: Fine-tuned submodularity vs. dataset distillation baselines [17] at varying IPC.

distillation, but rather demonstrate that it is possible to fine-tune a SF to outperform even certain distillation baselines as subset sizes increase. Further details can be found in [17].

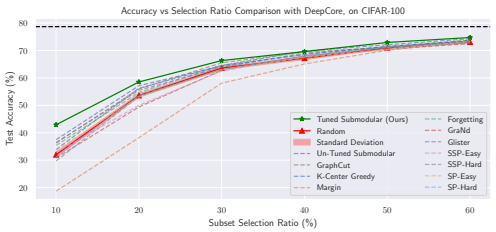
Figure 5: Test Accuracy Comparison with Subset Selection Baselines as reported in DC-Bench [17]. Each row is a particular dataset, whereas each column indicates a given subset size expressed in terms of images per class (IPC). The selected subsets are class balanced.

Tuned Submodularity Outperforms *all* Baselines in DeepCore. DeepCore [29] is a coreset benchmark with $\approx$ 12 subset selection methods [46, 70, 38] etc. The subset size is defined in terms of a percentage (p) of the whole dataset (class-balanced subsets selected). We train a ResNet18 model for 200 epochs on subsets following the setting in DeepCore for a fair comparison. We can see from Figure 6a (CIFAR-10) and Figure 6b (CIFAR-100) that Tuned Submodularity outperforms all baselines reported in DeepCore by $\approx 2\%$. It must be noted that random is a strong baseline as reported in the original DeepCore paper but this is outperformed by Fine-Tuned Submodularity.

Tuned Submodularity Outperforms Random & NNKMeans Baselines for ViT Training. We eval-



(a) Fine-tuned Submodularity compared with the DeepCore benchmark [29] with varying subset sizes for CIFAR-10.



(b) Fine-tuned Submodularity compared with the DeepCore benchmark [29] with varying subset sizes for CIFAR-100

Figure 6

uate fine-tuned submodularity against random subset selection and state-of-the-art NNKMeans on the recent Dual Attention Vision Transformer [20] (DaViT) on full-resolution ImageNet (Figure 7). This demonstrates the applicability of submodular subset selection for training vision transformer based architectures in addition to convolutional based ones. Figure 7 shows fine-tuned submodularity consistently outperforming random and (what is often considered state-of-the-art) NNKMeans baselines across various selection percentages. We leave benchmarking against the wide array of other subset selection baselines on DaViT training as future work, due to the computationally expensive nature of training ViT models.

Fine-tuning Submodularity is Faster than NNKMeans. Figure 8 illustrates the timing numbers for NNKMeans vs. fine-tuned submodularity, as well as speedups of fine-tuned submodular over NNKMeans. Fine-tuned submodularity has a speedup of $> 2\times$ over NNKMeans for every selection ratio even though NNKMeans performs worse. Thus, fine-tuning submodularity is both faster and better.

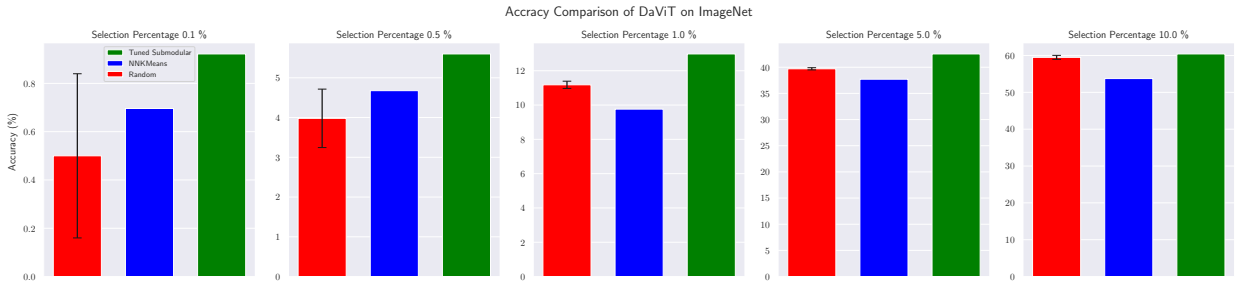


Figure 7: Comparison of fine-tuned submodular against random and state-of-the-art NNKMeans baselines for training a vision transformer DaViT [20] on full-resolution ImageNet-1K. The fine-tuned submodular approach consistently outperforms both baselines across all selection percentages, demonstrating its effectiveness in subset selection for such vision transformer-based architectures.

4 Related Literature

Coresets are (often weighted) subsets of data constructed to approximate characteristics such as loss, geometric structure etc. of the whole dataset for applications such as model training, clustering, and so on. Grad-Match [46] and Glister [47] construct coresets by matching proxy objectives such as validation loss and model gradients between the subset and whole dataset. Papers [45, 88, 48] include applications

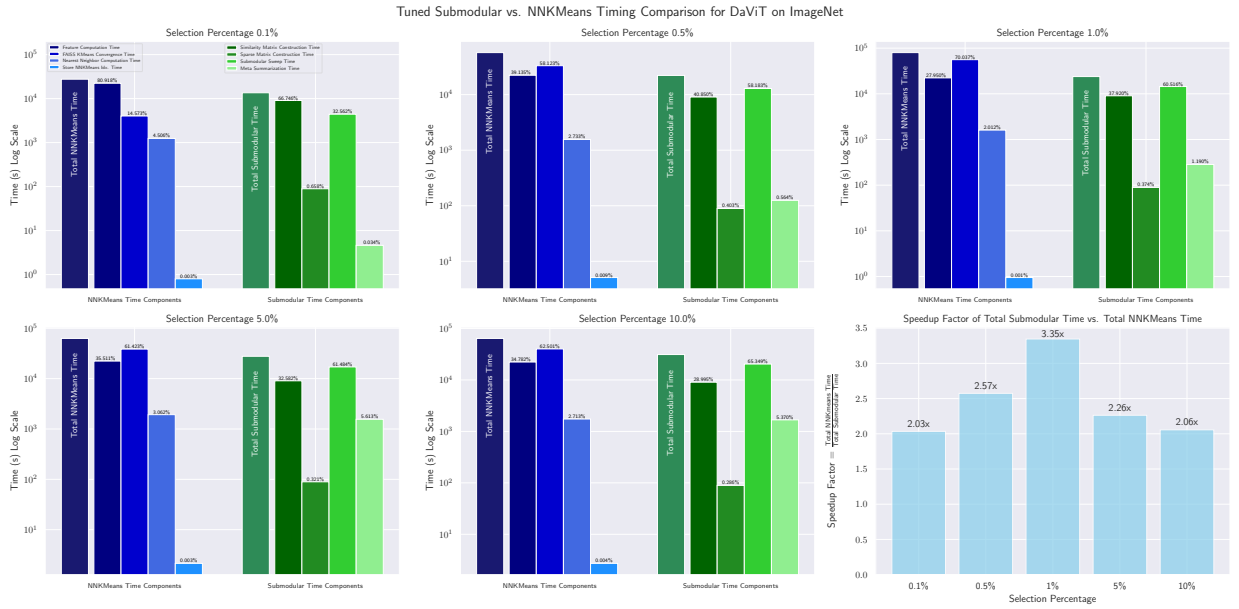


Figure 8: Figure showing fine-tuned submodularity is $> 2\times$ faster than NNKMeans for subset selection for training the DaViT model on ImageNet. NNKMeans is expensive due to the very high-dimensional gradient based features that we use showing NNKMeans poorly scales to such high-dimensional cases.

of Grad-Match & Glister for efficient hyperparameter tuning, replay buffer selection in continual learning, and in semi-supervised learning, respectively. CRAIG [70] constructs a facility location (FL) [98, 23] function based on a similarity matrix between gradients, and selects subsets using submodular maximization. Papers [41, 55, 51] use submodular mutual information maximization [38] to select subsets for downstream tasks like domain adaptation, semi-supervised learning, and rare category object detection. The paper [90] uses sensitivity sampling but this becomes computationally intractable for larger datasets. The paper [74] uses periodic random sampling after every, say R, epochs, for faster model training & convergence. Their setting is fundamentally different from ours since they repeatedly sample random subsets every few epochs, which potentially allows them to explore many more training samples, while we select a single fixed subset at the beginning of model training. Other classic coreset work [65, 32, 11, 12, 1, 30] have stronger theoretical performance guarantees, but evaluate only on smaller and lower dimensional datasets. We point the reader to [25] for a detailed review of coreset approaches. *In contrast to previous coreset work, we are the first to present a principled fine-tuning strategy for submodular functions for the purpose of data subset selection in a supervised learning setting.*

Submodular Functions (SF) and optimization algorithms have been widely studied in theoretical & empirical work like [60, 59, 73, 94, 18, 67, 69, 37, 27]. The classic paper [73] proved the constant factor $1 - 1/e$ approximation guarantee for monotone cardinality constrained submodular maximization using a greedy algorithm on function gains. Accelerated greedy [67] further improved the runtime by using a priority queue to store element gains (we leverage this algorithm in our work). [69] introduced a faster stochastic version of the greedy algorithm and showed it to work well in practice. Furthermore, numerous works on submodular minimization have been proposed [86, 66, 36, 35] We direct the reader to a detailed review of

submodular optimization [22, 27, 63, 7] for further information.

Submodularity in Machine Learning. SFs have been used to select representative subsets of data in modalities such as speech, vision, natural language, etc. The papers [60, 95, 62, 96] are a line of work exploring use of SF maximization for the automatic speech recognition task. Additionally [9, 49, 58, 61, 57, 56, 14, 3, 84] employ SFs for NLP tasks such as machine translation and multi-document extractive summarization [72].

5 Conclusions, Limitations and Future Directions

We demonstrated that it is necessary to fine-tune submodular functions when applying them to data subset selection to achieve optimal performance. Our findings suggest that when tuned appropriately, SFs outperform existing subset selection baselines such as DC-Bench [17], DeepCore [29], as well as on Vision Transformer Model training [20] on data subsets. We note that evaluating a facility location is $\mathcal{O}(n^2)$ making it computationally infeasible for large ground set sizes, but recent work addresses this [5]. Although our current work is limited to the scale of the ImageNet-1K dataset ($\approx$ 1.3M samples) currently, we envision easily extending our work to the scale of larger datasets such as ImageNet-21K, multimodal vision-language datasets such as DataComp [28], and so on. Scaling submodularity involves computational challenges, but it can be tackled with FL functions instantiated with sparse matrices, using feature based functions (which are evaluable in $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$), and faster submodular optimization algorithms. Practitioners may use our highly modular & easy to use codebase to integrate newer datasets, modalities, subset selection algorithms, ultimately contributing to data-efficient machine learning.

References

[1] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52(1):1–30, 2005.

[2] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *arXiv preprint arXiv:1203.6402*, 2012.

[3] Ramakrishna Bairi, Rishabh Iyer, Ganesh Ramakrishnan, and Jeff Bilmes. Summarization of multi-document topic hierarchies using submodular mixtures. In Chengqing Zong and Michael Strube, editors, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 553–563, Beijing, China, July 2015. Association for Computational Linguistics.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[5] Gantavya Bhatt, Arnav Das, and Jeff Bilmes. Deep submodular peripteral networks, 2024.

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[7] Jeff Bilmes. Submodularity in machine learning and artificial intelligence, 2022.

[8] Jeff Bilmes. Submarine, 2024.

[9] Jeff A Bilmes, Hui Lin, and Andrew Guillory. Applications of submodular functions in speech and nlp. In *MLSLP*, 2011.

[10] Semen Andreevich Budennyy, Vladimir Dmitrievich Lazarev, Nikita Nikolaevich Zakharenko, Aleksei N Korovin, OA Plosskaya, Denis Valer'evich Dimitrov, VS Akhripkin, IV Pavlov, Ivan Valer'evich Oseledets, Ivan Segundovich Barsola, et al. Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai. In *Doklady Mathematics*, volume 106, pages S118–S128. Springer, 2022.

[11] Trevor Campbell and Tamara Broderick. Bayesian coreset construction via greedy iterative geodesic ascent. In *International Conference on Machine Learning*, pages 698–706. PMLR, 2018.

[12] Trevor Campbell and Tamara Broderick. Automated scalable bayesian inference via hilbert coresets. *The Journal of Machine Learning Research*, 20(1):551–588, 2019.

[13] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[14] Yllias Chali, Moin Tanvee, and Mir Tafseer Nayeem. Towards abstractive multi-document summarization using submodular function-based framework, sentence compression and merging. In Greg Kondrak and Taro Watanabe, editors, *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 418–424, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[15] Yen-Chi Chen. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1(1):161–187, 2017.

[16] Yixin Chen, Tonmoy Dey, and Alan Kuhnle. Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. *Advances in Neural Information Processing Systems*, 34:25528–25539, 2021.

[17] Justin Cui, Ruochen Wang, Si Si, and Cho-Jui Hsieh. Dc-bench: Dataset condensation benchmark. *Advances in Neural Information Processing Systems*, 35:810–822, 2022.

[18] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 1057–1064, Madison, WI, USA, 2011. Omnipress.

[19] 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.

