Compression and Curriculum Strategies for Efficient Learning in Deep Neural Networks

by

Madan Ravi Ganesh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in The University of Michigan
2022

Doctoral Committee:

Professor Jason J. Corso, Co-Chair,
Assistant Professor Andrew Owens, Co-Chair,
Assistant Professor David Fouhey,
Professor Jessy W. Grizzle,
Assistant Professor Salimeh Yasaei Sekeh, University of Maine

Madan Ravi Ganesh

madantrg@umich.edu

ORCID iD: 0000-0003-1708-5509

DEDICATION

This dissertation would not exist without the help of those who supported, guided, and taught me how to persevere. First, I would like to thank Prof. Manikantan K., for giving me a chance to undertake research as an undergraduate student and planting the seed of research in my mind, which has grown into a lifelong passion. Next, I want to thank Prof. Jason Corso and Prof. Salimeh Yasaei Sekeh for taking a chance on me when I was still an unknown quantity and sticking with me through the years. In addition, I would like to thank all the friends and brothers I made during my undergraduate and graduate school years who have supported me and shown me love and grace through various stages of life. Thank you to Akansha Paramesh for breaking through my guard and giving me support and guidance during the darkest of times in my life. To my mother, sister, brother-in-law, wonderful nephew, and Sharmi: Thank you for being my rock. Last but not least, I love you dad, and I know you are here with us always.

ACKNOWLEDGEMENTS

Thank yo to the committee members-Prof. Jason J. Corso, Prof. Andrew Owens, Prof. Salimeh Yasaei Sekeh, Prof. David Fouhey, and Prof. Jessy Grizzle-for providing advice and constructive feedback throughout the dissertation process.

The MINT project was supported by NSF IIS 1522904, NIST 60NANB17D191, and NSF 1920908. It reflects the opinions and conclusions of its authors and does not represent any position of the funding bodies.

The SNACS project was supported by a Google Faculty Research Award, NIST 60NANB17D191, NSF 1920908 and 2053480. The findings in this project are those of the authors only and do not represent the position of the funding bodies.

The work in LILAC was supported by NSF NRI IIS 1522904 and NIST 60NANB17D191. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

The research underlying CAPER was supported by NSF CI-NEW: Collaborative Research: COVE, NSF CAREER 2144960, and NSF DMS 2053480. It reflects the opinions and conclusions of its authors, but not necessarily the funding agents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The lifecycle of a deep learning application consists of five phases: Data collection, Architecture prototyping, Training, Analysis, and Deployment. There is a significant cost—both human and computational—in all phases of this life cycle. Given the increasing dominance of deep learning across industry and commerce, reducing these costs while maintaining high performance would have a significant impact. To that end, this work focuses on Architecture prototyping and Training, and proposes new techniques that improve their efficiency, by reducing the number of Floating Point Operations (FLOPs), and performance.

Prototyping deep neural networks (DNNs) for hardware-constrained environments is done either manually, through architecture search, or pruning. Manual and architecture search algorithms require long processing times and large-scale resources to obtain optimal solutions, which limit their usability. While pruning algorithms operate more efficiently than previous approaches, they are not effective at modeling the uncertainty in information flow between layers and their downstream impact when pruning. In Chapters 3 and 4, we propose a single-shot model pruning approach that uses a probabilistic framework to model the uncertainty and decrease the redundancy in information passed between layers.

Within our framework, we use conditional mutual information (CMI) to measure the strength of contributions between filters in adjacent layers. In addition, we incorporate information from the weight matrices to balance the contributions from CMI, computed from the activations. Further, we tackle the practical challenges built into pruning pipelines like, the time complexity to determine the upper pruning limit or sensitivity for each layer of the DNN. Our main takeaway is a state-of-the-art single-shot model pruning pipeline, which has a performance of 72.60% on ResNet50-ILSVRC2012 with a sparsity of 68.93%. Overall, our pruning approach reduces the number of FLOPs computed during inference by 51.52%.

The second phase we focus on is Training, which scales its time and resource consumption based on factors like dataset, epochs, and many others. Several algorithms like low-precision computations and distributed training focus on making training more efficient. However, they require either a large number of computational resources or rely on approximations that do not fully match the performance of their original counterparts. Instead, we follow the curriculum learning paradigm, which regulates the DNN-Dataset interaction from the data side to improve performance while

simultaneously affecting computational load and other properties of training.

In Chapters 5 and 6, our primary focus is obtaining high-performing solutions with minimal modifications. Then, we expand our goals to include improved efficiency and adversarial robustness—important traits for real-world deployment. To concurrently tackle such interconnected goals, we introduce a feature-based curriculum in Chapter 6 that uses the difference in activation values, between the original and noise-perturbed inputs, to identify and remove samples susceptible to attacks. By comparing our curriculum against standard and adversarial training regimes we highlight how our curriculum improves performance in both categories. Overall, our curriculum-based approach to Training reduces 63.8 TFLOPs.

By proposing techniques that target the prototyping and training phases of the DNN lifecycle, we reduce the number of computations performed, and thereby the burden imposed by their repeated use when developing DNN-based solutions. By imposing multiple constraints during their development and training, we enable shorter and more resource-friendly development of DNNs; we also ensure the addition of robustness using an orthogonal perspective to traditional adversarial training that doesn't compromise the performance of DNNs.

CHAPTER 1

Introduction

1.1 Motivation

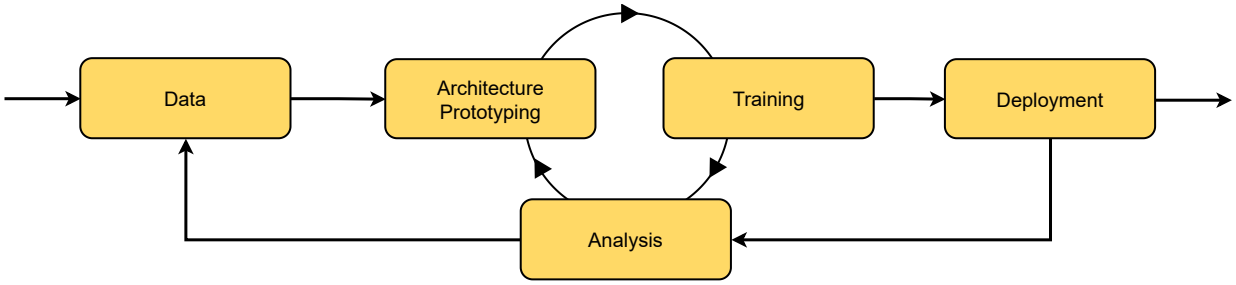


Figure 1.1: Illustration of the five phases of a DNN solution's development life cycle, 1) Data collection, 2) Architecture Prototyping, 3) Training, 4) Analysis, and 5) Deployment. These phases are designed to constantly provide feedback to each other to iteratively refine and obtain a solution.

The field of deep learning has received a lot of engagement in recent years, primarily due to the ever-expanding capabilities of deep neural networks (DNNs). Across domains like object recognition [1], detection [2], face recognition [3], [4], geolocation using photos [5], lip-reading [6] and many more, DNNs constantly push the boundaries of performance that can be achieved and, in doing so, improve over human-level performance. Inspired by the potential of DNNs to solve real-world problems by directly learning patterns from data, businesses are actively pursuing DNN-based solutions to help optimize their workflow, the services they offer and improve the performance of their products. The continuous increase in investment in AI [7], [8] and the growing number of executives who have reported a decrease in costs and an increase in revenue [9] when incorporating AI further reinforce the burgeoning demands for AI in the industry.

With every business problem comes unique requirements, constraints, and domain-specific information, which necessitates context-specific solutions. While adopting a solution across similar domains is convenient, transferring knowledge across starkly different applications is difficult. For example, consider the limitations of traditional deep learning models when transferring to unique domains like medical and hyper-spectral imaging or alternative tasks within a domain [10]–[12].

The challenge of varied priors, constraints, data types, and underlying physical systems is difficult to overcome when transferring DNN solutions. Thus, we need to develop custom deep learning solutions to satiate the demands of industrial problems. This, in turn, brings a sharp focus on understanding how to develop DNN solutions and what are its underlying processes.

In general, there are five phases in the development life cycle of a DNN solution as shown in Fig. 1.1, 1) Data, 2) Architecture Prototyping, 3) Training, 4) Analysis, and 5) Deployment. The Data and Deployment phases act as the entry and exit points to the entire development life cycle, with the Architecture Prototyping, Training, and Analysis phases forming the core. Typically, the development process begins with the Data phase, which includes the collection, preparation, and processing of relevant data. The difficulty associated with this phase, measured by the quality and quantity of data needed, is dictated by the real-world context where the solution will be placed. The process of prototyping novel architectures needs to account for multiple external factors such as available time budget, deployment cost, and resource availability, to name a few. Training imparts desired properties to a model, and Analysis highlights the flaws and improvements across all of the life cycle's phases. Finally, the deployment phase includes the supporting infrastructure, maintenance, and transfer of the designed solution to hardware, which then can be employed to solve real-world problems.

The tightly coupled modules of Architecture Prototyping and Training are simultaneously fundamental to the entire life cycle while offering some of the largest bottlenecks in terms of the amount of resources, cost, and time consumed. These phases offer a straightforward entry point to improving the development life cycle while other phases like data collection, analysis, and deployment are often labor intensive and less approachable. An improvement in the efficiency of the aforementioned core phases will have large implications for the operational requirements of developing a DNN as well as the final model itself.

1.2 Efficiency: The Guiding Principle

This dissertation focuses on improving the efficiency of Training and Architecture Prototyping by reducing the number of FLOPs consumed, in each phase as well as the solution, while maintaining a high level of performance. Reducing the number of FLOPs in each phase is important since the process of designing and obtaining a DNN solution requires repeated execution of the core phases of the developmental life cycle. Often, a typical development life cycle for a single machine learning model is 6-18 months [13], [14], with some requiring longer investments in labor, time, and money. In addition, when placed in the real world, the outcome of such dedicated developmental life cycles is constrained to perform well on limited resources, often with real-time inference as a pre-requisite. Examples of limited resources include restrictions on the total energy available,

memory budget, low-precision computations, and a lower number of computations to improve inference time. Improving the efficiency on both the above-mentioned fronts leads to a shorter development process and a lower footprint solution that is efficient in its execution, thereby making the entire process and outcome resource and time friendly.

In this dissertation, we modify each phase individually to highlight their contributions to the overall efficiency gain. When prototyping architectures, we explore the notion of using a single pass to adapt and sparsify existing DNN architectures to match user or hardware constraints. Doing so offers a reprieve from developing efficient architectures from scratch, which is a significantly more time process. In addition, using a single pruning pass to generate sparse networks minimizes computational overheads. From the dataset and training perspective, we explore the notion of using a more optimal subset of the original training data to reduce the overall computational load and training time while maintaining a high level of performance. We formally introduce each of these approaches and describe them further below.

1.2.1 Architecture Prototyping

In general, prototyping DNN architectures is a highly iterative process that uses the feedback from training, among other phases, to inform its evolution. Manual approaches to this process often embed novel ideas and expand on insights from existing work [15]–[21]. However, an important caveat of their development process is that they are time and labor-intensive. Optimization-based architecture search approaches [22]–[25] offer a reprieve from the manual labor involved in prototyping new architectures by defining constraints in a broad search space within which algorithms are allowed to find an optimal DNN architecture. But, optimization-based development has extreme convergence times, on the order of days to weeks, and is built on the assumption that large-scale resources are available. Both these approaches to prototyping new DNNs pose a clear disadvantage, given our emphasis on efficiency.

DNN pruning adapts any existing DNN architecture to user- or hardware-based constraints by removing redundant portions of the DNN. This process builds on the assumption that standard DNN architectures are over-parameterized (Fig. 1.2), with a steep increase in the number of parameters compared to the size of commonly available datasets to improve performance, and contain a large amount of redundant information that we can remove without compromising performance. While pruning does not deliver a strictly novel DNN architecture from scratch, like manual or even optimization-based approaches, it does allow the user to leverage the space of existing work in the field of deep learning, spanning decades of research. In addition, pruning can quickly assess redundancy and adapt existing architectures, unlike the previous methods. By combining the advantages offered by pruning, we can ensure a faster development process and a final DNN architecture that requires minimal resources to function. Hence, we adopt DNN pruning

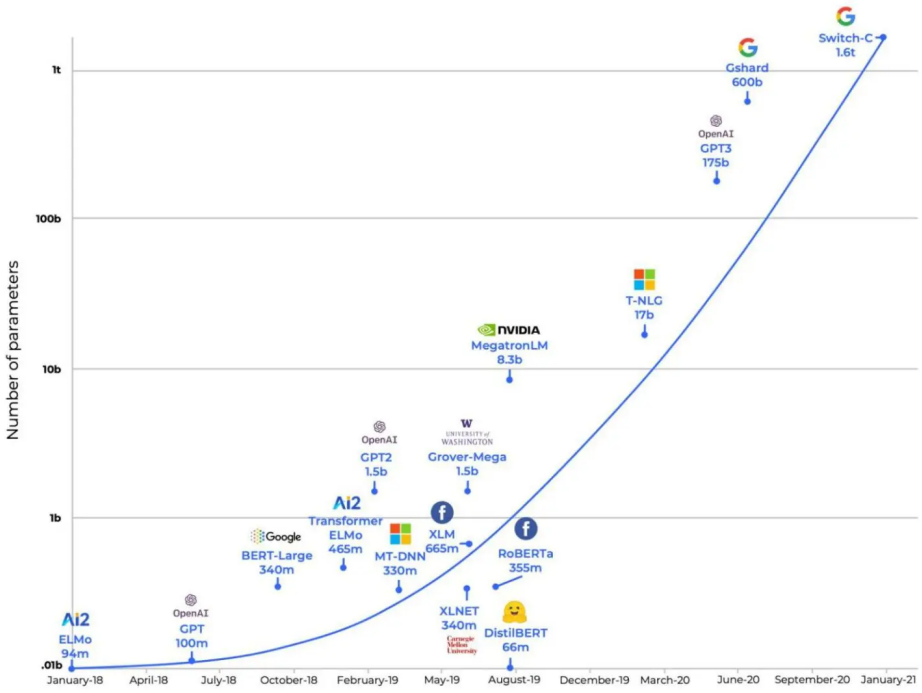


Figure 1.2: The exponential growth of the number of parameters in DNNs of state-of-the-art algorithms [26]. However, existing works in DNN pruning have shown that there is a lot of redundancy in large DNNs.

as our approach to prototyping DNN architectures that we can deploy in resource-constrained environments.

1.2.2 Training

Training is the critical process that defines the evolution of a DNN's weights. It derives its influence from several factors that include the choice of a dataset, optimizer, hyper-parameters, precision of computations, and others. Its primary focus is on imparting high generalization performance alongside other desirable properties like adversarial robustness [27], [28], improved prediction confidence [29], [30], and many others [31], [32]. Given our emphasis on improving the efficiency of the training process and the number of different factors that influence it, there a multiple approaches to realizing our goal. Examples of existing approaches towards realizing this goal include alternative optimization schemes [33]–[35], distributed training [36], [37], low-precision computations [38], [39], etc.

Alternative optimization schemes [40], [41] focus on improving the convergence in the general non-convex problem space of DNNs. However, they do not always guarantee faster convergence than standard mini-batch SGD and often neglect evaluating other desirable properties we expect

from real world solutions. Distributed learning approaches assume the availability of large-scale resources and focus on identifying more efficient communication patterns across each node of the hardware. While they offer additional value when placed alongside other approaches, individually they do not adhere to our goal of efficiency. Low-precision computations offer an inexpensive and faster alternative to standard double precision floating point operations. However, their common implementations do not match the performance of their high-precision counter-parts or require a large number of computations to achieve similar levels of performance [42], [43].

Curriculum learning targets efficiency from the perspective of the dataset, by organizing and scheduling the data shown to the DNN during training, similar to the "curriculum" used to teach humans from a young age [44], [45]. This approach helps on two fronts, maintaining high performance and improving efficiency. When organizing data, curriculum learning uses difficulty measures to assess the quality of samples and curate a more optimal subset of training data. The optimal subset requires lower memory to store and improves accuracy. In this dissertation, we propose a curriculum learning variant inspired by negative mining that focuses on improving generalization performance with minimal overheads. Then, we extend our takeaways to further maximize our returns from curriculum learning by using noise injection to curate an optimal training subset that simultaneously tackles performance, efficiency, and adversarial robustness. Our approach offers a reprieve from the large memory consumed by training data, and the time to preprocess and load it by simply removing a noisy subset of data. In affecting the original dataset, a curriculum's regulatory effect permeates through the DNN and readily offers a complementary strategy to the other methods discussed.

1.3 Contributions

In this dissertation, our main contributions revolve around making two key phases more efficient: *Architecture Prototyping* (Section 1.3.1) and *Training* (Section 1.3.2). Our primary goal when tackling the prototyping phase is to use DNN pruning to reduce the *number of parameters* that make up a DNN, including a reduction in the total number of Floating Point Operations (FLOPs) and memory footprint while maintaining high generalization performance. By making the process of pruning more efficient and obtaining a sparse outcome, we improve on two fronts concurrently. When discussing the efficiency of the training process, our primary goal is to maintain high performance while reducing the overheads associated with curriculum learning, including the reduction of *FLOPs* used to train the DNN.

1.3.1 Jumpstarting Architecture Prototyping: Neural Network Pruning

Mutual information (MI) [46], [47] offers a grounded and analytical perspective on the relationship between two variables, specifically how observing information from one variable informs us

about the second variable. Conditional Mutual Information (CMI) is a logical expansion of MI, which measures the relationship between two variables when a third was already observed. When placed in the context of neural network pruning, CMI offers a unique measure of the relationship between two filters from adjacent layers in the context of all the contributions from different filters in the previous layer.

In this dissertation, we introduce a probabilistic framework for pruning that uses CMI as a measure of dependency between filters. In using CMI, we control the amount of information passed down to succeeding layers. Larger values indicate stronger relationships, highlighting filters that are essential contributors to the information generated. Filters with smaller CMI values are removed since they contribute less, relatively. This idea forms the basis of all our works on neural network pruning. In addition to CMI, one-shot pruning is a critical philosophy underlying our probabilistic framework. In one-shot pruning, we do not allow the update of a DNN's parameters until pruning across the entire DNN is complete. Taking it a step further, pruning itself is performed only once. We adhere to these philosophies since they embed the notion of efficiency in to the process itself, allowing it to become the guiding principle.

Building around the idea of using CMI, we propose a novel formulation of CMI that is faster than our original formulation and incorporates the contributions of the underlying weight matrix, the primary approach to pruning across a large amount of prior work. With a functional core idea, we then solve more practical issues built-in to the pruning pipeline like the manual effort to determine the upper pruning limits of each layer in the DNN and analyze the sensitivity of filters to being pruned, all while ensuring the entire process is automated. Putting our contributions under a single-shot pruning framework allows us to generate slim models, thereby accounting for efficiency and performance in both process and outcome.

1.3.2 Extracting More from Less: Curriculum Learning

Data-level curriculum learning follows the idea of organizing and scheduling the delivery of samples to the DNN during the training phase. Individual data samples are evaluated based on difficulty as measured using a pre-defined criterion like changes in gradient, loss, prediction values, etc. [48], [49]. The main focus of data-level curriculum approaches is maximizing the generalization performance.

Our first contribution towards achieving improved performance is exploring the notion of incrementally introducing class labels. In this work, we exploit ideas similar to negative mining by using a subset of the training data as a single "negative" class and improving the feature representations learned during the early training phase, when the ground-truth labels of different categories are introduced to the DNN incrementally. Most importantly, we introduce the notion of learning curricula over labels as opposed to samples and how this improved the learning process.

We did so while maintaining the same number of training epochs as mini-batch SGD. Effectively, we were able to extract more performance with no overheads.

Our combined observations on the adversarial susceptibility of pruned neural networks and the takeaways from our curriculum learning foray spurred our second contribution in extracting more performance from less information. We propose a novel algorithm that simultaneously targets improved generalization performance and adversarial robustness while using only a subset of the data, thereby tackling Performance, Efficiency and Robustness concurrently. For this purpose, we use the difference in activation values between the original training data and their noise-perturbed counterparts to identify and remove samples susceptible to noise. Crucial to our idea was the assumption that samples highly susceptible to noise exist near the decision boundaries and thus have a strong impact on the evolution of a DNN. By highlighting and removing samples highly susceptible to noise, we improve the learning process and directly affect the generalization performance and adversarial robustness of the DNN solution. Thus, we improve the efficiency of the training process while making no assumptions about the type of noise used to perturb the training samples.

1.4 Thesis and Impact Statement

Through innovations in neural network model compression-via-pruning and curriculum learning, our work enforces a higher level of efficiency in process, for architecture development and training, as well as outcome than prior work. Specifically, we introduce and build upon a probabilistic pruning framework based on CMI, that balances the contributions from the weight matrices underlying the DNN with those obtained from mutual information. In addition, we use this framework to solve more practical concerns built-in to pruning frameworks like the large number of trials to figure out the upper pruning limit of each layer in the DNN as well as the sensitivity of each filter to being pruned. Combing these contributions, we provide an automated single-shot pruning framework capable of handling any feedforward DNN. Further, our work in curriculum learning addresses improvements in both performance and adversarial susceptibility while reducing the number of FLOPs computed during training.

The direct impacts of our contributions are the reduction in time and cost involved in the iterative design process of developing DNN solutions. Our solution is built to be flexible and quick in adapting any existing DNN architecture to a variety of constraints. On a macro-scale, we believe that using multiple factors to constrain the development of DNNs, from a nascent stage, is a definite way to build-in important properties. By doing so, we can address lower level goals like efficiency and calibration as well as higher level goals like a sustainable development process with a lower carbon footprint.

CHAPTER 2

Related Work

In the following sections, we divide the discussion of related works into four main subsections.

Sections 2.1 and 2.2 emphasize works in the domain of neural network compression and relevant literature on multivariate dependency measures. Sections 2.3.1 and 2.3.4 describe prior work in curriculum learning and adversarial training, and discuss how these ideas are interlinked Throughout discussing various prior works, we provide context and comparisons to the ideas discussed in the dissertation.

2.1 Neural Network Compression-via-Pruning

At the highest level, there are many strategies we can use to help compress DNNs while maintaining high performance. Examples of such approaches include low-rank approximations of weight matrices [50], quantization of weights stored [51], knowledge distillation from high capacity to slim networks [52], and pruning [53]–[55]. Low-rank approximation and quantization fundamentally change the values stored in the weight matrices by modifying their precision or approximating them. While these methods offer a speed-up in inference time, their final performance isn't sufficiently high compared to other approaches in the field of DNN pruning. Knowledge distillation relies on the existence of a fully pre-trained model from which logits or features can be distilled and thus, requires a large amount of storage memory and resources. All of the approaches outlined violate our principle of efficiency while offering diminished performance returns.

In this dissertation, we focus on DNN pruning since it comprises a flexible framework that offers a variety of resolutions at which we can prune a network, from individual weights to entire channels, with minimal memory requirements and without significantly weakening generalization performance. Based on the underlying philosophy, there are two broad categories among pruning approaches: 1) methods that use a deterministic constraint on the weight matrices, and 2) methods that use a probabilistic framework to reduce the redundancy and maintain the flow of information between layers

2.1.1 Deterministic Constraints

Direct Constraint on Weight Matrices: Some of the earliest works in pruning use the second-order relationship between the objective function and weights of the network to evaluate and remove unimportant values [56], [57]. The recent surge in size and complexity of DNN architectures makes it impossible to compute second-order relationships during training and pruning since they have a complexity of $\mathcal{O}(W^2)$. Since then, directly thresholding weights using their magnitude [53], [58] or $||.||_n$-based constraints [59], [60] have become more popular due to their simplicity and ease of implementation. While most of the existing methods exploited the relationship between the magnitude and importance of a filter or between the objective function and various weights, more recent advances in the field opt to use data-driven logic to derive the importance of various filter weights. The earliest examples of such methods include ThiNet [54], which poses the reconstruction of features with the removal of channels as an optimization problem, NISP [61], where the feature rankings from the final layer are propagated through the DNN and used derive the importance of various filters, and APoZ [62] which evaluates the percentage of zeros in the feature map to assess which filters can be pruned.

