Data-Efficient Augmentation for Training Neural Networks

Tian Yu Liu
Department of Computer Science
University of California, Los Angeles
tianyu@cs.ucla.edu

Baharan Mirzasoleiman
Department of Computer Science
University of California, Los Angeles
baharan@cs.ucla.edu

Abstract

Data augmentation is essential to achieve state-of-the-art performance in many deep learning applications. However, the most effective augmentation techniques become computationally prohibitive for even medium-sized datasets. To address this, we propose a rigorous technique to select subsets of data points that when augmented, closely capture the training dynamics of full data augmentation. We first show that data augmentation, modeled as additive perturbations, improves learning and generalization by relatively enlarging and perturbing the smaller singular values of the network Jacobian, while preserving its prominent directions. This prevents overfitting and enhances learning the harder to learn information. Then, we propose a framework to iteratively extract small subsets of training data that when augmented, closely capture the alignment of the fully augmented Jacobian with labels/residuals. We prove that stochastic gradient descent applied to the augmented subsets found by our approach has similar training dynamics to that of fully augmented data. Our experiments demonstrate that our method achieves 6.3x speedup on CIFAR10 and 2.2x speedup on SVHN, and outperforms the baselines by up to 10% across various subset sizes. Similarly, on TinyImageNet and ImageNet, our method beats the baselines by up to 8%, while achieving up to 3.3x speedup across various subset sizes. Finally, training on and augmenting 50% subsets using our method on a version of CIFAR10 corrupted with label noise even outperforms using the full dataset. [1]

1 Introduction

Standard (weak) data augmentation transforms the training examples with e.g. rotations or crops for images, and trains on the transformed examples *in place of* the original training data. While weak augmentation is effective and computationally inexpensive, *strong* data augmentation (in addition to weak augmentation) is a key component in achieving nearly all state-of-the-art results in deep learning applications [35]. However, strong data augmentation techniques often increase the training time by orders of magnitude. First, they often have a very expensive pipeline to find or generate more complex transformations that best improves generalization [5, 15, 22, 40]. Second, *appending transformed examples* to the training data is often much more effective than training on the (strongly or weakly) transformed examples *in-place* of the original data. For example, appending *one* transformed example to the training data is often much more effective than training on *two* transformed examples *in place* of every original training data, while both strategies have the same computational cost (*c.f.* Appendix D.6). Hence, to obtain the state-of-the-art performance, multiple augmented examples are added for every single data point and to each training iteration [14, 40]. In this case, even if producing transformations are cheap, such methods increases the size of the training data by orders of magnitude.

[1] Our code can be found at https://github.com/tianyu139/data-efficient-augmentation

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

As a result, state-of-the-art data augmentation techniques become computationally prohibitive for even medium-sized real-world problems. For example, the state-of-the-art augmentation of [40], which appends every example with its highest-loss transformations, increases the training time of ResNet20 on CIFAR10 by 13x on an Nvidia A40 GPU (*c.f.* Sec. 6).

To make state-of-the-art data augmentation more efficient and scalable, an effective approach is to carefully select a small subset of the training data such that augmenting only the subset provides similar training dynamics to that of full data augmentation. If such a subset can be quickly found, it would directly lead to a significant reduction in storage and training costs. First, while standard in-place augmentation can be applied to the entire data, the strong and *expensive transformations* can be only produced for the examples in the subset. Besides, only the transformed elements of the subset can be *appended* to the training data. Finally, when the data is larger than the training budget, one can train on random subsets (with standard in-place augmentation) and augment coresets (by strong augmentation and/or appending transformations) to achieve a superior performance.

Despite the efficiency and scalability that it can provide, this direction has remained largely unexplored. Existing studies are limited to fully training a network and subsampling data points based on their loss or influence, for augmentation in subsequent training runs [20]. However, this method is prohibitive for large datasets, provides a marginal improvement over augmenting random subsets, and does not provide any theoretical guarantee for the performance of the network trained on the augmented subsets. Besides, when the data contains mislabeled examples, augmentation methods that select examples with maximum loss, and append their transformed versions to the data, degrade the performance by selecting and appending several noisy labels.

A major challenge in finding the most effective data points for augmentation is to theoretically understand how data augmentation affects the optimization and generalization of neural networks. Existing theoretical results are mainly limited to simple linear classifiers and analyze data augmentation as enlarging the span of the training data [40], providing a regularization effect [4, 9, 37, 40], enlarging the margin of a linear classifier [32], or having a variance reduction effect [6]. However, such tools do not provide insights on the effect of data augmentation on training deep neural networks.

Here, we study the effect of label invariant data augmentation on training dynamics of overparameterized neural networks. Theoretically, we model data augmentation by *bounded additive perturbations* [32], and analyze its effect on neural network Jacobian matrix containing all its first-order partial derivatives [1]. We show that label invariant additive data augmentation *proportionally* enlarges but more importantly *perturbs* the singular values of the Jacobian, particularly the smaller ones, while maintaining prominent directions of the Jacobian. In doing so, data augmentation *regularizes* training by adding bounded but varying perturbations to the gradients. In addition, it *speeds up* learning harder to learn information. Thus, it prevents overfitting and improves generalization. Empirically, we show that the same effect can be observed for various strong augmentations, e.g., AutoAugment [7], CutOut [10], and AugMix [14].[2]

Next, we develop a rigorous method to iteratively find small weighted subsets (coresets) that when augmented, closely capture the alignment between the Jacobian of the full augmented data with the label/residual vector. We show that the most effective subsets for data augmentation are the set of examples that when data is mapped to the gradient space, have the most centrally located gradients. This problem can be formulated as maximizing a submodular function. The subsets can be efficiently extracted using a fast greedy algorithm which operates on small dimensional gradient proxies, with only a small additional cost. We prove that augmenting the coresets guarantees similar training dynamics to that of full data augmentation. We also show that augmenting our coresets achieve a superior accuracy in presence of noisy labeled examples.

We demonstrate the effectiveness of our approach applied to CIFAR10 (ResNet20, WideResNet-28-10), CIFAR10-IB (ResNet32), SVHN (ResNet32), noisy-CIFAR10 (ResNet20), Caltech256 (ResNet18, ResNet50), TinyImageNet (ResNet50), and ImageNet (ResNet50) compared to random and max-loss baselines [20]. We show the effectiveness of our approach (in presence of standard augmentation) in the following cases:

- **When producing augmentations is expensive and/or they are appended to the training data:**

[2] We note that our results are in line with that of [34], that in parallel to our work, analyzed the effect of linear transformations on a two-layer convolutional network, and showed that it can make the hard to learn features more likely to be captured during training.

We show that for the state-of-the-art augmentation method of [40] applied to CIFAR10/ResNet20 it is 3.43x faster to train on the whole dataset and only augment our coresets of size 30%, compared to training and augmenting the whole dataset. At the same time, we achieve 75% of the accuracy improvement of training on and augmenting the full data with the method of [40], outperforming both max-loss and random baselines by up to 10%.

- **When data is larger than the training budget:** We show that we can achieve 71.99% test accuracy on ResNet50/ImageNet when training on and augmenting only 30% subsets for 90 epochs. Compared to AutoAugment [7], despite using only 30% subsets, we achieve 92.8% of the original reported accuracy while boasting 5x speedup in the training time. Similarly, on Caltech256/ResNet18, training on and augmenting 10% coresets with AutoAugment yields 65.4% accuracy, improving over random 10% subsets by 5.8% and over only weak augmentation by 17.4%.
- **When data contains mislabeled examples:** We show that training on and strongly augmenting 50% subsets using our method on CIFAR10 with 50% noisy labels achieves 76.20% test accuracy. Notably, this yields a superior performance to training on and strongly augmenting the full data.

2 Additional Related Work

Strong data augmentation methods achieve state-of-the-art performance by finding the set of transformations for every example that best improves the performance. Methods like AutoAugment [7], RandAugment [8], and Faster RandAugment [8] search over a (possibly large) space of transformations to find sequences of transformations that best improve generalization [7, 8, 24, 40]. Other techniques involve a very expensive pipeline for generating the transformations. For example, some use Generative Adversarial Networks to directly learn new transformations [2, 24, 27, 33]. Strong augmentations like Smart Augmentation [22], Neural Style Transfer-based [15], and GAN-based augmentations [5] require an expensive forward pass through a deep network for input transformations. For example, [15] increases training time by 2.8x for training ResNet18 on Caltech256. Similarly, [40] generates multiple augmentations for each training example, and selects the ones with the highest loss.

Strong data augmentation methods either replace the original example by its transformed version, or append the generated transformations to the training data. Crucially, appending the training data with transformations is much more effective in improving the generalization performance. Hence, the most effective data augmentation methods such as that of [40] and AugMix [14] append the transformed examples to the training data. In Appendix D.6, we show that even for cheaper strong augmentation methods such as AutoAugment [7], while replacing the original training examples with transformations may decrease the performance, appending the augmentations significantly improves the performance. Appending the training data with augmentations, however, increase the training time by orders of magnitude. For example, AugMix [14] that outperforms AutoAugment increases the training time by at least 3x by appending extra augmented examples, and [40] increases training time by 13x due to appending and forwarding additional augmented examples through the model.

3 Problem Formulation

We begin by formally describing the problem of learning from augmented data. Consider a dataset $\mathcal{D}_{train} = (\boldsymbol{X}_{train}, \boldsymbol{y}_{train})$, where $\boldsymbol{X}_{train} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n) \in \mathbb{R}^{d \times n}$ is the set of n normalized data points $\boldsymbol{x}_i \in [0,1]^d$, from the index set V, and $\boldsymbol{y}_{train} = (y_1, \cdots, y_n) \in \{y \in \{\nu_1, \nu_2, \cdots, \nu_C\}\}$ with $\{\nu_j\}_{j=1}^C \in [0,1]$.

The additive perturbation model. Following [32] we model data augmentation as an arbitrary bounded additive perturbation $\boldsymbol{\epsilon}$, with $\|\boldsymbol{\epsilon}\| \leq \epsilon_0$. For a given ϵ_0 and the set of all possible transformations $\mathcal{A}$, we study the transformations selected from $\mathcal{S} \subseteq \mathcal{A}$ satisfying

$$\mathcal{S} = \{T_i \in \mathcal{A} \mid \|T_i(\boldsymbol{x}) - \boldsymbol{x}\| \leq \epsilon_0 \; \forall \boldsymbol{x} \in \boldsymbol{X}^{train}\}. \tag{1}$$

While the additive perturbation model cannot represent all augmentations, most real-world augmentations are bounded to preserve the regularities of natural images (e.g. AutoAugment [7] finds that a 6 degree rotation is optimal for CIFAR10). Thus, under local smoothness of images, additive perturbation can model bounded transformations such as small rotations, crops, shearing, and pixel-wise transformations like sharpening, blurring, color distortions, structured adversarial perturbation [24]. As such, we see the effects of additive augmentation on the singular spectrum holds even under

real-world augmentation settings (*c.f.* Fig. 3 in the Appendix). However, this model is indeed limited when applied to augmentations that cannot be reduced to perturbations, such as horizontal/vertical flips and large translations. We extend our theoretical analysis to augmentations modeled as arbitrary linear transforms (e.g. as mentioned, horizontal flips) in B.5.

The set of augmentations at iteration t generating r augmented examples per data point can be specified, with abuse of notation, as $\mathcal{D}^t_{aug} = \{\bigcup_{i=1}^{r} (T^t_i(\boldsymbol{X}_{train}), \boldsymbol{y}_{train})\}$, where $|\mathcal{D}^t_{aug}| = rn$ and $T^t_i(\boldsymbol{X}_{train})$ transforms all the training data points with the set of transformations $T^t_i \subset \mathcal{S}$ at iteration t. We denote $\boldsymbol{X}^t_{aug} = \{\bigcup_{i=1}^{r} T^t_i(\boldsymbol{X}_{train})\}$ and $\boldsymbol{y}^t_{aug} = \{\bigcup_{i=1}^{r} \boldsymbol{y}_{train}\}$.

Training on the augmented data. Let $f(\boldsymbol{W}, \boldsymbol{x})$ be an arbitrary neural network with m vectorized (trainable) parameters $\boldsymbol{W} \in \mathbb{R}^m$. We assume that the network is trained using (stochastic) gradient descent with learning rate η to minimize the squared loss $\mathcal{L}$ over the original and augmented training examples $\mathcal{D}^t = \{\mathcal{D}_{train} \cup \mathcal{D}^t_{aug}\}$ with associated index set V^t, at every iteration t. I.e.,

$$\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}) := \frac{1}{2} \sum_{i \in V^t} \mathcal{L}_i(\boldsymbol{W}^t, \boldsymbol{x}_i) := \frac{1}{2} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}^t} \|f(\boldsymbol{W}^t, \boldsymbol{x}_i) - y_i\|_2^2. \tag{2}$$

The gradient update at iteration t is given by

$$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \nabla \mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}), \quad \text{s.t.} \qquad \nabla \mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}) = \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})(f(\boldsymbol{W}^t, \boldsymbol{X}) - \boldsymbol{y}), \tag{3}$$

where $\boldsymbol{X}^t = \{\boldsymbol{X}_{train} \cup \boldsymbol{X}^t_{aug}\}$ and $\boldsymbol{y}^t = \{\boldsymbol{y}_{train} \cup \boldsymbol{y}^t_{aug}\}$ are the set of original and augmented examples and their labels, $\mathcal{J}(\boldsymbol{W}, \boldsymbol{X}) \in \mathbb{R}^{n \times m}$ is the Jacobian matrix associated with f, and $\boldsymbol{r}^t = f(\boldsymbol{W}^t, \boldsymbol{X}) - \boldsymbol{y}$ is the residual.