[20] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*, pages 74–92. Springer, 2022.

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[22] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*, pages 11–26. Springer, 2003.

[23] Reza Zanjirani Farahani and Masoud Hekmatfar. *Facility location: concepts, models, algorithms and case studies.* Springer Science & Business Media, 2009.

[24] Uriel Feige, Vahab S Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[25] Dan Feldman. Introduction to core-sets: an updated survey. *arXiv preprint arXiv:2011.09384*, 2020.

[26] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[27] Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

[28] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *arXiv preprint arXiv:2304.14108*, 2023.

[29] Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. In *Database and Expert Systems Applications: 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part I*, page 181–195, Berlin, Heidelberg, 2022. Springer-Verlag.

[30] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[31] 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.

[32] Jiawei Huang, Ruomin Huang, Wenjie Liu, Nikolaos Freris, and Hu Ding. A novel sequential coreset method for gradient descent algorithms. In *International Conference on Machine Learning*, pages 4412–4422. PMLR, 2021.

[33] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer, 2011.

[34] GI Ivchenko and SA Honov. On the jaccard similarity test. *Journal of Mathematical Sciences*, 88:789–794, 1998.

[35] Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112:45–64, 2008.

[36] Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1230–1237. SIAM, 2009.

[37] Rishabh Iyer, Stefanie Jegelka, and Jeff Bilmes. Fast semidifferential-based submodular function optimization. In *International Conference on Machine Learning*, pages 855–863. PMLR, 2013.

[38] Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR, 2021.

[39] Rishabh K Iyer, Stefanie Jegelka, and Jeff A Bilmes. Curvature and optimal algorithms for learning and minimizing submodular functions. *Advances in Neural Information Processing Systems*, 26, 2013.

[40] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[41] Athresh Karanam, Krishnateja Killamsetty, Harsha Kokel, and Rishabh Iyer. Orient: Submodular mutual information measures for data subset selection under distribution shift. *Advances in neural information processing systems*, 35:31796–31808, 2022.

[42] Vishal Kaushal, Rishabh Iyer, Suraj Kothawade, Rohan Mahadev, Khoshrav Doctor, and Ganesh Ramakrishnan. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1289–1299. IEEE, 2019.

[43] Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh Iyer. Submodlib: A submodular optimization library. *arXiv preprint arXiv:2202.10680*, 2022.

[44] Kirshnateja Killamsetty, Alexandre V Evfimievski, Tejaswini Pedapati, Kiran Kate, Lucian Popa, and Rishabh Iyer. Milo: Model-agnostic subset selection framework for efficient model training and tuning. *arXiv preprint arXiv:2301.13287*, 2023.

[45] Krishnateja Killamsetty, Guttu Sai Abhishek, Aakriti Lnu, Ganesh Ramakrishnan, Alexandre Evfimievski, Lucian Popa, and Rishabh Iyer. Automata: Gradient based data subset selection for compute-efficient hyper-parameter tuning. *Advances in Neural Information Processing Systems*, 35:28721–28733, 2022.

[46] Krishnateja Killamsetty, Sivasubramanian Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pages 5464–5474. PMLR, 2021.

[47] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8110–8118, 2021.

[48] Krishnateja Killamsetty, Xujiang Zhao, Feng Chen, and Rishabh Iyer. Retrieve: Coreset selection for efficient and robust semi-supervised learning. *Advances in Neural Information Processing Systems*, 34:14488–14501, 2021.

[49] Katrin Kirchhoff and Jeff Bilmes. Submodularity for data selection in machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 131–141, 2014.

[50] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[51] Suraj Kothawade, Saikat Ghosh, Sumit Shekhar, Yu Xiang, and Rishabh Iyer. Talisman: targeted active learning for object detection with rare classes and slices using submodular mutual information. In *European Conference on Computer Vision*, pages 1–16. Springer, 2022.

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

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

[54] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[55] Changbin Li, Suraj Kothawade, Feng Chen, and Rishabh Iyer. Platinum: Semi-supervised model agnostic meta-learning using submodular mutual information. In *International Conference on Machine Learning*, pages 12826–12842. PMLR, 2022.

[56] Jingxuan Li, Lei Li, and Tao Li. Multi-document summarization via submodularity. *Applied Intelligence*, 37:420–430, 2012.

[57] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In Ron Kaplan, Jill Burstein, Mary Harper, and Gerald Penn, editors, *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, Los Angeles, California, June 2010. Association for Computational Linguistics.

[58] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520, 2011.

[59] Hui Lin, Jeff Bilmes, and Shasha Xie. Graph-based submodular selection for extractive summarization. In *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 381–386. IEEE, 2009.

[60] Hui Lin and Jeff A Bilmes. How to select a good training-data subset for transcription: submodular active selection for sequences. In *Interspeech*, pages 2859–2862, 2009.

[61] Hui Lin and Jeff A Bilmes. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871*, 2012.

[62] Hui-Ching Lin and Jeff A. Bilmes. Optimal selection of limited vocabulary speech corpora. In *Interspeech*, 2011.

[63] Yajing Liu, Edwin KP Chong, Ali Pezeshki, and Zhenliang Zhang. Submodular optimization problems and greedy strategies: A survey. *Discrete Event Dynamic Systems*, 30:381–412, 2020.

[64] Alexandra Luccioni, Alexandre Lacoste, and Victor Schmidt. Estimating carbon emissions of artificial intelligence [opinion]. *IEEE Technology and Society Magazine*, 39(2):48–51, 2020.

[65] Tung Mai, Cameron Musco, and Anup Rao. Coresets for classification–simplified and strengthened. *Advances in Neural Information Processing Systems*, 34:11643–11654, 2021.

[66] S Thomas McCormick. Submodular function minimization. *Handbooks in operations research and management science*, 12:321–391, 2005.

[67] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*, pages 234–243. Springer, 2005.

[68] Pitu B. Mirchandani and Richard L. Francis, editors. *Discrete Location Theory*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York / Chichester / Brisbane / Toronto / Singapore, 1990. Includes bibliographies and index.

[69] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[70] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

[71] Mohammed Ali mnmoustafa. Tiny imagenet, 2017.

[72] N Moratanch and S Chitrakala. A survey on extractive text summarization. In *2017 international conference on computer, communication and signal processing (ICCCSP)*, pages 1–6. IEEE, 2017.

[73] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–i. *Math. Program.*, 14(1):265–294, dec 1978.

[74] Patrik Okanovic, Roger Waleffe, Vasilis Mageirakos, Konstantinos E Nikolakakis, Amin Karbasi, Dionysis Kalogerias, Nezihe Merve Gürel, and Theodoros Rekatsinas. Repeated random sampling for minimizing the time-to-accuracy of learning. *arXiv preprint arXiv:2305.18424*, 2023.

[75] Susan Hesse Owen and Mark S Daskin. Strategic facility location: A review. *European journal of operational research*, 111(3):423–447, 1998.

[76] Fabrício José Pontes, GF Amorim, Pedro Paulo Balestrassi, AP Paiva, and João Roberto Ferreira. Design of experiments and focused grid search for neural network parameter optimization. *Neurocomputing*, 186:22–34, 2016.

[77] Vishak Prasad, Colin White, Paarth Jain, Sibasis Nayak, Rishabh K Iyer, and Ganesh Ramakrishnan. Speeding up nas with adaptive subset selection. In *First Conference on Automated Machine Learning (Late-Breaking Workshop)*, 2022.

[78] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[79] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[80] Noveen Sachdeva and Julian McAuley. Data distillation: A survey. *arXiv preprint arXiv:2301.04272*, 2023.

[81] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[82] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[83] John Shawe-Taylor and Nello Cristianini. Machine learning and kernel methods. *American Mathematical Society*, 686:337–404, 1950.

[84] Yusuke Shinohara. A submodular optimization approach to sentence set selection. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4112–4115, 2014.

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

[86] Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. *Advances in Neural Information Processing Systems*, 23, 2010.

[87] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[88] Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. Gcr: Gradient coreset based replay buffer selection for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 99–108, 2022.

[89] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[90] Murad Tukan, Samson Zhou, Alaa Maalouf, Daniela Rus, Vladimir Braverman, and Dan Feldman. Provable data subset selection for efficient neural network training. *arXiv preprint arXiv:2303.05151*, 2023.

[91] Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. Graph cut based image segmentation with connectivity priors. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[92] Jan Vondrák. Submodularity and curvature: The optimal algorithm (combinatorial optimization and discrete algorithms). *RIMS Kôkyûroku Bessatsu*, 23:253–266, 2010.

[93] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[94] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1954–1963, Lille, France, 07–09 Jul 2015. PMLR.

[95] Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, and Jeff Bilmes. Submodular subset selection for large-scale speech training data. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3311–3315. IEEE, 2014.

[96] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 721–726, 2013.

[97] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009.

[98] G. W. Wolf. Facility location: concepts, models, algorithms and case studies. *International Journal of Geographical Information Science*, 25(2):331–333, 2011.

[99] Ling-Yun Wu, Xiang-Sun Zhang, and Ju-Liang Zhang. Capacitated facility location problem with general setup cost. *Computers & Operations Research*, 33(5):1226–1241, 2006.

[100] Faliu Yi and Inkyu Moon. Image segmentation: A survey of graph-cut methods. In *2012 international conference on systems and informatics (ICSAI2012)*, pages 1936–1941. IEEE, 2012.

[101] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, 2021.

[102] Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6503–6512, 2021.

[103] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021.

Appendices: On Fine-Tuning Submodular Functions for Data Subset Selection

A Details on Hyperparameters

Feature Extractors. We use the following three types of features -

- **Activation Based.** Activations from the pretrained vision transformer part of CLIP [78].
- **Activation Based.** Activations from the penultimate layer of a one-epoch trained model. The model is chosen in accordance to the benchmark that we are comparing against. For example, to compare against DC-Bench [17], we train a depth 3 convnet (or depth 4, depending on the dataset) for one epoch and use the penultimate layer activation as features. We choose the convnet since we train the same model to compare against DC-Bench (for a fair comparison). We do the same using ResNet18 [31] to compare against the DeepCore [29] benchmark.
- **Gradient Based.** For comparing against DC-Bench and DeepCoreWe, use gradients of all the model parameters with respect to the loss of a given data-points as features, similar to [70]. However, while [70] use only the gradients with respect to the last layer parameters, our feature vector has gradients with respect to *all* the model parameters. The model is trained for one-epoch, similar to [17]. More specifically, if (x, y) is the datapoint, $\mathcal{L}$ is the loss, θ are all the model parameters, $\mathcal{M}^1$ is a model trained for one epoch, and z is the feature vector, then

$$z = \texttt{flatten}(\nabla_\theta \mathcal{L}(\mathcal{M}^1(x), y)) \tag{7}$$

  For DaViT training, we train the ViT model for 2 epochs, and compute gradients with respect to only the last 2 layers of the model (still giving us a 790K dimensional gradient vector) due to computational resource constrains.

Similarity Kernels. We use the following similarity kernels. All similarity kernels are constructed using Submarine [8] (for low dimensional features) and FAISS [40] (for high dimensional features). -

- **Simeuclid.** Simeuclid is based on the euclidean distance measure. Given two feature vectors z_i and z_j, exponent parameter γ, simeuclid similarity matrix S is defined as

$$S(i,j) = \max_{i,j}(||z_i - z_j||_2^\gamma) - ||z_i - z_j||_2^\gamma \tag{8}$$

  where $||.||_2$ is L2 norm, and γ is a parameter that we tune over.

  In practice we use $\gamma \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 5 \times 10^{-1}, 1, 2, 5, 10\}$.

- **RBF.** Given feature vectors z_i and z_j, kernel width w, matrix normalization specification d, the radial

basis function similarity kernel [83] is defined as

$$S(i,j) = \exp\left(\frac{||z_i - z_j||_2}{w \times d}\right) \tag{9}$$

We tune over the w and d parameters. In practice we use $w \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 5 \times 10^{-1}, 1, 2, 5, 10\}$, and $d \in$ {Min, Max, None, Sum, Mean} where the specification defined by d is computed on the euclidean distance matrix $E(i,j) = ||z_i - z_j||_2$.

- **Cosine.** Given the standard cosine similarity matrix $C(i,j) = \frac{z_i \cdot z_j}{||z_i||_2 ||z_j||_2}$ and standard vector dot product operator $\cdot$, we use the following variants of C

$$C_1 = 1 + C \tag{10}$$

$$C_2 = \texttt{ReLU}(C) \tag{11}$$

where $\texttt{ReLU}(x) = \max(0, x)$. $C1$ and $C2$ ensure that all entries in the matrices positive. Positive values in $C1$ and $C2$ are necessary for the facility location function to be submodular. There are no additional parameters associated with the cosine similarity variants that we need to tune.