The common consensus among a broad swathe of prior work in this domain is to apply the deterministic constraints for pruning during or after the training phase. Attempting to maximize the benefits of training sparse networks, there has been a recent shift towards applying constraints for pruning at initialization. Tanaka *et al.* [63] iteratively prune weights with the lowest "synaptic strength", Lee *et al.* [64] prune weights that are least salient for the loss, and Wang *et al.* [65] prune weights that most harm or least benefit gradient flow. However, this sub-domain of pruning is in relatively early stages of development, and thus its results have not improved over other existing approaches [66]. Regardless of where the direct constraints are placed, they often do not account for the downstream impact of pruning or are built on the assumption of a purely deterministic relationship between filters. Instead, we use a probabilistic framework to maintain the flow of information between layers while also expanding to a hybrid formulation that uses a weight-based to overcome these issues.

Modification of Objective Function: We can induce sparsity in the weight matrix by modifying the objective function to include regularization terms that force the weights to swing between extremely small or normal values. This method is distinct from directly applying constraints on the weight matrices since it affects the natural evolution of the weights instead of only applying a binary mask. The regularization terms used to modify the objective function range from simple constraints like $l_{0/1}$-norms on individual weights or structures (channel, filter, layer) [67], [68] to more complex norms like $l_{2,1}$ [69] and group-lasso-based objective functions in [70]–[72]. Expanding upon these ideas, there are a number of works that use more nuanced constraints like balancing individual vs. group sparsity [73], [74], and and adding discrimination-aware losses at intermediate layers to

enhance and easily identify important channels [75].

Some of the most exciting and recent trends in pruning focus on the fusion of concepts like meta-learning [76], where sparsity-inducing regularizers are used to learn latent vectors that help decide on evolution and choice of weight values, and GANs, where an adversarial learning paradigm optimizes a loss based on the comparison of features derived from a baseline network, to generate a sparse DNN [55]. Apart from optimizing over a fundamentally different set of objective functions, which are harder to optimize, methods that modify the objective function require multiple iterations of pruning and fine-tuning to ensure stability during training. In this dissertation, we want a more controlled setup to study and compare the effects of pruning a network against its original counterpart. Thus, we avoid strong comparisons against methods that modify the objective function.

2.1.2 Probabilistic Frameworks

Among existing works in pruning, probabilistic frameworks offer an alternative modelling perspective to pruning that accounts for the uncertainty in relationships between filters and layers. Under the umbrella of probabilistic frameworks, there are two broad categories of approaches, bayesian and non-bayesian. Bayesian methods use variational bayesian inference as the basis of their approach, with a focus on estimating the posterior distribution of weights using the ELBO algorithm [77], [78]. Fundamentally, these approaches offer a theoretically sound basis for pruning. However, they require strong assumptions on the prior distribution of weights, which helps induce sparsity across the network. In addition, their performance on large-scale datasets has more room to grow, with VGG being the largest DNN used in Louizos *et al.* [78], and Zhao *et al.* [77] providing minimal comparisons against ResNet50 baselines on ILSVRC2012.

Non-bayesian approaches to pruning pursue information-theoretic measures to model the flow of information through the network. When comapred to bayesian approaches, information-theoretic measures make lesser assumptions and are applicable under a variety of conditions. Examples of non-bayesian approaches include Luo and Wu [79], who suggest entropy of activations as a measure to establish the importance of filters, and VIBNet [80], where they use the information bottleneck principle to minimize the redundancy between adjacent layers. While they are adept at reducing redundancy and maintaining the flow of information between layers, they have similar weaknesses as the bayesian approaches in terms of their low performance at extreme sparsity levels.

Information-theoretic measures offer multiple flexible formulations that have the potential to incorporate multiple sources of data and establish a theoretically grounded measure of information flow in the DNN. Thus, we use CMI as the basis of our pruning approaches and expand our comparisons to include lightweight and heavyweight networks in order to showcase its strengths. Further, we explore multiple formulations of CMI to steadily improve the speed of pruning, overall sparsity and final performance.

2.2 Multivariate Dependency Measures

Mutual Information (MI) is the foundational idea behind our works on neural network pruning. It is a measure used to estimate multivariate dependencies, and we use it to understand the relationship between filters in adjacent layers and remove filters that contribute less information. Plugin and direct estimation are the two general approaches used to estimate MI. Plugin estimators like Kernel Density Estimators (KDEs) [81], KNN estimators [82], and others [83], [84] form the bulk of early works in computing multivariate dependency. However, plugin estimators need to accurately estimate the probability density function of input variables. This, when combined with their large run-time complexity, renders them highly unscalable to high-dimensional information.

To overcome the issues faced by plugin estimators, direct estimators for Renyi-entropy and MI [83], [84], and the Henze-Penrose divergence measure [85] have been proposed. They provide manageable run-time complexity while avoiding direct knowledge of the density function. Crucial to the functioning of many direct-estimation methods is the use of graph-theoretic ideas, such as the Nearest Neighbour Ratios [86], which uses the k-NN graph to estimate MI, and the minimum spanning tree used to estimate the GMI [87]. These graph-based approaches help make the evaluation of MI more computationally tractable.

While most methods fall into either plugin or direct categories, Morteza *et al.* [88] propose a hybrid approach. This approach combines the fast run-time implementation of hash-tables, which have an average complexity of $\mathcal{O}(1)$, with an error convergence rate akin to plugin methods, thus merging the advantages of both the estimation approaches. In this dissertation, we begin by exploring the use of the conditional GMI [87] measure to assess the dependency between filters and remove filters that contribute less. Once we establish its viability for one-shot pruning, we then derive the conditional version of the MI measure proposed in Morteza *et al.* [88] and use it to combine information from the weights as well as the activations to measure the flow of information.

2.3 Efficient Training

Training guides the evolution of a DNN's weights through the interaction between a chosen dataset and DNN architecture using backpropagation [89]. This phase dictates the quality of the learned model including its performance, calibration, and adversarial robustness among a number of other desirable targets. In this dissertation, our approach to improving the training phase closely resembles the ideas discussed in curriculum learning, which controls the training process by affecting how the dataset is organized and presented to the DNN. While standard curriculum learning emphasizes the improvement of performance, we extend our discussion to adversarial training and how alternative curricula enhance the robustness generated from standard

adversarial training. Finally, we push the boundary of existing work by combining improvements in performance, adversarial robustness and efficiency, to design a curriculum packaged as a plug-and-play module that requires minimal modifications to incorporate into a standard training pipeline for DNNs. In the following sections, we discuss relevant works from multiple domains that have inspired our works.

2.3.1 Curriculum Learning

Curriculum learning for DNNs is inspired by the teaching paradigm used by humans to organize and present information in an orderly fashion [44], [45]. The earliest formulations of curriculum learning emphasized faster convergence to a high quality solution [90]–[92]. Often, data was organized based on the "difficulty" of samples using external ranking methods that include manual [90], [93], [94] and predictions from alternative DNNs [95]–[97], or internal rewards like loss-based [98]–[100], and others [49], [101].

Focusing more on the quality of convergence in DNNs, a number of works relax the assumption of faster convergence while exploring various ways to organize and schedule data [101], [102]. However, more recently there has been a shift towards using feedback from the model being trained to modify the organization of the training data, including the variation of the amount of data used to train the model [48], [49], [103]. However, across most works in curriculum learning there has always been an emphasis on improving the generalization performance of the final solution, be it on a standard dataset, small datasets [104] or corrupted datasets [102]. However, general curriculum learning pays little attention to adversarial robustness.

In this dissertation, we explore a label-based curriculum that breaks the mould of sample-level difficulty while establishing improved performance. Additionally, we consider robustness to adversarial attacks a key trait required of DNNs, especially when we consider their application in a safety critical real-world context like medical or health applications. From a methodological point of view, our approach moves away from using gradients, loss value, predictions or the change in those values to identify difficult samples [49], [105]–[107] and instead focus on label- and feature-level curricula that can be easily extended to different architectures. To boost efficiency, we focus on hard sampling to permanently remove samples from the training set instead of recycling them during the training phase [48].

2.3.2 Label Smoothing

Label smoothing techniques regularize deep networks by penalizing the objective function based on a pre-defined criterion. Such criteria include using a mixture of true and noisy labels [108], penalizing highly confident outputs [109], and using an alternate deep network's outcomes as

ground-truth [110]. Bagherinezhad *et al.* [111] proposed the idea of using logits from trained models instead of just one-hot vectors as ground-truth. Complementary work by Miyato *et al.* [112] used the local distributional smoothness, based on the robustness of a model's distribution around a data point, to smooth labels. The work closest to our work in this dissertation was proposed in Szegedy *et al.* [113], where an alternative smooth target distribution was used to replace the one-hot vector across the entire dataset. In our work, we propose to only alter the ground-truth label vector for samples that are misclassified. By targeting samples that are misclassified, we improve the entropy of their ground-truth vectors, thus facilitating the DNN to correct itself. We identify misclassified examples using a prior copy of the model being trained, which helps avoid external computational overhead and only uses a small set of operations.

2.3.3 Incremental Learning and Negative Mining

The core structure of our label-based curriculum is strongly inspired from incremental and continual learning. Often, incremental/continual learning focus on learning over evolving data distributions with the addition of constraints on the storage memory [114], [115], distillation of knowledge across different distributions [116], [117], assumption of a single pass over data [118], [119], etc. In our approach, we depart from the assumption of evolving data distributions. Instead, we adopt the experimental pipeline used in incremental learning to introduce new labels at regular intervals, while still using the entire dataset and thus maintaining the same data distribution.

To extract more performance out of any standard dataset, we take cues from negative mining [120]–[122]. In general, negative mining approaches generate "negatively" associated pairs of data samples by randomly associating samples, using uncertainty-based metrics [123], [124] and incorporating intra-class correlation as a method to pick samples that are rarely associable [120]. Since we primarily focus on learning from classification datasets, we pursue the the idea of introducing ground-truth labels incrementally and using the data whose ground-truth labels have not been introduced as negative samples. Doing so encourages the diversity of negative samples and allows the DNN to learn stronger representations for the classes which have been introduced.

2.3.4 Adversarial Training

Adversarial training approaches expose the DNN to a variety of adversarial perturbations during the training phase to increase their robustness to being attacked [27], [125]. The approaches themselves span a large spectrum of ideas based on their choice of target and method of approach. For example, the most common type of adversarial training involves choosing an adversarial attack to create perturbations to the input during training [126], [127]. At the other end of the spectrum, a number of works emphasize the theoretical robustness guarantees when conforming to l_p-bounded

attacks. The advent of adversarial training spawned a variety of different adversarial attacks [128]–[130] resulting in a symbiotic cycle of development between these two domains.

A common issue with adversarial training is the strong decrease in performance on natural testing data that accompanies the improved robust accuracy. To combat the trade-off between clean and robust accuracy in standard adversarial training a number of recent works propose the use of early stopping [131] or a student-teacher setup to learn from smooth logit distributions derived from pre-trained DNNs [132]. While these approaches focus on improving the natural and robust accuracies, their test bed does not cover a wide range of adversaries, thus limiting the scope of their study.

To that end, curriculum-based adversarial training approaches were introduced to improve the convergence of the final DNN over traditional adversarial training. Examples of such methods include gradually increasing the strength of adversaries to improve robustness [133], using the least adversarial data among confidently misclassified samples [134] and many more [135]. An important reason they perform well could be due to their ability to reduce overfitting and their use of relatively weaker attacks during the start of the training phase [136]. However, they rarely provide time or efficiency comparisons to standard or alternative adversarial training regimes.

Most recently, efficient adversarial training has received a lot of attention since it is guided by a number of desirable targets like improved adversarial robustness and efficiency. Wang *et al.* [137] propose a dynamic and efficient adversarial training methodology that automatically learns to adjust the magnitude of perturbations during the training process. While their theoretical analysis, computational complexity, and performance comparisons offer strong insights, their results are limited to fixed DNN backbones. Shafahi *et al.* [138] offer an inexpensive alternative of recycling gradient computations performed during backpropagation to generate adversarial examples. Wong *et al.* [139] review FGSM-based adversarial training and offer multiple key suggestions that extend FGSM's viability to quickly obtain highly robust DNNs. Each of the above methods that propose a more efficient adversarial training approach focus on modifying the algorithm used to generate adversaries while retaining the complete training set. However, in this dissertation we address training efficiency by using a feature-based curriculum to directly reduce the training data available, thus offering a complementary approach that can work alongside any traditional or efficient adversarial training algorithm.

CHAPTER 3

Mutual Information-based Neural Network Pruning

3.1 Motivation

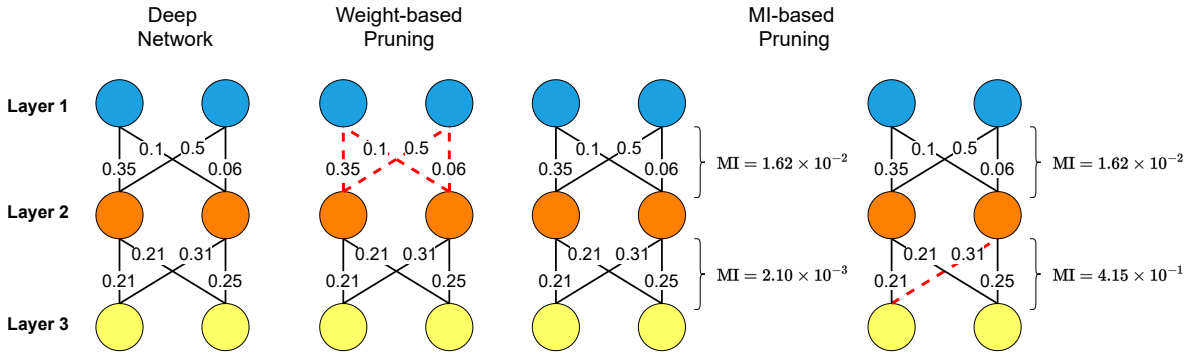


Figure 3.1: Weight-based pruning does not consider the dependency between layers. Instead, it suggests the removal of connections with small weight values. Mutual information-based pruning computes the value of information passed between layers, quantified by the MI value, and suggests the removal of weights from the latter layer. Doing so improves the flow of information while reducing redundancy. In this example, we calculate MI using the entropy of weight values.

Most prior pruning works that apply a deterministic constraint on the weights of the network, which form the bulk of existing work on neural network pruning, assume that the values stored in the weight matrix of a filter indicate its importance. Evaluating a filter's importance purely from its weights is insufficient since it does not consider the dependencies between filters across layers. For example, a subset of filters might contribute important information downstream and thus need to be retained, or a filter's contribution could be compensated elsewhere in the layer. Regardless of how the deterministic constraints are modeled, they do not account for any form of uncertainty.

An important takeaway from the deficiencies of common deterministic constraints on the weight matrix is that a higher weight value does not always represent its true importance. Consider the example shown in Fig. 3.1, where a simple weight-based criterion suggests the removal of small valued weights. However, the MI score in this example, which we use to measure the information shared between layers and thus the dependency between pairs of filters, values the first layer's

weights over the latter layer. If we follow the intuition provided by the MI scores, our pruned network ensures minimal redundancy and passes the majority of information to the next layer.

3.2 Mutual Information-based Neuron Trimming

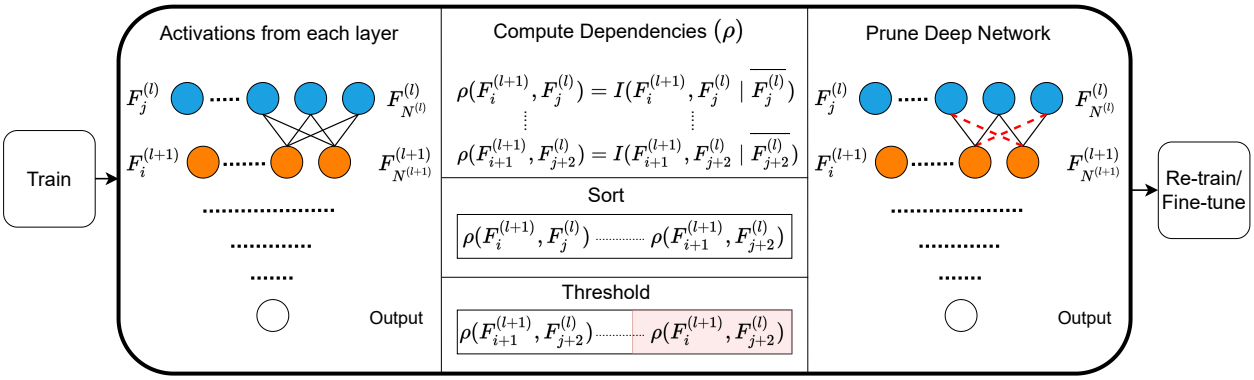


Figure 3.2: Illustration of the different components of MINT when pruning filters in layer $l+1$. Between every pair of filters in layers $(l, l+1)$, we compute the conditional geometric mutual information (GMI), using the activations from each filter as the importance score. We define the total number of filters in each layer by $N^{(l)}$ and $N^{(l+1)}$. The conditional GMI score indicates the importance of a filter in layer l's contribution towards a filter in layer $l+1$. We then threshold filters based on the importance scores to ensure that we only retain filters that pass the majority of the information to successive layers. Finally, we retrain the network once to recover performance.

Using the intuition provided by MI, we propose Mutual Information-based Neuron Trimming (MINT) as a novel approach to pruning deep networks by stochastically accounting for the dependency between layers. In Fig. 3.2, we outline the general framework of one-shot pruning we follow and the main components of our algorithm. In one-shot pruning, we partition the entire process, from one end to the other, into three phases, 1) training, 2) pruning, and 3) re-training. An important facet of having three distinct phases is that the pruning of weights happens during the middle phase, and we *do not update the weights of the network at any point during pruning*. The re-training phase, mirroring the training setup, allows for the compensation of any lost accuracy. All three phases are executed only *once*. We emphasize the necessity of following such a one-shot pruning approach: It is critical to reducing the time spent adapting and pruning existing DNN architectures.

Within the pruning stage, we use a graph-based criterion (Friedman-Rafsky Statistic [140]) to estimate the conditional geometric mutual information (GMI), inspired by Sekeh and Hero [87], to measure the dependency between filters of successive layers. On evaluating all such dependencies; between filters of every pair of adjacent layers in the network, we sort and threshold a desired percentage of the importance scores, thereby identifying and removing unimportant weights and filters. By retaining filters that contribute the majority of the information passed between layers, we

ensure minimal impact of pruning on downstream layers. In the following sections, we outline our algorithm before delving into its details.

3.2.1 Setup

First, we define the general setup of a prototypical DNN and the mathematical formulae associated with its various components. For a DNN containing a total of L layers, we compute the dependency (ρ) between pairs of filters in every adjacent set of layers. Here, the layer l is closer to the input while layer $l+1$ is closer to the output among the chosen pair of consecutive layers. The activations for a given layer $l+1$ are computed as,

$$F^{(l+1)}(\mathbf{x}^{(l)}) = \sigma\left(\mathbf{w}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}\right), \tag{3.1}$$

where $\mathbf{x}^{(l)} \in \mathbb{R}^{m \times d}$, m is the total number of samples, d is the feature dimension and $\mathbf{x}^{(l)}$ is the input to a given layer used to compute the activations. $\sigma()$ is an activation function, $\mathbf{w}^{(l)} \in \mathbb{R}^{N^{(l+1)} \times N^{(l)} \times H \times W}$, and $\mathbf{b}^{(l)}$ are the weights and bias. Moving forward, we avoid using the layer superscript in different components for the sake of readability.

3.2.2 Algorithm-specific Notations

- $F_i^{(l+1)}$: The activations from the selected filter i in layer $l+1$.
- $N^{(l+1)}$: Total number of filters in layer $l+1$.
- $S_{F_i^{(l+1)}}$: The set of indices that indicate the values that are retained in the weight vector of the selected filter.
- $\rho()$: The dependency between two filters, computed using $I(\mathbf{X}; \mathbf{Y}|\mathbf{Z})$ (importance score).
- $\overline{F_j^{(l)}}$: The set of all filters excluding $F_j^{(l)}$ in layer l.
- δ: Threshold on importance score to ensure only strong contributions are retained.

3.2.3 Core Algorithm

In every iteration of MINT (Alg. 1), we find the set of weight values in $\mathbf{w}$ to retain while we zero out the remaining.

- For a given pair of consecutive layers $(l, l+1)$, we compute the dependency between every filter in layer $l+1$ with filters in layer l. The main intent of framing the algorithm in this perspective is that the activations from layers closer to the input have a direct effect on downstream layers, while the reverse is not true for a forward pass of the network.

ALGORITHM 1: MINT pruning algorithm for filters of layers $(l, l+1)$

for *Every pair of layers* $(l, l+1), l \in 1, 2, \ldots, L-1$ **do**
  for $F_i^{(l+1)}, i \in 1, 2, \ldots N^{(l+1)}$ **do**
    Initialize $S_{F_i^{(l+1)}} = \emptyset$;
    for $F_j^{(l)}, j \in 1, 2, \ldots N^{(l)}$ **do**
      $\rho(F_i^{(l+1)}, F_j^{(l)}) = I(F_i^{(l+1)}, F_j^{(l)} \mid \overline{F_j^{(l)}})$;
      if $\rho(F_i^{(l+1)}, F_j^{(l)}) \geq \delta$ **then**
        $S_{F_i^{(l+1)}} = S_{F_i^{(l+1)}} \cup \text{index}(F_j^{(l)})$
      end
    end
  end
end

- Using the activations $F()$ for the selected filters, $(F_i^{(l+1)}, F_j^{(l)})$, we compute the conditional GMI (Eqn. 3.5) or $\rho()$ between them, given all the remaining filters in layer l. This conditional dependency captures the relationship between filters in the context of all the contributions from the preceding layer. Since the activations of layer $l+1$ are weighted combinations of activations from all the filters in the previous layer, we need to account for this when considering the dependence of activations between two selected filters.

- Based on the strength of each $\rho(F_i^{(l+1)}, F_j^{(l)})$, the contribution of filters from the previous layer is either retained/removed. We define a threshold δ for this purpose, a key hyper-parameter.

- $S_{F_i^{(l+1)}}$ stores the indices of all filters from layer l that are retained for a selected filter $F_i^{(l+1)}$. The weights for the retained filters are left the same, while we zero out the weights for the entire kernel in the remaining filters. In the context of fully connected layers, we retain or zero out specific weight values.

Group Extension While evaluating dependencies between every pair of filters allows us to take a close look at their relationships, it does not scale well to deeper or wider architectures. To address this issue, we evaluate filters in groups rather than individually. We define G as the total number of groups in a layer, where each group contains an equal number of filters. We explore in depth the impact of varying the number of groups in Section 3.3.4. Although there are multiple approaches to grouping filters, in this work, we restrict ourselves to sequential grouping, where we construct groups from consecutive filters. There is no explicit requirement for a pre-grouping step before our algorithm so long as we use a balanced grouping of filters.

Finer Details We construct MINT based on the assumption that majority of information from the preceding layer is passed forward, and the filter in consideration can selectively retain contributions for a subset of previous filters. This assumption allows us to work on isolated pairs of layers with minimal impact on downstream layers because retaining filters with high MI will ensure the passage of a large percentage of the information to the next layer. By retaining as much information as possible between layers we ensure that the amount of critical information passed to layers further downstream is maintained. We provide an implementation of our method on GitHub [1].

3.2.4 Conditional Geometric Mutual Information Estimation

In this section, we describe the conditional GMI estimator used to compute $\rho()$ in Alg. 1. We proceed by first defining conditional GMI estimation as featured in [87] before providing details on a close approximation of their method we use to calculate multivariate dependencies in our algorithm.