We further assume that $\mathcal{J}$ is smooth with Lipschitz constant L. I.e., $\|\mathcal{J}(\boldsymbol{W}, \boldsymbol{x}_i) - \mathcal{J}(\boldsymbol{W}, \boldsymbol{x}_j)\| \leq L\| \boldsymbol{x}_i - \boldsymbol{x}_j\| \;\; \forall \; \boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{X}$. Thus, for any transformation $T_j \in \mathcal{S}$, we have $\|\mathcal{J}(\boldsymbol{W}, \boldsymbol{x}_i) - \mathcal{J}(\boldsymbol{W}, T_j(\boldsymbol{x}_i))\| \leq L\epsilon_0$. Finally, denoting $\mathcal{J} = \mathcal{J}(\boldsymbol{W}, \boldsymbol{X}_{train})$ and $\tilde{\mathcal{J}} = \mathcal{J}(\boldsymbol{W}, T_j(\boldsymbol{X}_{train}))$, we get $\tilde{\mathcal{J}} = \mathcal{J} + \boldsymbol{E}$, where $\boldsymbol{E}$ is the perturbation matrix with $\|\boldsymbol{E}\|_2 \leq \|\boldsymbol{E}\|_F \leq \sqrt{n} L \epsilon_0$.

4 Data Augmentation Improves Learning

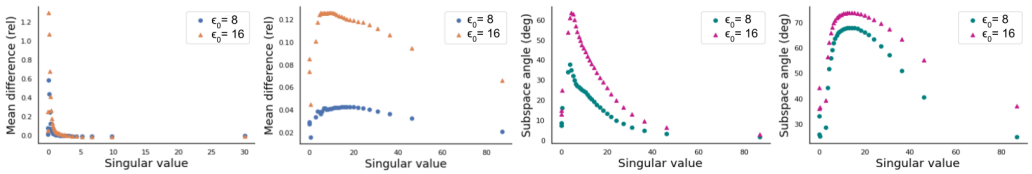
In this section, we analyze the effect of data augmentation on training dynamics of neural networks, and show that data augmentation can provably prevent overfitting. To do so, we leverage the recent results that characterize the training dynamics based on properties of neural network Jacobian and the corresponding Neural Tangent Kernel (NTK) [16] defined as $\boldsymbol{\Theta} = \mathcal{J}(\boldsymbol{W}, \boldsymbol{X})\mathcal{J}(\boldsymbol{W}, \boldsymbol{X})^T$. Formally:

$$\boldsymbol{r}^t = \sum_{i=1}^{n} (1 - \eta\lambda_i)(\boldsymbol{u}_i \boldsymbol{u}_i^T)\boldsymbol{r}^{t-1} = \sum_{i=1}^{n} (1 - \eta\lambda_i)^t (\boldsymbol{u}_i \boldsymbol{u}_i^T)\boldsymbol{r}^0, \tag{4}$$

where $\boldsymbol{\Theta} = \boldsymbol{U}\Lambda\boldsymbol{U}^T = \sum_{i=1} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^T$ is the eigendecomposition of the NTK [1]. Although the constant NTK assumption holds only in the infinite width limit, [21] found close empirical agreement between the NTK dynamics and the true dynamics for wide but practical networks, such as wide ResNet architectures [41]. Eq. (4) shows that the training dynamics depend on the alignment of the NTK with the residual vector at every iteration t. Next, we prove that for small perturbations ϵ_0, data augmentation prevents overfitting and improves generalization by proportionally enlarging and perturbing smaller eigenvalues of the NTK relatively more, while preserving its prominent directions.

4.1 Effect of Augmentation on Eigenvalues of the NTK

We first investigate the effect of data augmentation on the singular values of the Jacobian, and use this result to bound the change in the eigenvalues of the NTK. To characterize the effect of data augmentation on singular values of the perturbed Jacobian $\tilde{\mathcal{J}}$, we rely on Weyl's theorem [39] stating that under bounded perturbations $\boldsymbol{E}$, no singular value can move more than the norm of the perturbations. Formally, $|\tilde{\sigma}_i - \sigma_i| \leq \|\boldsymbol{E}\|_2$, where $\tilde{\sigma}_i$ and σ_i are the singular values of the perturbed and original Jacobian respectively. Crucially, data augmentation affects larger and smaller singular values differently. Let $\boldsymbol{P}$ be orthogonal projection onto the column space of $\mathcal{J}^T$, and $\boldsymbol{P}_\perp = \boldsymbol{I} - \boldsymbol{P}$ be the projection onto its orthogonal complement subspace. Then, the singular values of the perturbed



(a) MNIST - $\Delta\sigma_i$ (b) CIFAR - $\Delta\sigma_i$ (c) MNIST - Subs. Angle (d) CIFAR - Sub. Angle

Figure 1: Effect of augmentations on the singular spectrum of the network Jacobian of ResNet20 trained on CIFAR10, and a MLP on MNIST, trained till epoch 15. (a), (b) Difference in singular values and (c), (d) singular subspace angles between the original and augmented data with bounded perturbations with $\epsilon_0 = 8$ and $\epsilon_0 = 16$ for different ranges of singular values. Note that augmentations with larger bound ϵ_0 results in larger perturbations to the singular spectrum.

Jacobian $\tilde{\mathcal{J}}^T$ are $\tilde{\sigma}_i^2 = (\sigma_i + \mu_i)^2 + \zeta_i^2$, where $|\mu_i| \leq \|\boldsymbol{P}\boldsymbol{E}\|_2$, and $\sigma_{\min}(\boldsymbol{P}_\perp \boldsymbol{E}) \leq \zeta_i \leq \|\boldsymbol{P}_\perp \boldsymbol{E}\|_2$, $\sigma_{\min}$ the smallest singular value of $\mathcal{J}^T$ [36]. Since the eigenvalues of the projection matrix $\boldsymbol{P}$ are either 0 or 1, as the number of dimensions m grows, for bounded perturbations we get that on average $\mu_i^2 = \mathcal{O}(1)$ and $\zeta_i^2 = \mathcal{O}(m)$. Thus, the second term dominates and increase of small singular values under perturbation is proportional to $\sqrt{m}$. However, for larger singular values, first term dominates and hence $\tilde{\sigma}_i - \sigma_i \cong \mu_i$. Thus in general, small singular values can become proportionally larger, while larger singular values remain relatively unchanged. The following Lemma characterizes the *expected* change to the eignvalues of the NTK.

Lemma 4.1. *Data augmentation as additive perturbations bounded by small ϵ_0 results in the following expected change to the eigenvalues of the NTK:*

$$\mathbb{E}[\tilde{\lambda}_i] = \mathbb{E}[\tilde{\sigma}_i^2] = \sigma_i^2 + \sigma_i(1 - 2p_i)\|\boldsymbol{E}\| + \|\boldsymbol{E}\|^2/3 \tag{5}$$

where $p_i := \mathbb{P}(\tilde{\sigma}_i - \sigma_i < 0)$ is the probability that σ_i decreases as a result of data augmentation, and is smaller for smaller singular values.

The proof can be found in Appendix A.1.

Next, we discuss the effect of data augmentation on singular vectors of the Jacobian and show that it mainly affects the non-prominent directions of the Jacobian spectrum, but to a smaller extent compared to the singular values.

4.2 Effect of Augmentation on Eigenvectors of the NTK

Here, we focus on characterizing the effect of data augmentation on the eigenspace of the NTK. Let the singular subspace decomposition of the Jacobian be $\mathcal{J} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. Then for the NTK, we have $\boldsymbol{\Theta} = \mathcal{J}\mathcal{J}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^T$ (since $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$). Hence, the perturbation of the eigenspace of the NTK is the same as perturbation of the left singular subspace of the Jacobian $\mathcal{J}$. Suppose σ_i are singular values of the Jacobian. Let the perturbed Jacobian be $\tilde{\mathcal{J}} = \mathcal{J} + \boldsymbol{E}$, and denote the eigengap $\gamma_0 = \min\{\sigma_i - \sigma_{i+1} : i = 1, \cdots, r\}$ where $\sigma_{r+1} := 0$. Assuming $\gamma_0 \geq 2\|\boldsymbol{E}\|_2$, a combination of Wedin's theorem [38] and Mirsky's inequality [26] implies

$$\|\boldsymbol{u}_i - \tilde{\boldsymbol{u}}_i\| \leq 2\sqrt{2}\|\boldsymbol{E}\|/\gamma_0. \tag{6}$$

This result provides an upper-bound on the change of every left singular vectors of the Jacobian.

However as we discuss below, data augmentation affects larger and smaller singular directions differently. To see the effect of data augmentation on every singular vectors of the Jacobian, let the subspace decomposition of Jacobian be $\mathcal{J} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U}_s\boldsymbol{\Sigma}_s\boldsymbol{V}_s^T + \boldsymbol{U}_n\boldsymbol{\Sigma}_n\boldsymbol{V}_n^T$, where $\boldsymbol{U}_s$ associated with nonzero singular values, spans the column space of $\mathcal{J}$, which is also called the signal subspace, and $\boldsymbol{U}_n$, associated with zero singular values ($\boldsymbol{\Sigma}_n = 0$), spans the orthogonal space of $\boldsymbol{U}_s$, which is also called the noise subspace. Similarly, let the subspace decomposition of the perturbed Jacobian be $\tilde{\mathcal{J}} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^T = \tilde{\boldsymbol{U}}_s\tilde{\boldsymbol{\Sigma}}_s\tilde{\boldsymbol{V}}_s^T + \tilde{\boldsymbol{U}}_n\tilde{\boldsymbol{\Sigma}}_n\tilde{\boldsymbol{V}}_n^T$, and $\tilde{\boldsymbol{U}}_s = \boldsymbol{U}_s + \Delta\boldsymbol{U}_s$, where $\Delta\boldsymbol{U}_s$ is the perturbation of the singular vectors that span the signal subspace. Then the following general first-order expression for the perturbation of the orthogonal subspace due to perturbations of the Jacobian characterize the change of the singular directions: $\Delta\boldsymbol{U}_s = \boldsymbol{U}_n\boldsymbol{U}_n^T\boldsymbol{E}\boldsymbol{V}_s\boldsymbol{\Sigma}_s^{-1}$ [23]. We see that singular vectors associated to larger singular values are more robust to data augmentation, compared to others. Note that in general singular vectors are more robust than singular values.

Fig. 1 shows the effect of perturbations with $\epsilon_0 = 8, 16$ on singular values and singular vectors of the Jacobian matrix for a 1 hidden layer MLP trained on MNIST, and ResNet20 trained on CIFAR10. As calculating the entire Jacobian spectrum is computationally prohibitive, data is subsampled from 3 classes. We report the effect of other real-world augmentation techniques, such as random crops, flips, rotations and Autoaugment [7] - which includes translations, contrast, and brightness transforms - in Appendix C. We observe that data augmentation increases smaller singular values relatively more. On the other hand, it affects prominent singular vectors of the Jacobian to a smaller extent.

4.3 Augmentation Improves Training & Generalization

Recent studies have revealed that the Jacobian matrix of common neural networks is low rank. That is there are a number of large singular values and the rest of the singular values are small. Based on this, the Jacobian spectrum can be divided into information and nuisance spaces [31]. Information space is a lower dimensional space associated with the prominent singular value/vectors of the Jacobian. Nuisance space is a high dimensional space corresponding to smaller singular value/vectors of the Jacobian. While learning over information space is fast and generalizes well, learning over nuisance space is slow and results in overfitting [31]. Importantly, recent theoretical studies connected the generalization performance to small singular values (of the information space) [1].

Our results show that label-preserving additive perturbations relatively enlarge the smaller singular values of the Jacobian in a *stochastic* way and with a high probability. This benefits generalization in 2 ways. First, this stochastic behavior prevents overfitting along any particular singular direction *in the nuisance space*, as stochastic perturbation of the *smallest* singular values results in a stochastic noise to be added to the gradient at every training iteration. This prevents overfitting (thus a larger training loss as shown in Appendix D.5), and improves generalization [8, 9]. Theorem B.1 in the Appendix characterizes the expected training dynamics resulted by data augmentation. Second, additive perturbations improve the generalization by enlarging the smaller (useful) singular values that lie in the *information space*, while preserving eigenvectors. Hence, it enhances learning along these (harder to learn) components. The following Lemma captures the improvement in the generalization performance, as a result of data augmentation.

Lemma 4.2. *Assume gradient descent with learning rate η is applied to train a neural network with constant NTK and Lipschitz constant L, on data points augmented with additive perturbations bounded by ϵ_0 as defined in Sec. 3. Let $\sigma_{\min}$ be the minimum singular value of Jacobian $\mathcal{J}$ associated with training data $\boldsymbol{X}_{train}$. With probability $1-\delta$, generalization error of the network trained with gradient descent on augmented data $\boldsymbol{X}_{aug}$ enjoys the following bound:*

$$\sqrt{\frac{2}{(\sigma_{\min} + \sqrt{n}L\epsilon_0)^2}} + \mathcal{O}\left(\log \frac{1}{\delta}\right). \tag{7}$$

The proof can be found in Appendix A.2.