Transforms on Similarity Kernel. In addition to tuning over different choices of feature extractors, similarity matrices & corresponding parameters, we perform additional transforms on the similarity matrix such as -

- **K-Nearest Neighbor Transform.** We assymetrize the similarity matrix S using the K-Nearest neighbor (KNN) transform. For each row of the similarity matrix S, the KNN transform preserves only the top K similarity values and zeros out the remaining values.

$$S_{knn}(i,:) = \texttt{top\_k\_sim}(S(i,:)) \tag{12}$$

The KNN transform is useful for preventing early saturation of the conditional curvature of the FL function, as seen from Figure 9. In practice we tune the K-Value (k) in the KNN transform over the following values - $k \in \frac{n}{s} \times \{0.25, 0.5, 1, 1.25, 1.5, 1.75, 2.0, s\}$, where n is the ground set size, s is the summary size. $\frac{n}{s}$ intuitively denotes the number of data-points that are to be "represented for" by each summary point. We choose $\frac{n}{s}$ as the reference point, and tune over values lesser and greater than $\frac{n}{s}$ by a few multiplicative factors, as shown earlier.

- **Gravity Transform.** The gravity transform can be interpreted as a soft version of the KNN transform. Instead of zero-ing out similarities after a certain KValue, the gravity transform increases similarity values above a certain percentile, while reducing similarity values after below that percentile. Given

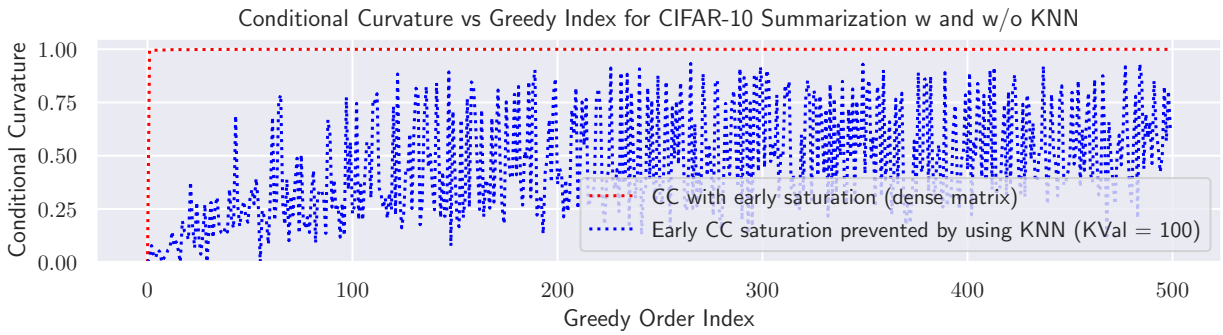


Figure 9: Conditional curvature obtained by running greedy maximization algorithm on similarity matrix with and without the K-Nearest Neighbor transform. It can be seen that using KNN (blue) prevents saturation of conditional curvature (red).

$g \in (-100, 100)$ ("strength" of gravity transform) and a fulcrum value $f \in [0, 100]$ (which essentially indicates a percentile point towards or away from which the similarity values are pulled/pushed), the gravity transform is expressed mathematically as

$$f_{\alpha,\beta}(x) = \frac{1}{(x^{1/\log_2(\beta)} - 1)^{\alpha} + 1} \tag{13}$$

where $\alpha = \frac{200}{g+100} - 1$ and $\beta = \frac{f}{100}$. Intuitively, the gravity transform can be thought of as "pulling"/"pushing" similarity values towards the point defined by f on this histogram of similarities, depending on whether the gravity value is positive/negative. In practice, we use $f \in \{1, 50, 75\}$ and we tune only over negative gravity values, $g \in \{-99, -50, -10\}$.

The KNN and Gravity Transform is used exactly as it is implemented in Submarine [8].

Mixture of Submodular Functions. In our experiments we use a weighted mixture of submodular functions of the form. We use a mixture of three weighted submodular functions each corresponding to one type of feature extraction method (2 activation based, and 1 gradient based). This weighted mixture is of the form -

$$f_{\phi} = w_1 f_{\phi}^1 + w_2 f_{\phi}^2 + w_3 f_{\phi}^3 \tag{14}$$

We tune over different values of weights w_1, w_2, and w_3. The f_{ϕ}^1, f_{ϕ}^2 and f_{ϕ}^3 correspond to a facility location functions corresponding to each of the three types of feature extractors. All other parameters are the same (*i.e.,* ϕ). We use the following combinations for weights -

$$(w_1, w_2, w_3) \in \{(2, 0, 8), (2, 2, 6), (2, 4, 4), \tag{15}$$

$$(2, 6, 2), (2, 8, 0), (4, 0, 6), (4, 2, 4), (4, 4, 2), (4, 6, 0), \tag{16}$$

$$(6, 0, 4), (6, 2, 2), (6, 4, 0), (8, 0, 2), (8, 2, 0), (10, 0, 0)\} \tag{17}$$

We utilize Weights & Biases [6] for experiment tracking and performing hyperparameter sweeps.

B Specific Hyperparameter Sweeps Performed

Searching over all hyperparameter configurations as defined in Appendix A is not computationally feasible due to the exponential nature of hyper-parameter spaces with respect to the number of hyper-parameters, and available computational resources. Hyperparameters described in Appendix A result in $\approx 300K$ unique parameter configurations for *each* dataset, and selection ratio, tuning over which is intractable. We thus divide the hyperparmater space into smaller and tractable subspaces, and apply summary generation, meta-summarization, and greedy order training on the smaller configuration spaces, and report the best obtained test-set accuracies amongst the smaller hyperparameter spaces.

C Partition Matroid

Matroid. A matroid $(E, \mathcal{I})$ is defined over a ground set E and a set of independent sets $\mathcal{I}$, and each $I \in \mathcal{I}, I \subseteq E$. The set of independent sets $\mathcal{I}$ satisfy the following properties

- $\varnothing \in \mathcal{I}$
- If $A \in \mathcal{I}$, and $B \subset A$, then $B \in \mathcal{I}$.
- If $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|B| > |A|$, then $\exists\, b \in B \setminus A$, such that $A \cup \{b\} \in \mathcal{I}$.

More details on matroids can be found in [22].

Partition Matroid. A partition matroid's $(\mathcal{P} = (E, \mathcal{I}))$ independent sets have the property

$$\mathcal{I} = \{X \subset E : |X \cap E_i| < k_i, \forall i = 1, 2, \ldots, l\} \tag{18}$$

where $E_i \subseteq E$ and $E_i \cap E_j = \varnothing$, and $\bigcup_i E_i = E$ (*i.e.,* a partition of E), and k_i's are limits imposed on each partition (l is the number of partitions). The rank function r_p for a partition matroid is defined as

$$r_p(X) = \sum_{i=1}^{i=l} \max(|X \cap E_i|, k_i) \tag{19}$$

We define a partition matroid $\mathcal{P}_\mathcal{C}$ for our submodular maximization as follows

- E_i's are partitions defined using class labels C, i.e. each partition consists of data-points of a specific class, say c_i.

- The limits k_i for each partitions are equal to the subset size for each class (which is nothing but the total subset size divided by the number of classes, since the subsets are class balanced).

Given a total subset size s ($= \sum_{i=1}^{i=l} k_i$), we impose the following constrains

- $r_p(X^*) = s$ and
- $|X^*| = s$

Both of the above constrains ensure that X^* is class balances and also $|X^*| = s$. Here X^* is the set that results from greedy submodular maximization. Hence, our submodular maximization problem can be defined as

$$X^* = \underset{r_p(X)=s, |X|=s}{\arg\max} \; f(X) \tag{20}$$

In all our experiments we optimize Equation 20 to obtain subsets. The partition matroid constrain is used as a submarine [8] command line argument, since the partition matroid constrained optimization implementation is available in Submarine.

D Envelope Plot Generation Method

Envelope Plot Quantities. Algorithm 2 illustrates our strategy to obtain the envelope plots. We plot the following quantities in the envelope plots: (1) Running maximum ($C_{\max}^{\text{greedy}}$) and minimum ($C_{\min}^{\text{greedy}}$) test accuracy of the greedy order as obtained by running the greedy maximization algorithm on the Jaccard similarity matrix during meta-summarization, (2) Mean and ± 1 standard deviation of running maximum ($C_{\max}^{\text{rand}}$) and minimum ($C_{\min}^{\text{rand}}$) of 1000 random permutations of test accuracy to simulate random search of the space of generated summaries. These quantities are plotted against the index (x-axis) which denotes the index at which the hyperparameter search methods (i.e., greedy, random) obtain a certain accuracy. This allows us to compare how *fast* a certain search method obtains a certain test accuracy.

Envelope Plot Annotations. Furthermore, we annotate each envelope plot at certain points (explained next) allowing us to visually inspect the advantage of a greedy training order over random orders. We annotate the envelope plots at the with the following points (1) The points at which the running maximum and minimum of the greedy order achieves the global maximum and minimum test accuracy respectively, of the hyperparameter space. We denote these points as $g_{\max}$ and $g_{\min}$ (2) To quantify the effectiveness of the random hyperparameter search order, we annotate the following points:

- **Mean Index.** We calculate the index at which the running maximum and minimum of a certain random order amongst the 1000 random orders (permutations) achieves the global maximum and minimum. We calculate the mean of all such indices for each of the 1000 random orders (*i.e., permutations*), and

Algorithm 2 Envelope Plot Generation Method

Input: A_g (array of test accuracies in greedy order), p (number of random permutations), ℓ (length of A_g)
Output: μ^{rand}_{cmax}, σ^{rand}_{cmax}, μ^{rand}_{cmin}, σ^{rand}_{cmin}
$A_{\max} \leftarrow \max(A_g)$, $A_{\min} \leftarrow \min(A_g)$ ▷ Initializae global max/min accuracies
$P \leftarrow \texttt{permute}(A_g, p)$ ▷ P is the matrix of p permutations of A_g i.e., $P \in \mathbb{R}^{p\times\ell}$
$C^{\text{rand}}_{\max} \leftarrow \texttt{empty}(\mathbb{R}^{p\times\ell}), C^{\text{rand}}_{\min} \leftarrow \texttt{empty}(\mathbb{R}^{p\times\ell})$ ▷ Running max/min of random orders
$C^{\text{greedy}}_{\max} \in \texttt{empty}(\mathbb{R}^{1\times\ell}), C^{\text{greedy}}_{\min} \in \texttt{empty}(\mathbb{R}^{1\times\ell})$ ▷ Running max/min of greedy order.
for $i = 1$ to p **do** ▷ Iterating over all p permutations
  for $j = 1$ to ℓ **do** ▷ Iterating over the accuracies in each permutation
    $C^{\text{rand}}_{\max}(i,j) \leftarrow \max_{j'\in\{1,2,\ldots,j\}} P(i,j')$ ▷ Populate random order running max. array
    $C^{\text{rand}}_{\min}(i,j) \leftarrow \min_{j'\in\{1,2,\ldots,j\}} P(i,j')$ ▷ Populate random order running min. array
for $j = 1$ to ℓ **do**
  $C^{\text{greedy}}_{\max}(j) \leftarrow \max_{j'\in\{1,2,\ldots,j\}} A_g(j')$ ▷ Populate greedy order running max. array
  $C^{\text{greedy}}_{\min}(j) \leftarrow \min_{j'\in\{1,2,\ldots,j\}} A_g(j')$ ▷ Populate greedy order running min. array
for $j = 1$ to ℓ **do** ▷ Calculate mean and std. dev. of max/min of random orders
  $\mu^{\text{rand}}_{\text{cmax}}(j) \leftarrow \frac{1}{p}\sum_{i=1}^{p} C^{\text{rand}}_{\max}(j,i)$
  $\sigma^{\text{rand}}_{\text{cmax}}(j) \leftarrow \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left(C^{\text{rand}}_{\max}(j,i) - \mu^{\text{rand}}_{\text{cmax}}(j)\right)^2}$
  $\mu^{\text{rand}}_{\text{cmin}}(j) \leftarrow \frac{1}{p}\sum_{i=1}^{p} C^{\text{rand}}_{\min}(j,i)$
  $\sigma^{\text{rand}}_{\text{cmin}}(j) \leftarrow \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left(C^{\text{rand}}_{\min}(j,i) - \mu^{\text{rand}}_{\text{cmin}}(j)\right)^2}$
Plot: $\mu^{\text{rand}}_{\text{cmax}}(j)$, $\sigma^{\text{rand}}_{\text{cmax}}(j)$, $\mu^{\text{rand}}_{\text{cmin}}(j)$, $\sigma^{\text{rand}}_{\text{cmin}}(j)$, $C^{\text{greedy}}_{\max}(j)$, $C^{\text{greedy}}_{\min}(j)$ as functions of $j \in \{1, 2, \ldots, \ell\}$.