Definition We first define a general form of GMI denoted by I_p: For parameters $p \in (0, 1)$ and $q = 1 - p$ consider two random variables $\mathbf{X} \in \mathbb{R}^{d_x}$ and $\mathbf{Y} \in \mathbb{R}^{d_y}$ with joint and marginal distributions $f(\mathbf{x}, \mathbf{y})$, $f(\mathbf{x})$, and $f(\mathbf{y})$ respectively. The GMI between $\mathbf{X}$ and $\mathbf{Y}$ is given by

$$I_p(\mathbf{X}; \mathbf{Y}) = \frac{1}{4pq} \times \left[\iint \frac{\left(f_{XY}(\mathbf{x}, \mathbf{y}) - q f_X(\mathbf{y}) f_Y(\mathbf{y})\right)^2}{p f_{XY}(\mathbf{x}, \mathbf{y}) + q f_X(\mathbf{x}) f_Y(\mathbf{y})} \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y} - (p - q)^2 \right]. \tag{3.2}$$

Considering the special case of $p = q = 1/2$ in Eqn. 3.2 we obtain,

$$I(\mathbf{X}; \mathbf{Y}) = 1 - 2 \iint \frac{f_{XY}(\mathbf{x}, \mathbf{y}) f_X(\mathbf{x}) f_Y(\mathbf{y})}{f_{XY}(\mathbf{x}, \mathbf{y}) + f_X(\mathbf{x}) f_Y(\mathbf{y})} \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y}. \tag{3.3}$$

The conditional form of this measure, proposed in [87], is,

$$I(\mathbf{X}; \mathbf{Y} | \mathbf{Z}) = \mathbb{E}_Z \left[I(\mathbf{X}; \mathbf{Y} | \mathbf{Z} = \mathbf{z}) \right], \quad \text{where} \tag{3.4}$$

$$I(\mathbf{X}; \mathbf{Y} | \mathbf{Z} = \mathbf{z}) = 1 - 2 \iint \frac{f_{XY|Z}(\mathbf{x}, \mathbf{y} | \mathbf{z}) f_{X|Z}(\mathbf{x} | \mathbf{z}) f_{Y|Z}(\mathbf{y} | \mathbf{z})}{f_{XY|Z}(\mathbf{x}, \mathbf{y} | \mathbf{z}) + \ f_{X|Z}(\mathbf{x} | \mathbf{z}) f_{Y|Z}(\mathbf{y} | \mathbf{z})} \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y}. \tag{3.5}$$

Relating the Estimator to MINT The estimation procedure, outlined in Algorithm 2, is based on the minimal spanning tree (MST). If we assume a data set $\mathcal{U}_N = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i\}_{i=1}^N$, we divide it randomly into three equal parts of n samples each, $\mathcal{U}_n$, $\mathcal{U}_2$, and $\mathcal{U}_3$. Denote $\overline{\mathcal{U}}_{2n} := \mathcal{U}_2 \cup \mathcal{U}_3$.

In step 2 of Algorithm 2, we use set $\overline{\mathcal{U}}_{2n}$ in the Nearest Neighbour Bootstrap algorithm [141] (Algorithm 3). We generate a data set $\widetilde{\mathcal{U}}_n$ consisting of n samples given a data set $\overline{\mathcal{U}}_{2n}$ with $2n$

[1] https://github.com/MichiganCOG/MINT

ALGORITHM 2: Conditional MI Estimator

Require: Divide $\mathcal{U}_N$ in two subsets $\mathcal{U}_n$ and $\overline{\mathcal{U}}_{2n}$ with sample sizes n and $2n$, respectively.

1: Divide $\mathcal{U}_N$ in two subsets $\mathcal{U}_n$ and $\overline{\mathcal{U}}_{2n}$ with sample sizes n and $2n$, respectively.
2: Find set $\widetilde{\mathcal{U}}_n$ from Nearest-Neighbor bootstrap (Alg. 3) with n sample size using $\overline{\mathcal{U}}_{2n}$.
3: $\widehat{\mathcal{U}}_n \leftarrow \mathcal{U}_n \cup \widetilde{\mathcal{U}}_n$
4: Construct MST on $\widehat{\mathcal{U}}_n$
5: $\mathfrak{R}_n \leftarrow \#$ edges connecting a node in $\mathcal{U}_n$ to a node of $\widetilde{\mathcal{U}}_n$
6: $\widehat{I} \leftarrow 1 - \dfrac{\mathfrak{R}_n}{n}$

Ensure: Conditional MI estimator $\widehat{I}$.

ALGORITHM 3: Nearest-Neighbor Bootstrap

Require: Data set $\overline{\mathcal{U}}_{2n} := (\mathbf{X}_{2n}, \mathbf{Y}_{2n}, \mathbf{Z}_{2n}) = \left\{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)_{i=1}^{2n}\right\}$.

1: Divide $\overline{\mathcal{U}}_{2n}$ into two equally sized distinct subsets $\overline{\mathcal{U}}_2$ and $\overline{\mathcal{U}}_3$.
2: $\widetilde{\mathcal{U}}_n = \emptyset$.
3: For $\mathbf{u}_2 = (\mathbf{x}_2, \mathbf{y}_2, \mathbf{z}_2) \in \overline{\mathcal{U}}_2$ do
4: Let $\mathbf{u}_3 = (\mathbf{x}_3, \mathbf{y}_3, \mathbf{z}_3) \in \overline{\mathcal{U}}_3$ be the sample s.t. $\mathbf{z}_3$ is the Nearest Neighbor of $\mathbf{z}_2$ in the dataset $\mathcal{U}_3$.
5: Let $\widetilde{\mathbf{u}}_3 = (\mathbf{x}_2, \mathbf{y}_3, \mathbf{z}_2)$ and $\widetilde{\mathcal{U}}_n = \widetilde{\mathcal{U}}_n \cup \{\widetilde{\mathbf{u}}_3\}$.

Ensure: n conditional independent sample $\widetilde{\mathcal{U}}_n$.

i.i.d samples from a distribution $f(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Essentially, this generates conditionally independent samples. The estimator of $I(\mathbf{X}; \mathbf{Y}|\mathbf{Z})$ is the FR test statistic, $\mathfrak{R}_n := \mathfrak{R}_n(\mathcal{U}_n, \widetilde{\mathcal{U}}_n)$ which is the total number of dichotomous edges in a graph constructed by first generating an Euclidean minimal spanning tree (MST) on the concatenated data set $\mathcal{U}_n \cup \widetilde{\mathcal{U}}_n$.

MINT takes advantage of the estimator by providing a set of activations from a neuron as the data to the estimator. Thus, for groups of neurons, we have a multi-dimensional vector of activations, which correspond to $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$. Following Algorithm 2, we get a scalar value as the CMI measure. Note: Within the MINT, we apply the conditional GMI estimator, indicated as the function $\rho()$, on a set of activations obtained from each filter we consider.

Computational Complexity For pruning algorithms we generally consider training-time cost as less important than test-time impact. The computational complexity of our pruning method has a bottleneck of $\mathcal{O}(E \log V)$, which is created by the algorithm used to generate the MST. Here, vertices (V) correspond to the number of neurons in a group. Also, there is an additional dependency on the number of activations used to approximate the value of a neuron (dimensionality in the MST). To provide some context for our computational complexity, if we consider the computation of a hessian [56], with no assumptions on the underlying structure, the time complexity would be of the order $\mathcal{O}(N^2)$, where N denotes the size of the weight matrix. Despite the computational complexity

bottleneck, our proposed algorithm offers a significantly high compression ratio, which leads to strong test-time impact.

3.3 Evaluation

In this section, we outline the various protocols used to evaluate our algorithm before discussing its performance in comparison to existing benchmarks (Section 3.3.3), analyzing the impact of critical hyper-parameters (Section 3.3.4) and characterizing our algorithm from multiple perspectives to provide a more holistic evaluation of its behavior (Section 3.3.5).

3.3.1 Datasets, Models and Metrics

Dataset Our experiments are conducted on three distinct datasets, MNIST [142], which contains hand-written images of digits, CIFAR-10 [143], a 32×32 pixel set of natural images across 10 categories and ILSVRC2012 [144], the 1000 class variant of the ImageNet 2012 dataset.

Models We evaluate a Multi-Layer Perceptron model's performance on the MNIST dataset, VGG16 [145] and ResNet56 [146] on CIFAR-10, and ResNet50 on ILSVRC2012. These represent the common dataset-DNN benchmarks used to validate the performance of pruning algorithms.

Metrics We use three distinct metrics to evaluate the quality of our pruning algorithm.

- **Parameters Pruned (%)**: The percentage of parameters removed from the baseline network. A higher value alongside good performance indicates a superior method.
- **Test Accuracy (%)**: The best performance on the testing set, upon training, for baseline networks, and re-training, for pruning methods.
- **Memory Footprint (Mb)**: Memory consumed when storing the weights of a network in CSR format under "npz" files.

An ideal pruning algorithm is expected to have a high value for Parameters Pruned ($\%$) and Test Accuracy ($\%$) while retaining a small Memory Footprint (Mb).

3.3.2 Experimental Setup

We outline the hyper-parameters used during each stage of our experimental pipeline, training, pruning and retraining. The values are provided here in an effort to help users replicate our experiments.

	MLP	VGG16	ResNet56
Epochs	30	300	300
Batch Size	256	128	128
Learning Rate	0.1	0.1	0.01
Schedule	10, 20	90, 180, 260	150, 225
Optimizer	SGD	SGD	SGD
Weight Decay	0.0001	0.0005	0.0002
Multiplier	0.1	0.2	0.1

Table 3.1: Training setups used to obtain pre-trained network weights.

	MLP	VGG16
Epochs	30	300
Batch Size	256	128
Learning Rate	0.1	0.1
Schedule	[10, 20]	[90, 180, 260]
Optimizer	SGD	SGD
Weight Decay	0.0001	0.0005
Multiplier	0.1	0.2

Table 3.2: Retraining setups used to obtain final performance listed in Table 3.4.

	ResNet56	ResNet50
Epochs	300	130
Batch Size	128	64
Learning Rate	0.1	0.1
Schedule	[90, 180, 260]	[30, 60, 90, 100]
Optimizer	SGD	SGD
Weight Decay	0.0002	0.0001
Multiplier	0.1	0.1

Table 3.3: Retraining setups used to obtain final performance listed in Table 3.4.

Training Setup: In Table 3.1 we outline the setups used to obtain trained models from which mutual information (MI) estimates are computed. A standard pretrained model from PyTorch [147] zoo was used for the ResNet50-ILSVRC2012 experiments.

Pruning Setup: For the conditional geometric mutual information (GMI) estimate we provide a set of parameters describing the number of groups per layer, G as well as the number of samples per class, m. We use the average activation across a filter's dimensions as the samples for GMI computations.

- MLP - MNIST: Between FC 1 and FC2, G is set to 250 and 100 while between FC 2 and FC 3 it is set to 300 and 10. 650 samples per class are used to compute the conditional GMI estimates.
- VGG16 - CIFAR10: All the layers between Convolution 4 and Linear 1 use $G = 64$ while $m = 650$. $\gamma = 0.45$ is used for Convolution layers 5, 6, 7, and 8.
- ResNet56 - CIFAR10: m is set to 500 samples per class. All the layers between Convolution 1 and Convolution 19 use $G = 16$, Convolution 20 to 37 use $G = 32$ while layers up to Convolution 55 use $G = 64$. We skip pruning Convolution layer 16, 20, 38, and 54.
- ResNet50 - ILSVRC2012: m is set to 5 samples per class. All the convolution layer use $G = 64$. We skip pruning the final linear layer.

Retraining Setup: Tables 3.2 and 3.3 outline the setups used to obtain the final values used in Table 3.4. While retraining, the MLP took approximately 1-2 minutes, VGG16 took $\sim$ 1.5 hours, ResNet56 took $\sim$ 2.5 hours and ResNet50 on ILSVRC2012 took close to a week on 1 GPU.

3.3.3 Comparison Against Existing Work

As a first step in showcasing MINT's abilities, we compare it against State-Of-The-Art (SOTA) baselines in network pruning. The baselines in Table 3.4 are arranged in ascending order of the percentage of parameters pruned, from top-down. Our algorithm outperforms most of the SOTA pruning baselines across the number of pruned parameters by maintaining high accuracy and reducing the memory footprint of the network. We note that while most of the pruning baselines listed use multiple prune-retrain steps to achieve their result, we use only a *single step* to match or outperform them.

In Fig. 3.3 we take a deeper look at how the overall compression $\%$ is spread throughout the network. Comparing Figs. 3.3a, 3.3b and 3.3c, we can establish the strong influence of datasets and network architecture on where redundancies are stored. In the cases of VGG16 and ResNet56, training with CIFAR-10 leads to the storage of possibly redundant information in the latter portion of the networks. The early portions of the network are extremely sensitive to pruning. ResNet50 when trained on ILSVRC2012 forces compression to be more spread out across the network, possibly indicating the spread of redundant features at different levels of the network.

	Method	Params. Pruned(%)	Test Acc. (%)	Memory(Mb)
MLP MNIST	Baseline	N.A.	98.59	0.537
	SSL [70]	90.95*	98.47	N.A.
	Network Slimming [67]	96.00*	98.51	N.A.
	MINT ($\delta = 0.645$)	96.20*	98.47	0.022
VGG16 CIFAR-10	Baseline	N.A.	93.98	53.868
	Pruning Filters [148]	64.00	93.40	N.A
	SSS [71]	73.80	93.02	N.A
	GAL [55]	82.20	93.42	N.A.
	MINT ($\delta = 0.850$)	83.46	93.43	9.020
ResNet56 CIFAR-10	Baseline	N.A.	92.55	3.109
	GAL [55]	11.80	93.38	N.A.
	Pruning Filters [148]	13.70	93.06	N.A.
	OED [70]	43.50	93.29	N.A.
	NISP [61]	43.68	93.01	N.A.
	MINT ($\delta = 0.184$)	52.41	93.47	1.552
	MINT ($\delta = 0.208$)	57.01	93.02	1.461
ResNet50 ILSVRC2012	Baseline	N.A.	76.13	91.157
	GAL [55]	16.86	71.95	N.A.
	OED [70]	25.68	73.55	N.A.
	SSS [71]	27.05	74.18	N.A.
	NISP [61]	43.82	71.99	N.A.
	ThiNet [54]	51.45	71.01	N.A.
	MINT ($\delta = 0.1000$)	43.01	71.50	52.365
	MINT ($\delta = 0.1101$)	49.00	71.12	47.513
	MINT ($\delta = 0.1103$)	49.62	71.05	46.925

Table 3.4: MINT is easily able to compete with SOTA pruning methods across all our evaluated benchmarks, using only a **single prune-retrain step**. Baselines use multiple prune-retrain steps and are arranged in increasing order of Parameters Pruned(%). We highlight a subset of available methods in pruning literature in the table. * indicates comparison of layer 2's weights.

3.3.4 Hyper-parameter Empirical Analysis

We take a closer look at three critical hyper-parameters that help MINT scale well to deep networks, (a) the number of groups in a layer, G, (b) the number of samples per class, m, used to compute the conditional GMI, and (c) γ, a hyper-parameter used to limit pruning for each layer of a DNN. We look into how each of them impacts the percentage of parameters pruned while ensuring

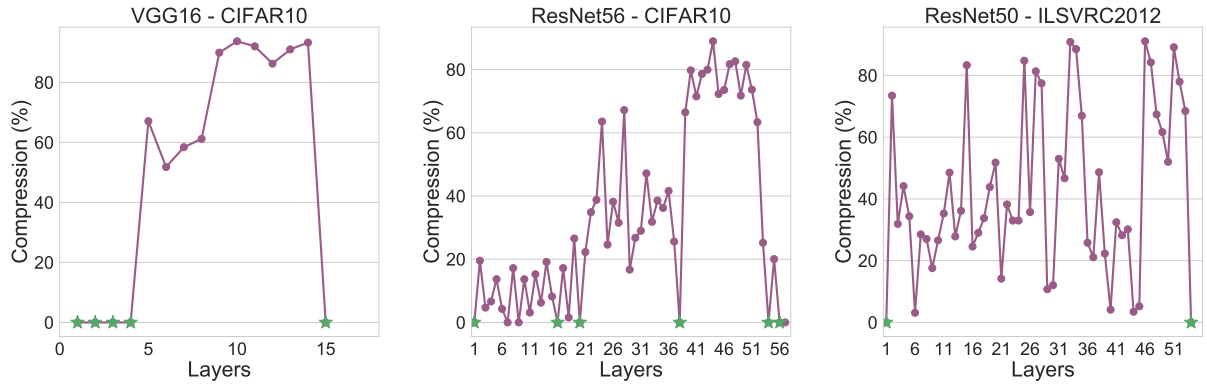


Figure 3.3: Illustration of compression % per layer for the best MINT-compressed networks from Table 3.4. We observe a characteristic peak in compression towards the later layers of both VGG16 and ResNet56 when trained on CIFAR-10. However, compression is spread over the course of the entire network for ResNet50. The green stars indicate layers we avoid pruning due to high sensitivity.

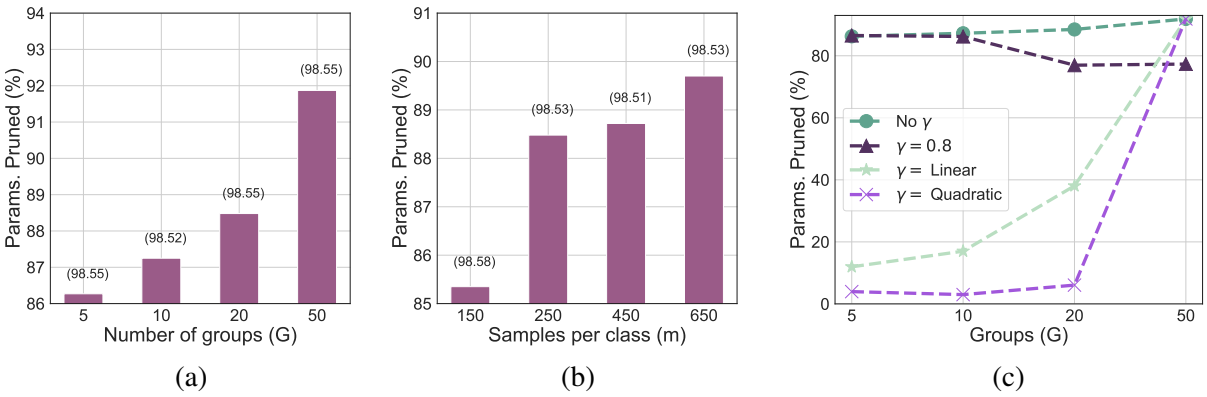


Figure 3.4: (a) An increase in the number of groups per layer allows for finer grouping of filters, which leads to more accurate GMI estimates and thresholding. Thus, there is a steady increase in the number of parameters that we can removed to achieve $> 98.50\%$ performance. (b) Keeping $G = 20$, we observe that increasing the number of samples per class improves the GMI estimate accuracy, which in turn allows for better thresholding and an increase in Parameters Pruned$(\%)$. The values on top of the bar plots are Test Accuracy$(\%)$. (c) Varying γ using a linear or quadratic dependence on G shows a positive correlation to parameters pruned, while using a constant value forces irregular behaviour.

we maintain $> 98.50\%$ accuracy on MNIST-MLP.

Group size G directly corresponds to the number of filters that are grouped together when computing conditional GMI and thresholding. Higher G leads to a lower number of filters per

group, which should allow for a more fine-grained computation of multivariate dependency and thereby more precise pruning. In this experiment, we set $m = 250$. Results in Fig. 3.4a match our expectations by illustrating the increase in the percentage of parameters pruned to maintain the desired performance.

Samples per class The number of samples per class directly impacts the final number of activations used to compute the conditional GMI. The GMI estimator should improve its estimates as the number of samples per class and the total number of samples increase. In this experiment, we fix $G = 20$. Fig. 3.4b validates our expectation by showing a steady improvement in the percentage of parameters pruned as the number of samples per class, and total number of samples increases.

Limit on pruning per layer In the description of Alg. 1, we use δ as a threshold on $\rho()$ values to help retain only those dependencies that contribute a majority of information to the next layer. However, in practice since the $\rho()$ values are not normalized w.r.t. each other, there are cases when certain layers can be fully removed when the δ parameter is large. In order to protect the network from such behaviour, we use an additional threshold called γ to cap the maximum percentage of filters that can be removed from a selected layer. If the pruning ratio within a layer exceeds the value of γ, then γ is used to re-evaluate δ such that it falls within the desired pruning ratio. The new temporary value of δ is computed using the $\rho()$ values within the pair of layers considered, and is only used prune filters in the selected pair of layers.

Fig. 3.4c illustrates the impact of γ on the percentage of parameters pruned in the MNIST-MLP experimental setup. Here, we observe a relatively steady increase in the maximum number of parameters pruned as the number of groups (G) increases, when no γ is provided or when a linear/quadratic dependence on G is used to compute γ. However, when γ is fixed at 0.8, we observe irregular behaviour, where the maximum percentage of parameters removed decreases with an increase in G. We posit that this behaviour arises from two factors, (a) the varied sensitivity of different layers is not compensated for by a fixed γ value, and (b) the difference in G values lead to varied GMI estimates which account for more uncertainty in measurement.

3.3.5 Characterization of MINT

While there are a number of standard metrics used to compare neural network pruning methods, the original intent in compressing them was to deploy them in real-world scenarios. In such contexts, characterizations like robustness to adversarial attacks, the ability to reflect true confidence in predictions, and other such properties are highly desirable. To understand the behavior of MINT-pruned models in real-world settings, we empirically analyze the variation in features captured, calibration statistics and adversarial robustness of MINT-pruned models.

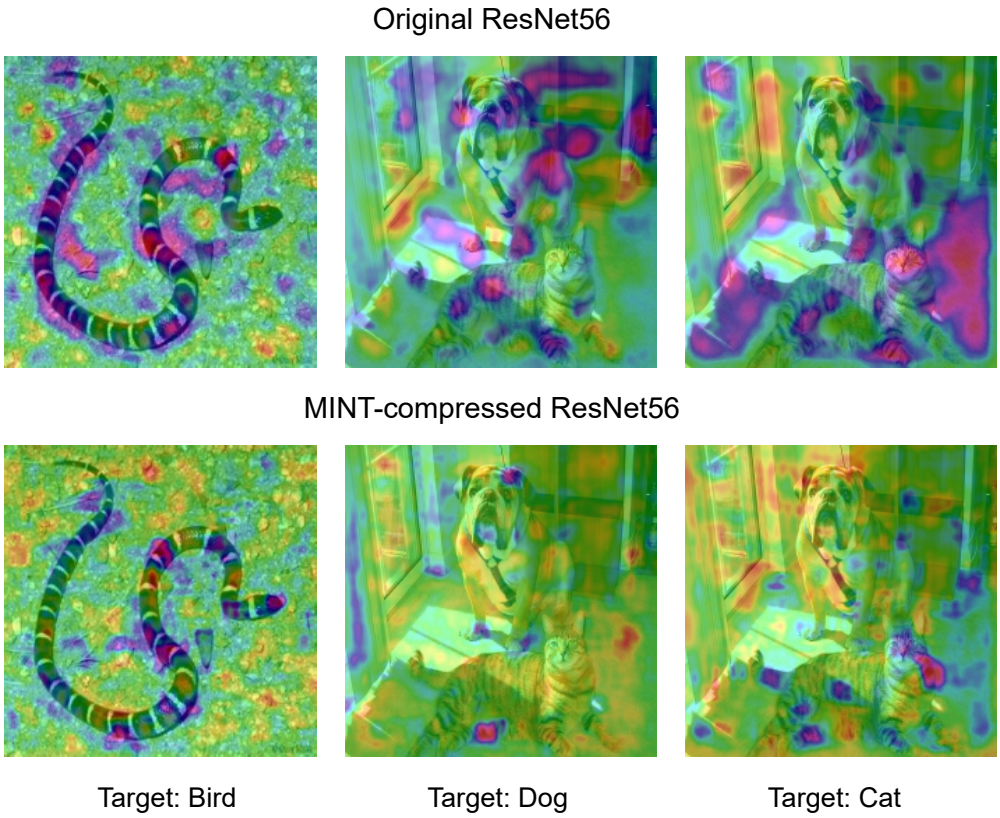
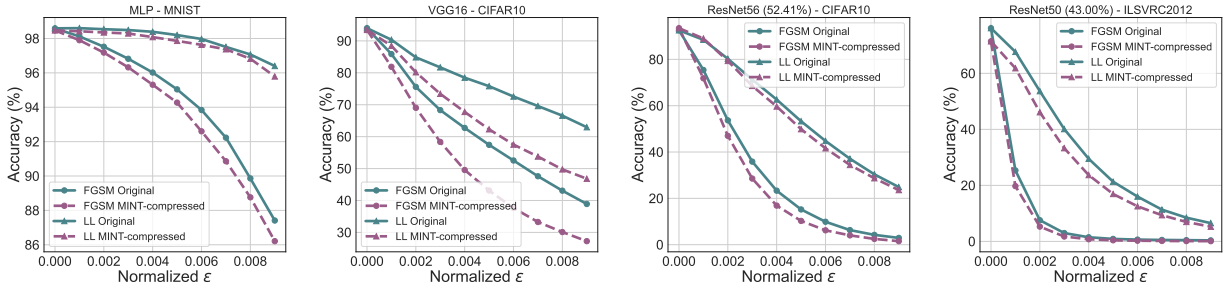


Figure 3.5: Visualizations using GradCAM [149] illustrate the decrease in effective portions of the image that contribute towards specific target classes in MINT-compressed ResNet56 (row 2) when compared to the original un-pruned network (row 1).