5 Effective Subsets for Data Augmentation

Here, we focus on identifying subsets of data that when augmented similarly improve generalization and prevent overfitting. To do so, our key idea is to find subsets of data points that when augmented, closely capture the alignment of the NTK (or equivalently the Jacobian) corresponding to the full augmented data with the residual vector, $\mathcal{J}(\boldsymbol{W}^t, \boldsymbol{X}_{aug}^t)^T \boldsymbol{r}_{aug}^t$. If such subsets can be found, augmenting only the subsets will change the NTK and its alignment with the residual in a similar way as that of full data augmentation, and will result in similar improved training dynamics. However, generating the full set of transformations $\boldsymbol{X}_{aug}^t$ is often very expensive, particularly for strong augmentations and large datasets. Hence, generating the transformations, and then extracting the subsets may not provide a considerable overall speedup.

In the following, we show that weighted subsets (coresets) S that closely estimate the alignment of the Jacobian associated to the original data with the residual vector $\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_{train})\boldsymbol{r}_{train}$ can closely estimate the alignment of the Jacobian of the full augmented data and the corresponding residual $\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_{aug}^t)\boldsymbol{r}_{aug}^t$. Thus, the most effective subsets for augmentation can be directly found from the training data. Formally, subsets S_*^t weighted by $\boldsymbol{\gamma}_S^t$ that capture the alignment of the full Jacobian

Algorithm 1 CORESETS FOR EFFICIENT DATA AUGMENTATION

Require: The dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, number of iterations T.
Ensure: Output model parameters $\boldsymbol{W}^T$.

for $t = 1, \cdots, T$ **do**
  $\boldsymbol{X}_{aug}^t = \emptyset$.
  for $c \in \{1, \cdots, C\}$ **do**
    $S_c^t = \emptyset$, $[\boldsymbol{G}_{S_c^t}]_{i.} = c_1 \mathbf{1} \ \ \forall i$.
    while $\|\boldsymbol{G}_{S_c^t}\|_F \geq \xi$ **do** ▷ Extract a coreset from class c by solving Eq. (9)
      $S_c^t = \{S_c^t \cup \arg\max_{s \in V \setminus S_c^t} (\|\boldsymbol{G}_{S_c^t}\|_F - \|\boldsymbol{G}_{\{S_c^t \cup \{s\}\}}\|_F)\}$
    end while
    $\gamma_j = \sum_{i \in V_c} \mathbb{I}[j = \arg\min_{j' \in S} \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_i) r_i - \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_{j'}) r_{j'}\|]$ ▷ Coreset weights
    $\boldsymbol{X}_{aug}^t = \{\boldsymbol{X}_{aug} \cup \{\cup_{i=1}^r T_i^t(\boldsymbol{X}_{S_c^t})\}\}$ ▷ Augment the coreset
    $\boldsymbol{\rho}_j^t = \gamma_j^t / r$
  end for
  Update the parameters $\boldsymbol{W}^t$ using weighted gradient descent on $\boldsymbol{X}_{aug}^t$ or $\{\boldsymbol{X}_{train} \cup \boldsymbol{X}_{aug}^t\}$.
end for

with the residual by an error of at most ξ can be found by solving the following optimization problem:

$$S_*^t = \arg\min_{S \subseteq V} |S| \qquad \text{s.t.} \qquad \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}^t)\boldsymbol{r}^t - \text{diag}(\boldsymbol{\gamma}_S^t)\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S^t)\boldsymbol{r}_S^t\| \leq \xi. \tag{8}$$

Solving the above optimization problem is NP-hard. However, as we discuss in the Appendix A.5, a near optimal subset can be found by minimizing the Frobenius norm of a matrix $\boldsymbol{G}_S$, in which the ith row contains the euclidean distance between data point i and its closest element in the subset S, in the gradient space. Formally, $[\boldsymbol{G}_S]_{i.} = \min_{j' \in S} \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_i) r_i - \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_{j'}) r_{j'}\|$. When $S = \emptyset$, $[\boldsymbol{G}_S]_{i.} = c_1 \mathbf{1}$, where c_1 is a big constant. Intuitively, such subsets contain the set of medoids of the dataset in the gradient space. Medoids of a dataset are defined as the most centrally located elements in the dataset [18]. The weight of every element $j \in S$ is the number of data points closest to it in the gradient space, i.e., $\gamma_j = \sum_{i \in V} \mathbb{I}[j = \arg\min_{j' \in S} \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_i) r_i - \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_{j'}) r_{j'}\|]$. The set of medoids can be found by solving the following *submodular*[3] cover problem:

$$S_*^t = \arg\min_{S \subseteq V} |S| \quad s.t. \quad \|\boldsymbol{G}_S\|_F \leq \xi. \tag{9}$$

The classical greedy algorithm provides a logarithmic approximation for the above submodular maximization problem, i.e., $|S| \leq (1 + ln(n))$. It starts with the empty set $S_0 = \emptyset$, and at each iteration τ, it selects the training example $s \in V \setminus S_{\tau-1}$ that maximizes the marginal gain, i.e., $S_\tau = S_{\tau-1} \cup \{\arg\max_{s \in V \setminus S_{\tau-1}} (\|\boldsymbol{G}_{S_{\tau-1}}\|_F - \|\boldsymbol{G}_{\{S_{\tau-1} \cup \{s\}\}}\|_F)\}$. The $\mathcal{O}(nk)$ computational complexity of the greedy algorithm can be reduced to $\mathcal{O}(n)$ using randomized methods [28] and further improved using lazy evaluation [25] and distributed implementations [30]. The rows of the matrix $\boldsymbol{G}$ can be efficiently upper-bounded using the gradient of the loss w.r.t. the input to the last layer of the network, which has been shown to capture the variation of the gradient norms closely [17]. The above upper-bound is only marginally more expensive than calculating the value of the loss. Hence the subset can be found efficiently. Better approximations can be obtained by considering earlier layers in addition to the last two, at the expense of greater computational cost.

At every iteration t during training, we select a coreset from every class $c \in [C]$ separately, and apply the set of transformations $\{T_i^t\}_{i=1}^r$ only to the elements of the coresets, i.e., $X_{aug}^t = \{\cup_{i=1}^r T_i^t(\boldsymbol{X}_{S^t})\}$. We divide the weight of every element j in the coreset equally among its transformations, i.e. the final weight $\rho_j^t = \gamma_j^t / r$ if $j \in S^t$. We apply the gradient descent updates in Eq. (3) to the weighted Jacobian matrix of $\boldsymbol{X}^t = \boldsymbol{X}_{aug}^t$ or $\boldsymbol{X}^t = \{\boldsymbol{X}_{train} \cup \boldsymbol{X}_{aug}^t\}$ (viewing $\boldsymbol{\rho}^t$ as $\boldsymbol{\rho}^t \in \mathbb{R}^n$) as follows:

$$\boldsymbol{W}^{t+1} = \boldsymbol{W}^t - \eta \left(\text{diag}(\boldsymbol{\rho}^t)\mathcal{J}(\boldsymbol{W}^t, \boldsymbol{X}^t)\right)^T \boldsymbol{r}^t. \tag{10}$$

The pseudocode is illustrated in Alg. 1.

The following Lemma upper bounds the difference between the alignment of the Jacobian and residual for augmented coreset vs. full augmented data.

[3] A set function $F : 2^V \to \mathbb{R}^+$ is submodular if $F(S \cup \{e\}) - F(S) \geq F(T \cup \{e\}) - F(T)$, for any $S \subseteq T \subseteq V$ and $e \in V \setminus T$. F is *monotone* if $F(e|S) \geq 0$ for any $e \in V \setminus S$ and $S \subseteq V$.

Table 1: Training ResNet20 (R20) and WideResnet-28-10 (W2810) on CIFAR10 (C10) using small subsets, and ResNet18 (R18) on Caltech256 (Cal). We compare accuracies of training on and strongly (and weakly) augmenting subsets. For CIFAR10, training and augmenting subsets selected by max-loss performed poorly and did not converge. Average number of examples per class in each subset is shown in parentheses. Appendix D.4 shows baseline accuracies from only weak augmentations.

Model/Data	C10/R20				C10/W2810	Cal/R18					
Subset	0.1% (5)	0.2% (10)	0.5% (25)	1% (50)	1% (50)	5% (3)	10% (6)	20% (12)	30% (18)	40% (24)	50% (30)
Max-loss	$< 15\%$	$< 15\%$	$< 15\%$	$< 15\%$	$< 15\%$	19.2	50.6	71.3	75.6	77.3	78.6
Random	33.5	42.7	58.7	74.4	57.7	41.5	61.8	72.5	75.7	77.6	78.5
Ours	**37.8**	**45.1**	**63.9**	**74.7**	**62.1**	**52.7**	**65.4**	**73.1**	**76.3**	**77.7**	**78.9**

Lemma 5.1. *Let S be a coreset that captures the alignment of the full data NTK with residual with an error of at most ξ as in Eq. 8. Augmenting the coreset with perturbations bounded by $\epsilon_0 \leq \frac{1}{n^{\frac{3}{2}}\sqrt{L}}$ captures the alignment of the fully augmented data with the residual by an error of at most*

$$\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_{aug})\boldsymbol{r} - diag(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_{S^{aug}})\boldsymbol{r}_S\| \leq \xi + \mathcal{O}\left(\sqrt{L}\right). \tag{11}$$

5.1 Coreset vs. Max-loss Data Augmentation

In the initial phase of training the NTK goes through rapid changes. This determines the final basin of convergence and network's final performance [11]. Regularizing deep networks by weight decay or data augmentation mainly affects this initial phase and matters little afterwards [12]. Crucially, augmenting coresets that closely capture the alignment of the NTK with the residual during this initial phase results in less overfitting and improved generalization performance. On the other hand, augmenting points with maximum loss early in training decreases the alignment between the NTK and the label vector and impedes learning and convergence. After this initial phase when the network has good prediction performance, the gradients for majority of data points become small. Here, the alignment is mainly captured by the elements with the maximum loss. Thus, as training proceeds, the intersection between the elements of the coresets and examples with maximum loss increases. We visualize this pattern in Appendix D.11. The following Theorem characterizes the training dynamics of training on the full data and the augmented coresets, using our additive perturbation model.

Theorem 5.2. *Let $\mathcal{L}_i$ be β-smooth, $\mathcal{L}$ be λ-smooth and satisfy the α-PL condition, that is for $\alpha > 0$, $\|\nabla\mathcal{L}(\boldsymbol{W})\|^2 \geq \alpha\mathcal{L}(\boldsymbol{W})$ for all weights $\boldsymbol{W}$. Let f be Lipschitz in $\boldsymbol{X}$ with constant L', and $\bar{L} = \max\{L, L'\}$. Let G_0 be the gradient at initializaion, $\sigma_{\max}$ the maximum singular value of the coreset Jacobian at initialization. Choosing $\epsilon_0 \leq \frac{1}{\sigma_{\max}\sqrt{\bar{L}n}}$ and running SGD on full data with augmented coreset using constant step size $\eta = \frac{\alpha}{\lambda\beta}$, result in the following bound:*

$$\mathbb{E}[\|\nabla\mathcal{L}^{f+c_{\text{aug}}}(\boldsymbol{W}^t)\|] \leq \frac{1}{\sqrt{\alpha}}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{t}{2}}\left(2G_0 + \xi + \mathcal{O}\left(\frac{\sqrt{\bar{L}}}{\sigma_{\max}}\right)\right).$$

The proof can be found in Appendix A.4.

Theorem 5.2 shows that training on full data and augmented coresets converges to a close neighborhood of the optimal solution, with the same rate as that of training on the fully augmented data. The size of the neighborhood depends on the error of the coreset ξ in Eq. (8), and the error in capturing the alignment of the full augmented data with the residual derived in Lemma 5.1. The first term decrease as the size of the coreset grows, and the second term depends on the network structure.

We also analyze convergence of training only on the augmented coresets, and augmentations modelled as arbitrary linear transformations using a linear model [40] in Appendix B.5.

6 Experiments

Setup and baselines. We extensively evaluate the performance of our approach in three different settings. Firstly, we consider training only on coresets and their augmentations. Secondly, we investigate the effect of adding augmented coresets to the full training data. Finally, we consider adding augmented coresets to random subsets. We compare our coresets with max-loss and random subsets as baselines. For all methods, we select a new augmentation subset every R epochs. We note that the original max-loss method [20] selects points using a fully trained model, hence it can only

Table 2: Caltech256/ResNet18 with same settings as Tab. 1 with default weak augmentations but varying strong augmentations.

Augmentation	Random			Ours		
	30%	40%	50%	30%	40%	50%
CutOut	43.32	62.84	76.21	**55.53**	**66.10**	**76.91**
AugMix	40.77	61.81	72.17	**52.72**	**64.91**	**73.01**
Perturb	48.51	66.20	75.34	**58.29**	**67.47**	**76.50**

Table 3: Training on full data and strongly (and weakly) augmenting random subsets, max-loss subsets and coresets on TinyImageNet/ResNet50, $R = 15$.

Random			Max-loss			Ours		
20%	30%	50%	20%	30%	50%	20%	30%	50%
50.97	52.00	54.92	51.30	52.34	53.37	**51.99**	**54.30**	**55.16**

Table 4: Accuracy improvement by augmenting subsets found by our method vs. max-loss and random, over improvement of full (weak and strong) data augmentation (F.A.) compared to weak augmentation only (W.A.). The table shows the results for training on CIFAR10(C10)/ResNet20 (R20), SVHN/ResNet32(R32), and CIFAR10-Imbalanced(C10-IB)/ResNet32, with $R = 20$.