Algorithm 3 Annotation of Mean Indices

Input: P (matrix of random permutations from Algorithm 2), p (number of permutations), ℓ (length of A_g)
Output: $r^{\text{mean}}_{\max}, r^{\text{mean}}_{\min}$ ▷ Mean index of global max/min point of random orders.
Initialize: $r^{\text{mean}}_{\max} = 0$, $r^{\text{mean}}_{\min} = 0$
for $i = 1$ to p **do**
  $r^{\text{mean}}_{\max} = r^{\text{mean}}_{\max} + \min(\arg\max_{j\in\{1,2,\ldots,\ell\}} C^{\text{rand}}_{\max}(i,j))$ ▷ Sum lowest max. achieving indices
  $r^{\text{mean}}_{\min} = r^{\text{mean}}_{\min} + \min(\arg\min_{j\in\{1,2,\ldots,\ell\}} C^{\text{rand}}_{\min}(i,j))$ ▷ Sum lowest min. achieving indices
$r^{\text{mean}}_{\max} = \frac{r^{\text{mean}}_{\max}}{p}$ ▷ Final values of average max. index
$r^{\text{mean}}_{\min} = \frac{r^{\text{mean}}_{\min}}{p}$ ▷ Final values of average min. index

annotate these indices, for both the global maximum and minimum. These indices are denoted by $r^{\text{mean}}_{\max}$ and $r^{\text{mean}}_{\min}$ for the global maximum and minimum respectively. These annotations allow us to benchmark of how random hyperparameter search performs on an average with respect to the greedy order. This procedure is illustrated in Algorithm 3.

- **95%-ile Index.** Next, we annotate the smallest index at which at least 95% of the random hyperparameter search orders (*i.e. permutations*) achieved the global maximum and minimum of test accuracies. We denote these points by $r^{95\%}_{\max}$ and $r^{95\%}_{\min}$ respectively. The 95 percentile points allow us to compare greedy order global maximum with the index after which with high probability (which we quantify as 95% of the random orders having achieved the global maximum (minimum)) random orders have achieved the global maximum (minimum) of test accuracies. Algorithm 4 demonstrates this procedure.

Algorithm 4 Annotation of 95% Indices

Input: C_{max}^{rand}, C_{min}^{rand}, $A_{\max}$, $A_{\min}$, p, ℓ ▷ Quantities as defined in Line 1 Algorithm 2
Output: $r_{\max}^{95\%}, r_{\min}^{95\%}$ ▷ 95%−ile indices for max. and min.
Initialize: $c_{\max} = 0, c_{\min} = 0$ ▷ Counter of permutations that have reached global max./min.
Initialize: $t_{\max} = 0.95 \times p$, $t_{\min} = 0.95 \times p$ ▷ Target 95-percentile count
for $i = 1$ to ℓ **do**
    for $j = 1$ to p **do**
        if $C_{\max}^{\text{rand}}(j, i) \geq A_{\max}$ **then** ▷ Check if running max reached global max
            $c_{\max} \leftarrow c_{\max} + 1$
    if $c_{\max} \geq t_{\max}$ **then**
        $r_{\max}^{95\%} \leftarrow i$
        exit all nested loops ▷ Exit loop if count has reached 95-percentile threshold
for $i = 1$ to ℓ **do** ▷ Similar procedure as above for min.
    for $j = 1$ to p **do**
        if $C_{\min}^{\text{rand}}(j, i) \geq A_{\min}$ **then**
            $c_{\min} \leftarrow c_{\min} + 1$
    if $c_{\min} \geq t_{\min}$ **then**
        $r_{\min}^{95\%} \leftarrow i$
        exit all nested loops

E Block Iteration based Similarity Matrix Construction

Why use blocking-based iterative similarity matrix construction? Gradient based similarity matrix calculation involves high-dimensional feature vectors since gradient of model parameter with respect to the loss is often a very high dimensional vector, due to the large number of model parameters. The high dimensionality of the feature vectors makes it challenging to accommodate the entire design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ in the computer memory (either CPU or GPU memory), which will subsequently be used to calculate the similarity matrix $\mathbf{S}$. This effect is especially pronounced in the case of calculating the similarity matrix for gradient based features of the DaViT [20] model due to the large number of parameters in ViT based models. We calculate the gradients with respect to only the last two layers of the model. The dimensionality of the gradient-based feature vector in this setting is $790,000$ which amounts to a design matrix of size 4 Terabytes, using single-precision floating point (32 bit) representation, which is more than our compute infrastructure.

To overcome this compute-imposed limitation, we break down our design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ into smaller chunks which are feasible to fit into our compute memory and calculate the similarity values of these smaller chunks with each other in a doubly-nested loop based iterative fashion. These calculated similarity matrices are then assigned at appropriate spots in the larger similarity matrix. This provides us an alternate approach to construct large similarity matrices, specifically for the ImageNet dataset. Algorithm 5 illustrates our blocking-based procedure.

Algorithm 5 Similarity Matrix Calculation for Gradient Based Features - Iterative Blocking Trick

Input: Network model $\mathcal{M}$, loss function $\mathcal{L}$, dataloader $\mathcal{D}$, mini-batch size b
Output: Similarity matrix $\mathbf{S}$
Initialize: Similarity matrix $\mathbf{S}$ of dimensions $(|\mathcal{D}|, |\mathcal{D}|)$ with zeros
Initialize: Batch start indices for outer and inner loops: $\text{start}_{\text{outer}} = 0$ and $\text{start}_{\text{inner}} = 0$
for each mini-batch B_{outer} in $\mathcal{D}$ **do** ▷ Outer loop to iterate through matrix rows in batches
    $\text{start}_{\text{inner}} = 0$
    $G_{\text{outer}} \leftarrow \textsc{ComputeGradients}(B_{\text{outer}}, \mathcal{N}, \mathcal{L})$ ▷ Use Algorithm 6
    for each mini-batch B_{inner} in $\mathcal{D}$ **do** ▷ Inner loop to iterate through matrix columns in batches
        $G_{\text{inner}} \leftarrow \textsc{ComputeGradients}(B_{\text{inner}}, \mathcal{N}, \mathcal{L})$
        $\mathbf{S}_{\text{block}} = \texttt{compute\_similarity\_matrix}(G_{\text{outer}}, G_{\text{inner}})$ ▷ Pairwise batch similarity ($\mathbb{R}^{b \times b}$)
        $\mathbf{S}[\text{start}_{\text{outer}} + b, \text{start}_{\text{inner}} + b] \leftarrow S_{\text{block}}$ ▷ Assign submatrix with similarity values
        $\text{start}_{\text{inner}} \leftarrow \text{start}_{\text{inner}} + b$ ▷ Update column counter
    $\text{start}_{\text{outer}} \leftarrow \text{start}_{\text{outer}} + b$ ▷ Update row counter
return S

Algorithm 6 Compute Gradients

function $\textsc{ComputeGradients}(B, \mathcal{M}, \mathcal{L})$ ▷ Input: Mini-batch, model, loss function
    $(\mathbf{X}, \mathbf{y}) \leftarrow B$ ▷ Unpack batch
    $\hat{\mathbf{y}} \leftarrow \mathcal{M}(\mathbf{X})$ ▷ Forward pass
    $\ell \leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ ▷ Compute loss
    $G \leftarrow \text{Autograd}(\ell, \mathcal{M}.\text{parameters}())$ ▷ Batch gradients using autograd package
    return G

F Greedy Order Meta-Summarization Running Accuracy Plots

In this section, we provide additional plots akin to that which was shown in Figure 3 but for other datasets, models, and summary sizes. We see in these plots that in all cases, the greedy meta-summarization order of the summaries finds the best summary quite early, significantly earlier than the 95% confidence. In other words, if we were to take the summaries in a random order (which is equivalent to random hyperparameter search), we would find the best (and worst) performing summary much later than when we search hyperparameters in the greedy order.

The meta-summarization order results for CIFAR-10 are shown for various IPCs in Figure 10, Figure 11, and Figure 13 and histograms corresponding to these results are shown in Figure **??**, Figure 12, and Figure 14.

The meta-summarization order results for CIFAR-100 are shown for various IPCs in Figure 15 and Figure 17 and histograms corresponding to these results are shown in Figure 16 and Figure 18.

The meta-summarization order results for TinyImagenet are shown for various IPCs in Figure 19 and Figure 21 and histograms corresponding to these results are shown in Figure 20 and Figure 22.

The meta-summarization order results for Imagenet (64×64) are shown for various IPCs in Figure 23, Figure 25, Figure 27, Figure 29 and histograms corresponding to these results are shown in Figure 24, Figure 26, Figure 28, and Figure 30

We note that the plots for the other cases (e.g., other IPCs, Selection Percentages, Benchmarks etc.) are quite similar and thus are not shown here since we have already provided a proliferation of such plots and adding more does not make the point any clearer. The bottom line is that the greedy order for

meta-summarization, and testing the summaries in this order, yields a process that finds the best performing summary much more quickly than random search.

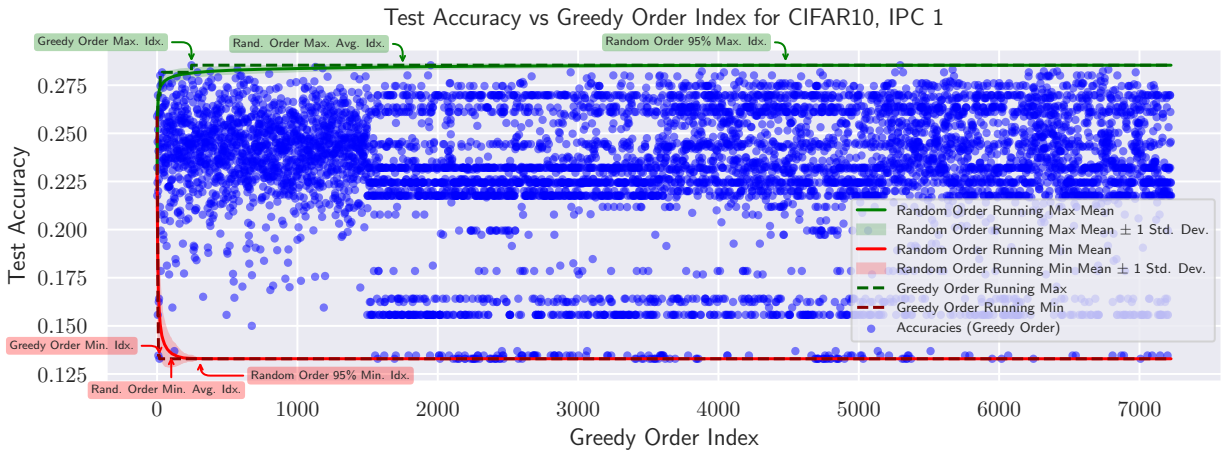
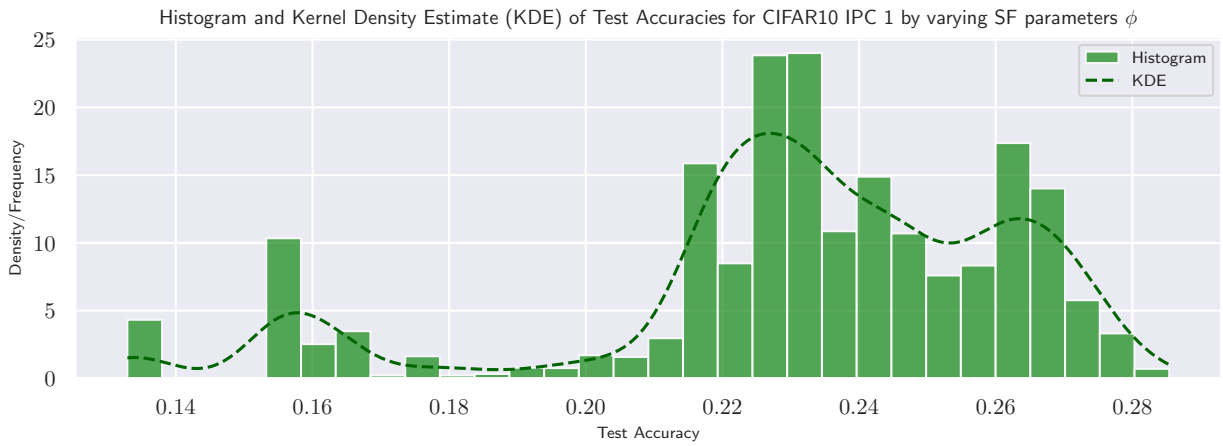


Figure 10

F.1 CIFAR-10



G Tabular Results

G.1 DC-Bench

Table 1 shows that Fine-Tuned Submodular functions outperform all reported coreset baselines in DC-Bench [17] across all datasets and all IPC (images per class). Visualization of these results can be seen in Figure 5 in the main text.