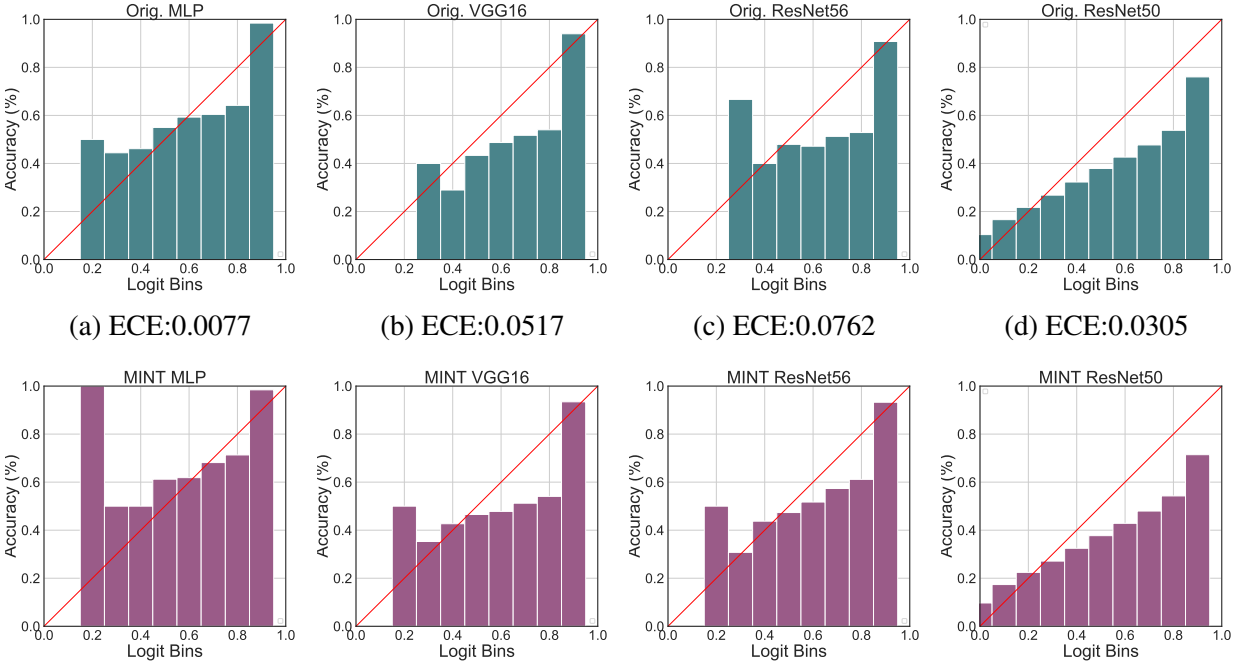
Feature Representations While the core idea behind MINT is to retain filters that contribute the majority of the information passed to the next layer, in using a subset of the available filters we remove a certain portion of the information passed downstream. Fig. 3.5 compares the portions of the image that contribute towards the desired target class, using GradCAM [149], between the baseline (top row) and MINT-compressed networks (bottom row). We observe that the use of a subset of filters in the compressed network has reduced the effective portions of the image that contribute towards a decision. In addition, there are minor modifications to the features captured as well. This idea is illustrated by the removal of contributions from the background portions of the image, in the baseline network, and an emphasis on the jowls of the dog in the MINT-compressed network (middle column).

Adversarial Susceptibility To understand the impact of pruning networks in the context of adversarial attacks, we use two common adversarial attack algorithms, Iterative FGSM [150], which doesn't exclusively target a class, and Iterative-LL [151], which targets the selection of the least likely class. Fig. 3.6 shows the response of the original and MINT-compressed networks to both



(a) (b) (c) (d)

Figure 3.6: By enforcing the use of an important subset of filters from all the available ones, MINT-compressed networks begin to overvalue their importance. MINT-compressed networks seem more susceptible to targeted and non-targeted adversarial attacks when compared to the original network. Here, ϵ refers to the ϵ ball in l_∞ norm.



(a) ECE:0.0077 (b) ECE:0.0517 (c) ECE:0.0762 (d) ECE:0.0305

(e) ECE:0.0054 (f) ECE:0.0500 (g) ECE:0.0383 (h) ECE:0.0069

Figure 3.7: We observe that MINT-compressed networks act as a regularizer to decrease the Expected Calibration Error (ECE), when compared to the original network and better match the ideal curve. Here, calibration statistics measure the agreement between the confidence output of the network and the true probability. The red line indicates the ideal trend.

attacks. We clearly observe that MINT-compressed networks are more vulnerable to targeted and non-targeted attacks. We posit that the reduction in the number of filters used and in the available redundant features are the reason MINT-compressed networks are vulnerable to adversarial attacks.

Calibration Error Calibration statistics [152], [153] measure the agreement between the confidence provided by the network and the actual probability. These measures provide an orthogonal perspective to adversarial attacks since they measure statistics only for in-domain images, while adversarial attacks alter the input. Fig. 3.7 highlights the decrease in Expected Calibration Error (ECE) for the MINT-compressed networks in comparison to their original counterparts. The plot illustrates that the histogram trend is closer to matching the ideal trend indicated by the linear red curve. After pruning, the sparse networks seem to behave similar to a regularizer by focusing on a smaller subset of features and decreasing the ECE. On the other hand, the original networks contain many levels of redundancies which could translate to overfitting and having higher ECE.

3.4 Conclusion

In this work, we introduced MINT as a novel approach to network pruning in which the dependency between filters of adjacent layers, computed using conditional GMI, is used as a measure of importance. This idea helps retain filters that pass the majority of the information through layers, thereby maintaining high test set accuracy. By building a stochastic measure of dependency, MINT-compressed networks achieve highly competitive pruning performances to SOTA baselines. This, in spite of competing against methods which use multiple fine-tuning/retraining steps while MINT uses a single prune-retrain step. When characterizing the behavior of MINT-compressed networks, we observe that it behaves like a regularizer and improves the calibration error of the network. However, a reduction in the number of filters used and redundancies makes pruned networks susceptible to adversarial attacks.

CHAPTER 4

Hybrid Neural Network Pruning

4.1 Motivation

Our prior work in MINT helped establish the use of mutual information (MI) as a means to yield high-performing pruned neural networks. However, the use of a minimum spanning tree (MST) algorithm as the backbone for computing MI creates a large bottleneck in terms of computational complexity. Specifically, the computational complexity of an MST is $\mathcal{O}(E \log V)$, where vertices (V) correspond to the number of neurons in a group. In addition, the dimensionality of an MST is also affected by the number of activations used to approximate the information content of a neuron. Both these factors force long run times during the pruning phase. While the commonly accepted trade-off for pruning algorithms is that training time cost is less important than test-time impact, in this dissertation we strive to consistently improve the efficiency of the pruning process and further reduce the amount of time spent in the pruning phase.

When reviewing the two broad approaches to neural network pruning, we observed that pruning approaches based on applying deterministic constraints on weight matrices are straightforward to implement and leverage the underlying structure of the weight matrices, but they often do not account for the downstream impact of pruning filters. On the other hand, probabilistic frameworks focus on reducing the redundancy between layers using information theoretic measures or variational bayesian inference but are not fast or efficient at modeling the sensitivity of filters at an individual level. In a sense, the two types of methods are converses: one's weakness is remedied by the other. Yet, to the best of our knowledge, there is no recent work that combines both approaches and improves upon them. This provides an opportunity to simultaneously leverage the strengths of both approaches to pruning while mitigating their weaknesses.

Finally, there are several unresolved practical issues that exist in pruning pipelines, e.g., the labor-intensive process of analyzing the sensitivity of different layers to pruning or determining the upper limit on pruning percentage for each layer of the DNN. Each of these issues requires repetitive testing and evaluation, which consumes a lot of time, resources, and energy from both a human and computational point of view. If we include the number of resources and time spent in

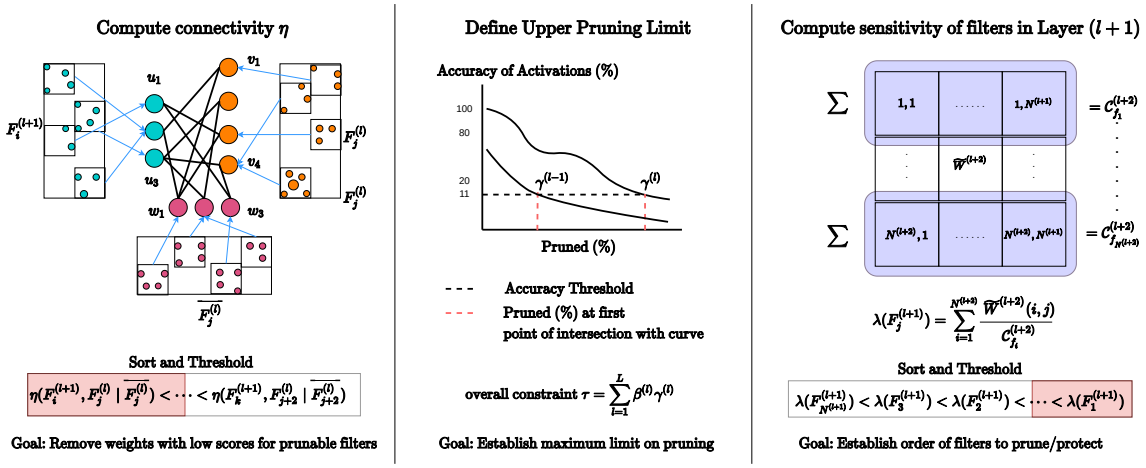


Figure 4.1: Illustration of the three major components of SNACS that help prune connections between layer l and $l+1$. First, we propose the hash-based ACMI estimator to compute the connectivity scores between filters in layer $l+1$ and all the filters in layer l. These connectivity scores are thresholded to obtain the set of filters that we prune. Next, to protect the network from being excessively pruned, we define a custom set of operating constraints, based on the degradation of activation quality at various pruning levels, to decide on the upper pruning percentage limit for layer $l+1$. Finally, we compute the sensitivity of filters in $l+1$ as the sum of normalized weights between chosen filters in layer $l+1$ and all the filters in layer $l+2$. We sort and threshold the sensitivity values to create a subset of sensitive filters that we protect from pruning. Combining the information from all three highlighted components, we prune layer $l+1$.

iteratively pruning and fine-tuning DNNs, the computational overhead to execute a pruning pipeline is exorbitant. By tackling these practical issues, we can not only reduce the resources and time consumed to prune DNNs, but we can also pave the way to fully automating the pruning pipeline.

4.2 Slimming Neural Networks Using Adaptive Connectivity Scores

Building atop the probabilistic framework proposed in MINT, we propose *Slimming Neural networks using Adaptive Connectivity Scores* (SNACS) as a hybrid single-shot pruning method in which we unify the benefits of the two broad categories of pruning approaches while solving practical issues associated with the pruning pipeline. Fig. 4.1 outlines the three main components of our proposed algorithm.

Firstly, we introduce the *Adaptive Conditional Mutual Information* (ACMI) measure, which incorporates weights as a scaling function within the framework of conditional mutual information [154], [155]. The ACMI measure evaluates the connectivity between pairs of filters across adjacent layers and prunes unimportant filters. In this work, we explore different ways to balance the contributions from the weight matrices and the mutual information computed from activations.

Secondly, to remove the manual effort involved in setting the upper pruning limit of each layer

in a DNN, we jointly try to optimize them using a custom set of operating constraints. These constraints are based on the degradation in quality of activations at various levels of compression, as dictated by the ACMI measure. By jointly constraining the optimization of various layers in a DNN, the upper pruning limits follow the hierarchical pattern of features learned in neural networks, where we can prune the layers closest to the output the most.

Finally, we encapsulate the importance of a filter using our newly proposed *Sensitivity* criterion, defined as the sum of a filter's contributions (normalized weights) to filters in the succeeding layer. Using this measure, we curate a subset of relatively less sensitive filters that can be pruned based on their connectivity scores, while we protect highly-sensitive filters from any form of pruning. The sensitivity criterion behaves like a strong prior in deciding the set of filters that contribute important information and thus need to be protected from any form of pruning. In the following sections, we outline our algorithm before describing each component in detail.

4.2.1 Algorithm-specific Notations

We assume that a given DNN has a total of L layers where,

- SENSITIVE FILTERS() : Function that returns the indices of a subset of filters that need to be protected from pruning, computed using sensitivity (Section 4.2.5).
- $F_i^{(l+1)}$: Activations from the selected filter i in layer $l+1$. We also overload this notation to represent the indexing scheme of the selected filter.
- $N^{(l+1)}$: Total number of filters in layer $l+1$.
- $S_{F_i^{(l+1)}}$: The set of filter indices whose values are pruned from the weight vector.
- $\eta()$: Connectivity score between two filters computed using ACMI (Section 4.2.3).
- $\overline{F_j^{(l)}}$: Activations from the set of all filters excluding $F_j^{(l)}$ in layer l. We also overload this notation to represent the indexing scheme of the selected filters.
- δ: Threshold on connectivity scores to ensure only strong connections are retained.
- $\gamma^{(l+1)}$: Upper limit on pruning percentage for layer $l+1$ defined using the constraints in Section 4.2.4.
- $W^{(l+2)}$: Weight matrix of layer $l+2$.
- $\widetilde{W}^{(l+2)}(i,j)$: Indexing element in i^{th} row and j^{th} column of the weight matrix of layer $l+2$ averaged over the height and width dimensions.

4.2.2 Core Algorithm

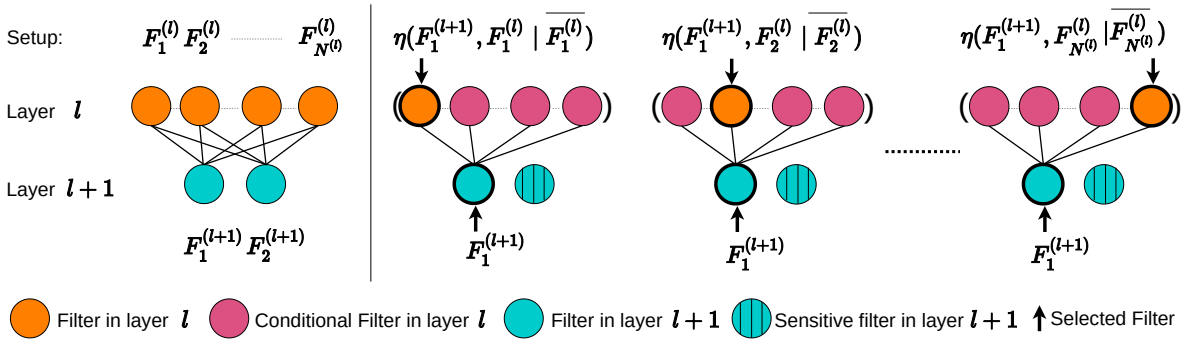


Figure 4.2: An illustrative example of computing ACMI, $\eta()$, between activations of filters in layers $l+1$ and l. In each $\eta()$ computation, the arrows indicate the filters between which we compute the connectivity score while taking into consideration the activations from the remaining filters in layer l. These steps are repeated for every possible pair of filters except for highly sensitive filters in layer $l+1$, where η need not be computed since their connections (lines between filters) are not pruned.

ALGORITHM 4: SNACS pruning between filters of layers $(l, l+1)$

for *Every pair of layers* $(l, l+1), l \in 1, 2, \ldots, L-1$ **do**
  Compute $\gamma^{(l+1)}$;
  for $F_i^{(l+1)}, i \in \{1, 2, \ldots N^{(l+1)}\} \setminus$ SENSITIVE_FILTERS $\left(\{1, 2, \ldots N^{(l+1)}\}\right)$ **do**
    Initialize $S_{F_i^{(l+1)}} = \emptyset$;
    for $F_j^{(l)}, j \in 1, 2, \ldots N^{(l)}$ **do**
      Compute $\eta(F_i^{(l+1)}, F_j^{(l)} | \overline{F_j^{(l)}})$;
      if $\left(\eta(F_i^{(l+1)}, F_j^{(l)} | \overline{F_j^{(l)}}) \leq \delta$ ***and*** $\sum_i |S_{F_i^{(l+1)}}| / (N^{(l+1)} N^{(l)}) < \gamma^{(l+1)}\right)$ **then**
        $S_{F_i^{(l+1)}} = S_{F_i^{(l+1)}} \cup \text{index}(F_j^{(l)})$
      end
    end
  end
end

The overall goal of SNACS is to find the set of filters that contribute minimally to the flow of information between layers and prune their values from the weight matrix. Similar to MINT, we apply SNACS between every pair of filters in adjacent layers of a pre-trained DNN where,

- We identify a subset of sensitive filters in layer $l+1$ that need to be protected from pruning and iterate over the remaining insensitive filters in layer $l+1$.
- To measure the connectivity score, η, between filters in layers l and $l+1$, we apply our proposed hash-based ACMI estimator to the activations from each set of filters. Fig.4.2 shows

an example of this process. The connectivity score evaluates the strength of the relationship between two filters in the context of contributions from all the remaining filters in layer l.

- If the connectivity score is lower than a threshold level δ, and the number of pruned filters does not exceed the pre-determined upper limit, denoted by $\gamma^{(l+1)}$, we add the index of the filter to $S_{F_i^{(l+1)}}$. The weights for retained and protected filters/neurons are untouched, while we zero out the weights for the entire kernel/elements in pruned filters/neurons.

In the practical implementation of Alg. 4, we determine the value of δ by thresholding η values from a chosen layer to remove sufficient weights and match the predetermined γ^{l+1}. Once we prune the filters that contribute the least across all the layers of the DNN, we proceed to re-training the network using a setup that mirrors the training phase of the pre-trained DNN. Across Alg. 4, we note that SNACS does not contain a continual feedback loop to update weights when pruning. Instead, we take only a single retraining pass after pruning. Compared to iterative pruning approaches, which often continually fine-tune to compensate for the performance lost due to pruning, SNACS falls firmly in the domain of single-shot pruning methods.

Complexity of Algorithm There are 2 primary factors which affect the complexity of Alg. 4, 1) the number of groups associated with each layer l and $l+1$, and 2) the total number of layers in the DNN. The internal double **FOR** loop has an upper bound of $\mathcal{O}(N^{(l)}N^{(l+1)})$ if the number of groups defined matches the number of filters in each layer. The outer **FOR** loop, used to iterate over pairs of adjacent layers, is executed a total of $L-1$ times.

4.2.3 Adaptive Conditional Mutual Information

In the following sections, we introduce Adaptive Mutual Information, a non-linear dependency measure that is based on f-divergence [155], [156], extend it to a conditional formulation, and discuss the hash-table-based estimator used to compute ACMI.

Definition Let $\mathcal{X}$ and $\mathcal{Y}$ be Euclidean spaces and let P_{XY} be a probability measure on the space $\mathcal{X}\times\mathcal{Y}$. Here, P_X and P_Y define the marginal probability measures. Similar to Yuri *et al.* [154], for a given function $(x,y)\in\mathcal{X}\times\mathcal{Y}\mapsto\varphi(x,y)\geq 0$, the Adaptive Mutual Information (AMI), denoted by $I_\varphi(X;Y)$, is defined as,

$$I_\varphi(X;Y)=\mathop{\mathbb{E}}_{P_XP_Y}\left[\varphi(X,Y)g\left(\frac{dP_{XY}}{dP_XP_Y}\right)\right], \tag{4.1}$$

where $\frac{dP_{XY}}{dP_XP_Y}$ is the Radon-Nikodym derivative, and $g:(0,\infty)\mapsto\mathbb{R}$ is a convex function and $g(1)=0$. Note that when $\frac{dP_{XY}}{dP_XP_Y}\to 1$ then $I_\varphi\to 0$.

Bounds on AMI Recall the definition of AMI (Eqn. 4.1). For the particular case of g, $g(t) = \frac{(t-1)^2}{2(t+1)}$, we have

$$I_\varphi(X;Y) = \frac{1}{2} \mathop{\mathbb{E}}_{P_X P_Y} \left[\varphi(X,Y) \left(\frac{dP_{XY}}{dP_X P_Y} + 1 \right) \right] - 2 \mathop{\mathbb{E}}_{P_X P_Y} \left[\varphi(X,Y) h \left(\frac{dP_{XY}}{dP_X P_Y} \right) \right], \tag{4.2}$$

where $h(t) = \dfrac{t}{t+1}$. When $\frac{dP_{XY}}{dP_X P_Y} = 1$, then the minimum value of I_φ is zero. Further, when P_{XY} and $P_X P_Y$ have no overlapping space then the second term in (4.2) becomes zero. Therefore, bounds on I_φ is given as,

$$0 \leq I_\varphi(X,Y) \leq \frac{1}{2} \mathop{\mathbb{E}}_{P_X P_Y} \left[\varphi(X,Y) \left(\frac{dP_{XY}}{dP_X P_Y} + 1 \right) \right]. \tag{4.3}$$

Adaptive Conditional Mutual Information Let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ be Euclidean spaces and let P_{XYZ} be a probability measure on the space $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. We presume $P_{XY|Z}$, $P_{X|Z}$, and $P_{Y|Z}$ are the joint and marginal conditional probability measures, respectively. P_Z defines the marginal probability measure on the space $\mathcal{Z}$. Following Yuri *et al.* [154], the Adaptive Conditional Mutual Information (ACMI), denoted by $I_\varphi(X;Y|Z)$, is defined as,

$$I_\varphi(X;Y|Z) = \mathop{\mathbb{E}}_{P_Z P_{X|Z} P_{Y|Z}} \left[\varphi(X,Y,Z) g \left(\frac{dP_{XY|Z}}{dP_{X|Z} P_{Y|Z}} \right) \right]. \tag{4.4}$$

In SNACS, we focus on the particular case of $g(t) = \frac{(t-1)^2}{2(t+1)}$, as introduced in Berisha and Hero [157]. Using this formulation allows for $I_\varphi \in [0,1]$ and symmetric behaviour, which are critical to the functioning of our algorithm, as well as a number of other properties which allow for statistical analysis as highlighted in Berisha *et al.* [158]. Note that when $\varphi = 1$, the ACMI in (4.4) becomes the conditional geometric MI measure proposed in Salimeh and Hero [159]. Next, we propose a hash-based estimator of ACMI to approximate the connectivity score between filters.

Hash-based Estimator of ACMI Consider N i.i.d samples $\left\{(X_i, Y_i, Z_i)\right\}_{i=1}^N$ drawn from P_{XYZ}, which is defined on the space $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. We define a dependence graph $G(X,Y,Z)$ as a directed multi-partite graph, consisting of three sets of nodes V, U, and W, with cardinalities denoted as $|V|$, $|U|$, and $|W|$, respectively and with the set of all edges E_G. The variable W here is different from the DNN weight matrix. Following similar arguments to Morteza *et al.* [88], we map each point in the sets $\mathbf{X} = \{X_1, \ldots, X_N\}$, $\mathbf{Y} = \{Y_1, \ldots, Y_N\}$, and $\mathbf{Z} = \{Z_1, \ldots, Z_N\}$ to the nodes in the sets V, U, and W, respectively, using the hash function H.

Here, $H(x) = H_2(H_1(x))$, where the vector valued hash function $H_1 : \mathbb{R}^d \mapsto \mathbb{Z}^d$ is defined as $H_1(x) = [h_1(x), \ldots, h_1(x_d)]$, for $x = [x_1, \ldots, x_d]$ and $h_1(x_i) = \lfloor \frac{x_i + b}{\epsilon} \rfloor$, for a fixed $\epsilon > 0$, and

random variable $b \in [0, \epsilon]$. The random hash function $H_2 : \mathbb{Z}^d \mapsto \mathcal{F}$ is uniformly distributed on the output $\mathcal{F} = \{1, 2, \ldots, F\}$ where for a fixed tunable integer c_H, $F = c_H N$.