Dataset	W.A.	F.A.	Random			Max-loss			Ours		
	Acc	Acc	5%	10%	30%	5%	10%	30%	5%	10%	30%
C10/R20	89.46	93.50	21.8%	39.9%	65.6%	32.9%	47.8%	73.5%	**34.9%**	**51.5%**	**75.0%**
C10-IB/R32	87.08	92.48	25.9%	45.2%	74.6%	31.3%	39.6%	74.6%	**37.4%**	**49.4%**	**74.8%**
SVHN/R32	95.68	97.07	5.8%	36.7%	64.1%	**35.3%**	**49.7%**	76.4%	31.7%	48.3%	**80.0%**

select one subset throughout training. To maximize fairness, we modify our max-loss baseline to select a new subset at every subset selection step. For all experiments, standard weak augmentations (random crop and horizontal flips) are always performed on both the original and strongly augmented data.

6.1 Training on Coresets and their Augmentations

First, we evaluate the effectiveness of our approach for training on the coresets and their augmentations. Our main goal here is to compare the performance of training on and augmenting coresets vs. random and max-loss subsets. Tab. 1 shows the test accuracy for training ResNet20 and Wide-ResNet on CIFAR10 when we only train on small augmented coresets of size 0.1% to 1% selected at every epoch ($R = 1$), and training ResNet18 on Caltech256 using coresets of size 5% to 50% with $R = 5$. We see that the augmented coresets outperform augmented random subsets by a large margin, particularly when the size of the subset is small. On Caltech256/ResNet18, training on and augmenting 10% coresets yields 65.4% accuracy, improving over random by 5.8%, and over only weak augmentation by 17.4%. This clearly shows the effectiveness of augmenting the coresets. Note that for CIFAR10 experiments, training on the augmented max-loss points did not even converge in absence of full data.

Generalization across augmentation techniques. We note that our coresets are not dependent on the type of data augmentation. To confirm this, we show the superior generalization performance of our method in Tab. 2 for training ResNet18 with $R = 5$ on coresets vs. random subsets of Caltech256, augmented with CutOut [10], AugMix [14], and noise perturbations (color jitter, gaussian blur). For example, on 30% subsets, we obtain 28.2%, 29.3%, 20.2% relative improvement over augmenting random subsets when using CutOut, AugMix, and noise perturbation augmentations, respectively.

6.2 Training on Full Data and Augmented Coresets

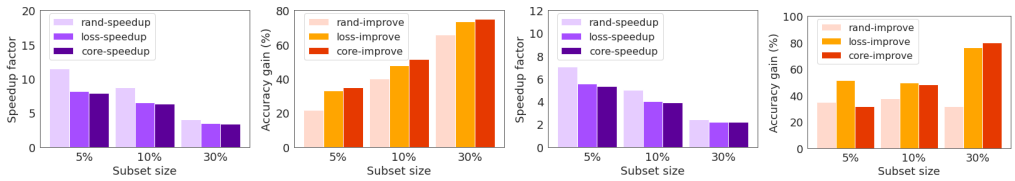
Next, we study the effectiveness of our method for training on full data and augmented coresets. Tab. 4 demonstrates the percentage of accuracy improvement resulted by augmenting subsets of size 5%, 10%, and 30% selected from our method vs. max-loss and random subsets, over that of full data augmentation. We observe that augmenting coresets effectively improves generalization, and outperforms augmenting random and max-loss subsets across different models and datasets. For example, on 30% subsets, we obtain 13.1% and 2.3% relative improvement over random and max-loss on average. We also report results on TinyImageNet/ResNet50 (R=15) in Tab. 3, where we show that augmenting coresets outperforms max-loss and random baselines, e.g. by achieving 3.7% and 4.4% relative improvement over 30% max-loss and random subsets, respectively.

Training speedup. In Fig. 2, we measure the improvement in training time in the case of training on full data and augmenting subsets of various sizes. While our method yields similar or slightly lower speedup to the max-loss and random baselines, our resulting accuracy outperforms the baselines on average. For example, for SVHN/Resnet32 using 30% coresets, we sacrifice 11% of the relative speedup to obtain an additional 24.8% of the relative gain in accuracy from full data augmentation, compared to random baseline. Notably, we get 3.43x speedup for training on full data and augmenting

Table 5: Training ResNet20 on CIFAR10 with 50% label noise, $R = 20$. Accuracy without strong augmentation is 70.72 ± 0.20 and the accuracy of full (weak and strong) data augmentation is 75.87 ± 0.77. Note that augmenting 50% subsets outperforms augmenting the full data (marked **).

Subset	Random	Max-loss	Ours
10%	72.32 ± 0.14	71.83 ± 0.13	$\mathbf{73.02 \pm 1.06}$
30%	74.46 ± 0.27	72.45 ± 0.48	$\mathbf{74.67 \pm 0.15}$
50%	75.36 ± 0.05	73.23 ± 0.72	$\mathbf{76.20 \pm 0.75}^{**}$

30% coresets, while obtaining 75% of the improvement of full data augmentation. We provide wall-clock times for finding coresets from Caltech256 and TinyImageNet in Appendix D.7.



(a) CIFAR10/ResNet20 (b) CIFAR10/ResNet20 (c) SVHN/ResNet32 (d) SVHN/ResNet32

Figure 2: Accuracy improvement and speedups by augmenting subsets found by our method vs. max-loss and random on (a), (b) ResNet20/CIFAR10 and (c), (d) ResNet32/SVHN.

Augmenting noisy labeled data. Next, we evaluate the robustness of our coresets to label noise. Tab. 5 shows the result of augmenting coresets vs. max-loss and random subsets of different sizes selected from CIFAR10 with 50% label noise on ResNet20. Notably, our method not only outperforms max-loss and random baselines, but also achieves superior performance over full data augmentation.

6.3 Training on Random Data and Augmented Coresets

Finally, we evaluate the performance of our method for training on random subsets and augmenting coresets, applicable when data is larger than the training budget. We report results on TinyImageNet and ImageNet on ResNet50 (90 epochs, $R = 15$). Tab. 6 shows the results of training on random subsets, and augmenting random subsets and coresets of the same size. We see that our results hold for large-scale datasets, where we obtain 7.9%, 4.9%, and 5.3% relative improvement over random baseline with 10%, 20%, 30% subset sizes respectively on TinyImageNet, and 7.6%, 2.3%, and 1.3% relative improvement over random baseline with 10%, 30%, and 50% subset sizes on ImageNet. Notably, compared to AutoAugment, despite using only 30% subsets, we achieve 71.99% test accuracy, which is 92.8% of the original reported accuracy, while boasting 5x speedup in training.

Table 6: Training on random subsets and strongly (and weakly) augmenting random and max loss subsets vs coresets for TinyImageNet (left) and ImageNet (right) with ResNet50.

Random			Max-loss			Ours			Random			Maxloss			Ours		
10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	30%	50%	10%	30%	50%	10%	30%	50%
28.64	38.97	44.10	27.64	**41.40**	45.75	**30.90**	40.88	**46.42**	63.67	70.39	72.35	65.43	71.55	72.77	**68.53**	**71.99**	**73.28**

7 Conclusion

We showed that data augmentation improves training and generalization by relatively enlarging and perturbing the smaller singular values of the neural network Jacobian while preserving its prominent directions. Then, we proposed a framework to iteratively extract small coresets of training data that when augmented, closely capture the alignment of the fully augmented Jacobian with the label/residual vector. We showed the effectiveness of augmenting coresets in providing a superior generalization performance when added to the full data or random subsets, in presence of noisy labels, or as a standalone subset. Under local smoothness of images, our additive perturbation can be applied to model many bounded transformations such as small rotations, crops, shearing, and pixel-wise transformations like sharpening, blurring, color distortions, structured adversarial perturbation [24]. However, the additive perturbation model is indeed limited when applied to augmentations that cannot be reduced to perturbations, such as horizontal/vertical flips and large translations. Further theoretical analysis of complex data augmentations is indeed an interesting direction for future work.

8 Acknowledgements

This research was supported in part by the National Science Foundation CAREER Award 2146492, and the UCLA-Amazon Science Hub for Humanity and AI.

References

[1] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

[2] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[3] Raef Bassily, Mikhail Belkin, and Siyuan Ma. On exponential convergence of sgd in non-convex over-parametrized learning. *arXiv preprint arXiv:1811.02564*, 2018.

[4] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[5] Christopher Bowles, Roger Gunn, Alexander Hammers, and Daniel Rueckert. Gansfer learning: Combining labelled and unlabelled data for gan based data augmentation. *arXiv preprint arXiv:1811.10669*, 2018.

[6] Shuxiao Chen, Edgar Dobriban, and Jane H Lee. Invariance reduces variance: Understanding data augmentation in deep learning and beyond. *arXiv preprint arXiv:1907.10905*, 2019.

[7] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[8] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[9] Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, and Christopher Ré. A kernel theory of modern data augmentation. In *International Conference on Machine Learning*, pages 1528–1537. PMLR, 2019.

[10] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[11] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33, 2020.

[12] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. *Advances in Neural Information Processing Systems*, 32:10678–10688, 2019.

[13] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[14] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[15] Philip TG Jackson, Amir Atapour Abarghouei, Stephen Bonner, Toby P Breckon, and Boguslaw Obara. Style augmentation: data augmentation via style randomization. In *CVPR Workshops*, volume 6, pages 10–11, 2019.

[16] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[17] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pages 2525–2534. PMLR, 2018.

[18] L Kaufman, PJ Rousseeuw, and Y Dodge. Clustering by means of medoids in statistical data analysis based on the, 1987.

[19] Byungju Kim and Junmo Kim. Adjusting decision boundary for class imbalanced learning. *IEEE Access*, 8:81674–81685, 2020.

[20] Michael Kuchnik and Virginia Smith. Efficient augmentation via data subsampling. In *International Conference on Learning Representations*, 2018.

[21] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *NeurIPS*, 2019.

[22] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart augmentation learning an optimal data augmentation strategy. *Ieee Access*, 5:5858–5869, 2017.

[23] Fu Li, Hui Liu, and Richard J Vaccaro. Performance analysis for doa estimation algorithms: unification, simplification, and observations. *IEEE Transactions on Aerospace and Electronic Systems*, 29(4):1170–1184, 1993.

[24] Calvin Luo, Hossein Mobahi, and Samy Bengio. Data augmentation via structured adversarial perturbations. *arXiv preprint arXiv:2011.03010*, 2020.

[25] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pages 234–243. Springer, 1978.

[26] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59, 1960.

[27] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[28] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[29] 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.

[30] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

[31] Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

[32] Shashank Rajput, Zhili Feng, Zachary Charles, Po-Ling Loh, and Dimitris Papailiopoulos. Does data augmentation lead to positive margin? In *International Conference on Machine Learning*, pages 5321–5330. PMLR, 2019.

[33] Alexander J Ratner, Henry R Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. *Advances in neural information processing systems*, 30:3239, 2017.

[34] Ruoqi Shen, Sébastien Bubeck, and Suriya Gunasekar. Data augmentation as feature manipulation: a story of desert cows and grass cows. *arXiv preprint arXiv:2203.01572*, 2022.

[35] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[36] GW Stewart. A note on the perturbation of singular values. *Linear Algebra and Its Applications*, 28:213–216, 1979.

[37] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. *arXiv preprint arXiv:1307.1493*, 2013.

[38] Per-Åke Wedin. Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics*, 12(1):99–111, 1972.

[39] Hermann Weyl. The asymptotic distribution law of the eigenvalues of linear partial differential equations (with an application to the theory of cavity radiation). *mathematical annals*, 71(4):441–479, 1912.

[40] Sen Wu, Hongyang Zhang, Gregory Valiant, and Christopher Ré. On the generalization effects of linear transformations in data augmentation. In *International Conference on Machine Learning*, pages 10410–10420. PMLR, 2020.