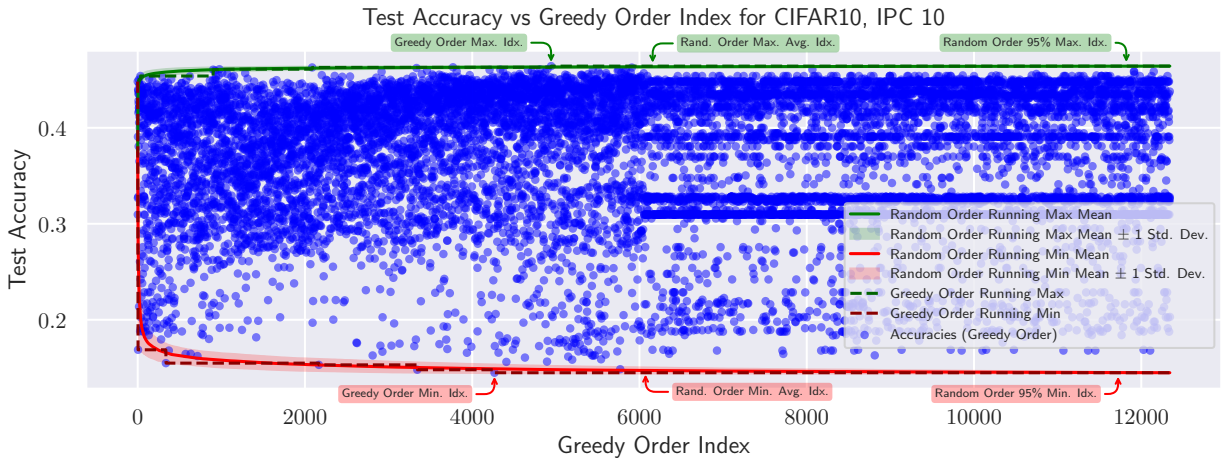


Figure 11

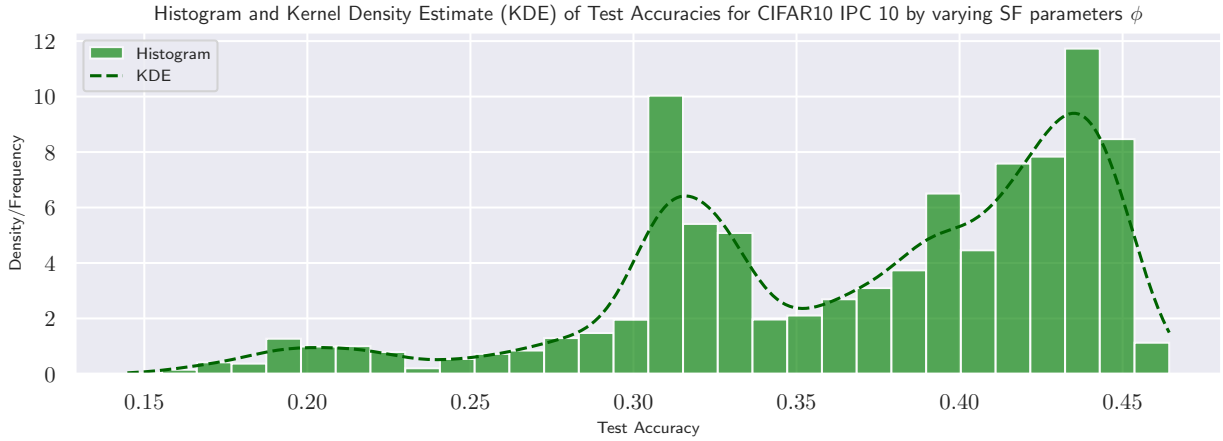


Figure 12

G.2 DeepCore

Table 2 shows that Fine-Tuned Submodular functions outperform all reported baselines in DeepCore [29] across all datasets and all IPC (images per class). Visualization of these results can be seen in Figure 6a in the main text.

G.3 Vision Transformer Training on Subsets

Table 4 compares Fine-tuned submodularity with NNKMeans and Random Subsets. Table 4 is the tabular representation of the data in Figure 7. Table 5 is the tabular representation of Figure 8. We show that in addition to outperforming NNKMeans, Fine-tuned submodularity is $> 2\times$ faster than NNKMeans for subset selection (*i.e,* to generate a summary).

NNKMeans vs. Fine-Tuned Submodularity Test-set accuracy. We conjecture that NNKMeans performs poorly due to bad initialization of K-centers before starting the K-Means algorithm. We use FAISS [40] to perform NNKMeans. We use FAISS since it is highly optimized and hence much better for running the

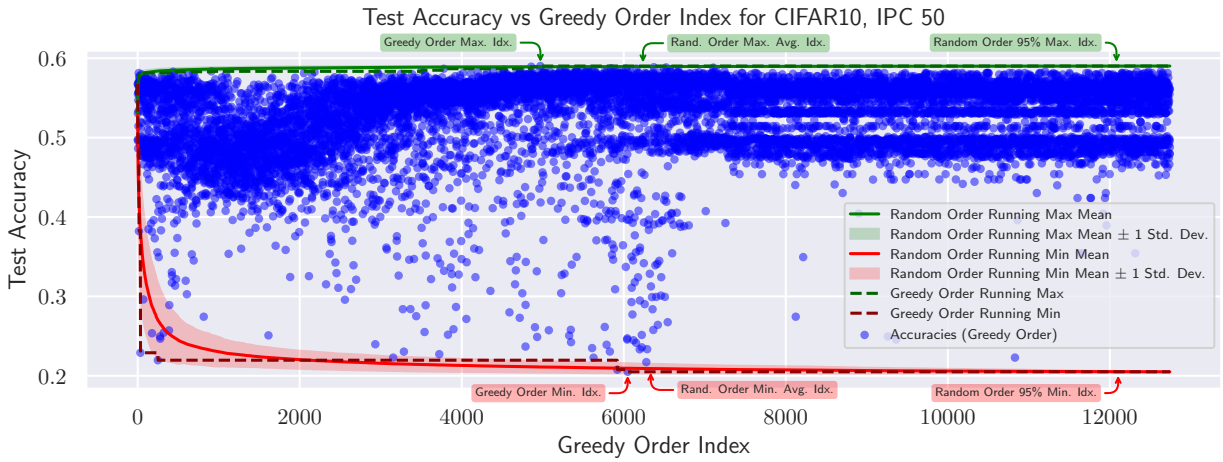


Figure 13

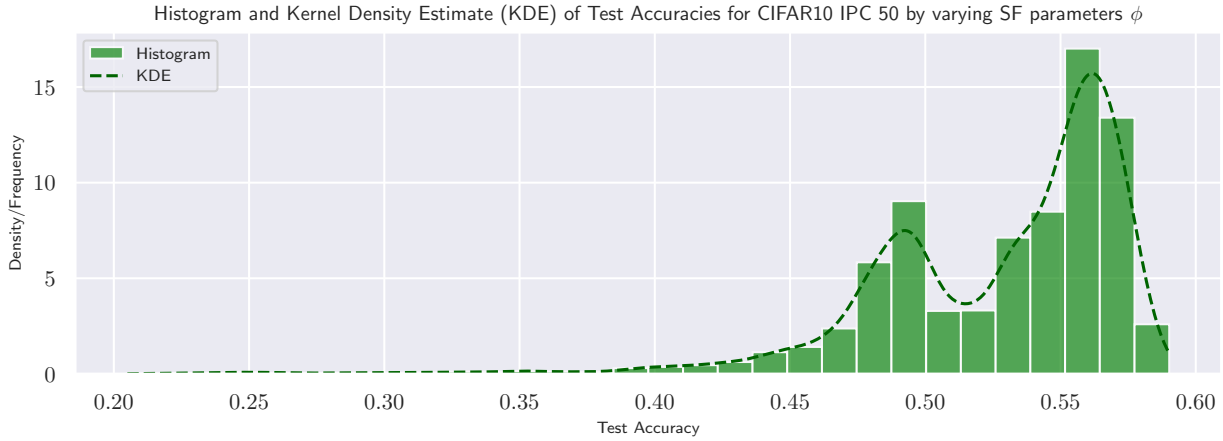


Figure 14

K-Means algorithm at scale. FAISS utilizes GPU cores to run the K-Means algorithm. The default FAISS initialization of the K-Means is uniformly at random (which can be arbitrarily bad), which might be the root cause of NNKMeans failing catastrophically. Given that NNKMeans is extremely expensive to run on high dimensional features and large scale datasets (as seen from Figure 5 and Table 5), we run it for only one seed initialization of initial K-Means. Additionally, for some classes in ImageNet we observe that NNKMeans selects redundant data-points even after fully converging, demonstrating a potential problem with NNKMeans. This is further exacerbated by the fact that downstream training of DaViT on ImageNet subsets is computationally expensive, denying us the opportunity to train many different subsets generated by NNKMeans to obtain the best one. We leave benchmarking with better NNKMeans initialization such as K-Means++ [2] to future work.

NNKMeans vs. Fine-Tuned Submodularity Timing Analysis. Fine-tuned submodularity is $> 2\times$ faster than NNKMeans since NNKMeans needs to recompute the gradient based features for each class for every selection ratio each time we run NNKMeans since the features can't be stored on disk due to space limitations (total size of design matrix is $\approx$ 4TB due to high dimensional gradient features). This is in

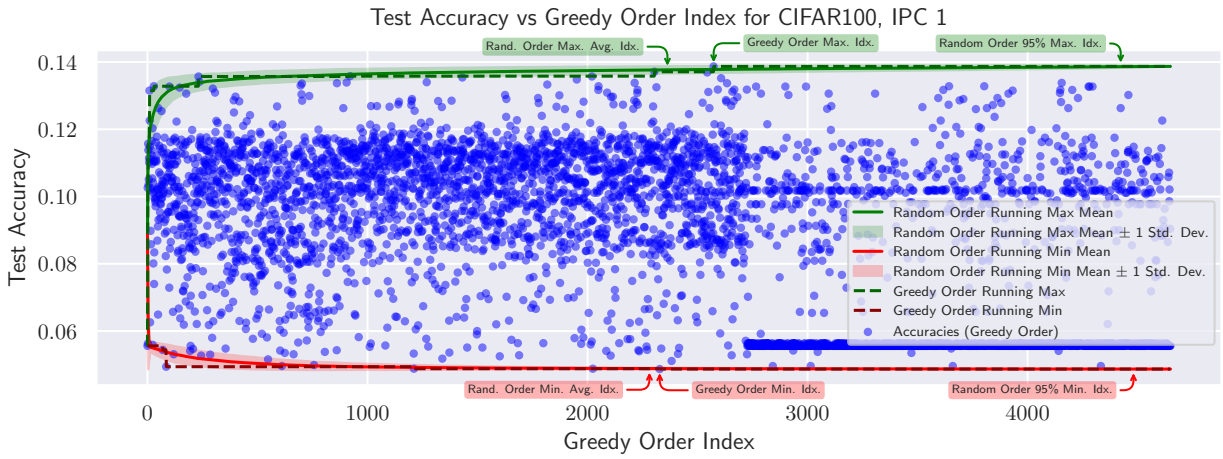


Figure 15

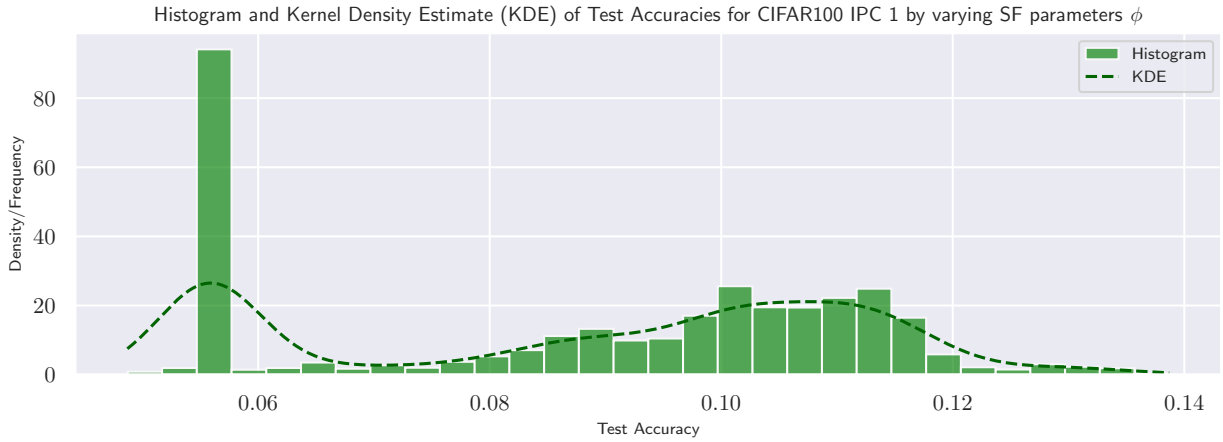


Figure 16

contrast to Fine-Tuned submodularity, which requries only a one-time computation of a similarity martrix using the blocking method (explained in E) which is $<$ 20GB. Once, we construct this similarity matrix, we can generate subsets of any size as we wish using the greedy algorith. Hence, while reporting the time to construct the similarity matrix as in Table 5, we report amortized time across all selection ratios for a fair comparison. Furthermore, we also note that generating summaries using fine-tuned submodularity consumes only CPU compute, whereas NNKMeans using FAISS consumes GPU cycles, making it financially expensive from the perspective of energy consumption. Despite NNKMeans having access to GPU parallelism, Fine-Tuned submodular is faster.

H Best Performing Hyperparameter Settings

We show the best performing hyperparameter settings for different datasets, selection percentages, architectures and benchmarks in Table 6, Table 7, Table 8 and Table 9. For cells in which a certain hyperparameter is not applicable, the cell is marked with a dash (-).