After the projection of values on to the dependence graph $G(X, Y, Z)$, we define the following cardinality,

$$N_{ijk} = \#\{(X_t, Y_t, Z_t) \;\; s.t. \;\; H(X_t) = i, \\ H(Y_t) = j, H(Z_t) = k\}, \tag{4.5}$$

which is the number of joint collisions of the nodes (X_t, Y_t, Z_t) at the triple (v_i, u_j, ω_k). Let N_{ik}, N_{jk}, and N_k be the number of collisions at the vertices (v_i, ω_k), (u_j, ω_k), and ω_k, respectively. By using N_{ijk}, N_{ik}, N_{jk}, and N_k, we define the following ratios,

$$r_{ijk} := \frac{N_{ijk}}{N}, \;\; r_{ik} := \frac{N_{ik}}{N}, \;\; r_{jk} := \frac{N_{jk}}{N}, \;\; r_k := \frac{N_k}{N}. \tag{4.6}$$

Finally, using the above ratios we propose the following hash-based estimator of the ACMI measure (4.4):

$$\widehat{I}_\varphi(X; Y|Z) = \sum_{e_{ijk} \in E_G} \varphi(i, j, k) \; \frac{r_{ik} \, r_{jk}}{r_k} g\left(\frac{r_{ijk} \, r_k}{r_{ik} \, r_{jk}}\right), \tag{4.7}$$

summed over all edges e_{ijk} of $G(X, Y, Z)$ having non-zero ratios.

Theorem 1. *For given $g(t) = \frac{(t-1)^2}{2(t+1)}$ and under the assumptions:* **(A1)** *The support sets $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are bounded.* **(A2)** *The function φ is bounded.* **(A3)** *The continuous marginal, joint, and conditional density functions are belong to Hölder continuous class, [160]. For fixed d_X, d_Y, and d_Y, as $N \to \infty$ we have*

$$\widehat{I}_\varphi(X; Y|Z) \longrightarrow I_\varphi(X; Y|Z), \quad a.s. \tag{4.8}$$

The proof of Theorem 1 is available in the Appendix A.

Implementation Overall, X, Y, and Z denote sets of activations derived from different filters, and we obtain a scalar value (connectivity score) as the outcome of the ACMI estimator in (4.7). The flexibility in defining function φ offers a way to connect the probabilistic framework of MI to existing weight-based pruning approaches. In the experimental results, we explore a variety of options for φ and empirically determine that a function defined on the weight matrix helps achieve the highest pruning performance.

Complexity By extending the discussion provided in Morteza *et al.* [88], we find that the estimation process is dependent on two main factors, the total number of samples, N, and the dimensionality of each sample. From the original paper, we find that the computational complexity

is linearly dependent on the number of samples as well as the dimensionality of the samples. In our setup the dimensionality of a sample is capped by $\overline{F_j^{(l)}}$ which includes activations from all the filters in a layer excluding j. The exact value of this variable is dependent on the neural network architecture over which ACMI is calculated.

4.2.4 Defining Upper Pruning Limit of Layers

To protect different layers of the DNN from being excessively pruned, we propose a set of operating constraints to automate the joint definition of the upper pruning percentage limits of every layer in the DNN. Our approach follows the trends in degradation of the quality of activations when we prune a layer to varying extents. At each layer, we collect the performances of an SVM model with an RBF kernel ($\alpha_c^{(l)}$), trained on a subset of activations from the un-pruned version of the layer and tested on the same subset from the pruned version of the layer at various compression levels c, where $c \in \{1, 2, 3, \ldots, 99\}$. Here, we use the ground-truth labels from the dataset to train the SVM model.

Once we have the performances of SVM models across all layers, we cycle through them, from $100 - 0\%$, to find the optimal threshold value such that the sum of compression levels of all the layers, dictated by the selected threshold, adds up to our overall target pruning percentage. Each layer's pruning percentage is dictated by the highest compression level where the SVM model's performance exceeds the chosen threshold. The general trend we observe is higher the compression level, the lower the SVM model's performance. Thus, picking smaller performance thresholds leads to the selection of higher compression levels in a layer. We select the highest compression level from a range of possible values to avoid noisy and inconsistent behavior in SVM performances. Mathematically, we optimize,

$$\tau = \sum_{l=1}^{L} \beta^{(l)} \gamma^{(l)}, \tag{4.9}$$

where, $\beta^{(l)}$ is the ratio of the number the of parameters in layer l to the total number of parameters across the DNN, and τ denotes the desired pruning percentage across the DNN. Fig. 4.3 illustrates this process using an example of four layers.

It is important to note that the statistics computed from the SVM models across all layers can be executed in parallel, at an average of 36s per SVM model. This is an important distinction in comparison to prior work, where optimization involves computing the permutation of pruning percentages across various layers (order of 99^l). Across each such permutation, the entire network needs to be retrained/fine-tuned, which can take anywhere from a couple of hours (CIFAR-10) to a week (ILSVRC-2012). This cost is significantly higher when compared to the simple forward pass across the DNN and training time for an RBF-SVM model used in our approach. Our core

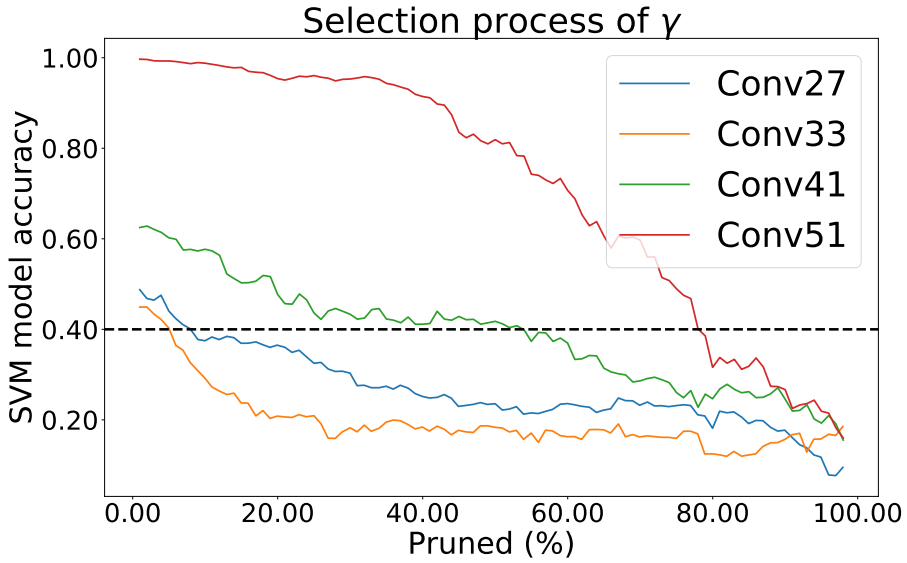


Figure 4.3: The selection process for upper pruning limits for each layer of a DNN is based on using a fixed threshold (dotted line) over the SVM models' performances such that the weighted sum of Pruned(%) allocated to each layer, the x coordinate where the threshold intersects the curve latest, matches the overall sparsity τ.

contribution to this work is a systematic approach to deciding the upper pruning percentage limits across all layers of the DNN. Previous works often relegate this information to the final chosen values without disclosing how they arrived at them. We provide the γ values for all layers of each DNN architecture in Section 4.3.2.

4.2.5 Sensitivity of Filters

A common assumption made during pruning is that all filters in a layer have the same downstream impact and hence can be characterized solely using the magnitude of their weights. In contrast, probabilistic pruning aims to maintain the flow of information between a pair of layers, but they consider all filters equally important. Taking into account each filter's impact on succeeding layers is an effective tool to assess their importance and protect filters that contribute the majority of information from being pruned.

We define a sensitivity criterion, $\lambda(F_i^{l+1})$, that can be used to sort filters in their order of importance. Using this, we curate a subset of filters that are critical and hence need to be protected from pruning while the remaining filters are pruned using the steps in Alg. 4. To evaluate the sensitivity of filters in layer $l+1$, we look at the weight matrix of its downstream layer $l+2$, $W^{(l+2)}$, and assess the contributions from filters in $l+1$ to those in $l+2$. Here, $W^{(l+2)} \in \mathbb{R}^{N^{(l+2)} \times N^{(l+1)} \times H \times W}$, where H and W are the height and width of the filters in layer $l+2$. For a given filter, the sum

of normalized contributions across all the filters in $l+2$ is its overall sensitivity, $\lambda(F_i^{(l+1)})$. It is defined as,

$$\lambda(F_i^{l+1}) = \sum_{f_c=1}^{N^{(l+2)}} \widetilde{W}^{(l+2)}(f_c, i) / \mathcal{C}^{(l+2)}(f_c), \tag{4.10}$$

$$\text{where } \mathcal{C}^{(l+2)}(f_c) = \sum_{f_p=1}^{N^{(l+1)}} \widetilde{W}^{(l+2)}(f_c, f_p). \tag{4.11}$$

Here, $\mathcal{C}^{(l+2)}$ is the normalization constant used to relate the weights of filters from $l+1$ contributing to the same filter in $l+2$, and $\widetilde{W}^{(l+2)}$ is the weight matrix of $l+2$ averaged over the height and width.

Once we obtain the order of sensitivity values for filters in a given layer, we define a threshold of highly-sensitive filters that remain untouched after empirically comparing the improvement in performance at similar pruning levels with and without protecting sensitive filters. This comparison is critical to ensure that only sensitive filters, which contribute the majority of the information downstream, remain untouched. This idea also helps improve the overall compression performance since less sensitive filters can be pruned more without compromising the quality of information flowing between layers to a large degree. After empirically comparing the degradation in performance of the SVM model, between the case when all the filters are pruned and the case when we protect a variable percentage of sensitive filters, we determine the set of highly-sensitive filters to protect from pruning and return their indices to Alg. 4.

4.3 Evaluation

We divide our results into three subsections, formatted as an ablative study. Section 4.3.3 focuses on the validation of the ACMI estimator and evaluation of its run-time and choice of φ, to highlight the impact of using our ACMI estimator in place of the MST-based estimator used in MINT. Here, the upper pruning limits are manually defined, with the help of artificial limits placed on the SVM model accuracy, to mimic prior work. In Section 4.3.4, we detail the results of applying SNACS (ACMI + Automated upper pruning percentage limits) across four Dataset-DNN combinations. Within this section we focus on drawing strong comparisons against single-shot pruning approaches while also highlighting how competitive SNACS is amongst approaches that use a modified objective function or iterative pruning. Finally, in Section 4.3.5 we discuss the impact of adding our sensitivity measure as a way to prioritize and fully protect important filters from being pruned. We begin by outlining the various datasets, preprocessing techniques, models and metrics used in our experiments. Our implementation is available at `https://github.com/MichiganCOG/SNACS`.

4.3.1 Datasets, Preprocessing, Models and Metrics

Datasets and Preproceesing We use two distinct datasets in our experiments, CIFAR-10 [143] and ILSVRC2012 [144]. CIFAR-10 dataset is a 10 class subset of the original 80 million tiny images dataset. The dataset split contains 50000 images for training, split as 5000 images/class, and 10000 images for testing where there are 1000 images/class. Each image in the dataset is originally $32 \times 32 \times 3$. For preprocessing, we randomly crop the image after padding 4 pixels, then we randomly flip the image horizontally before normalizing its values using mean (0.4914, 0.4822, 0.4465) and std. (0.2470, 0.2435, 0.2616) for each channel respectively. During testing, we normalize the images and provide them to the DNN.

The ILSVRC2012 dataset contains 1000 different classes of images totalling to about 1.2 million images overall for training and 50000 images for validation. The number of images per class varies between 732 to 1300. For preprocessing, we randomly crop the image in to $224 \times 224 \times 3$, then we randomly flip the image horizontally before normalizing its values using mean (0.485, 0.456, 0.406) and std. (0.229, 0.224, 0.225) for each channel respectively. During testing, we resize the original image to $256 \times 256 \times 3$, take a center crop of size $224 \times 224 \times 3$ before normalizing it and providing it to the DNN.

Models We evaluate VGG16 [145], MobileNetv2 [161], and ResNet56 [146]'s performance on CIFAR-10, and ResNet50's on ILSVRC2012. These represent the common dataset-DNN benchmarks used to validate the performance of pruning algorithms.

Metrics We use three distinct metrics to evaluate the quality of our pruning algorithm.

- **Pruning (%)**: The percentage of parameters removed when compared to the total number of parameters in the un-pruned DNN (Conv and FC only).
- **Test Accuracy (%)**: The best performance on the testing set, upon training, for baseline networks, and re-training, for pruning methods.
- **FLOPs** (%): The percentage of FLOPs reduced when compared to the non-pruned original DNN.

Apart from the above metrics, we also use *run-time* to compare speed of estimators. An ideal pruning algorithm is expected to have a high value for all metrics list above.

4.3.2 Experimental Setup

Throughout our experiments we use four major Dataset-DNN combinations, CIFAR10-VGG16, CIFAR10-ResNet56, CIFAR10-MobileNetv2 and ILSVRC2012-ResNet50. Table 4.1 lists the main

	VGG16	ResNet56	MobileNetv2
Epochs	300	300	350
Batch Size	128	128	128
Learning Rate	0.1	0.01	0.1
Schedule	90, 180, 260	150, 225	150, 250
Optimizer	SGD	SGD	SGD
Weight Decay	0.0005	0.0002	0.00004
Multiplier	0.2	0.1	0.1

Table 4.1: Training setups used to obtain pre-trained network weights.

	VGG16	ResNet56	ResNet50
Epochs	300	300	100
Batch Size	128	128	64
Learning Rate	0.1	0.1	0.1
Schedule	[90, 180, 260]	[90, 180, 260]	[30, 60, 90]
Optimizer	SGD	SGD	SGD
Weight Decay	0.0005	0.0005	0.0001/0.00003
Multiplier	0.1	0.2	0.1
Label Smoothing	0.35	0.15	0.8,0.85,0.8,0.8

Table 4.2: Base retraining setup used to obtain final performance listed in Table 4.7.

hyper-parameters used to train the VGG16, ResNet56, and MobileNetv2 networks and obtain their baseline performances. Pre-trained weights for ILSVRC2012-ResNet50 are used to compute ACMI values. Tables 4.2 and 4.3 list the basic hyper-parameters used to retrain the VGG16, ResNet56, MobileNetv2 and ResNet50 networks and obtain their final performance.

Procedure for Upper Pruning Percentage Limit of Layers Across all the experiments, when using our set of operating constraints to define γ, we collect the performance of an SVM model across $c \in \{1, 2, \ldots, 99\}$.

4.3.2.1 Hyper-parameters for Evaluation of Estimator

Run-Time To compare the improvement offered by our hash-based ACMI estimator, we choose the Minimum Spanning Tree-based (MST) CMI estimator from MINT as the nearest competitive baseline. In this experiment, we apply both estimators over the 9^{th} convolution layer of VGG16. To ensure fair comparison, we use ACMI with $\varphi = 1$ as well as $\|\text{weights}\|_2$ where weights are scaled

	MobileNetv2
Epochs	350
Batch Size	128
Learning Rate	0.1
Schedule	150,250
Optimizer	SGD
Weight Decay	0.00004
Multiplier	0.1
Label Smoothing	0.7

Table 4.3: Base retraining setup used to obtain final performance listed in Table 4.7.

	1	$\|\text{weights}\|_2$	$\|\text{weights}\|_2^2$	$\exp(\frac{-\|\text{weights}\|_2^2}{2})$	$\|\text{act}\|_2$
δ	0.9865	0.9925	0.9925	0.988	0.995
$\gamma^{(1)}$	00.00	00.00	00.00	00.00	00.00
$\gamma^{(2)}$	00.00	00.00	00.00	00.00	00.00
$\gamma^{(3)}$	21.02	21.02	21.02	21.02	00.00
$\gamma^{(4)}$	51.02	51.02	51.02	51.02	96.02
$\gamma^{(5)}$	61.03	51.02	51.02	71.02	51.02
$\gamma^{(6)}$	86.03	91.01	91.01	86.03	96.02
$\gamma^{(7)}$	91.01	91.01	91.01	91.01	86.03
$\gamma^{(8)}$	91.01	91.01	91.01	91.01	91.01
$\gamma^{(9)}$	96.02	96.02	96.02	96.02	96.02
$\gamma^{(10)}$	91.01	91.01	91.01	91.01	91.01
$\gamma^{(11)}$	91.01	91.01	91.01	91.01	91.01
$\gamma^{(12)}$	66.01	66.01	66.01	66.01	61.03
$\gamma^{(13)}$	91.01	91.01	91.01	91.01	91.01
$\gamma^{(14)}$	00.00	00.00	00.00	00.00	00.00
Pruned $(\%)$	84.02	84.12	84.17	**84.46**	76.13

Table 4.4: Hyper-parameters specific to the φ function used final performance; the best possible final performance $\geq 93.43\%$. Here, act refers to the activations and γ values are represented as $\%$.

to be between $[0, 1]$ within each layer, use the grouping formulation introduced in MINT as well as a manual threshold δ on the ACMI values. Here, we vary G values for both the layer l and $l + 1$ (8 and 9) over 16, 32, 64, 128 and 256. We use an average run-time from 10 trials, except for groups 128 and 256 for the MST-based estimator for which we use 2 trials. Most importantly, we set 200

	$\|\text{weights}\|_2\|\text{act}\|_2$	$\exp(-\frac{\|\text{weights}\|_2^2\|\text{act}\|_2^2}{2})$
δ	0.880	0.919
$\gamma^{(1)}$	00.00	00.00
$\gamma^{(2)}$	00.00	00.00
$\gamma^{(3)}$	41.01	36.03
$\gamma^{(4)}$	56.03	61.03
$\gamma^{(5)}$	61.03	56.03
$\gamma^{(6)}$	81.03	86.03
$\gamma^{(7)}$	86.03	96.02
$\gamma^{(8)}$	91.01	86.03
$\gamma^{(9)}$	96.02	91.01
$\gamma^{(10)}$	96.02	96.02
$\gamma^{(11)}$	91.01	81.03
$\gamma^{(12)}$	61.03	71.02
$\gamma^{(13)}$	91.01	86.03
$\gamma^{(14)}$	00.00	00.00
Pruned $(\%)$	82.59	76.99

Table 4.5: Hyper-parameters specific to the φ function used final performance; the best possible final performance $\geq 93.43\%$. Here, act refers to the activations and γ values are represented as $\%$.

samples per class which results in a total of 2000 samples of activations used by the estimators.

Selection of φ We implement a number of possible functions and evaluate them over the CIFAR10-VGG16 experimental setup. The exact hyper-parameters used to obtain ACMI values and the final test accuracy are provided in Tables 4.2, 4.3, 4.4 and 4.5. We maintain $G = 64$ throughout these experiments. The retraining performances are based on the highest Pruning $(\%)$ at which the model has a test accuracy that matches or exceeds 93.43% (from MINT).

4.3.2.2 Hyper-parameters for Large Scale Comparison

The basic setup to obtain the final results presented in Table 4.7 are listed under Tables 4.2 and 4.3. The main differences in the pruning setup between these experiments and the ones listed under Estimator evaluation are, 1) we avoid using a separate δ parameter and instead prune layers up to $\gamma^{(l)}$, and 2) we use label smoothing [113]. In Fig. 4.4, we illustrate the γ values obtained through our set of operating constraints used to define the upper pruning percentage limit for all layers in the DNN.

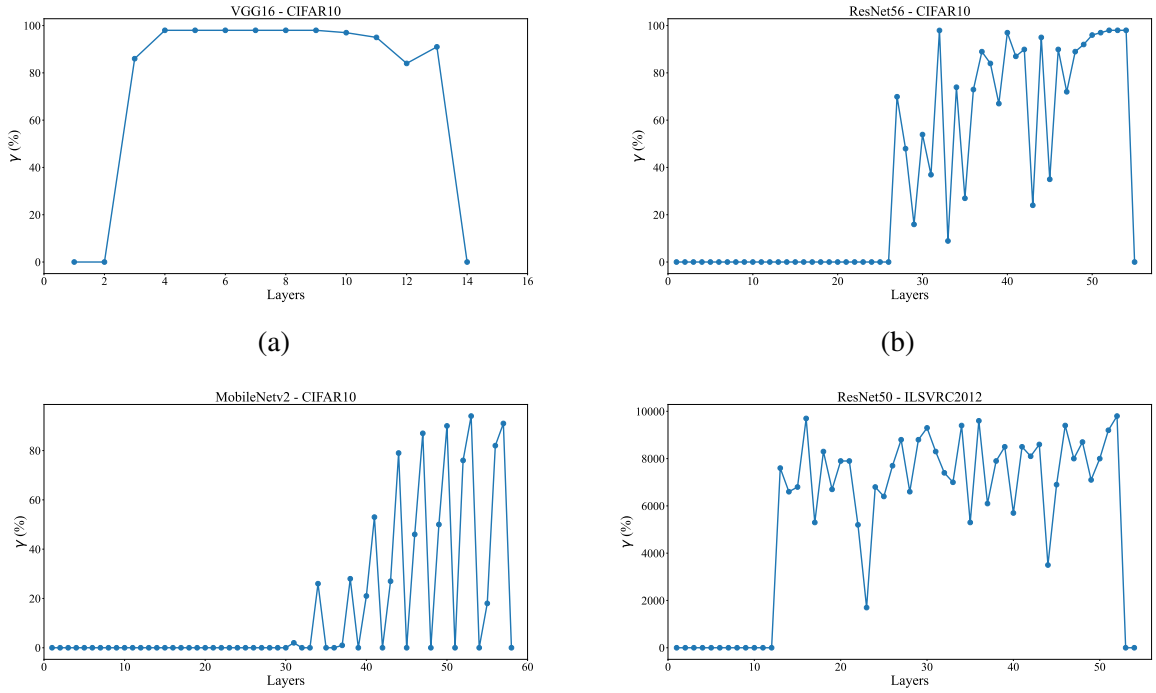
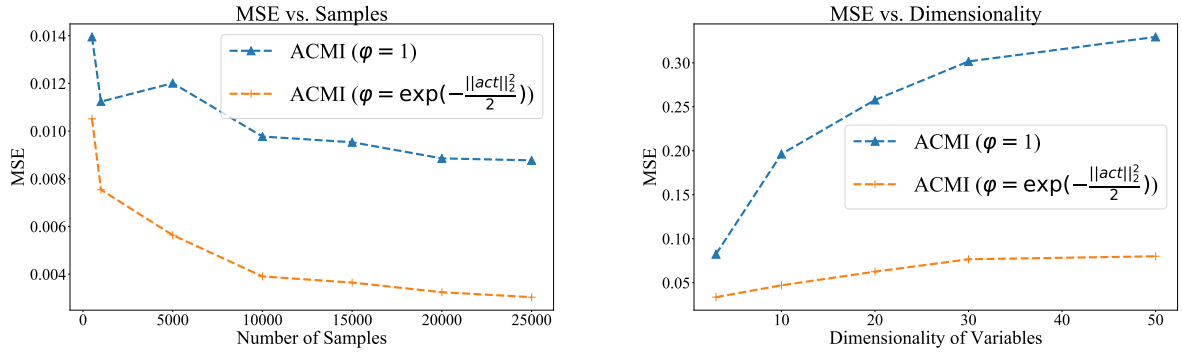


Figure 4.4: Illustration of the γ values obtained through our operating constraints used to define the upper pruning percentage limits for a DNN.



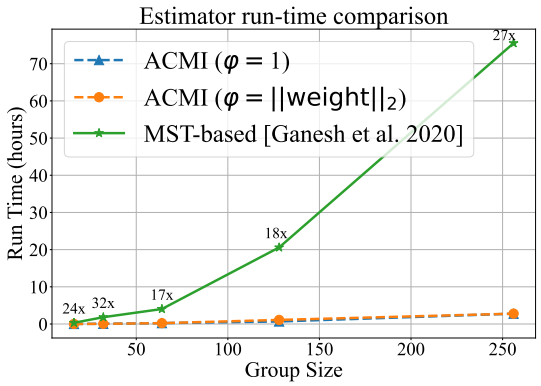
(a) MSE vs. No. of Samples

(b) MSE vs. Dimensionality

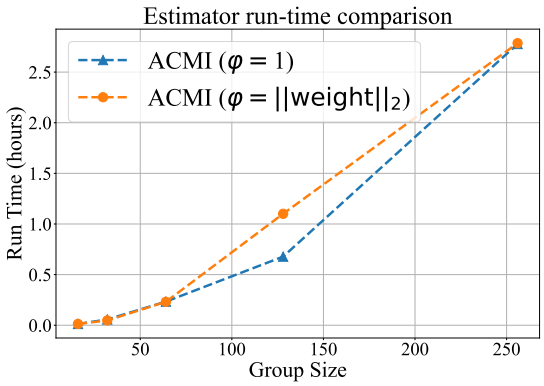
Figure 4.5: (Fig. 4.5a) An increase in the number of samples while dimensionality of input variables are held constant shows steadily decreasing MSE. (Fig. 4.5b) Increasing the dimensionality of input variables while the total number of samples are constant shows a steady decline of the MSE. Overall, the trends observed in both experiments match the expectations from a valid estimator.