[41] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In Supp Mat.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material:
Data-Efficient Augmentation for Training Neural Networks

A Proof of Main Results

A.1 Proof for Lemma 4.1

Proof. Let $\delta_i := \tilde{\sigma}_i - \sigma_i$, where $\mathbb{P}(\delta_i < 0) = p_i$. Assuming uniform probability between $-\|\boldsymbol{E}\|$ to 0, and between 0 to $\|\boldsymbol{E}\|$, we have pdf $\rho_i(x)$ for δ_i:

$$\rho_i(x) = \begin{cases} \frac{p_i}{\|\boldsymbol{E}\|}, & \text{if } -\|\boldsymbol{E}\| \leq x < 0 \\ \frac{1-p_i}{\|\boldsymbol{E}\|}, & 0 \leq x \leq \|\boldsymbol{E}\| \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Taking expectation,

$$\mathbb{E}(\tilde{\sigma}_i - \sigma_i) = \mathbb{E}(\delta_i) = \int_{-\infty}^{\infty} x\rho_i(x)dx \tag{13}$$

$$= \int_{-\|\boldsymbol{E}\|}^{0} x\frac{p_i}{\|\boldsymbol{E}\|}dx + \int_{0}^{\|\boldsymbol{E}\|} x\frac{1-p_i}{\|\boldsymbol{E}\|}dx \tag{14}$$

$$= -\frac{\|\boldsymbol{E}\|p_i}{2} + \frac{(1-p_i)\|\boldsymbol{E}\|}{2} \tag{15}$$

$$= \frac{(1-2p_i)\|E\|}{2} \tag{16}$$

We also have

$$\mathbb{E}(\delta_i^2) = \int_{-\infty}^{\infty} x^2\rho_i(x)dx \tag{17}$$

$$= \int_{-\|\boldsymbol{E}\|}^{0} x^2\frac{p_i}{\|\boldsymbol{E}\|}dx + \int_{0}^{\|\boldsymbol{E}\|} x^2\frac{1-p_i}{\|\boldsymbol{E}\|}dx \tag{18}$$

$$= \frac{\|\boldsymbol{E}\|^2 p_i}{3} + \frac{(1-p_i)\|\boldsymbol{E}\|^2}{3} \tag{19}$$

$$= \frac{\|\boldsymbol{E}\|^2}{3} \tag{20}$$

Thus, we have

$$\mathbb{E}(\tilde{\lambda}_i) = \mathbb{E}(\tilde{\sigma}_i^2) \tag{21}$$

$$= \mathbb{E}((\sigma_i + \delta_i)^2) \tag{22}$$

$$= \mathbb{E}(\sigma_i^2 + 2\sigma_i\delta_i + \delta_i^2) \tag{23}$$

$$= \sigma_i^2 + 2\sigma_i\mathbb{E}[\delta_i] + \mathbb{E}[\delta_i^2] \tag{24}$$

$$= \sigma_i^2 + 2\sigma_i\frac{(1-2p_i)\|\boldsymbol{E}\|}{2} + \frac{\|\boldsymbol{E}\|^2}{3} \tag{25}$$

$$= \sigma_i^2 + \sigma_i(1-2p_i)\|\boldsymbol{E}\| + \frac{\|\boldsymbol{E}\|^2}{3} \tag{26}$$

□

A.2 Proof of Corollary 4.2

Under the assumptions of Theorem 5.1 of [1], i.e. where the minimum eigenvalue of the NTK is $\lambda_{\min}(\mathcal{J}\mathcal{J}^T) \geq \lambda_0$ for a constant $\lambda_0 > 0$, and training data $\boldsymbol{X}$ of size n sampled i.i.d. from distribution D and 1-Lipschitz loss $\mathcal{L}$, we have that with probability $\delta/3$, training the over-parameterized neural network with gradient descent for $t \geq \Omega\left(\frac{1}{n\lambda_0}\log\frac{n}{\delta}\right)$ iterations results in the following population loss $\mathcal{L}_D$ (generalization error)

$$\mathcal{L}_D(\boldsymbol{W}^t, \boldsymbol{X}) \leq \sqrt{\frac{2\boldsymbol{y}^T(\mathcal{J}\mathcal{J}^T)^{-1}\boldsymbol{y}}{n}} + \mathcal{O}\left(\frac{\log\frac{n}{\lambda_0\delta}}{n}\right), \tag{27}$$

with high probability of at least $1-\delta$ over random initialization and training samples.

Hence, using $\lambda_{\min}, \sigma_{\min}$ to denote minimum eigen and singular value respectively of the NTK corresponding to full data, we get

$$\mathcal{L}_{D_{train}}(\boldsymbol{W}^t, \boldsymbol{X}_{train}) \leq \sqrt{\frac{2\frac{1}{\lambda_{\min}}\|y\|^2}{n}} + \mathcal{O}\left(\log\frac{1}{\delta}\right) \tag{28}$$

$$\leq \sqrt{\frac{2}{\sigma_{\min}^2}} + \mathcal{O}\left(\log\frac{1}{\delta}\right). \tag{29}$$

For augmented dataset $\boldsymbol{X}_{aug}$, we have $\tilde{\sigma_i} \leq \sigma_i + \sqrt{n}L\epsilon_0$, hence the improvement in the generalization error is at most

$$\mathcal{L}_{D_{aug}}(\boldsymbol{W}^t, \boldsymbol{X}_{\text{aug}}) \leq \sqrt{\frac{2}{(\sigma_{\min} + \sqrt{n}L\epsilon_0)^2}} + \mathcal{O}\left(\log\frac{1}{\delta}\right). \tag{30}$$

Combining these two results, we obtain Corollary 4.2.

A.3 Proof of Lemma 5.1

Proof.

$$\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_{aug})\boldsymbol{r} - \text{diag}(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_{S^{aug}})\boldsymbol{r}_S\| \tag{31}$$

$$= \|(\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}) + \boldsymbol{E})\boldsymbol{r} - (\text{diag}(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_S) + \boldsymbol{E}_S)\boldsymbol{r}_S\| \tag{32}$$

$$\leq \|(\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r} - \text{diag}(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S) + \boldsymbol{E}\boldsymbol{r} - \boldsymbol{E}_S\boldsymbol{r}_S\| \tag{33}$$

$$\leq \|(\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r} - \text{diag}(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S)\| + \|\boldsymbol{E}\boldsymbol{r}\| + \|\boldsymbol{E}_S\boldsymbol{r}_S\| \tag{34}$$

Applying definition of coresets, we obtain

$$\|(\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r} - \text{diag}(\boldsymbol{\rho}^t)\mathcal{J}^t(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S)\| + \|\boldsymbol{E}\boldsymbol{r}\| + \|\boldsymbol{E}_S\boldsymbol{r}_S\| \tag{35}$$

$$\leq \xi + \|\boldsymbol{E}\boldsymbol{r}\| + \|\boldsymbol{E}_S\boldsymbol{r}_S\| \tag{36}$$

$$\leq \xi + 2n^{\frac{3}{2}}L\epsilon_0 \tag{37}$$

$$\leq \xi + 2\sqrt{L} \tag{38}$$

□

A.4 Proof of Theorem 5.2

Proof. In this proof, as shorthand notation, we use $\boldsymbol{X}_f$ and $\boldsymbol{X}_{train}$ interchangeably. We further use $\boldsymbol{X}_c$ to represent the coreset selected from the full data, and $\boldsymbol{X}_{c_{aug}}$ to represent the augmented coreset.

By Theorem 1 of [3], under the α-PL assumption for $\mathcal{L}$ and interpolation assumption (i.e. for every sequence $\boldsymbol{W}^1, \boldsymbol{W}^2, \ldots$ such that $\lim_{t\to\infty}\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}) = 0$, we have that the loss for each data point $\lim_{t\to\infty}\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{x}_i) = 0$), the convergence of SGD with constant step size is given by

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{f+c_{\text{aug}}})\|^2] \leq \left(1 - \frac{\alpha\eta}{2}\right)^t \mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{f+c_{\text{aug}}}) \tag{39}$$

$$\leq \frac{1}{\alpha}\left(1 - \frac{\alpha\eta}{2}\right)^t \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{f+c_{\text{aug}}})\|^2 \tag{40}$$

Using Jensen's inequality, we have

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{f+c_{\text{aug}}})\|] \tag{41}$$

$$\leq \sqrt{\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{f+c_{\text{aug}}})\|^2]} \tag{42}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{f+c_{\text{aug}}})\| \tag{43}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \left(\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_f)\|+\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c_{\text{aug}}})\|\right) \tag{44}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \left(G_0 + \|(\mathcal{J}(\boldsymbol{W}^0, \boldsymbol{X}_c) + \boldsymbol{E})(\boldsymbol{y} - f(\boldsymbol{W}^0, \boldsymbol{X}_c + \boldsymbol{\epsilon}))\|\right) \tag{45}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \tag{46}$$

$$\left(G_0 + \|(\mathcal{J}(\boldsymbol{W}^0, \boldsymbol{X}_c) + \boldsymbol{E})^T(\boldsymbol{y} - f(\boldsymbol{W}^0, \boldsymbol{X}_c) - \nabla_x f(\boldsymbol{W}^0, \boldsymbol{X}_c)^T\boldsymbol{\epsilon} - \mathcal{O}(\boldsymbol{\epsilon}^T\boldsymbol{\epsilon})\|\right) \tag{47}$$

$$= \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} (G_0 + \|\nabla L(\boldsymbol{W}^0, \boldsymbol{X}_c) - (\mathcal{J}(\boldsymbol{W}^0, \boldsymbol{X}_c)^T(\nabla_x f(\boldsymbol{W}^0, \boldsymbol{X}_c)^T\boldsymbol{\epsilon} + \mathcal{O}(\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}))+ \tag{48}$$

$$\boldsymbol{E}(\boldsymbol{y} - f(\boldsymbol{W}^0, \boldsymbol{X}_c + \boldsymbol{\epsilon}))\|) \tag{49}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} (G_0 + \|\nabla L(\boldsymbol{W}^0, \boldsymbol{X}_c) - (\mathcal{J}(\boldsymbol{W}^0, \boldsymbol{X}_c)^T(\nabla_x f(\boldsymbol{W}^0, \boldsymbol{X}_c)^T\boldsymbol{\epsilon} + \mathcal{O}(\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}))\|+ \tag{50}$$

$$\sqrt{2}\|\boldsymbol{E}\|) \tag{51}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} (G_0 + \|\nabla L(\boldsymbol{W}^0, \boldsymbol{X}_c)\|+\sigma_{\max}\bar{L}\sqrt{n}\epsilon_0 + \sigma_{\max}\mathcal{O}(n\epsilon_0^2)) + \sqrt{2n}\bar{L}\epsilon_0) \tag{52}$$

$$= \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} (G_0 + \|\nabla L(\boldsymbol{W}^0, \boldsymbol{X}_f)\|+\xi + \sigma_{\max}\bar{L}\sqrt{n}\epsilon_0 + \sigma_{\max}\mathcal{O}(n\epsilon_0^2)) + \sqrt{2n}\bar{L}\epsilon_0) \tag{53}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} (2G_0 + \xi + \sigma_{\max}\bar{L}\sqrt{n}\epsilon_0 + \sigma_{\max}\mathcal{O}(n\epsilon_0^2)) + \sqrt{2n}\bar{L}\epsilon_0) \tag{54}$$

□

A.5 Finding Subsets

Let S be a subset of training data points. Furthermore, assume that there is a mapping $\pi_{w,S} : V \to S$ that for every $\boldsymbol{W}$ assigns every data point $i \in V$ to its closest element $j \in S$, i.e. $j = \pi_{w,S}(i) = \arg\max_{j' \in S} s_{ij'}(\boldsymbol{W})$, where $s_{ij}(\boldsymbol{W}) = C - \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_i)r_i - \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_j)r_j\|$ is the similarity between gradients of i and j, and $C \geq \max_{ij} s_{ij}(\boldsymbol{W})$ is a constant. Consider a matrix $\boldsymbol{G}_{\pi_{w,S}} \in \mathbb{R}^{n\times m}$, in which every row i contains gradient of $\pi_w(i)$, i.e., $[\boldsymbol{G}_{\pi_{w,S}}]_{i.} = \mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{x}_{\pi_{w,S}(i)})r_{\pi_{w,S}(i)}$. The Frobenius norm of the matrix $\boldsymbol{G}_{\pi_w}$ provides an upper-bound on the error of the weighted subset S in capturing the alignment of the residuals of the full training data with the Jacobian matrix. Formally,

$$\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_{train})\boldsymbol{r}^t_{train} - \boldsymbol{\gamma}_{S^t}\mathcal{J}^T(\boldsymbol{W}^t, [\boldsymbol{X}_{train}]_{.S^t})\boldsymbol{r}_{S^t}\| \leq \|\boldsymbol{G}_{\pi_{w,S}}\|_F, \tag{55}$$

where the weight vector $\boldsymbol{\gamma}_{S^t} \in \mathbb{R}^{|S|}$ contains the number of elements that are mapped to every element $j \in S$ by mapping $\pi_{w,S}$, i.e. $\gamma_j = \sum_{i\in V} \mathbb{1}[\pi_{w,S}(i) = j]$. Hence, the set of training points that closely estimate the projection of the residuals of the full training data on the Jacobian spectrum can be obtained by finding a subset S that minimizes the Frobenius norm of matrix $\boldsymbol{G}_{\pi_{w,S}}$.

B Additional Theoretical Results

B.1 Convergence analysis for training on augmented full data

Theorem B.1. *Gradient descent with learning rate η applied to a neural network with constant NTK and Lipschitz constant L, and data points $\mathcal{D}_{aug}$ augmented with r additive perturbations bounded by ϵ_0 results in the following training dynamics:*

$$\mathbb{E}[\|\boldsymbol{y} - f(\boldsymbol{X}_{aug}, \boldsymbol{W}^t)\|_2] \leq \sqrt{\sum_{i=1}^{n} \left(1 - \eta\left(\sigma_i^2 + \sigma_i(1-2p_i)\|E\| + \frac{\|E\|^2}{3}\right)\right)^{2t} ((\boldsymbol{u}_i\boldsymbol{y})^2 + 2n\sqrt{2}\|E\|/\gamma_0)} \tag{56}$$

where $\boldsymbol{E}$ with $\|\boldsymbol{E}\| \leq \sqrt{n}L\epsilon_0$ is the perturbation to the Jacobian, and $p_i := \mathbb{P}(\tilde{\sigma}_i - \sigma_i < 0)$ is the probability that σ_i decreases as a result of data augmentation.