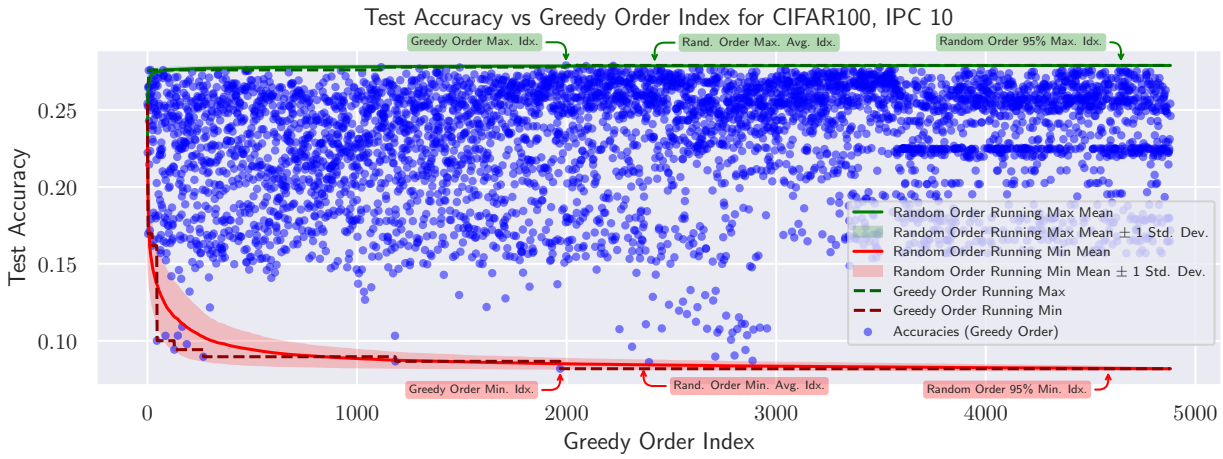


Figure 17

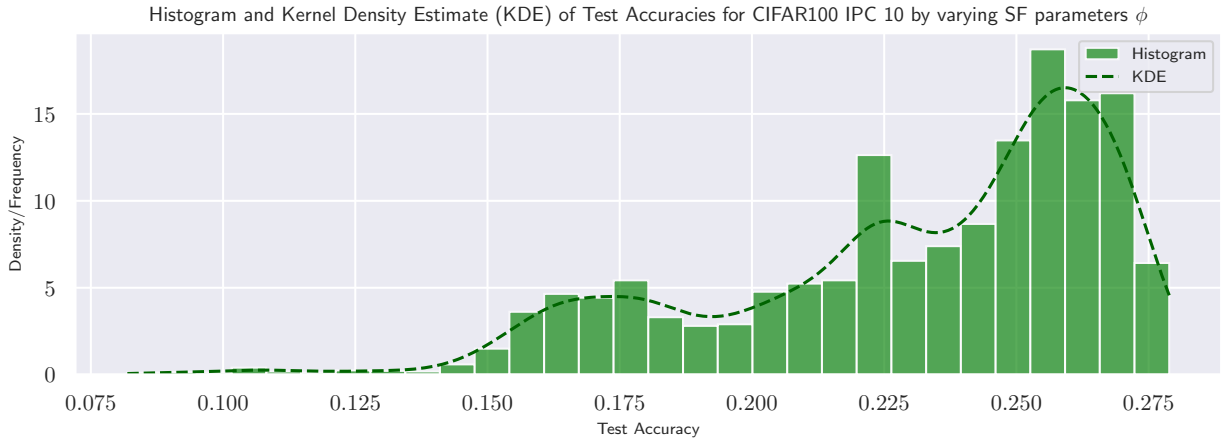


Figure 18

I Details on Computing Infrastructure

Submodular Summary Generation. All summaries using submodular greedy maximization algorithm were generated on Intel Xeon CPUs. These CPUs were available as a part of local compute as well from a cloud provider. We note submodular summary generation requires CPU only cloud instances.

Model Training on Summaries. Model training was performed on different GPUs based on computational requirements of the datasets and models. The GPUs used specifically were: NVIDIA A100, NVIDIA P100, NVIDIA V100, NVIDIA GeForce GTX 1080 Ti. These GPUs were available as a part of local compute as well from a cloud provider.