4.3.3 Estimator Evaluation

Validation To observe the performance of the estimator when the number of samples is varied, we set the dimensionality of X, Y to one, and Z to two. This setup is used to mimic the dimensionality



(a) Run-time comparison across MST and ACMI measures



(b) Run-time comparison across various φ

Figure 4.6: (4.6a) When comparing run-times between the MST-based estimator used in SNACS and our hash-based ACMI estimator, our estimator provides up to 27× speedup in run-time. (4.6b) Across different selections of the scaling function in our estimator, the run-times scale similarly as the number of groups increases.

difference, at a small scale, in our experiments. We vary the number of samples in the range $\in \{500, 1000, 5000, 10000, 15000, 20000, 25000\}$. To observe the impact of a change in dimensionality on the estimator's performance, we restrict the total number of samples to 5000 and vary the dimensions of X, Y, and Z across $\{3, 10, 20, 30, 50\}$. In both the setups, we sample data from a multivariate normal distribution where the covariance matrix is set to identity and $\mu = 0$.

Fig. 4.5 shows the results of our experiments where, in Fig. 4.5a, we observe a steady decrease in MSE as the number of samples is increased. This matches our expectation of a good estimator where an increase in the number of samples improves the overall estimation accuracy and thus, reduces the MSE. Fig. 4.5b illustrates the steady increase in MSE when the number of samples is held constant but the dimensionality of the input variables grows larger. Further, the trends from secondary curves with $\varphi = \exp(-\frac{\|\text{act}\|_2^2}{2})$ show that the inclusion of a scaling term improves the overall performance. Thus, our observations match the expected trends from a valid estimator.

Run-time Comparison We provide a comparison between the run-time taken to compute the dependency scores across convolution layer 9 in VGG16 using our proposed ACMI estimator and the MST-based estimator used in MINT. For this experiment, we use three distinct estimators, the MST-based estimator from MINT, our ACMI estimator with $\varphi = 1$ and $\varphi = \|\text{weight}\|_2$. Here, weight values are re-scaled between $[0, 1]$. To provide a fair comparison, we adopt the grouping concept introduced in MINT. From Fig. 4.6, we make two important observations, 1) run-time increases with an increase in group size across both estimators, and 2) relative to the run-time from the MST-based estimator, our estimator is faster by at least $17\times$. Thus, we show that our

φ function	Pruning (%)
constant $= 1$	84.02
$\|\text{weights}\|_2$	84.12
weights^2	84.17
$\exp(\frac{-\text{weights}^2}{2})$	**84.46**
$\|\text{act}\|_2$	76.13
$\|\text{weights}\|_2\|\text{act}\|_2$	82.59
$\exp(-\frac{\text{weights}^2\|\text{act}\|_2^2}{2})$	76.99

Table 4.6: We compare the maximum compression performance of a variety of φ functions when maintaining a test accuracy $\geq 93.43\%$. $\varphi = \exp(\frac{-\text{weights}^2}{2})$ performs the best, and we use this in all further experiments.

estimator significantly reduces the overall run-time required to compute conditional MI across a DNN. Further, the run-time for one of the largest computational bottlenecks is massively reduced irrespective of the scaling function used in ACMI.

Selection of φ There are several potential functions we can associate with φ. In Table 4.6, we illustrate a variety of functions and their performance, w.r.t. the Pruning (%) while maintaining an accuracy $\geq 93.43\%$ in the VGG16-CIFAR10 setup. The main differences between Section 4.3.3 and MINT [162] are the inclusion of ACMI and the manual definition of upper pruning percentage limits using artificially capped SVM model accuracies (0.8). From Table 4.6, we observe that most variants of φ outperform SNACS including $\varphi = 1$. Furthermore, we find that$\varphi = \exp(\frac{-\text{weights}^2}{2})$ performs the best when compared to all the options for φ we explore. Thus, we set this as the default φ throughout all further experiments.

4.3.4 Comparison Against Existing Work

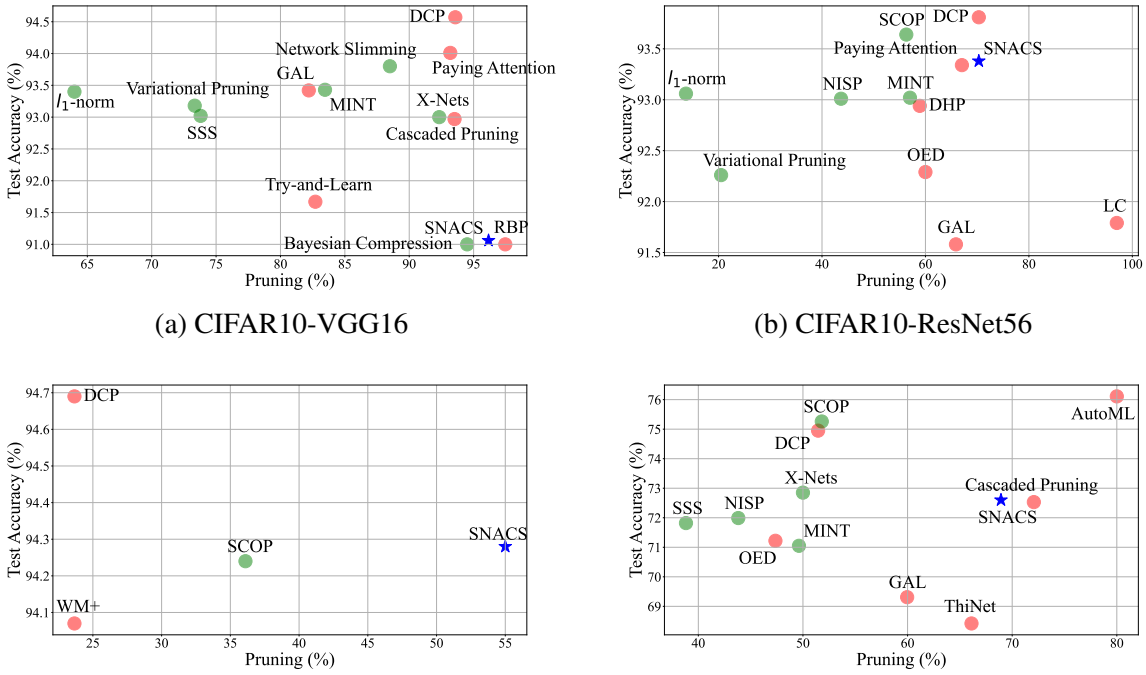
When compared to existing single-shot pruning methods, from Table 4.7, we observe that SNACS outperforms all of them by a significant margin to establish new SOTA performances. Our consistently high results establish our hybrid pruning framework as one of the top performing single-shot algorithms. A combination of improved estimates from the hash-based ACMI estimator (Table 4.6) and the joint definition of upper pruning percentage limits for each layer in the DNN are the main contributors to our high performance.

Fig. 4.7 helps put SNACS's performance in perspective of pruning approaches that use either sparsity-inducing objective functions or iterative re-training setups. In general, we expect a decrease in performance with an increase in the number of parameters pruned. Often, iterative approaches achieve the highest compression while suffering a minimal drop in testing accuracy, with methods

	Method	Pruning (%)	Test Acc. (%)	FLOPs (%)
CIFAR-10 VGG16	Baseline	N.A.	93.98	N.A.
	l_1-norm [59]	64.00	93.40	34.18
	Variational Pruning [77]	73.34	93.18	39.29
	SSS [71]	73.80	93.02	41.60
	MINT [162]	83.46	93.43	N.A.
	Network Slimming [67]	88.52	93.80	50.94
	X-Nets [163]	92.33	93.00	N.A.
	Bayesian Compression [78]	94.50	91.00	N.A.
	SNACS($\tau = 0.96$)	96.16	91.06	67.85
CIFAR-10 ResNet56	Baseline	N.A.	92.55	N.A.
	l_1-norm [59]	13.70	93.06	27.28
	Variational Pruning [77]	20.49	92.26	20.17
	NISP [61]	42.60	93.01	43.61
	FSDP [164]	50.00	92.64	N.A.
	SCOP [165]	56.30	93.64	56.00
	MINT [162]	57.01	93.02	N.A.
	SNACS($\tau = 0.685$)	68.59	93.38	36.89
CIFAR-10 MobileNetv2	Baseline	N.A.	93.66	N.A.
	SCOP [165]	36.10	94.24	40.30
	SNACS($\tau = 0.55$)	55.00	94.28	28.52
ILSVRC2012 ResNet50	Baseline	N.A.	76.13	N.A.
	SSS [71]	38.82	71.82	43.04
	NISP [61]	43.82	71.99	44.01
	MINT [162]	49.62	71.05	N.A.
	X-Nets [163]	50.00	72.85	50.00
	SCOP [165]	51.80	75.26	54.60
	SNACS ($\tau = 0.60$)	54.99	74.68	34.51
	SNACS ($\tau = 0.65$)	59.67	74.37	39.04
	SNACS ($\tau = 0.70$)	64.51	73.59	45.65
	SNACS ($\tau = 0.75$)	68.93	72.60	51.52

Table 4.7: Using a **single** train-prune-retrain cycle, SNACS is among the top performers across all the Dataset-DNN combinations. Baselines are ordered according to increasing Pruning (%).

that use joint optimization sprinkled across the entire range of Pruning (%) values. Single-shot methods are often the weakest performers, given that they get the fewest attempts to account for the loss in accuracy after pruning. However, across each dataset-DNN combination, our algorithm is



(a) CIFAR10-VGG16

(b) CIFAR10-ResNet56

(c) CIFAR10-MobileNetv2

(d) ILSVRC2012-ResNet50

Figure 4.7: Comparison of single-shot (green) vs. non single-shot (red) pruning approaches across our benchmarks. SNACS, despite being a single-shot approach, is highly competitive with the best performing iterative methods.

highly competitive with the best pruning approaches regardless of variations in optimizers, iterative pruning pipelines, modified objective functions, or layer-by-layer fine-tuning. SNACS remains competitive at high sparsity levels despite using a *single* prune-retrain step.

An important distinction between our pruning approach and other single-shot methods we compare against is that we avoid pruning early layers to a large extent, as shown in Fig. 4.8. Given that a large portion of FLOPs are concentrated in the early layers of the network, the percentage of FLOPs reduced by our SNACS is slightly lower when compared to methods like X-Nets [163], which preemptively prunes the network before training, or SSS [71], which optimizes a different objective function altogether. Interestingly, on closer inspection of Fig. 4.8, we observe a minimal correlation between the patterns of high and low γ values achieved in MINT and our work. While MINT showcases minimal pruning in the early and middle sets of layers, SNACS focuses on the middle and final set of layers, avoiding the early layers. We believe this variation stems from the fact that γ values in MINT are co-opted from prior works where the focus was on individual layers, while in SNACS, the joint definition of γs helps capture trends across multiple layers while trying to optimize the performance-sparsity tradeoff.

We observe that in SNACS, DNNs are more forgiving when pruning layers closer to the output

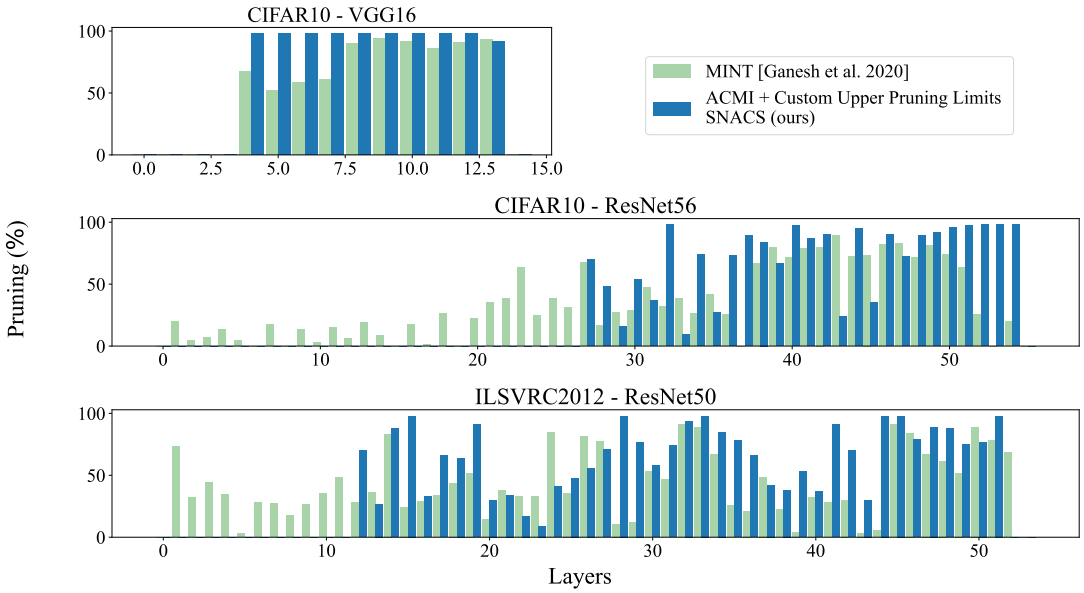
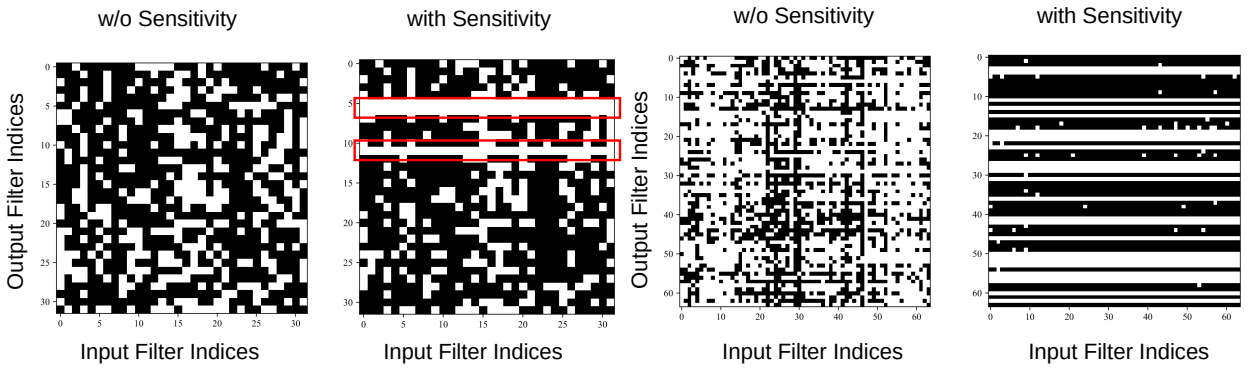


Figure 4.8: On observing the compression performance per layer in the ILSVRC2012-ResNet50 experiment, SNACS can achieve high Pruning (%) while focusing only on the middle and latter layers while avoiding the early layers. Interestingly, the pattern of pruning in MINT and SNACS is extremely different.

	Method	Pruning (%)	Test Accuracy (%)
CIFAR-10 ResNet56	Baseline	N.A.	92.55
	SNACS (ours)	68.59	93.38
	SNACS + sensitivity (ours)	**68.96**	**93.41**

Table 4.8: By saving a small percentage of sensitive filters, we can further improve the overall Pruning (%) while maintaining high Test Accuracy (%).

than input since the retraining phase allows them to overcome the loss of abstract concepts learned in later layers but not fundamental structures when compressing the earlier layers of the network. Our observations are matched by the discriminant scores in Gkalelis and Mezaris [164] and the median oracle ranking statistics per layer from Molchanov *et al.* [166]. However, these observations are in direct contrast to previous works which identify portions of the network closer to the input as being pruned first [55], [71]. We hypothesize that their outcomes stem from the modification of the objective function and subsequent training of the baseline network. In our approach and those in [164], [166], we remove filters based on a pre-defined criterion without modifying the loss function.



(a) Convolution 35 (b) Convolution 46

Figure 4.9: Illustrations of filters retained (white) and pruned (black) w/o and with sensitivity-based pruning. When protecting important filters from pruning, all its associated connections are maintained (red highlight). An interesting impact of sensitivity is that the connections pruned can be completely modified when compared to their counterpart w/o pruning. This is illustrated by the pruning mask of convolution 46.

4.3.5 Sensitivity Evaluation

Experiments in Sections 4.3.3 and 4.3.4 assume that all filters contributed equally to the information flow downstream and hence, the connectivity scores are the only constraint used for pruning. In this section, we highlight the impact of using the sensitivity criterion to prioritize the pruning of relatively weaker filters while protecting more sensitive filters from pruning on the CIFAR10-ResNet56 experimental setup. In Figs. 4.9a and 4.9b, we illustrate the 2D pruning masks generated by our algorithm, where the colors black and white represent filters that are removed and retained, respectively, and we observe three distinct behaviours. Firstly, when a filter is protected from pruning, an entire row representing all of its associated connections, are retained. Secondly, in addition to this we also observe an increase in the number of weights pruned from filters that are not protected. This is illustrated by an increase in the number of black pixels overall. Finally, when the sensitivity criterion is applied to layers which were previously not pruned to a large extent (Fig. 4.8 Convolution 32, 34, and many others) we observe a complete restructure in the way filters are pruned. Fig. 4.9b highlights this trend, which showcases an increase in the overall pruning of the layer as well as a stark difference in how it is pruned. All these observations put together lead to an overall improvement in the Pruning (%) with the inclusion of sensitivity, while maintaining high Test Accuracy (%) as shown in Table 4.8. In Table 4.9 we highlight the difference in γ values achieved in each case.

Across the results presented in Table 4.8, the percentage of filters protected from pruning is maintained at an optimal level. We determine the optimal combination of high sparsity and accuracy by constraining the % of filters saved to a value such that SVM model performance is higher than

	w/o Sensitivity	with Sensitivity
$\gamma^{(27)}$	0.699	0.699
$\gamma^{(28)}$	0.4794	0.4394
$\gamma^{(29)}$	0.1591	0.5507
$\gamma^{(30)}$	0.5390	0.7041
$\gamma^{(31)}$	0.3691	0.5117
$\gamma^{(32)}$	0.9794	0.1796
$\gamma^{(33)}$	0.089	0.3183
$\gamma^{(34)}$	0.7392	0.6611
$\gamma^{(35)}$	0.2695	0.4287
$\gamma^{(36)}$	0.7294	0.8261
$\gamma^{(37)}$	0.8896	0.7739
$\gamma^{(38)}$	0.8398	0.8198
$\gamma^{(39)}$	0.6699	0.799
$\gamma^{(40)}$	0.9699	0.9299
$\gamma^{(41)}$	0.8698	0.7927
$\gamma^{(42)}$	0.899	0.8999
$\gamma^{(43)}$	0.2399	0.2299
$\gamma^{(44)}$	0.9499	0.8957
$\gamma^{(45)}$	0.3498	0.5817
$\gamma^{(46)}$	0.899	0.8898
$\gamma^{(47)}$	0.7199	0.7099
$\gamma^{(48)}$	0.8898	0.8759
$\gamma^{(49)}$	0.9199	0.8813
$\gamma^{(50)}$	0.9599	0.9599
$\gamma^{(51)}$	0.9699	0.9699
$\gamma^{(52)}$	0.9799	0.9699
$\gamma^{(53)}$	0.9799	0.9799
$\gamma^{(54)}$	0.9799	0.9799
Compression$(\%)$	68.59	68.96

Table 4.9: Comparison of γ values in CIFAR10-ResNet56 when sensitive filters are protected.

the case when no filters are protected. The performance comparison is restricted to the SVM model only and no re-training is necessary. When we relax this constraint (Table 4.10), we observe that the performance levels drop by a significant amount while the sparsity level is lower than expected. This highlights the necessity of maintaining our constraints to obtain the optimal combination of high sparsity with accuracy.

Layer	% Saved	Sparsity (%)	Test Accuracy (%)
Layer 28	30	15.03	92.83
	34	15.03	93.05
	38	14.35	92.88
	45	**55.07**	**93.41**
	50	48.92	92.71
	54	45.89	93.24
	60	39.74	93.10
Layer 44	25	26.97	92.97
	30	54.83	93.13
	35	48.55	93.28
	40	**58.17**	**93.41**
	45	53.58	92.96
	50	48.99	93.50
	52.5	45.92	92.86

Table 4.10: Deviating the % of filters saved from our optimal constraints forces lower sparsity levels with bad testing performance. Optimal values are highlighted in **bold**.

4.3.6 Conclusion

In this work, we establish SNACS as a hybrid pruning framework where ACMI provides faster overall run-time and improved estimation accuracy. We offer new state-of-the-art levels of compression using a single train-prune-retrain cycle. At a fundamental level, by jointly constraining the definition of upper pruning limits across all layers of a DNN, we identify that early layers do not tolerate any form of pruning, and latter layers can be pruned to large extents. Finally, by using sensitivity as a strong prior for deciding which filters need to be pruned, we induce larger pruning performance. Our observations on the interplay between sensitivity and ACMI highlight an interesting direction for future work for hybrid pruning methods. Overall, we improve the run-time and performance of our previously established pruning framework while addressing a number of practical challenges associated with it.

CHAPTER 5

Incremental Label Curriculum

5.1 Motivation

Curriculum learning is a long-standing approach that extracts more performance with the same or lesser information. The main philosophy behind curriculum learning is the organization and scheduling of data with respect to difficulty. The notion of difficulty in samples has evolved since its original formulation, mirroring the perspective of human annotators, to being more model-centric. However, standard curriculum learning comes with the caveat of using only a subset of the data, be it easy or difficult, at varying segments of the training phase. This approach reduces the total amount of information made available to the DNN, not to mention the diversity in data while adding the overhead of identifying "difficult" samples. An alternative means of organizing training data while maintaining the entire training set could be leveraged to improve the learning process during the early training phase, thus ensuring higher generalization performance and improved quality of learned features.

In addition, traditional classification setups for DNNs find it difficult to learn and match the one-hot distribution of ground-truth vectors. We can attribute their inability to the function approximation generated using common activations as well as the content of the input image. Often, the image contains information pertaining to multiple categories of data in the input or feature embedding space, thus, making it more difficult to learn strict one-hot encodings. By smoothing out the one-hot distribution to include components that contribute to the content of the input image, the DNNs can learn to approximate distributions akin to what we would expect in the real world.

5.2 Learning from Incremental Labels and Adaptive Compensation

Inspired by an alternative outlook on Elman's [167] notion of "starting small", we propose LILAC, *Learning with Incremental Labels and Adaptive Compensation*, a novel algorithm that uses label-based curriculum and label-smoothing to regulate the learning process of DNNs. LILAC works in two phases, 1) *Incremental Label Introduction* (IL), which emphasizes gradually learning

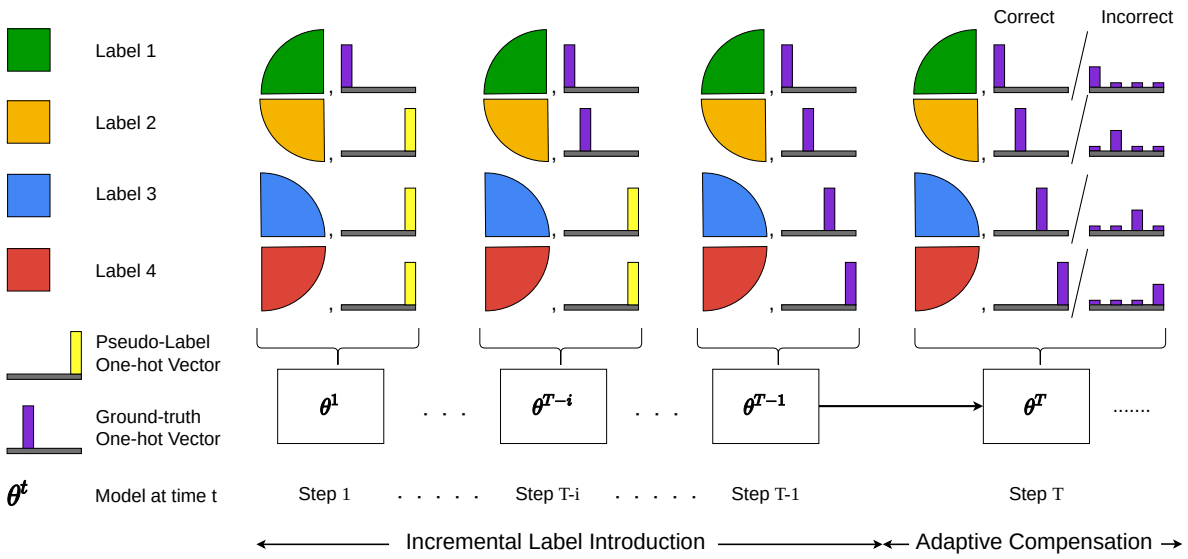


Figure 5.1: Illustration of the components of LILAC using a four-label dataset example. The *Incremental Label introduction* (IL) phase introduces new labels at regular intervals while using the data corresponding to unknown labels (pseudo-label) as negative samples. Once we have introduced all the labels, the *Adaptive Compensation* (AC) phase of training begins. Here, we use a prior copy of the network to classify training data. If a sample is misclassified, then a smoother distribution is used as its ground-truth vector in the current epoch.

labels instead of samples, and 2) *Adaptive Compensation* (AC), which regularizes the outcomes of previously misclassified samples by modifying their target vectors to smoother distributions (Fig. 5.1).