B.2 Proof of Theorem B.1

Using Jensen's inequality, we have

$$\mathbb{E}\left[\|\boldsymbol{y} - f(\boldsymbol{X}_{aug}, \boldsymbol{W}^t)\|_2\right] \tag{57}$$

$$= \mathbb{E}\left[\sqrt{\sum_{i=1}^{n}(1-\eta\tilde{\lambda}_i)^{2t}(\tilde{\boldsymbol{u}}_i^T\boldsymbol{y})^2} \pm \epsilon\right] \tag{58}$$

$$\leq \sqrt{\mathbb{E}\left[\sum_{i=1}^{n}(1-\eta\tilde{\lambda}_i)^{2t}(\tilde{\boldsymbol{u}}_i^T\boldsymbol{y})^2\right]} \tag{59}$$

$$\leq \sqrt{\sum_{i=1}^{n}\mathbb{E}\left[(1-\eta\tilde{\lambda}_i)^{2t}((\boldsymbol{u}_i\boldsymbol{y})^2 + 2n\sqrt{2}\|E\|/\gamma_0)\right]} \tag{60}$$

$$\leq \sqrt{\sum_{i=1}^{n}(1-\eta\mathbb{E}\left[\tilde{\lambda}_i\right])^{2t}((\boldsymbol{u}_i\boldsymbol{y})^2 + 2n\sqrt{2}\|E\|/\gamma_0)} \tag{61}$$

$$= \sqrt{\sum_{i=1}^{n}\left(1 - \eta\left(\sigma_i^2 + \sigma_i(1-2p_i)\|E\| + \frac{\|E\|^2}{3}\right)\right)^{2t} ((\boldsymbol{u}_i\boldsymbol{y})^2 + 2n\sqrt{2}\|E\|/\gamma_0)} \tag{62}$$

B.3 Convergence analysis for training on the coreset and its augmentation

Theorem B.2. *Let $\mathcal{L}_i$ be β-smooth, $\mathcal{L}$ be λ-smooth and satisfy the α-PL condition, that is for $\alpha > 0$, $\|\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X})\|^2 \geq \alpha\mathcal{L}(\boldsymbol{W}, \boldsymbol{X})$ for all weights $\boldsymbol{W}$. Let ξ upper-bound the normed difference in gradients between the weighted coreset and full dataset. Assume that the network $f(\boldsymbol{W}, \boldsymbol{X})$ is Lipschitz in $\boldsymbol{W}$, $\boldsymbol{X}$ with Lipschitz constant L and L' respectively, and $\bar{L} = \max\{L, L'\}$. Let G_0 the gradient over the full dataset at initialization, $\sigma_{\max}$ the maximum Jacobian singular value at initialization. Choosing perturbation bound $\epsilon_0 \leq \frac{1}{\sigma_{\max}\sqrt{\bar{L}n}}$ where $\sigma_{\max}$ is the maximum singular value of the coreset Jacobian and n is the size of the original dataset, running SGD on the coreset and its augmentation using constant step size $\eta = \frac{\alpha}{\lambda\beta}$, we get the following convergence bound:*

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|] \leq \frac{1}{\sqrt{\alpha}}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{t}{2}}\left(2G_0 + 2\xi + \mathcal{O}\left(\frac{\bar{L}}{\sigma_{\max}}\right)\right), \tag{63}$$

where $\boldsymbol{X}_{c+c_{aug}}$ represents the dataset containing the (weighted) coreset and its augmentation.

Proof. As in the proof for Theorem 5.2, we begin with the following inequality

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2] \leq \left(1-\frac{\alpha\eta}{2}\right)^t \mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}}) \tag{64}$$

$$\leq \frac{1}{\alpha}\left(1-\frac{\alpha\eta}{2}\right)^t \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2 \tag{65}$$

Thus, we can write

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}})\|] \tag{66}$$

$$\leq \sqrt{\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2]} \tag{67}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}})\| \tag{68}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \left(\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_c)\|+\|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c_{\text{aug}}})\|\right) \tag{69}$$

$$\leq \frac{1}{\sqrt{\alpha}}\left(1-\frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \left(G_0+\xi+\|(\mathcal{J}(\boldsymbol{W}^0, \boldsymbol{X}_c)+\boldsymbol{E})(-f(\boldsymbol{W}^0, \boldsymbol{X}_c+\boldsymbol{\epsilon}))\|\right) \tag{70}$$

The rest of the proof is similar to that of Theorem 5.2. □

B.4 Lemma for eigenvalues of coreset

The following Lemma characterizes the sum of eigenvalues of the NTK associated with the coreset.

Lemma B.3. *Let ξ be an upper bound of the normed difference in gradient of the weighted coreset and the original dataset, i.e. for full data $\boldsymbol{X}$ and its corresponding coreset $\boldsymbol{X}_S$ with weights γ_S, and respective residuals $\boldsymbol{r}$, $\boldsymbol{r}_S$, we have the bound $\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \boldsymbol{\gamma}_S\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S^t\| \leq \xi$. Let $\{\lambda_i\}_{i=1}^k$ be the eigenvalues of the NTK associated with the coreset. Then we have that*

$$\sqrt{\sum_{i=1}^k \lambda_i} \geq \frac{|\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t\|-\xi|}{\|\boldsymbol{r}_S^t\|}.$$

Proof. Let singular values of coreset Jacobian be σ_i. Let $\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t = \boldsymbol{\gamma}_S\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S^t+\xi_S$ where $\|\xi_S\| \leq \xi$.

Taking Frobenius norm, we get

$$\|\boldsymbol{\gamma}_S\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S)\boldsymbol{r}_S^t\| = \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \xi_S\| \tag{71}$$

$$\Rightarrow \|\boldsymbol{\gamma}_S\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S)\|\|\boldsymbol{r}_S^t\| \geq \|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \xi_S\| \tag{72}$$

$$\Rightarrow \|\boldsymbol{\gamma}_S\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X}_S)\| \geq \frac{\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \xi_S\|}{\|\boldsymbol{r}_S^t\|} \tag{73}$$

$$\Rightarrow \sqrt{\sum_{i=1}^s \sigma_i^2} \geq \frac{\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \xi_S\|}{\|\boldsymbol{r}_S^t\|} \tag{74}$$

$$\Rightarrow \sqrt{\sum_{i=1}^s \lambda_i} \geq \frac{\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t - \xi_S\|}{\|\boldsymbol{r}_S^t\|} \tag{75}$$

$$\Rightarrow \sqrt{\sum_{i=1}^s \lambda_i} \geq \frac{|\|\mathcal{J}^T(\boldsymbol{W}^t, \boldsymbol{X})\boldsymbol{r}^t\|-\xi|}{\|\boldsymbol{r}_S^t\|} \quad \text{by reverse triangle inequality} \tag{76}$$

□

We can make the following observations: For overparameterized networks, with bounded activation functions and labels, e.g. softmax and one-hot encoding, the norm of the residual vector is bounded,

and the gradient norm is likely to be much larger than residual, especially when dimension of gradient is large. In this case, the Jacobian matrix associated with small weighted coresets found by solving Eq. (9), have large singular values.

B.5 Augmentation as Linear Transformation: Linear Model Analysis

We introduce a simplified linear model to extend our theoretical analysis to augmentations modelled as linear transformation matrices F applied to the original training data. These augmentations are also originally studied by [40]. In this section, we specifically study the effect of these augmentations using a linear model when applied to coresets.

Lemma B.4 (Augmented coreset gradient bounds: Linear)**.** *Let f be a simple linear model with weights $\boldsymbol{W} \in \mathbb{R}^{d\times C}$ where $f(\boldsymbol{W}, \boldsymbol{x}_i) = \boldsymbol{W}^T\boldsymbol{x}_i$, trained on mean squared loss function $\mathcal{L}$. Let $F \in \mathbb{R}^{d\times d}$ be a common linear augmentation matrix with norm $\|F\|$ with augmentation $\boldsymbol{x}_i^{\text{aug}}$ given by $F\boldsymbol{x}_i$. Let coreset be of size k and full dataset be of size n. Further assume that the predicted label of $\boldsymbol{x}_i$ and its augmentation $F\boldsymbol{x}_i$ are sufficiently close, i.e. there exists ω such that $\boldsymbol{W}^T(F\boldsymbol{x}_i)$ = $\boldsymbol{W}^T\boldsymbol{x}_i + z_i$, $\|z_i\| \leq \omega\ \forall i$. Let ξ upper-bound the normed difference in gradients between the weighted coreset and full dataset. Then, the normed difference between the gradient of the augmented full data and augmented coreset is given by*

$$\|\sum_{i\in V}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_i^{\text{aug}}) - \sum_{j=1}^{k}\gamma_{s_j}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_{s_j}^{\text{aug}})\| \leq \|F\|(\xi + \sqrt{d}n\omega)$$

for some (small) constant ξ.

Proof. By our assumption, we can begin with,

$$\|\sum_{i\in V}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_i) - \sum_{j=1}^{k}\gamma_{s_j}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_{s_j})\| \leq \xi \tag{77}$$

Furthermore, by [29], we know that sum of the coreset weights γ_{s_j} is given by $\sum_{j=1}^{k=1}\gamma_{s_j} \leq n$.

Hence,

$$\|\sum_{i\in V}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_i^{\text{aug}}) - \sum_{j=1}^{k}\gamma_{s_j}\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{x}_{s_j}^{\text{aug}})\| \tag{78}$$

$$= \|\sum_{i\in V}(\mathcal{J}(\boldsymbol{W}, \boldsymbol{x}_i^{\text{aug}}))^T[\boldsymbol{W}^T(F\boldsymbol{x}_i) - y_i] - \sum_{j=1}^{k}\gamma_{s_j}(\mathcal{J}(\boldsymbol{W}, \boldsymbol{x}_{s_j}^{\text{aug}}))^T[\boldsymbol{W}^T(F\boldsymbol{x}_{s_j}) - y_{s_j}]\| \tag{79}$$

$$= \|\sum_{i\in V}F\boldsymbol{x}_i[\boldsymbol{W}^T(F\boldsymbol{x}_i) - y_i] - \sum_{j=1}^{k}\gamma_{s_j}F\boldsymbol{x}_{s_j}[\boldsymbol{W}^T(F\boldsymbol{x}_{s_j}) - y_{s_j}]\| \tag{80}$$

$$= \|F\sum_{i\in V}\boldsymbol{x}_i(\boldsymbol{W}^T\boldsymbol{x}_i - y_i) - F\sum_{j=1}^{k}\gamma_{s_j}\boldsymbol{x}_{s_j}(\boldsymbol{W}^T\boldsymbol{x}_{s_j} + z_i - y_{s_j})\| \tag{81}$$

$$= \|F\sum_{i\in V}\nabla L(\boldsymbol{W}, \boldsymbol{x}_i) - F\sum_{j=1}^{k}\gamma_{s_j}\nabla L(\boldsymbol{W}, \boldsymbol{x}_{s_j}) - F\sum_{j=1}^{k}\gamma_{s_j}\boldsymbol{x}_{s_j}z_{s_j}\| \tag{82}$$

$$\leq \|F\|\|\sum_{i\in V}\nabla L(\boldsymbol{W}, \boldsymbol{x}_i) - \sum_{j=1}^{k}\gamma_{s_j}\nabla L(\boldsymbol{W}, \boldsymbol{x}_{s_j})\| + \|F\|\|\sum_{j=1}^{k}\gamma_{s_j}\boldsymbol{x}_{s_j}z_{s_j}\| \tag{83}$$

$$\leq \|F\|\xi + \sqrt{d}\|F\|n\omega \tag{84}$$

$$= \|F\|(\xi + \sqrt{d}n\omega) \tag{85}$$

□

Corollary B.5. *In the simplified linear case above, the difference in gradients of the full training data with its augmentations ($\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{f+aug})$) and gradients of the coreset with its augmentations ($\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c+c_{aug}})$) can be bounded by*

$$\|\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{f+aug}) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c+c_{aug}})\| \leq (\|F\|+1)\xi + \sqrt{d}\|F\|n\omega$$

Proof. Applying Eq. (77) and Lemma B.4, we obtain

$$\begin{aligned}
&\|\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{f+aug}) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c+c_{aug}})\| && (86)\\
&= \|(\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_f) + \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{aug})) - (\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_c) + \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c_{aug}}))\| && (87)\\
&= \|(\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_f) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_c)) + (\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{aug}) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c_{aug}}))\| && (88)\\
&\leq \|(\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_f) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_c))\| + \|(\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{aug}) - \nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_{c_{aug}}))\| && (89)\\
&\leq \xi + \|F\|(\xi + \sqrt{d}n\omega) && (90)\\
&= (\|F\|+1)\xi + \sqrt{d}\|F\|n\omega && (91)
\end{aligned}$$

□

Theorem B.6 (Convergence of linear model)**.** *Let f be a linear model with weights $\boldsymbol{W}$ and augmentation be represented by the common linear transformation F. Let $\mathcal{L}_i$ be β-smooth, $\mathcal{L}$ be λ-smooth and satisfy the α-PL condition, that is for $\alpha > 0$, $\|\nabla\mathcal{L}(\boldsymbol{W}, \boldsymbol{X})\|^2 \geq \alpha\mathcal{L}(\boldsymbol{W}, \boldsymbol{X})$ for all weights $\boldsymbol{W}$. Let ξ upper-bound the normed difference in gradients between the weighted coreset and full dataset and ω bound $\boldsymbol{W}^T(F\boldsymbol{x}_i) = \boldsymbol{W}^T\boldsymbol{x}_i + z_i$, $\|z_i\| \leq \omega\ \forall i$. Let G_0' be the gradient over the full dataset and its augmentations at initialization. Then, running SGD on the size k coreset with its augmentation using constant step size $\eta = \frac{\alpha}{\lambda\beta}$, we get the following convergence bound:*

$$\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|] \leq \frac{1}{\sqrt{\alpha}}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{t}{2}}\left(G_0' + (\|F\|+1)\xi + \sqrt{d}\|F\|n\omega\right)$$

Proof. From [3], we have

$$\begin{aligned}
\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2] &\leq \left(1 - \frac{\alpha\eta}{2}\right)^t \mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}}) && (92)\\
&\leq \frac{1}{\alpha}\left(1 - \frac{\alpha\eta}{2}\right)^t \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2 && (93)\\
& && (94)
\end{aligned}$$

Using Jensen's inequality, we have

$$\begin{aligned}
&\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|] && (95)\\
&\leq \sqrt{\mathbb{E}[\|\nabla\mathcal{L}(\boldsymbol{W}^t, \boldsymbol{X}_{c+c_{\text{aug}}})\|^2]} && (96)\\
&\leq \frac{1}{\sqrt{\alpha}}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{t}{2}} \|\nabla\mathcal{L}(\boldsymbol{W}^0, \boldsymbol{X}_{c+c_{\text{aug}}})\| && (97)\\
&\leq \frac{1}{\sqrt{\alpha}}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{t}{2}}\left(G_0' + (\|F\|+1)\xi + \sqrt{d}\|F\|n\omega\right) && (98)
\end{aligned}$$

where the last inequality follows from applying Corollary B.5. □

C Singular spectrum analysis

C.1 Experiment details

We generate singular spectrum plots for both MNIST and CIFAR10 datasets in Figures 1 and 3. Due to the computational infeasbility of computing the network Jacobian for the full datasets in deep network settings, we instead construct and use a reduced version of these datasets by uniformly select 900 images from the first 3 classes. For our experiments on MNIST, we pretrain a MLP model with 1

hidden layer for 15 epochs. For our experiments on CIFAR10, we pretrain a ResNet20 model for 15 epochs. We then compute the singular spectrums for augmented and non-augmented data based on these pretrained networks.