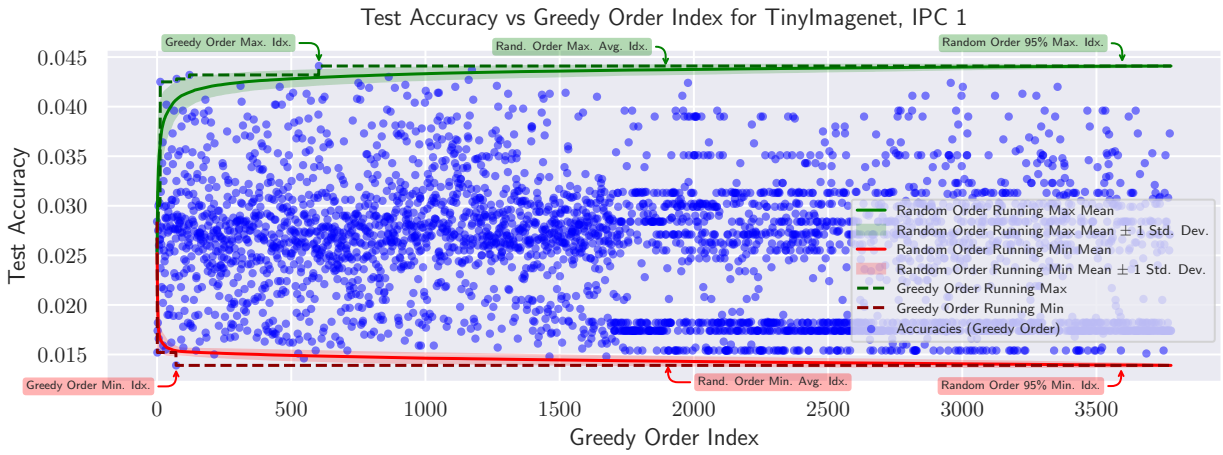


Figure 19

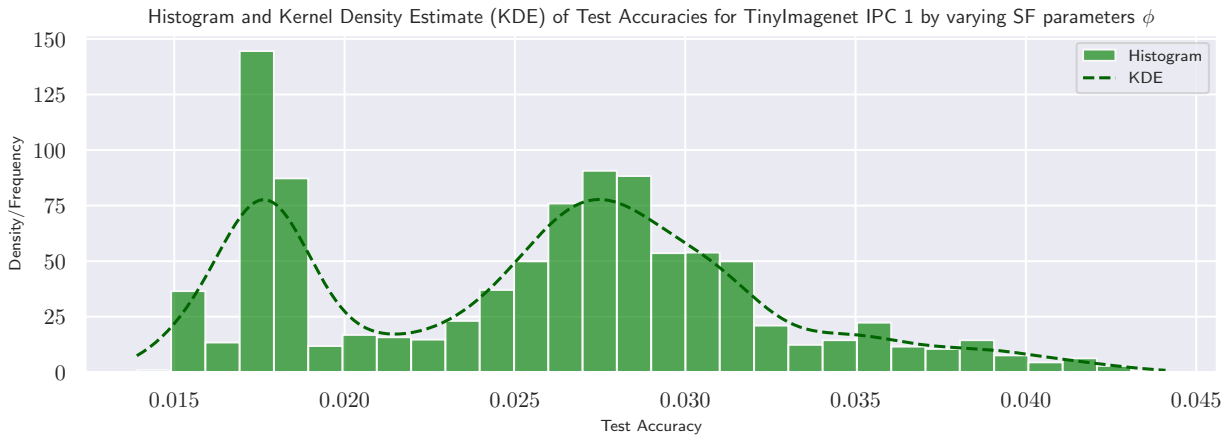


Figure 20

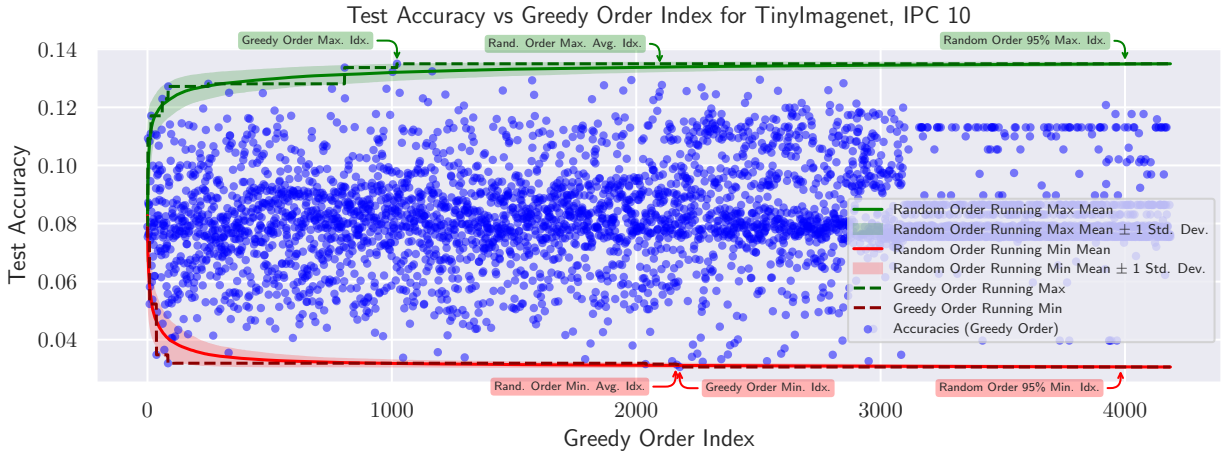


Figure 21

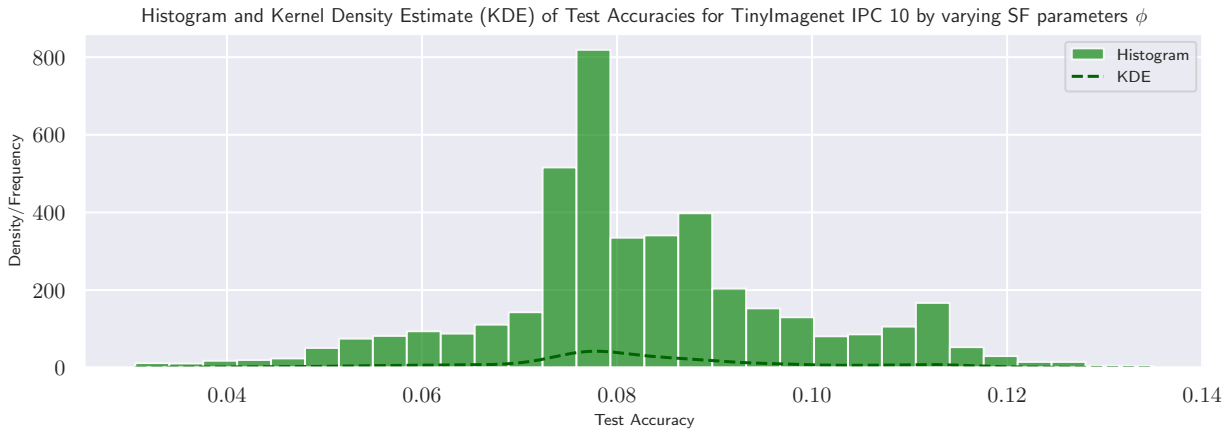


Figure 22

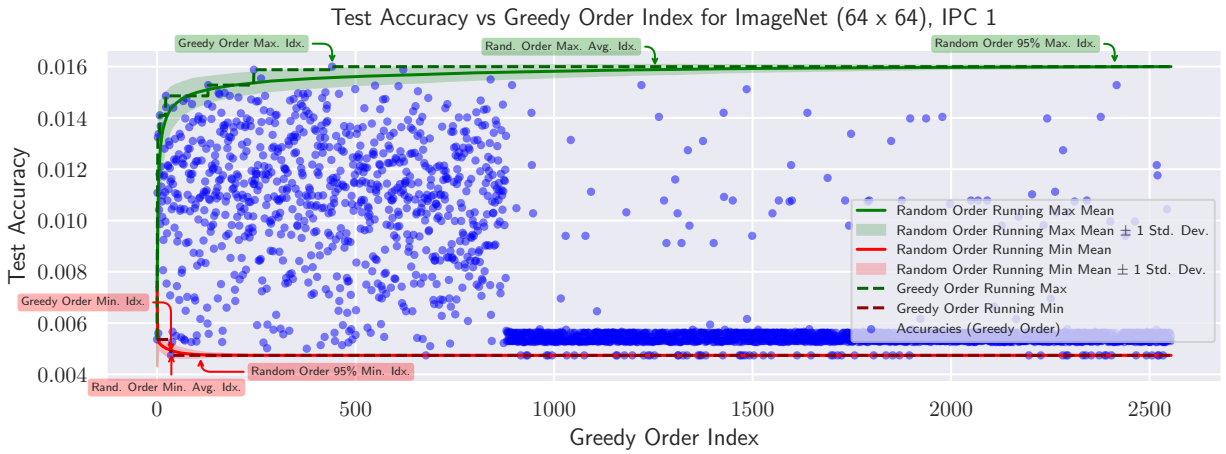


Figure 23

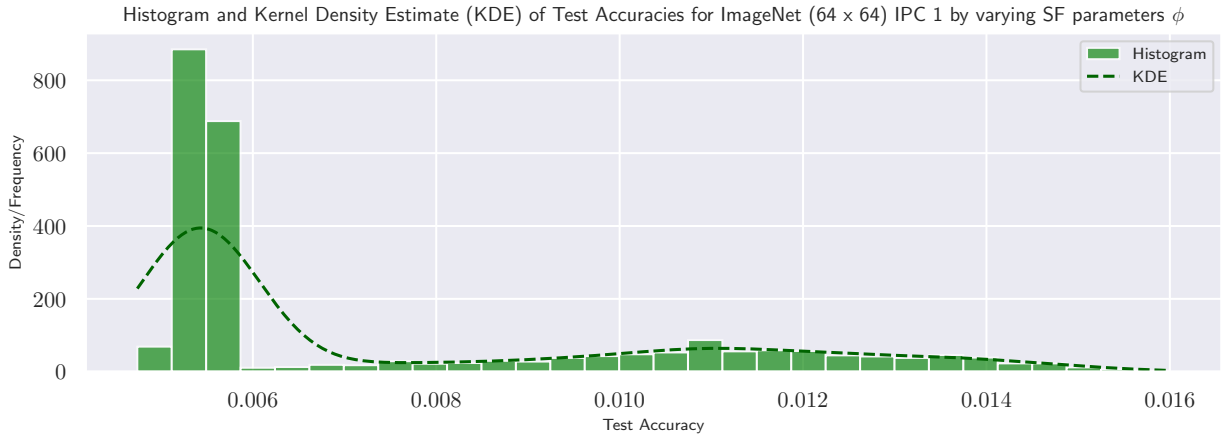


Figure 24

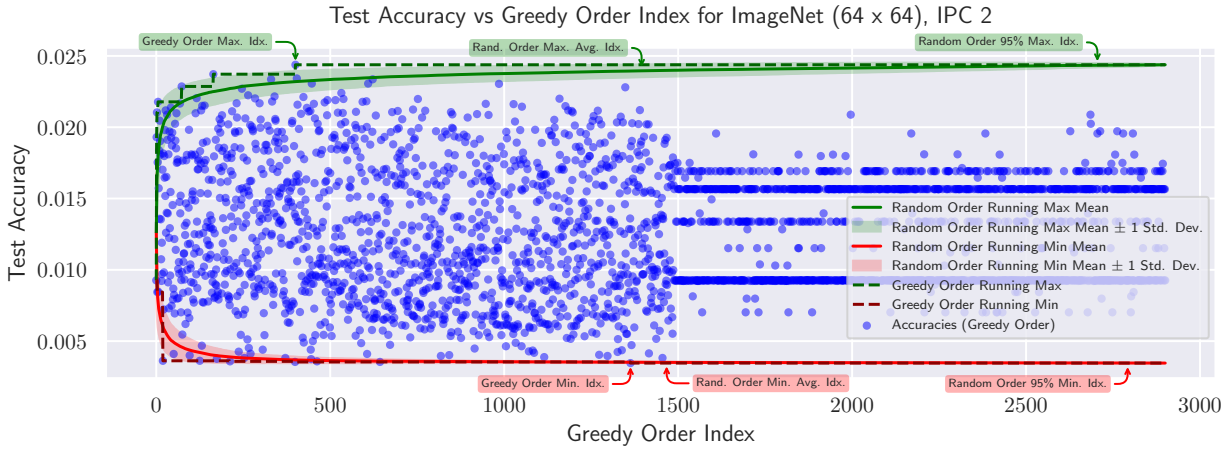


Figure 25

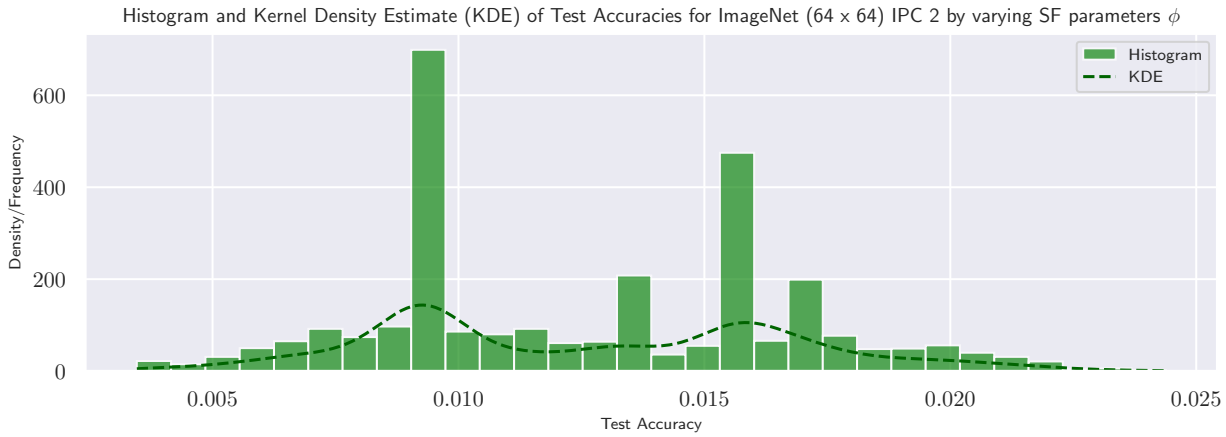


Figure 26

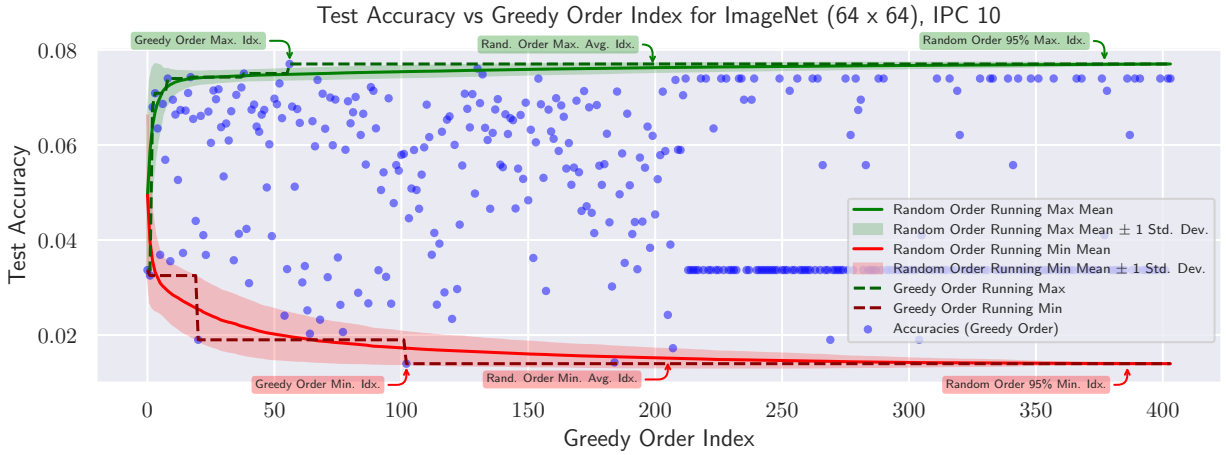


Figure 27

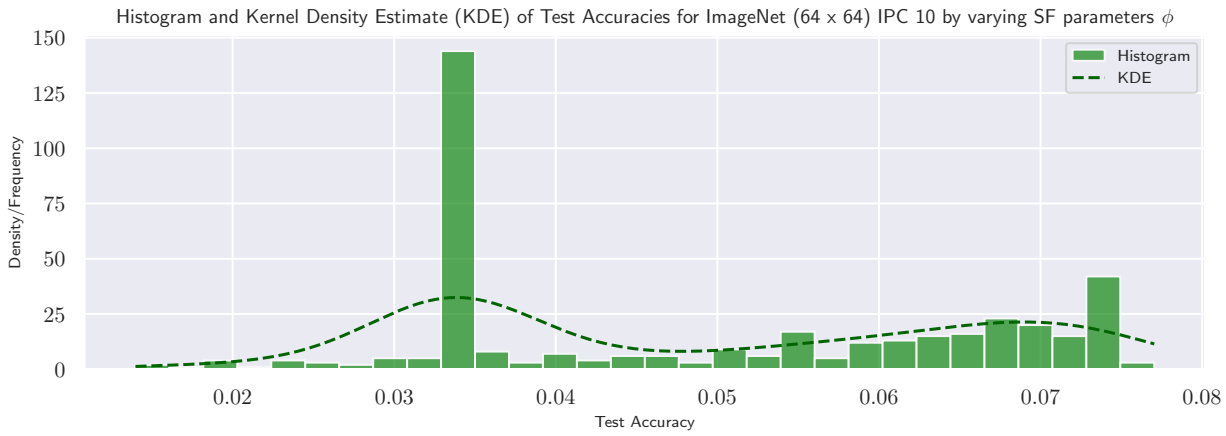


Figure 28

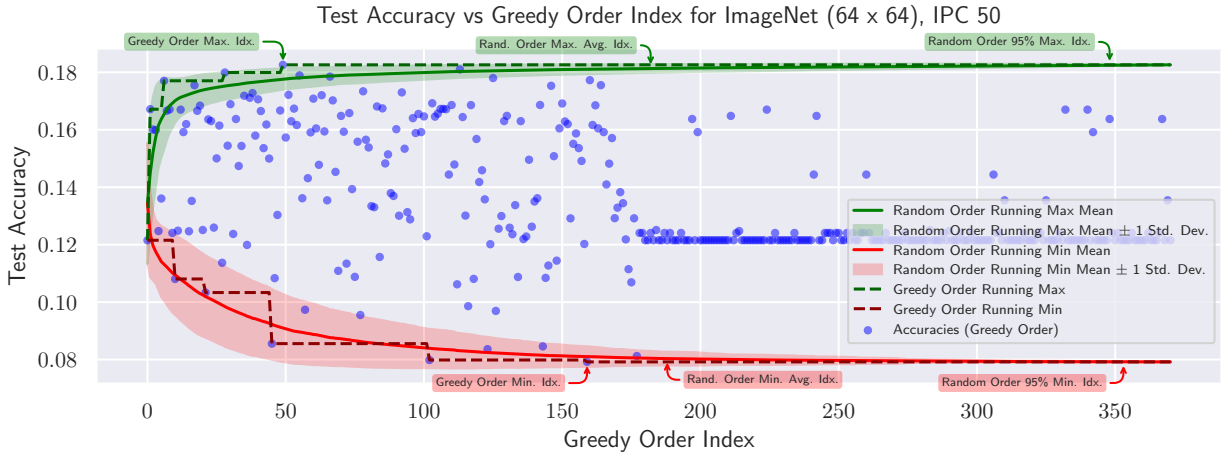


Figure 29

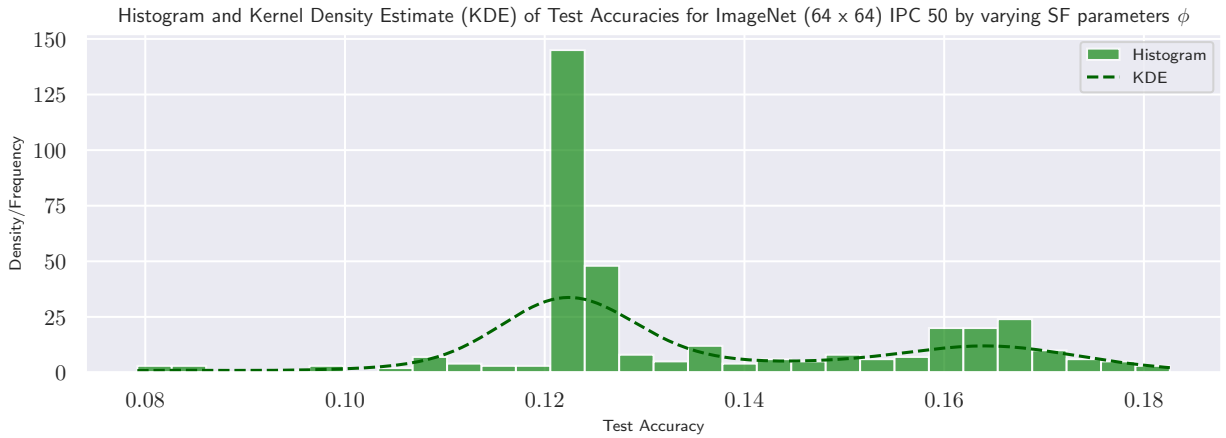


Figure 30

Dataset	IPC	Random	Herding	Forgetting	NNKMeans	*Tuned* Submodular (Ours)	Full Data
CIFAR-10	1	15.4 ± 0.28	21.5 ± 1.2	13.5 ± 1.2	25.16 ± 0.45	$\mathbf{28.26 \pm 0.74}$	86.0 ± 0.1
	10	31.00 ± 0.48	31.6 ± 0.7	23.3 ± 1.0	41.49 ± 0.73	$\mathbf{44.97 \pm 0.30}$	
	50	50.55 ± 0.32	40.4 ± 0.6	23.3 ± 1.1	56.00 ± 0.29	$\mathbf{57.50 \pm 0.26}$	
CIFAR-100	1	4.45 ± 0.15	8.4 ± 0.3	4.5 ± 0.2	10.89 ± 0.17	$\mathbf{13.35 \pm 0.09}$	56.7 ± 0.2
	10	18.64 ± 0.25	17.3 ± 0.3	15.1 ± 0.3	25.04 ± 0.30	$\mathbf{27.29 \pm 0.10}$	
	50	34.66 ± 0.41	33.7 ± 0.5	30.5 ± 0.3	38.64 ± 0.43	$\mathbf{41.61 \pm 0.27}$	
TinyImagenet	1	1.65 ± 0.11	2.8 ± 0.2	1.6 ± 0.1	3.03 ± 0.12	$\mathbf{4.2 \pm 0.13}$	39.83 ± 0.0
	10	6.88 ± 0.25	6.3 ± 0.2	5.1 ± 0.2	11.38 ± 0.26	$\mathbf{13.28 \pm 0.16}$	
	50	18.62 ± 0.22	16.7 ± 0.3	15.0 ± 0.3	22.02 ± 0.40	$\mathbf{25.04 \pm 0.32}$	
Imagenet (64×64 resolution)	1	0.5 ± 0.1	-	-	1.0 ± 0.0	$\mathbf{1.57 \pm 0.04}$	33.8 ± 0.3
	2	0.9 ± 0.1	-	-	1.8 ± 0.0	$\mathbf{2.46 \pm 0.04}$	
	10	3.6 ± 0.1	-	-	6.1 ± 0.0	$\mathbf{7.59 \pm 0.08}$	
	50	12.3 ± 2.3	-	-	13.62 ± 0.0	$\mathbf{18.36 \pm 0.18}$	