In the first phase, we partition data into two mutually exclusive sets: $\mathbb{S}$, a subset of ground-truth (GT) labels and their corresponding data; and $\mathbb{U}$, remaining data associated with a pseudo-label (ρ) and used as negative samples. Once we train the network using the current state of the data partition for a fixed interval, we reveal more GT labels and their corresponding data and repeat the training process. By contrasting data in $\mathbb{S}$ against the entire remaining dataset in $\mathbb{U}$, we consistently use all the available data throughout training, thereby overcoming one of the primary issues in curriculum learning. The setup of the IL phase, inspired by continual learning, allows us to flexibly space out the introduction of new labels and provide the network with enough time to develop a strong understanding of each class.

Once we reveal all the GT labels, we initiate the AC phase of training. In this phase, we replace the target one-hot vector of misclassified samples with a smoother distribution, thus limiting the alteration of target vectors to only necessary samples. We obtain the misclassified samples from a previous version of the network being trained. The smoother distribution provides easier values for the network to learn while using a previous copy of the network helps avoid external computational overhead. Our intended effect in using a smoother distribution is to increase the entropy of the

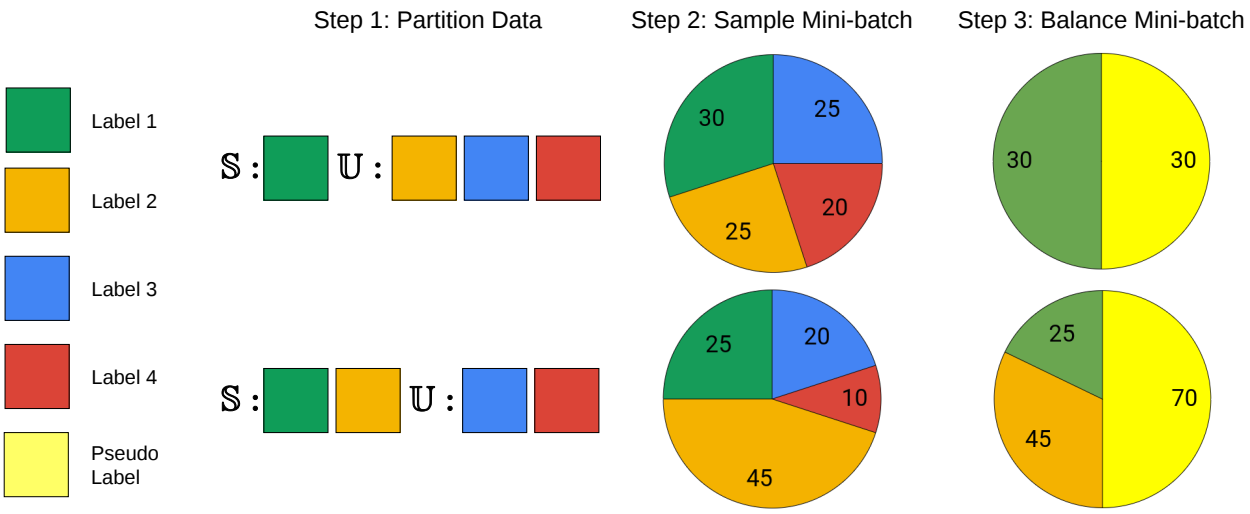


Figure 5.2: Illustration of the steps in the IL phase when (**Top**) only one GT label is in $\mathbb{S}$ and (**Bottom**) when two GT labels are in $\mathbb{S}$. The steps are 1) partition data, 2) sample a mini-batch of data and 3) balance the number of samples from $\mathbb{U}$ to match those from $\mathbb{S}$ in the mini-batch before training. Samples from $\mathbb{U}$ are assumed to have a uniform prior when being augmented/reduced to match the total number of samples from $\mathbb{S}$. Values inside each pie represent the number of samples. Across both cases, the number of samples from $\mathbb{S}$ determines the final balanced mini-batch size.

target vector. In the following sections, we describe the IL and AC phases in detail.

5.2.1 Incremental Label Introduction

In the IL phase, we partition data into two sets: $\mathbb{S}$, a subset of GT labels and their corresponding data; and $\mathbb{U}$, the remaining data marked as negative samples using a pseudo-label ρ. Over the course of multiple intervals of training, we reveal more GT labels to the network according to a predetermined schedule. Within a given interval of training, the data partition is held fixed, and we uniformly sample mini-batches from the entire training set based on their GT label. However, for samples from $\mathbb{U}$, we use ρ as their label. There is no additional change required in the objective function or the outputs of the model when we sample data from $\mathbb{U}$. By the end of this phase we reveal all GT labels to the network.

For a given dataset, we assume a total of L labels are provided in the ascending order of their value. Based on this ordering, we initialize the first b labels and their corresponding data as $\mathbb{S}$ and the data corresponding to the remaining $L - b$ labels as $\mathbb{U}$. Over the course of multiple training intervals, we reveal GT labels in increments of m, a hyper-parameter that controls the schedule of new label introduction. Revealing a GT label involves moving the corresponding data from $\mathbb{U}$ to $\mathbb{S}$ and using their GT label instead of ρ.

Within a training interval, we train the network for E epochs using the current state of the data

partition. First, we sample a mini-batch of data based on a uniform prior over their GT labels. Then, we modify their target vectors based on the partition to which a sample belongs. To ensure the balanced occurrence of samples from GT labels and ρ, we augment or reduce the number of samples from $\mathbb{U}$ to match those from $\mathbb{S}$ and use this curated mini-batch to train the network. After E epochs, we move m new GT labels and their corresponding data from $\mathbb{U}$ to $\mathbb{S}$ and repeat the entire process (Fig. 5.2).

5.2.2 Adaptive Compensation

Once all the GT labels have been revealed and the network has trained sufficiently, we begin the AC phase. In this phase, we use a smoother distribution for the target vector of samples that have been misclassified. Compared to one-hot vectors, optimizing over a smoother distribution, with an increased entropy, can bridge the gap between the unequal distances in the embedding space and overlaps in the label space [168]. This overlap can occur due to common image content or close proximity in the embedding space relative to other classes. Thus, improving the entropy of such target vectors can help modify the embedding space in the next epoch and compensate for the predictions in misclassified samples.

For a sample (x_i, y_i) in epoch $e \geq T$, we use predictions from the model at $e-1$ to determine the final target vector used in the objective function; specifically, we smoothen the target vector for a sample if and only if it was misclassified by the model at epoch $e-1$. Here, (x_i, y_i) denotes a training sample and its corresponding GT label for sample index i, and T represents a threshold epoch value until which the network is trained without adaptive compensation. We compute the final target vector for the i^{th} instance at epoch e, t_i^e, based on the model θ^{e-1} using the following equation,

$$t_i^e = \begin{cases} (\frac{\epsilon L-1}{L-1})\delta_{y_i} + (\frac{1-\epsilon}{L-1})\mathbb{1}, & \arg\max\left(f_{\theta^{e-1}}(x_i)\right) \neq y_i \\ \delta_{y_i}, & \text{otherwise} \end{cases} . \tag{5.1}$$

Here, δ_{y_i} represents the one-hot vector corresponding to GT label y_i, $\mathbb{1}$ is a vector of L dimensions with all entries as 1 and ϵ is a scaling hyper-parameter.

5.3 Evaluation

In this section, we discuss the performance of LILAC in the context of existing baselines (Section 5.3.4) before analyzing the impact of key hyper-parameters on the training setup (Section 5.3.5) and how they affect the IL and AC phases (Section 5.3.6). We begin by outlining the various protocols used to evaluate the performance of all the algorithms used in the experiments.

5.3.1 Datasets, Models and Metrics

Dataset We conduct our experiments on three distinct datasets, CIFAR-10, CIFAR-100 [143], and STL-10 [169]. CIFAR-10 and 100 are the 10 and 100 class subsets of the tiny images dataset, while STL-10 is a 10 class subset of the ILSVRC2012 dataset.

Models and Metrics We evaluate the performance of ResNet18 on CIFAR-10 and CIFAR-100, and ResNet34 [146] on STL-10 using *Average Recognition Accuracy (%)* over five trials and their *Standard Deviation*.

5.3.2 Baselines

1. Stochastic Gradient Descent with mini-batches (Batch Learning).
2. Standard Baselines
    - Fixed Curriculum: Following the methodology proposed in Bengio *et al.* [90], we create a "Simple" subset of the dataset, using data that are within a value of 1.1, as predicted by a linear one-vs-all SVR model. We train the deep network on the "Simple" dataset for a fixed period of time, which mirrors the total length of the IL phase, after which we use the entire dataset to train the network.
    - Label Smoothing: We follow the method proposed in Szegedy *et al.* [113].
3. Custom Baselines
    - Dynamic Batch Size (DBS): DBS randomly copies data available within a mini-batch to mimic variable batch sizes, similar to the IL phase. However, all GT labels are available to the model throughout the training process.
    - Random Augmentation (RA): This baseline samples from a single randomly chosen class in $\mathbb{U}$, available in the current mini-batch, to balance data between $\mathbb{S}$ and $\mathbb{U}$ in the current mini-batch. This approach is in contrast to LILAC, which uses samples from all classes in $\mathbb{U}$ that are available in the current mini-batch.
4. Ablative Baselines
    - *Only IL*: This baseline quantifies the contribution of incrementally learning labels when combined with batch learning.
    - *Only AC*: This baseline shows the impact of adaptive compensation, as a label smoothing technique, when combined with batch learning.

Parameters	CIFAR10/100	STL10
Epochs	300	450
Batch Size	128	128
Learning Rate	0.1	0.1
Lr Milestones	[90 180 260]	[300 400]
Weight Decay	0.0005	0.0005
Nesterov Momentum	Yes	Yes
Gamma	0.2	0.1

Table 5.1: List of hyper-parameters used to in batch learning. Note: All experiments use the SGD optimizer.

Types	Training	Performance (%)		
		CIFAR 10	CIFAR 100	STL 10
	Batch Learning	95.19 ± 0.190	78.32 ± 0.175	72.88 ± 0.642
Standard	Fixed Curriculum [90]	95.27 ± 0.112	77.89 ± 0.287	72.18 ± 0.601
	Label Smoothing [113]	95.27 ± 0.111	79.06 ± 0.179	72.55 ± 0.877
Custom	Random Augmentation	95.27 ± 0.076	75.37 ± 0.480	73.67 ± 0.708
	Dynamic Batch Size	95.22 ± 0.131	78.73 ± 0.264	72.66 ± 1.081
Ablative	*Only IL* (**ours**)	95.38 ± 0.135	78.73 ± 0.139	73.43 ± 0.903
	Only AC (**ours**)	95.38 ± 0.170	78.94 ± 0.179	72.94 ± 0.530
Overall	LILAC (**ours**)	$\mathbf{95.52 \pm 0.072}$	78.88 ± 0.201	$\mathbf{73.77 \pm 0.838}$
	LS + LILAC (**ours**)	95.34 ± 0.080	$\mathbf{79.08 \pm 0.307}$	73.59 ± 0.623

Table 5.2: Under similar setups, LILAC consistently achieves higher mean accuracy than batch learning across all evaluated benchmarks, a property not shared by other baselines.

5.3.3 Experimental Setup

In Table 5.1 we list the general hyper-parameters used to train the batch learning portion of every baseline. This setup covers the training beyond the IL phase for LILAC, DBS, RA, *Only IL* as well as the *Only AC* baseline. Across all the methods we ensure that the total number of training epochs, when all the labels in the dataset are known, is held constant.

5.3.4 Comparison Against Existing Work

Table 5.2 illustrates the improvement offered by LILAC over batch learning when using comparable setups. We further break down the contributions of each phase of LILAC. Both, *Only IL* and

Method	CIFAR-10
Wide Residual Networks [170]	96.11
Multilevel Residual Networks [171]	96.23
Fractional Max-pooling [172]	96.53
Densely Connected Convolutional Networks [173]	96.54
Drop-Activation [174]	96.55
Shake-Drop [175]	96.59
Shake-Drop + LILAC (ours)	**96.79**

Table 5.3: LILAC easily outperforms the Shake-Drop network ([175]) as well as other top performing algorithms on CIFAR-10 with *standard pre-processing (random crop + flip)*.

Only AC improve over batch learning, albeit to varying degrees. This highlights their individual strengths and importance. However, we observe a consistently high performance across all benchmarks only when we combine both phases. These results indicate that the two phases complement each other.

The Fixed Curriculum approach does not offer consistent improvements over the batch learning baseline across CIFAR-100 and STL-10, while the Label Smoothing approach does not outperform batch learning on the STL-10 dataset. Both of these standard baselines fall short, while LILAC consistently outperforms batch learning across all evaluated benchmarks. Interestingly, Label Smoothing provides the highest performance on CIFAR-100. Since the original formulation of LILAC is based on batch learning, we assume all GT vectors to be one-hot. Label Smoothing violates this assumption. Instead, when we tailor our GT vectors according to the Label Smoothing baseline, we outperform it with minimal hyper-parameter tuning, a testament to LILAC's applicability on top of conventional label smoothing.

The RA baseline highlights the importance of using all of the data in $\mathbb{U}$ as negative samples in the IL phase instead of using data from individual classes. This idea is reflected in the boost in performance offered by LILAC. The DBS baseline highlights the importance of fluctuating mini-batch sizes, which occur due to the balancing of data in the IL phase. Even with the availability of all labels and fluctuating batch sizes, the DBS baseline is easily outperformed by LILAC. This indicates the importance of the recursive structure used to introduce data in the IL phase and the use of data from U as negative samples. Overall, LILAC consistently outperforms batch learning across all benchmarks while existing comparable methods fail to do so. When we extend LILAC to the Shake-Drop [175] network architecture, with only standard pre-processing, we easily outperform other existing approaches with comparable setups, as shown in Table 5.3.

5.3.5 Hyper-parameter Empirical Analysis

Smoothness of Target Vector Throughout this work, we maintained the importance of using a smoother distribution as the alternate target vector during the AC phase. Table 5.4 (Top) illustrates the change in performance across varying degrees of smoothness in the alternate target vector. There is a clear increase in performance when ϵ values are between 0.7-0.4 (mid-range). On either side of this band of values the GT vector is either too sharp or too flat, which leads to a drop in performance.

Size of Label Groups We design LILAC to introduce as many or as few new labels as desired in the IL phase. We hypothesized that developing stronger representations can be facilitated by introducing a small number of new labels while contrasting them against a large variety of negative samples. Table 5.4 (Bottom) supports our hypothesis by illustrating the decrease in performance with an increase in the number of new labels introduced in each interval of the IL phase. Thus, we introduce two labels each for CIFAR-10 and STL-10 and only one new label per interval for CIFAR-100 throughout the experiments in Table 5.2.

Epochs in Training Interval When we vary E, the fixed training interval size in the IL phase, we observe a dataset-specific behaviour. For datasets with a lower number of total labels, a higher number of epochs provides better performance while, for datasets with more labels, a smaller number of epochs yields better performance. Holding the alternate learning rate consistent, pacing the introduction of new labels can have a tremendous impact on the subsequent hyper-parameters used in LILAC.

Label Order In Table 5.5, we compare three different orders of label introduction during the IL phase, 1) random label order, 2) difficulty-based label order, and 3) ascending label order. Here, difficulty-based label order is obtained from the overall classification scores per label, using the features from a trained model. Although these three orders do not constitute the exhaustive set of possible label orderings, within these three options there is no definitive order that boosts the performance of LILAC consistently. Thus, we employ ascending label order throughout our work.

5.3.6 Discussion: Impact of Each Phase

In this section, we take a closer look at the impact of each phase of LILAC and how they affect the quality of the learned representations. We extract features from the second to last layer of ResNet18/34 from 3 different baselines (Batch Learning, LILAC, and *Only IL*) and use these features to train a linear SVM model and a k-means clustering model with hungarian job assignment [176].

Figs. 5.3 and 5.4 highlight the two key phases of our algorithm. First, the plots on the left-hand side show a steady improvement in the performance of LILAC and the *Only IL* baseline once the IL

Property	Performance (%)		
	CIFAR-10	CIFAR-100	STL-10
$\epsilon = 0.9$	95.30 ± 0.072	78.48 ± 0.328	73.57 ± 0.980
$\epsilon = 0.8$	95.34 ± 0.141	78.52 ± 0.118	73.54 ± 0.984
$\epsilon = 0.7$	95.42 ± 0.189	78.72 ± 0.356	73.59 ± 0.872
$\epsilon = 0.6$	95.36 ± 0.096	78.75 ± 0.180	$\mathbf{73.77 \pm 0.838}$
$\epsilon = 0.5$	95.49 ± 0.207	78.88 ± 0.227	73.61 ± 0.810
$\epsilon = 0.4$	$\mathbf{95.52 \pm 0.072}$	$\mathbf{78.88 \pm 0.201}$	73.54 ± 0.959
$\epsilon = 0.3$	95.31 ± 0.125	78.66 ± 78.66	73.59 ± 0.955
$\epsilon = 0.2$	95.36 ± 0.095	78.47 ± 0.093	73.57 ± 0.963
m: 1	95.32 ± 0.156	$\mathbf{78.73 \pm 0.139}$	73.27 ± 0.220
m: 2 (4)	$\mathbf{95.38 \pm 0.135}$	78.34 ± 0.209	$\mathbf{73.43 \pm 0.903}$
m: 4 (8)	95.29 ± 0.069	78.37 ± 0.114	72.30 ± 0.543

Table 5.4: (**Top**) The mid-range ϵ values, 0.7-0.4, show an increase in performance, while the edges, due to either too sharp or too flat a distribution, show decreased performance. (**Bottom**) *Only IL* model results illustrate the importance of introducing a small number of new labels in each interval of the IL phase. Values in brackets are for CIFAR-100.

Property	Performance (%)		
	CIFAR-10	CIFAR-100	STL-10
$E = 1$	95.13 ± 0.175	78.21 ± 0.236	72.59 ± 0.476
$E = 3$	95.20 ± 0.200	$\mathbf{78.73 \pm 0.139}$	73.03 ± 0.380
$E = 5$	95.32 ± 0.044	78.57 ± 0.102	73.08 ± 0.996
$E = 7$	$\mathbf{95.32 \pm 0.156}$	78.44 ± 0.265	73.13 ± 1.460
$E = 10$	95.26 ± 0.185	77.98 ± 0.218	$\mathbf{73.27 \pm 0.220}$
Label Order: Rnd.	95.30 ± 0.146	78.35 ± 0.280	73.10 ± 0.861
Label Order: Difficulty	95.25 ± 0.156	78.42 ± 0.115	73.69 ± 0.849
Label Order: Asc.	95.32 ± 0.156	78.73 ± 0.139	73.27 ± 0.220

Table 5.5: (**Top**) Varying E, the fixed training interval size in the IL phase, shows a dataset-specific behavior, with the dataset with lesser labels preferring a larger number of epochs while the dataset with more labels preferring a smaller number of epochs. (**Bottom**) Comparing random label ordering and difficulty-based label ordering against the ascending order assumption used throughout our experiments, we observe no preference for any ordering pattern.

phase is complete and all the labels have been introduced to the network. When we compare the plots of CIFAR-10 and STL-10 against CIFAR-100, we see that all baselines follow the learning trend shown by batch learning, with CIFAR-100 lagging behind slightly. Since there are a large number of epochs required to introduce all the labels of CIFAR-100 to the network, the plots are significantly delayed compared to batch learning. Conversely, since there are very few epochs in the IL phase of CIFAR-10 and STL-10, we observe that the performance trend of *Only IL* and LILAC quickly matches that of batch learning. Overall, the final performances of both LILAC and the *Only IL* baseline are higher than batch learning, which supports the importance of the IL phase in learning strong representations.

The plots on the right-hand side highlight the similarity in behavior of *Only IL* and LILAC before AC. However, afterward, we observe that the performance of LILAC overtakes the *Only IL* baseline. This trend is a clear indicator of the improvement in representation quality when AC is applied. Additionally, from Fig. 5.3, we observe that inherently the STL-10 dataset results have a high standard deviation, which is reflected in the middle portion of the training phase, between the end of the IL and the beginning of the AC phase, and it is not a consequence of our approach. To further support the importance of the AC phase, we provide examples in Fig. 5.5 of randomly sampled data from the testing set that are incorrectly classified by the *Only IL* baseline and correctly classified by LILAC.

5.4 Key Takeaways

In this work, we proposed LILAC, which rethinks curriculum learning based on incrementally learning labels instead of samples. This approach helps kick-start the learning process from a substantially better starting point while making the learned embedding space amenable to adaptive compensation of target vectors. Both these techniques combine well in LILAC to show the highest performance on CIFAR-10 for simple data augmentations while easily outperforming batch and curriculum learning and label smoothing methods on comparable network architectures. Overall, we use the improved starting point to obtain a higher quality of solution, while maintaining the same number of epochs the DNN has complete knowledge of the GT label set. We provide an implementation of LILAC at `https://github.com/MichiganCOG/LILAC`.

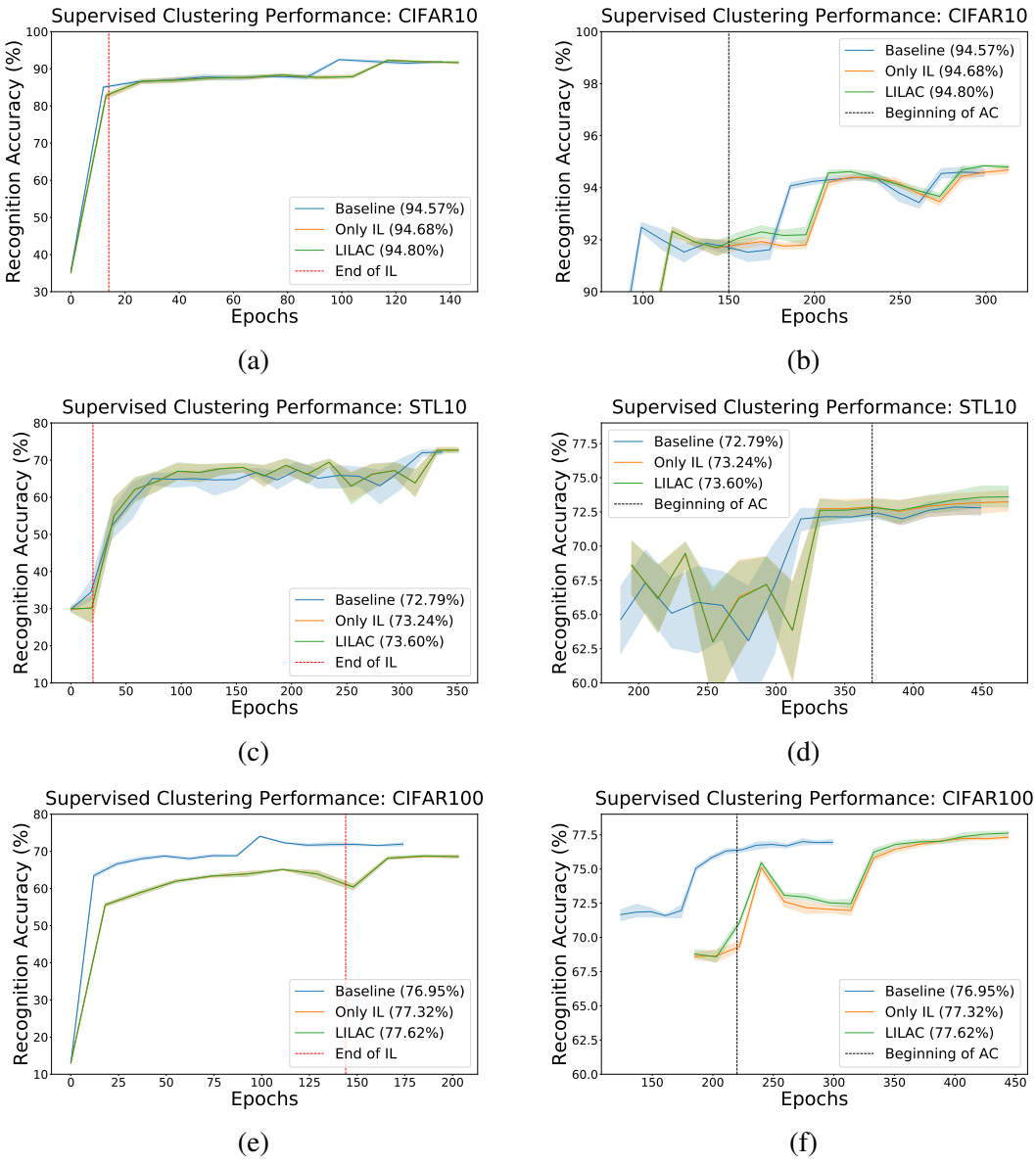


Figure 5.3: Plots on the (**Left**) show the common learning trend between all baselines, albeit slightly delayed for CIFAR-100, after the IL phase, while those on the (**Right**) show steady improvement in performance after applying AC when compared to the *Only IL* baseline. Final supervised classification performances on representations collected from LILAC easily outperform those from batch learning and *Only IL* methods.