Since it is difficult to perform a one-to-one matching of singular values produced from augmented and non-augmented datasets, we instead bin our singular values into 30 separate and uniformly distributed bins each containing the same number of singular values. To measure perturbation to singular values resulted from augmentation, we compute the mean difference between each bin. On the other hand, to measure perturbation to singular vectors, we compute mean subspace angle between the singular subspace spanned by singular vectors in each bin.

C.2 Real-world strong augmentations

We study the effects of real-world, unbounded augmentations on the singular spectrum of the network Jacobian. In particular, in additional to the plots in the main paper, we show the effect of strong augmentations through (1) random rotation (up to $30\circ$ and AutoAugment [7] for MNIST and (2) random horizontal flips/random crops and AutoAugment for CIFAR10. The policies implemented by AutoAugment include translations, shearing, as well as contrast and brightness transforms. We study the effects of these augmentations on the singular spectrum in Figure 3. Despite these augmentations being unbounded transformations, we note that the results of our theory still holds. In particular, it can be observed that data augmentation increases smaller singular values relatively more with a higher probability. On the other hand, data augmentation affects the prominent singular vectors of the Jacobian to a smaller extent, and preserves the prominent directions. As such, our argument empirically extends to real-world, unbounded label-invariant transformations characteristic of strong augmentations.

D Experiment Setup and Additional Experiments

D.1 Experiment setup

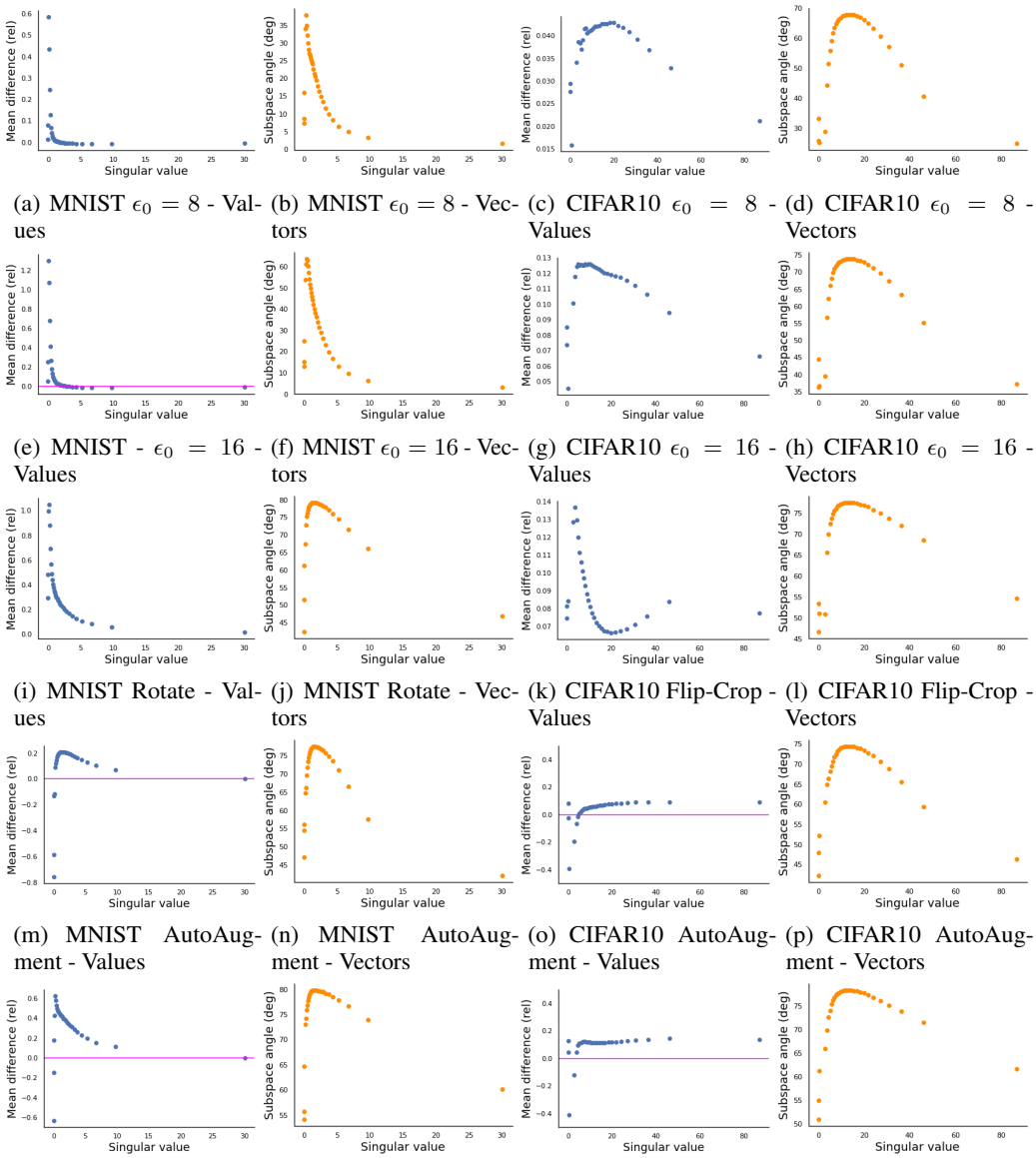
For all experiments, we train using SGD with 0.9 momentum and learning rate decay. For experiments on CIFAR10 and variants/ResNet20, we train for 200 epochs, for Caltech256 (ImageNet pretrained)/ ResNet18, we trained for 40 epochs starting at learning rate 0.001 and batch size 64. We also report results for Caltech256 without ImageNet pretraining in Sec. D.8, where we train for 400 epochs to ensure convergence with a starting learning rate of 0.05 and batch size 64. For experiments on ImageNet/ResNet50 and TinyImageNet/ResNet50, we use the standard 90 epoch learning schedule starting at learning rate of 0.1 and batch size 64.

Data and augmentation. We apply our method to training ResNet20 and Wide-ResNet-28-10 on CIFAR10, and ResNet32 on CIFAR10-IMB (Long-Tailed CIFAR10 with Imbalance factor of 100 following [19]) and SVHN datasets. We train Caltech256 [13] on ImageNet-pretrained ResNet18, and include experiments with random initialization in Appendix D. TinyImageNet and ImageNet are trained on with ResNet50. We use [40] for CIFAR10/SVHN, and AutoAugment [7] for Caltech256, TinyImageNet, and ImageNet as the strong augmentation method. Note that we append strong augmentations rather than apply them in-place, which we show to be more effective in Appendix D. All results are averaged over 5 runs using an Nvidia A40 GPU.

D.2 Experiment setup

For all experiments, we train using SGD with 0.9 momentum and learning rate decay. We also set weight decay as For experiments on CIFAR10 and variants/ResNet20, we train for 200 epochs, for Caltech256 (ImageNet pretrained)/ ResNet18, we trained for 40 epochs starting at learning rate 0.001 and batch size 64. We also report results for Caltech256 without ImageNet pretraining in Sec. D.8, where we train for 400 epochs to ensure convergence with a starting learning rate of 0.05 and batch size 64. For experiments on ImageNet/ResNet50 and TinyImageNet/ResNet50, we use the standard 90 epoch learning schedule starting at learning rate of 0.1 and batch size 64.

Data and augmentation. We apply our method to training ResNet20 and Wide-ResNet-28-10 on CIFAR10, and ResNet32 on CIFAR10-IMB (Long-Tailed CIFAR10 with Imbalance factor of 100 following [19]) and SVHN datasets. We train Caltech256 [13] on ImageNet-pretrained ResNet18,



(a) MNIST $\epsilon_0 = 8$ - Values (b) MNIST $\epsilon_0 = 8$ - Vectors (c) CIFAR10 $\epsilon_0 = 8$ - Values (d) CIFAR10 $\epsilon_0 = 8$ - Vectors

(e) MNIST - $\epsilon_0 = 16$ - Values (f) MNIST $\epsilon_0 = 16$ - Vectors (g) CIFAR10 $\epsilon_0 = 16$ - Values (h) CIFAR10 $\epsilon_0 = 16$ - Vectors

(i) MNIST Rotate - Values (j) MNIST Rotate - Vectors (k) CIFAR10 Flip-Crop - Values (l) CIFAR10 Flip-Crop - Vectors

(m) MNIST AutoAugment - Values (n) MNIST AutoAugment - Vectors (o) CIFAR10 AutoAugment - Values (p) CIFAR10 AutoAugment - Vectors

(q) MNIST Rotate + AutoAugment - Values (r) MNIST Rotate + AutoAugment - Vectors (s) CIFAR10 Flip + Crop + AutoAugment - Values (t) CIFAR10 Flip + Crop + AutoAugment - Vectors

Figure 3: Difference in mean singular values (Cols 1 & 3) between augmented and non-augmented data and mean angular difference (Cols 2 & 4) between subspaces spanned by singular vectors for augmented and non-augmented data.

and include experiments with random initialization in Appendix D. TinyImageNet and ImageNet are trained on with ResNet50. We use [40] for CIFAR10/SVHN, and AutoAugment [7] for Caltech256, TinyImageNet, and ImageNet as the strong augmentation method. Note that we append strong augmentations rather than apply them in-place, which we show to be more effective in Appendix D. All results are averaged over 5 runs using an Nvidia A40 GPU.

D.3 Full Results for Table 4

This section contains full experiment results including standard deviations and the full augmentation benchmark for Table 4. Augmenting coresets of size 10% achieves 51% of the improvement obtained from augmentation of the full data and further enjoys a 6x speedup in training time on CIFAR10.

This speedup becomes more significant when using strong augmentation techniques which are mostly computationally demanding, especially when applied to the entire dataset.

Table 7: Supplementary table for Table 4 - Test accuracy on CIFAR10 + ResNet20, SVHN + ResNet32, CIFAR10-Imbalanced + ResNet32 including standard deviation errors and full dataset augmentation accuracy.

Method	Size	CIFAR10	CIFAR10-IMB	SVHN
None	0%	$89.46 \pm 0.17\%$	$87.08 \pm 0.50\%$	$95.676 \pm 0.108\%$
Random	5%	$90.34 \pm 0.18\%$	$88.48 \pm 0.25\%$	$95.760 \pm 0.084\%$
	10%	$91.07 \pm 0.13\%$	$89.52 \pm 0.15\%$	$96.187 \pm 0.112\%$
	30%	$92.11 \pm 0.12\%$	$91.11 \pm 0.18\%$	$96.569 \pm 0.073\%$
Max-Loss	5%	$90.79 \pm 0.19\%$	$88.77 \pm 0.35\%$	$\mathbf{96.165 \pm 0.108}\%$
	10%	$91.39 \pm 0.08\%$	$89.22 \pm 0.48\%$	$\mathbf{96.370 \pm 0.076}\%$
	30%	$92.43 \pm 0.07\%$	$91.11 \pm 0.25\%$	$96.735 \pm 0.068\%$
Coreset	5%	$\mathbf{90.87 \pm 0.05}\%$	$\mathbf{89.10 \pm 0.41}\%$	$96.121 \pm 0.055\%$
	10%	$\mathbf{91.54 \pm 0.19}\%$	$\mathbf{89.75 \pm 0.52}\%$	$96.354 \pm 0.091\%$
	30%	$\mathbf{92.49 \pm 0.15}\%$	$\mathbf{91.12 \pm 0.26}\%$	$\mathbf{96.791 \pm 0.051}\%$
All	100%	$93.50 \pm 0.25\%$	$92.48 \pm 0.34\%$	$97.068 \pm 0.030\%$

D.4 Supplementary results for Tab. 1

Table 8: Supplementary results for Tab. 1. Training ResNet20 (R20) and WideResnet-28-10 (W2810) on CIFAR10 (C10) using small subsets, and ResNet18 (R18) on Caltech256 (Cal).