Table 1: Tuned Submodularity vs. Subset Selection Baselines in DC-Bench [17]. Tuned Submodularity outperforms all subset selection baselines across all datasets and subset sizes. Herding [97] and Forgetting [89] are computationally expensive for ImageNet and hence we do not benchmark against them for ImageNet.

Fraction	0.1%	0.5%	1%	5%	10%	20%
Random	21.0 ± 0.3	30.8 ± 0.6	36.7 ± 1.7	64.5 ± 1.1	75.7 ± 2.0	$\underline{87.1 \pm 0.5}$
CD	15.8	20.5	23.6	38.1	58.8	81.3
Herding	20.2	27.3	34.8	51.0	63.5	74.1
K-Center Greedy	18.5	26.8	31.1	51.4	75.8	87.0
Least Confidence	14.2	17.2	19.8	36.2	57.6	81.9
Entropy	14.6	17.5	21.1	35.3	57.6	81.9
Margin	17.2	21.7	28.2	43.4	59.9	81.7
Forgetting	21.4	29.8	35.2	52.1	67.0	86.6
GraNd	17.7	24.0	26.7	39.8	52.7	78.2
Cal	22.7	33.1	37.8	60.0	71.8	80.9
DeepFool	17.6	22.4	27.6	42.6	60.8	83.0
CRAIG	22.5	27.0	31.7	45.2	60.2	79.6
GradMatch	17.4	25.6	30.8	47.2	61.5	79.9
Glister	19.5	27.5	32.9	50.7	66.3	84.8
FL	22.3	31.6	38.9	60.8	74.7	85.6
GC	$\underline{24.3}$	$\underline{34.9}$	$\underline{42.8}$	$\underline{65.7}$	$\underline{76.6}$	84.0
Tuned Submodular (Ours)	**28.61**	**38.22**	**43.1**	**66.6**	**79.35**	**88.15**

Table 2: Tuned Submodular vs. baselines reported in the DeepCore [29] paper on CIFAR-10. **Bold** indicates the best performing method. $\underline{\text{Underline}}$ indicates the second best. Tuned Submodular outperforms all baselines across all selection ratios.

Fraction	**10%**	**20%**	**30%**	**40%**	**50%**	**60%**
Random	32.0 ± 0.9	53.6 ± 0.6	63.6 ± 0.5	67.2 ± 0.5	71.0 ± 0.3	73.1 ± 0.4
K-Center Greedy	33.9	56.2	64.5	69.8	72.1	$\underline{74.3}$
Margin	18.7	38.2	58.1	65.1	70.1	73.3
Forgetting	35.4	54.7	64.6	68.6	$\underline{71.5}$	73.7
GraNd	30.8	49.4	62.8	68.1	70.5	72.5
Glister	36.4	55.5	63.9	69.1	71.2	73.5
SSP-Easy	32.8	50.0	62.5	67.9	70.2	73.4
SSP-Hard	29.7	53.3	63.2	67.8	71.3	72.9
SP-Easy	33.6	53.0	63.0	67.4	70.5	73.3
SP-Hard	31.2	53.6	63.0	68.0	71.1	73.0
GraphCut	36.3	56.0	$\underline{65.5}$	$\underline{69.5}$	71.1	73.8
Un-Tuned Submodular	$\underline{37.47}$	$\underline{57.16}$	64.38	68.53	71.49	73.48
Tuned Submodular	**42.92**	**58.53**	**66.33**	**69.61**	**72.94**	**74.73**

Table 3: Tuned Submodular vs. baselines reported in the DeepCore [29] paper on CIFAR-100. **Bold** indicates the best performing method. $\underline{\text{Underline}}$ indicates the second best. Tuned Submodular outperforms all baselines across all selection ratios.

Fraction (%)	0.1%	0.5%	1%	5%	10%
Random	0.5 ± 0.34	3.98 ± 0.735	11.18 ± 0.21	39.7 ± 0.21	59.50 ± 0.54
NNKMeans	0.696	4.676	9.762	37.708	53.74
Tuned Submodular (Ours)	**0.922**	**5.6**	**12.98**	**42.0**	**60.426**

Table 4: Tuned Submodular vs. Random Subset Selection for training a Vision Transformer Model [20], on full resolution ImageNet. Tuned Submodular outperforms random subset selection across all selection ratios.

Selection Method	**Timing Components**	**0.1%**	**0.5%**	**1%**	**5%**	**10%**
		Time (s)	**Time (s)**	**Time (s)**	**Time (s)**	**Time (s)**
NNKMeans	Number of KMeans iterations to converge	1	15	25	20	20
	Gradient/Feature Computation Time	22322	22322	22322	22322	22322
	FAISS KMeans Convergence Time	4020	33152	55935	38610	40112
	Nearest Neighbor Computation Time	1243	1559	1607	1925	1741
	Store NNKMeans index time	0.8	5.1	0.949	2.14	2.72
	Total NNKMeans Time	27585.8	57038.1	79864.949	62859.14	64177.72
Submodular	Blocking Based Similarity Matrix Computation Time	45250	45250	45250	45250	45250
	Blocking Based Similarity Matrix Computation Time (Amortized)	9050	9050	9050	9050	9050
	Sparse Matrix Construction Time	446	446	446	446	446
	Sparse Matrix Construction Time (Amortized)	89.2	89.2	89.2	89.2	89.2
	Greedy Max Wall Clock Time Actual	4415.07	12889.76	14442.8	17077.57	20396.65
	Meta Summarization Time	4.614	125	284	1559	1676
	Total Submodular Time (incl. sweep)	13558.884	22153.96	23866	27775.77	31211.85

Table 5: Comparison of Timing of Tuned Submodularity vs. NNKMeans for training DaViT [20] on ImageNet.

Setting	**Feat Types**	**FL Weights**	**Model (Epoch)**	**KNN**	**Gravity Value**	**Gravity Fulcrum**	**Similarity Metric**	**SimEuclid Exp.**	**RBF KW**	**RBF Divide By**
CIFAR-10, IPC 1	Gradient	1	ConvNet (1)	1000	-	-	RBF	-	2	None
CIFAR-10, IPC 10	Gradient, Activation, Activation	0.4, 0.4, 0.2	ConvNet (1), CLIP, ConvNet (1)	5000	−10	1	Cosine, SimEuclid, SimEuclid	5	-	-
CIFAR-10, IPC 50	Gradient, Activation, Activation	0.2, 0.2, 0.6	ConvNet (1), CLIP, ConvNet (1)	2000	−10	50	Cosine, SimEuclid, SimEuclid	0.1	-	-
CIFAR-10, 0.1 %	Gradient	1	ResNet18 (1)	50	-	-	RBF	-	2	None
CIFAR-10, 0.5 %	Gradient	1	ResNet18 (1)	4000	-	-	RBF	-	1	None
CIFAR-10, 1 %	Gradient	1	ResNet18 (1)	4000	-	-	RBF	-	0.01	None
CIFAR-10, 5 %	Activation	1	CLIP	4000	-	-	SimEuclid	0.001	-	-
CIFAR-10, 10 %	Gradient	1	ResNet18 (1)	4000	-	-	RBF	-	2	Max
CIFAR-10, 20 %	Gradient	1	ResNet18 (1)	100	-	-	RBF	-	5	None

Table 6: Hyperparameter settings achieving best test-set accuracy for CIFAR-10, DC-Bench and DeepCore.

Setting	**Feat Types**	**FL Weights**	**Model (Epoch)**	**KNN**	**Gravity Val.**	**Gravity Fulcrum**	**Similarity Metric**	**SimEuclid Exp.**	**RBF KW**	**RBF Divide By**
CIFAR-100, IPC 1	Gradient	1	ConvNet (1)	300	-	-	RBF	-	2	Mean
CIFAR-100, IPC 10	Gradient, Activation, Activation	0.4, 0.2, 0.4	ConvNet (1), CLIP, ConvNet (1)	500	−50	1	Cosine, SimEuclid, SimEuclid	0.0001	-	-
CIFAR-100, IPC 50	Gradient, Activation, Activation	0.4, 0.2, 0.4	ConvNet (1), CLIP, ConvNet (1)	500	−50	75	Cosine, SimEuclid, SimEuclid	1	-	-
CIFAR-100, 10 %	Gradient	1	ResNet18 (1)	5	-	-	RBF	-	10	PNorm
CIFAR-100, 20 %	Gradient	1	ResNet18 (1)	50	-	-	RBF	-	10	Max
CIFAR-100, 30 %	Gradient	1	ResNet18 (1)	50	-	-	RBF	-	10	Max
CIFAR-100, 40 %	Gradient	1	ResNet18 (1)	20	-	-	RBF	-	0.0001	PNorm
CIFAR-100, 50 %	Gradient	1	ResNet18 (1)	20	-	-	RBF	-	5	Mean
CIFAR-100, 60 %	Gradient	1	ResNet18 (1)	5	-	-	RBF	-	0.001	Sum

Table 7: Hyperparameter settings achieving best test-set accuracy for CIFAR-100, DC-Bench and DeepCore.

Setting	Feat Types	FL Weights	Model (Epoch)	KNN	Gravity Val.	Gravity Fulcrum	Similarity Metric	SimEuclid Exp.	RBF KW	RBF Divide By
TinyImageNet, IPC 1	Activation, Activation	$0.6, 0.4$	ConvNet (1), CLIP	500	-10	50	SimEuclid, SimEuclid	0.5	-	-
TinyImageNet, IPC 10	Activation, Activation	$0.6, 0.4$	ConvNet (1), CLIP	500	-10	50	SimEuclid, SimEuclid	1	-	-
TinyImageNet, IPC 50	Activation, Activation	$0.5, 0.5$	ConvNet (1), CLIP	10	-99	1	SimEuclid, SimEuclid	0.01	-	-

Table 8: Hyperparameter settings achieving best test-set accuracy for TinyImageNet.

Setting	Feat Types	FL Weights	Model (Epoch)	KNN	Gravity Val.	Gravity Fulcrum	Similarity Metric	SimEuclid Exp.	RBF KW	RBF Divide By
ImageNet (64 x 64), IPC 1	Activation, Activation	$0.9, 0.1$	ConvNet (1), CLIP	1300	-75	50	SimEuclid, SimEuclid	1	-	-
ImageNet (64 x 64), IPC 2	Activation, Activation	$1, 0$	ConvNet (1), CLIP	1000	-75	50	SimEuclid, SimEuclid	1	-	-
ImageNet (64 x 64), IPC 10	Activation, Activation	$0.9, 0.1$	ConvNet (1), CLIP	1100	-50	50	SimEuclid, SimEuclid	0.01	-	-
ImageNet (64 x 64), IPC 50	Activation, Activation	$0.2, 0.8$	ConvNet (1), CLIP	1100	-75	75	SimEuclid, SimEuclid	0.5	-	-
ImageNet, 0.5 %	Gradient	1	DaViT (2)	100	-50	50	RBF	-	0.5	PNorm
ImageNet, 1 %	Gradient	1	DaViT (2)	100	-99	50	RBF	-	10	Sum
ImageNet, 5 %	Gradient	1	DaViT (2)	50	-99	50	RBF	-	0.001	PNorm
ImageNet, 10 %	Gradient	1	DaViT (2)	600	-99	75	RBF	-	10	Max

Table 9: Hyperparameter settings achieving best test-set accuracy for ImageNet, DC-Bench and DaViT training.