Model/Dataset	Subset	Random		Ours	
		Weak Aug.	Strong Aug.	Weak Aug.	Strong Aug.
C10/R20	0.1% (5)	31.7 ± 3.2	33.5 ± 2.7	29.6 ± 3.8	$\mathbf{37.8 \pm 4.5}$
	0.2% (10)	35.9 ± 2.1	42.7 ± 3.9	33.6 ± 3.2	$\mathbf{45.1 \pm 2.3}$
	0.5% (25)	51.1 ± 2.3	58.7 ± 1.3	55.8 ± 3.1	$\mathbf{63.9 \pm 2.1}$
	1% (50)	66.2 ± 1.0	74.4 ± 0.8	65.9 ± 4.0	$\mathbf{74.7 \pm 1.1}$
C10/W2810	1% (50)	61.3 ± 2.4	57.7 ± 0.8	59.9 ± 2.4	$\mathbf{62.1 \pm 3.1}$
Cal/R18	5% (3)	24.8 ± 1.5	41.5 ± 0.5	33.8 ± 1.7	$\mathbf{52.7 \pm 1.2}$
	10% (6)	49.5 ± 0.6	61.8 ± 0.8	55.7 ± 0.3	$\mathbf{65.4 \pm 0.3}$
	20% (12)	66.6 ± 0.2	72.5 ± 0.1	67.5 ± 0.3	$\mathbf{73.1 \pm 0.1}$
	30% (18)	72.0 ± 0.1	75.7 ± 0.2	71.9 ± 0.2	$\mathbf{76.3 \pm 0.2}$
	40% (24)	74.6 ± 0.3	77.6 ± 0.4	74.2 ± 0.4	$\mathbf{77.7 \pm 0.5}$
	50% (30)	76.1 ± 0.5	78.5 ± 0.3	76.1 ± 0.1	$\mathbf{78.9 \pm 0.2}$

D.5 Training dynamics vs generalization

Figure 4 demonstrates the relationship between training loss and validation accuracy resulted from data augmentation. While training loss of augmented datasets do not decrease as quickly as non-augmented datasets, generalization performance (shown by val. acc.) improves.

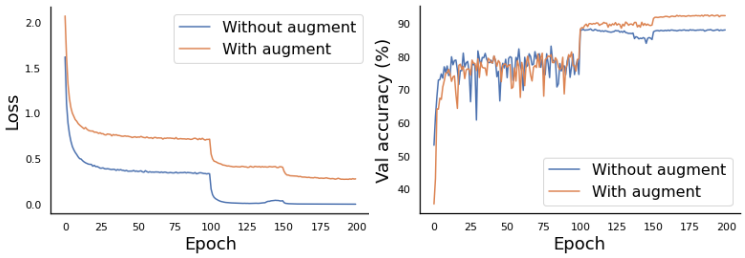


Figure 4: Training loss vs validation accuracy of CIFAR10/ResNet20 using AutoAugment.

D.6 Augmentations applied through appending vs in-place

Our experiments on Caltech256/ResNet18/AutoAugment (R=5) show that even for cheaper strong augmentation methods (AutoAugment), while in-place augmentation may decrease the performance, appending Random (R) and Coresets (C) augmentations (Append) outperforms in-place augmentation of 2x data points (In-place 2x) for various subset sizes.

Table 9: Caltech256/AutoAugment in-place vs. appending for Caltech256.

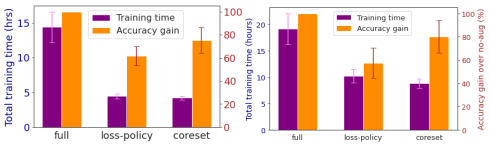
	No Aug.	In-place	In-place (2x)	Append
C5%	33.8%	26.4%	48.2%	**52.7%**
R5%	24.8%	17.4%	40.2%	**41.5%**
C10%	55.7%	48.2%	62.8%	**65.4%**
R10%	50.6%	40.2%	**62.0%**	61.8%
C30%	71.9%	68.8%	74.9%	**76.3%**
R30%	72.0%	68.7%	75.1%	**75.7%**

D.7 Speed-up measurements

We measure the improvement in training time in the case of training on full data and augmenting subsets of various sizes. While our method yields similar or slightly lower speed-up to the max-loss policy and random approach respectively, our resulting accuracy outperforms these two approaches. We show this in Fig. D.7. For example, for SVHN/Resnet32 using 30% coresets, we sacrifice 11% of the speed-up to obtain an additional 24.8% of the gain in accuracy from full data augmentation when compared to a random subset of the same size. We show the speed-up obtained for our method and various subset sizes in Tab. 10, and provide wall-clock times for our method in Tab. D.7.

Table 10: Speedup on CIFAR10 + ResNet20 (C10/R20), SVHN + ResNet32 (SVHN/R32).

Dataset	Full Aug.	Ours						Max loss.	Random.
	100%	5%	10%	15%	20%	25%	30%	30%	30%
C10 / R20	1x	7.93x	6.31x	4.46x	4.27x	3.41x	3.43x	3.48x	4.03x
SVHN / R32	1x	5.35x	3.93x	3.40x	2.80x	2.49x	2.18x	2.21x	2.43x



(a) CIFAR10/ResNet20 (b) SVHN/Resnet32

Figure 5: Speedup/Accuracy of augmenting 30% coresets compared to original max-loss policy for (a) ResNet20 trained on CIFAR10 and (b) ResNet32 trained on SVHN.

Table 11: Wall-clock times to find various sized coresets from *all classes* of Caltech256 and TinyImagene at 1 epoch. Note, training ResNet20/CIFAR10 with [40] takes 14.4 hrs. In practice, coresets can be found in parallel (p threads) from different classes, and selection happens every $R = 5-15$ epochs. Hence, the numbers divide by $p \times R$.

Caltech256			TinyImageNet		
10%	30%	50%	10%	30%	50%
10.50s	10.52s	10.53s	7.85s	8.09s	8.17s

D.8 End to end training on Caltech256

As Caltech256 contains many classes and higher resolution images, training on smaller subset without pretraining has a low accuracy. Thus, many works (e.g. Achille et al., 2020) finetune from ImageNet pretrained initialization. However, we show that our results still hold even when training form scratch. We demonstrate our results in Tab. 12, where we train Caltech256 on ResNet50 without pretraining for 400 epochs, and with $R = 40$, where our method consistently outperfoms random subsets for multiple subset sizes (5%, 10%, 30%, 50%).

Table 12: Caltech256 (w/o pretraining) /ResNet50, 400 epochs, $R = 40$

Random				Ours			
5%	10%	30%	50%	5%	10%	30%	50%
17.26	35.38	58.2	64.67	**20.58**	**38.20**	**60.30**	**65.17**

D.9 Training on full data and augmenting small subsets re-selected every epoch

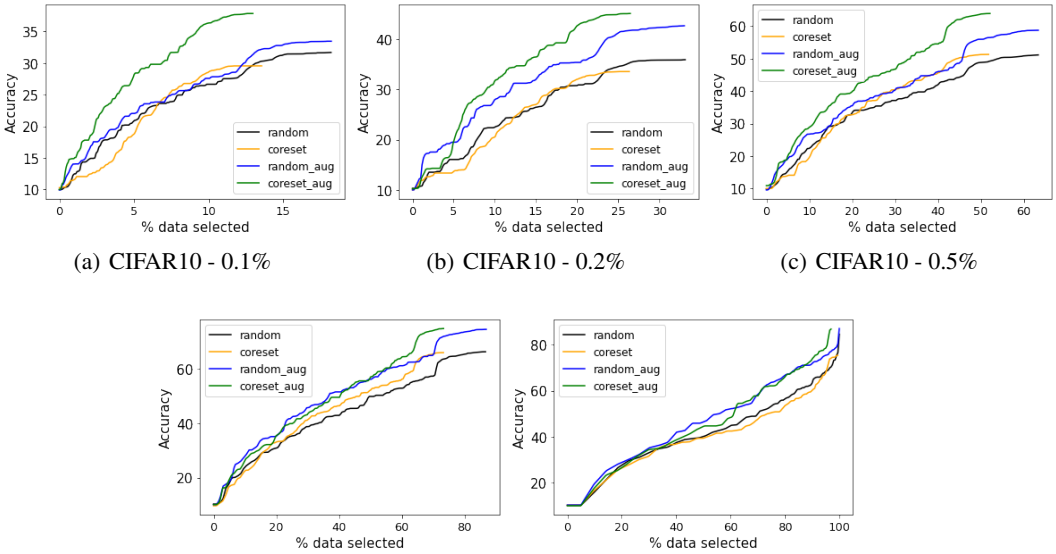
We apply our proposed method to select a new subset for augmentation every epoch (i.e. using $R = 1$) and compare our results with other approaches using accuracy and percentage of data not selected (NS). We see that while the max-loss policy selects a small fraction of data points over and over and random uniformly selects all the data points, our approach successfully finds the smallest subset of data points that are the most crucial for data augmentation. Hence, it can achieve a superior accuracy than max-loss policy, while augmenting only slightly more examples. This confirms the data-efficiency of our approach. This is especially evident when using coresets of size 0.2%. Furthermore, despite the random baseline using a significantly larger percentage of data, it is outperformed by our approach in both data-efficiency and accuracy. We emphasize that results in this table is different from that of Table 7, as default augmentations on the full training data are performed once every $R = 1$ epochs instead of every $R = 20$ epochs. Since selecting subsets at every epoch can be computationally expensive, we only perform these experiments on small coresets and hence still enjoy good speedups compared to full data augmentation. This shows that our approach is still effective at very small subset sizes, hence can be computationally efficient even when subsets are re-selected every epoch.

Table 13: Training on full data and selecting a new subset for augmentation every epoch ($R = 1$).

Subset	Random		Max-loss Policy		Ours	
	Acc	NS (%)	Acc	NS (%)	Acc	NS (%)
0%	91.96 ± 0.12	–	91.96 ± 0.12	–	91.96 ± 0.12	–
0.2%	92.22 ± 0.22	67.03 ± 0.04	91.94 ± 0.12	86.70 ± 0.15	$\mathbf{92.26} \pm 0.13$	79.19 ± 1.10
0.5%	92.06 ± 0.17	36.70 ± 0.18	92.20 ± 0.13	76.80 ± 0.31	$\mathbf{92.27} \pm 0.08$	63.23 ± 0.35

D.10 Additional visualizations for training on coresets and its augmentations - Measuring training dynamics over time

We include additional visualizations in Figure 6 for training on coresets and its augmentations as supplementary plots to Figure 9(c) and Table 1. We plot metrics obtained during each point (epoch)



(a) CIFAR10 - 0.1% (b) CIFAR10 - 0.2% (c) CIFAR10 - 0.5%

(d) CIFAR10 - 1% (e) CIFAR10 - 5%

Figure 6: Supplementary plots for Figure 9(c): Training on coreset and its augmentation compared to random baseline, measured using test accuracy against percentage of data used on CIFAR10 dataset across various subset sizes. Accuracy and percentage of data used are measured at every epoch and averaged over 5 runs.

of the training process based on percentage of data selected/used and test accuracy achieved. All metrics are averaged over 5 runs and obtained using $R = 1$. These plots demonstrate that coreset augmentation approaches outperform random augmentation baselines throughout the training process. Furthermore, they show that augmentation of coresets result in a larger increase in test accuracy compared to augmentation of randomly selected training examples, especially for small subset sizes.

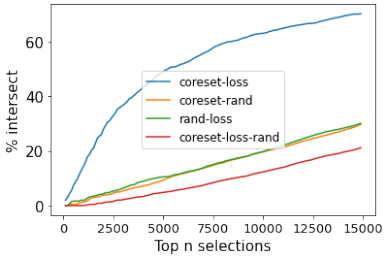


Figure 7: Intersection between max-loss and coresets in the top N points selected aggregated across the entire training process. Here, we show the increasing overlap between max-loss and coreset points as N grows.

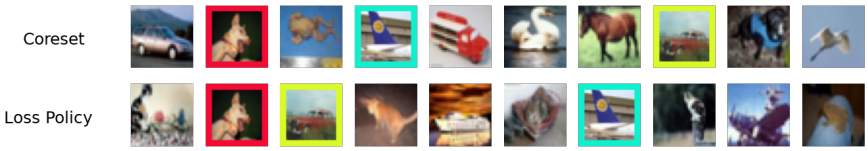
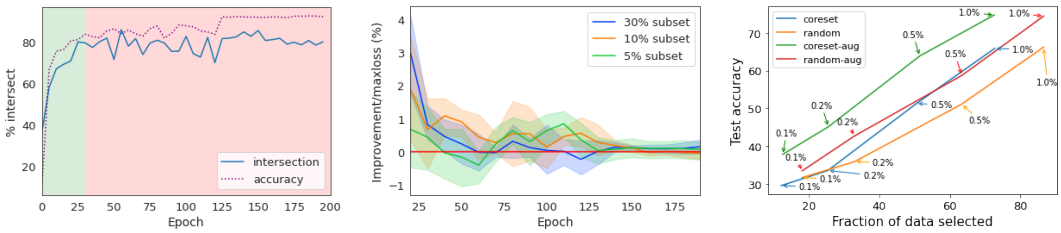


Figure 8: Qualitative evaluation of coreset and max-loss points.

D.11 Intersection of max-loss policy and coresets

Figure $9(a)$ depicts the increase in intersection between max-loss subsets and coresets over time. In addition, we also aggregate 30% subsets selected every $R = 20$ epochs using both approaches over the entire training process to compute intersection between the top N selected data points. Our plots



(a) Loss intersection (b) Improvement (c) Coresets vs random

Figure 9: Training ResNet20 on full data and augmented coresets extracted from CIFAR10. (a) Intersection between elements of coresets of size 30% and maximum loss subsets of the same size. The intersection increases after the initial phase of training, (b) Accuracy improvement for training on full data and augmented coresets over training on full data and max-loss augmentation. (c) Accuracy vs. fraction of data selected for augmentation during training Resnet20 on CIFAR10.

in Figure 7 suggest that a similar pattern holds in this setting. We also qualitatively visualize this in Figure 8.