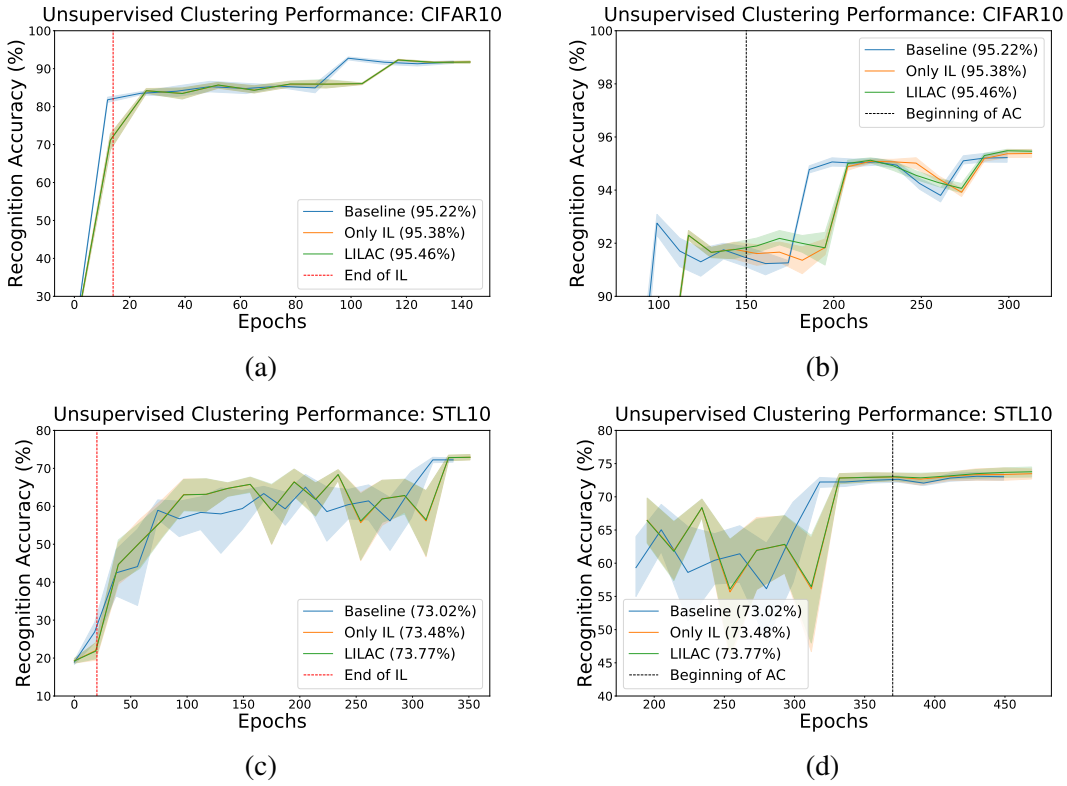
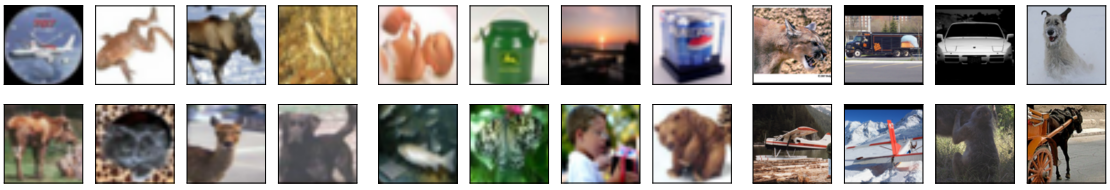


Figure 5.4: Unsupervised classification performance on representations collected from LILAC easily outperforms those collected from Batch Learning and *Only IL* methods. The plots on the left show the common learning trend between all baselines after IL while plots on the right show steady improvement in performance after applying AC when compared to the baselines.



(a) CIFAR-10 (b) CIFAR-100 (c) STL-10

Figure 5.5: Illustration of 8 randomly chosen samples that were incorrectly labelled by the *Only IL* baseline and correctly labelled by LILAC, highlighting the importance of AC.

CHAPTER 6

Targeting Performance, Efficiency and Robustness

6.1 Motivation

When developing deep neural network (DNN) solutions, accuracy or performance metrics are often a key point of emphasis. While performance is critical, the computational load and security of the final solution play an equally important role in a real-world setting. Our prior work on MINT and SNACS helped establish state-of-the-art benchmarks for one-shot pruning of neural networks resulting in low computational load at inference while maintaining high accuracy. However, when empirically analyzing the adversarial response of our pruned neural networks and standard mini-batch SGD, we observe that they are highly susceptible to being attacked. Thus, addressing their adversarial vulnerability is crucial to enable their widespread adoption in real-world settings.

While our work in pruning addressed efficiency from an architecture prototyping and inference perspective, the training process still scales uncompromisingly with the choice of model, dataset, GPU memory, and other factors. LILAC helped us understand the field of curriculum learning and how organizing and scheduling limited data can improve the learning process with minimal overhead. To further improve the efficiency of the training process, we explore the idea of finding an optimal subset of data to improve generalization performance [177]. By reducing the amount of data used to train a DNN we can simultaneously target improvements in performance and efficiency. We take it a step further by jointly constraining the development process to satisfy Performance (P), Efficiency (E), and Robustness (R) or PER goals so that we can significantly reduce the cost and resource consumption of DNN solutions while simultaneously making them more secure from attacks.

6.2 Concurrently Achieving Performance, Efficiency and Adversarial Robustness in Deep Neural Networks

To effectively tackle all three PER targets simultaneously, we propose CAPER, a method to Concurrently Achieve improvements in Performance, Efficiency, and Robustness. Our algorithm is

built on the assumption that there exists a subset of the original training data that negatively impacts the decision boundaries learned by the final model [177]. In CAPER, we identify and remove this subset of data by using a function of the distance between features, specifically between the original inputs and their noise-perturbed counterparts. With the addition of noise, a higher value of this heuristic indicates a large deviation from standard features, which correlates with samples easily affected by noise and closer to the learned decision boundaries. Thus, these samples have a large impact on the learning process. By removing these samples, we use only a subset of the available data to train the model. In doing so, we reduce the overall training time while regularizing the learning process, whereby we simultaneously improve the generalization performance and adversarial robustness.

CAPER's idea of retaining a subset of the training data draws strong parallels from conventional curriculum learning. Instead of using changes in loss/gradients/predictions, in CAPER, we use noise injection and the subsequent feature distance as a measure to remove samples. An interesting way to conceptualize our approach is by looking at it as the interaction between the dataset and DNN, where the DNN is held constant while we regularize the dataset by penalizing certain samples and removing them. This regularization effect on the learned decision boundary has the intended consequence of improved adversarial robustness. In this work, we analyze adversarial robustness from two perspectives, 1) where the source and target are the same model, and the alternative 2) where the source model can be different from the target. While the first scenario establishes robustness to a known model, the second scenario measures the robustness of DNNs to attacks designed on a variety of backbones, a property we define as in-Transferability. Through our experimental validation, we highlight the difference in how robustness is imparted to a DNN using CAPER when compared to standard adversarial training [27], [28], [178]. Keeping this distinction in mind, we build CAPER atop adversarial training regimes and highlight its complementary behavior.

In the following sections, we provide a summary of the standard training setup for classification before taking an in-depth look at CAPER. We begin by defining key notations.

6.2.1 Algorithm-specific Notation

- ϵ : Smoothing value for ground-truth label vector when loss is evaluated.
- τ : Training epoch at which CAPER is applied.
- m_i : Binary value indicating whether the i^{th} sample is retained or removed.
- $H()$: Function used to project features to lower dimensions.
- $D()$: Distance function.

- $\tilde{O}^{(l)}$: Sensitivity-based subset of filters used to capture features.
- ξ_i : Overall instability score for each sample in the training data.
- $\alpha()$: Window function that assigns multipliers to instability values from different layers.
- γ : Number of samples removed from training data.

6.2.2 Standard Setup

When training an L layer DNN for classification, the input variables are denoted by $\{(x_i, y_i)\}_{i=1}^{N} \sim (X, Y)$, where N represents the total number of samples. Here, $x_i \in \mathbb{R}^{H \times W \times 3}$, is the input RGB image and $y_i \in \{1, 2, \ldots, C\}$ is the ground-truth label in a dataset with C classes. The output of layer l is denoted by,

$$f^{(l)}(x_i^{(l-1)}) = \sigma(W^{(l)} x_i^{(l-1)} + b^{(l)}) \ , \tag{6.1}$$

assuming an activation function $\sigma()$, $f^{(l)} \in \mathbb{R}^{N \times O^{(l)} \times h^{(l)} \times w^{(l)}}$, where $O^{(l)}$ denotes the output dimension of layer l, $h^{(l)}$, $w^{(l)}$, $W^{(l)}$, and $b^{(l)}$ represent the output height, width, weights and biases of layer l, respectively. The general loss function used to train this setup is,

$$\mathcal{L}(X, Y) = \min_{W} \frac{1}{N} \sum_{i=1}^{N} \ell(F(x_i), y_i) \ , \tag{6.2}$$

where $F()$ denotes the output of the entire DNN. For classification, $\ell()$ is the multi-class cross-entropy loss.

6.2.3 Proposed Algorithm

In CAPER, we focus on removing a subset of the training data that negatively impacts performance. We begin by training a DNN using the complete training dataset up to τ epochs. At the chosen epoch $\tau << E$, where E is the total number of training epochs, we compare the distance between features of standard inputs and their noise-perturbed counterparts. Here, we generate the noise-perturbed counterparts by applying additive gaussian noise to the input images. The intuition behind this approach is that in identifying samples that are highly susceptible to noise, we highlight data points that are close to and have a strong impact on the decision boundary. Since these samples are readily susceptible to noise, removing them allows the DNN to learn better decision boundaries from a more regularized set of data, leading to improved generalization and adversarial robustness. In our approach, a large distance between the features highlights samples highly susceptible to noise. We use the distance values to generate a binary mask, remove those samples from the dataset and

continue to train using the remaining subset of data. We explain the processes underlying CAPER below.

6.2.3.1 Basic Setup

In CAPER, we modify the loss function used to learn the weights of the DNN by masking the contributions from the noisy subset of data after epoch τ:

$$\mathcal{L}(X, Y) = \min_{W} \frac{1}{||m||_0} \sum_{i=1}^{N} m_i \ell_\epsilon(F(x_i), y_i). \tag{6.3}$$

Here, $m \in \{0, 1\}^N$ is the binary mask vector defined using our heuristic based on the distance between features. Once we determine the value of m at epoch τ, it remains fixed throughout the remaining training epochs. An extremely small value of τ would force the capture features that aren't coherent, while large values of τ would significantly reduce the expected efficiency gain. Instead, we choose a relatively small but balanced value for τ to obtain coherent features and maximize our gain in efficiency. In addition, we use the cross-entropy loss modified by label smoothing [113] (ℓ_ϵ), where the smoothing operation on the one-hot ground-truth vector is,

$$y_{ls} = (1 - \epsilon) \times \mathbb{1}_y + \frac{\epsilon}{C} \ . \tag{6.4}$$

Here ϵ is the smoothing value and $\mathbb{1}_y$ is a one-hot vector at the ground-truth label.

6.2.3.2 Capturing Feature Distance

To ascertain the value of m, we begin by capturing the distance between features, specifically between the original input and their noise-perturbed counterparts, at a chosen epoch τ across different layers in the DNN. To generate the noise-perturbed counterparts, we apply additive gaussian noise to the input. Mathematically, we denote the capture of features from the desired layer l as,

$$\begin{aligned} f(x_i) &= \sigma(W x_i + b), \\ f(x_i + \delta_i) &= \sigma(W(x_i + \delta_i) + b). \end{aligned} \tag{6.5}$$

Here, $\delta_i \sim \mathcal{N}(0, 0.5)$, with dimensionality matching the input. Note: We drop the layer superscript to improve readability hereon. To avoid inconsistencies between the effects of applying δ_i independently at multiple layers, we apply δ_i directly on the image and observe its effects at downstream layers. Furthermore, to ensure that the noise is in the same feature space as the image, we apply the

noise to the normalized image.

Once we obtain the features from each layer, we compute the distance $D(.)$ between corresponding pairs of features. Here,

$$\Delta f(i) = D(H(f(x_i)), H(f(x_i + \delta_i))) = ||H(f(x_i)) - H(f(x_i + \delta_i))||_2, \tag{6.6}$$

where $H(.)$ is a projection function that maps the features into a lower dimensional space, and $\Delta f(i) \in \mathbb{R}^{1 \times O^{(l)}}$. The function $H : \mathbb{R}^{O^{(l)} \times h^{(l)} \times w^{(l)}} \rightarrow \mathbb{R}^{O^{(l)} \times P}$, where $P << h^{(l)} \times w^{(l)}$ and $O^{(l)}$ denotes the filter counts from layer l. While (6.6) depicts the l_2-norm version of the distance function, the formulation itself is not limited to it. Beyond capturing the distance, we further normalize $\Delta f(i)$ values across samples to ensure that the distances remain comparable. We propose normalizing them on a channel-wise basis using the following equation,

$$\Delta \hat{f}(i, q) = \frac{\Delta f(i, q) - \min\limits_{n \in 1, \ldots, N} \Delta f(n, q)}{\max\limits_{n \in 1, \ldots, N} \Delta f(n, q) - \min\limits_{n \in 1, \ldots, N} \Delta f(n, q)}. \tag{6.7}$$

Here, $i \in \{1, 2, \ldots, N\}$, $q \in \{1, 2, \ldots, O^{(l)}\}$, and $\Delta \hat{f}(i, q) \in [0, 1]$.

6.2.3.3 Sensitivity Constraint

When collecting features across all the filters of a layer (6.5) we implicitly make the assumption of uniform importance across all filters. However, from DNN pruning literature [59], [61] we know that there are a number of filters that provide redundant information. Reducing their contribution does not hurt the performance of DNNs. Following this line of thought, we adopt the notion of sensitivity from SNACS to capture features from a subset of filters that provide important information. While there are many different ways to utilize sensitivity, in this work we threshold the value of sensitivity to obtain a subset of the filters ($\tilde{O}^{(l)}$) from which we derive our features. Doing so allows us to leverage the learned structure of the weight matrices in identifying sensitive filters while reducing the overall memory consumed to store features. The exact number of filters used for each DNN backbone is provided in the experiment-specific setup (Section 6.3.2).

6.2.3.4 Computing the Binary Mask

While $\Delta \hat{f}$ captures the distance between features from a specific layer, we expand the formulation of CAPER to include sensitivity when aggregating distances across multiple layers of the DNN. To do so, we include $\xi_i^{(l)}$, the instability of a sample measured as the average $\Delta \hat{f}$ across filters in a

given layer.

$$\xi_i^{(l)} = \frac{\sum_{q=1}^{\tilde{O}^{(l)}} \Delta \hat{f}(i,q)}{\tilde{O}^{(l)}}, \quad i \in \{1, \ldots, N\}. \tag{6.8}$$

By combining the contributions of $\xi_i^{(1)}, \xi_i^{(2)}, \ldots, \xi_i^{(L)}$ across multiple layers we obtain the overall instability of a sample, ξ_i, given as,

$$\xi_i = \xi_i^{(1)}\alpha^{(1)} + \ldots + \xi_i^{(L)}\alpha^{(L)}, \tag{6.9}$$

where $\alpha()$ denotes a window function that provides scalar multipliers used to combined the instability values obtained from different layers.

To identify the optimal values of α we would have to solve the system of equations shown below,

$$\begin{bmatrix} \xi_1^{(1)} & \xi_1^{(2)} & \ldots & \xi_1^{(L)} \\ \vdots & \vdots & \ldots & \vdots \\ \xi_N^{(1)} & \xi_N^{(2)} & \ldots & \xi_N^{(L)} \end{bmatrix} \in \mathbb{R}^{(N \times L)} \begin{bmatrix} \alpha(1) \\ \vdots \\ \alpha(L) \end{bmatrix} \in \mathbb{R}^{(L)}_{\geq 0}, \tag{6.10}$$

where the final accuracy is the metric over which we need to optimize. Given the practical constraints in solving this system of equations, where the LHS is ill-defined and the size of the system matrix forces any operation on it to be expensive, we explore a restricted set of functions, including $\mathbb{1}_{1:\frac{L}{2}}$, $\mathbb{1}_{\frac{L}{2}:L}$, a gaussian distribution and finally $\mathbb{1}_L$, to find the best performing α. Once we set α, we can evaluate ξ_i. Using these values, we compute m as:

$$m_i = \begin{cases} 0 & \text{if } \xi_i \text{ is in the top } \gamma \text{ values of } \xi \\ 1 & \text{o.w} \ . \end{cases} \tag{6.11}$$

By controlling γ, we use m_i to reduce the amount of the training data held in memory and the overall FLOPs required during training. Once m is applied, the DNN is then trained with the remaining subset of data from epochs τ to E.

ILSVRC2012 Implementation CAPER scales across the size and depth of a DNN as well as the number of samples in a dataset. To efficiently execute CAPER on ILSVRC2012, which has over 1 million images, we re-purposed the algorithm to function in two phases instead of one. In the first phase, we compute $D(.)$ from (6.6) across samples of each label and summarize them using their mean value to ascertain the difference heuristic over labels. This is similar to assessing the prior over the ten worst performing labels. In the second phase, we refine our search space to samples across the ten labels with the highest difference in values and re-capture the heuristic across the samples from only these labels. Doing so allows us to avoid comparing statistics across a million

samples. Instead we simplify the comparison to samples across ten labels, which is approximately 13000. Thus, we reduce the overall amount of memory consumed.

6.3 Evaluation

The experimental results section is divided into three main parts, where each subsection aligns with one of our PER goals. The first discusses the performance of CAPER in the context of the state-of-the-art curriculum learning algorithm from Zhou *et al.* [48]. The second part emphasizes the adversarial robustness of CAPER, in the context of normal as well as adversarial training, under a variety of adversarial attacks. The third part demonstrates the improvement in efficiency.

6.3.1 Datasets, Models, Attacks, User-specific Hyper-parameters

We outline the datasets, DNNs, types of adversarial attacks, metrics and key hyper-parameters used across our experiments.

Datasets We use five primary datasets to evaluate our proposed method, CIFAR-10, CIFAR-100 [143], STL-10 [169], miniImagenet [179] and ILSVRC2012 [144]. Among these datasets, we restrict our adversarial robustness comparisons to CIFAR-10/100 to match existing literature. For miniImagenet, we use a custom-generated and balanced training-and-testing split that we make available alongside our code.

DNN Architectures We use four DNN architectures to evaluate CAPER in the context of standard curriculum learning, VGG16 [180], MobileNet [181], DenseNet [173], [182] and ResNet50 [16]. In addition to these architectures, we use ResNet18 and PreActResNet18 in adversarial robustness comparisons. We choose these networks to help represent a wide variety of architectural backbones. Each of the four main DNNs has two distinct versions, one suitable for the CIFAR datasets and another for the remaining datasets.[1]

Adversarial Attacks And Metrics We explore the effect of a variety of adversarial attacks like MIFGSM [183], FFGSM [139], DI2FGSM [184], APGDDLR [128], APGDCE, PGD [27] and CW [185] using the code from [134], [186]. To measure the performance of various algorithms, we use standard **Accuracy**(%) over the testing set. For adversarial robustness, we measure **Robust Accuracy**(%) over the perturbed testing set, illustrated by the radius of the polar plots. Finally, we use total **FLOPs**, measured as one pass over the entire DNN scaled across the entire training phase,

[1]Detailed descriptions of these model variants are provided at `https://github.com/MichiganCOG/Q_TART`.

to compare the improvement in efficiency across different training methods. Across all experiments, we provide average statistics over five trials for accuracies unless stated otherwise.

CAPER: Hyper-parameters Within CAPER, τ is an extremely important parameter that influences the amount of efficiency gain we expect. For experiments in Section 6.3.3, we set $\tau = 50$ for all DNN-Dataset combinations except ResNet50-CIFAR-10, for which we set it to 100. We generate results on ILSVRC12 using $\tau = 15$. Experiments under Section 6.3.4.1 use $\tau = 35$ and 15 when comparing against Rice *et al.* [131] and Cui *et al.* [132], respectively, and the remaining use $\tau = 50$. Throughout our experiments, we fix $H(.)$ as the mean value across the $h^{(l)} \times w^{(l)}$ channels. Within the ablation study used to understand the impact of alternative window functions, $\alpha() \in \{\mathbb{1}_{1:\frac{L}{2}}, \mathbb{1}_{\frac{L}{2}:L}$, a gaussian distribution, and finally $\mathbb{1}_L\}$. Apart from the ablation study, we use $\alpha = \mathbb{1}_L$ across the remaining experiments. Finally, we list all the values of γ, the number of samples removed from the training data, within each experimental subsection.

6.3.2 Experiment-specific Setup

For the purpose of repeatability we provide the hyper-parameters for different baselines used in our experiments below.

6.3.2.1 Curriculum Comparison

Tables 6.1 and 6.2, describe the hyper-parameters used for our baseline (SGD) models while Tables 6.3 and 6.4 describe the hyper-parameters used for the DIHCL algorithm [48]. For the ILSVRC2012 experiments, we use Epoch=100, Batch=64, Lr=0.1, Sched. = 30,60,90, Opt.=SGD, Decay=0.00003, Mult.=0.1 and Mtm= True, with $\tau = 15$. Code for the DIHCL algorithm was provided from `https://github.com/tianyizhou/DIHCL`. For CAPER, we re-use the hyper-parameters in Tables 6.1 and 6.2 while experimenting on values for γ and ϵ, after setting $\tau = 50$. Only for DenseNet on CIFAR-10 we set $\tau = 75$. The final values of ϵ and γ for the results in Tables 6.9 and 6.10 are,

- For the CIFAR-10 experiments, $\gamma = 125, 125, 50, 5$ and $\epsilon = 0.7, 0.1, 0.0, 0.3$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the CIFAR-100 experiments, $\gamma = 250, 50, 50, 50$ and $\epsilon = 0.5, 0.7, 0.1, 0.2$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the STL-10 experiments, $\gamma = 12, 12, 25, 12$ and $\epsilon = 0.3, 0.5, 0.0, 0.1$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300 / 200	350 / 200	300 / 300	300 / 300
Batch	128 / 128	128 / 128	64 / 64	128 / 128
Lr	0.1 / 0.1	0.1 / 0.1	0.1 / 0.1	0.1 / 0.1
Sched.	90,180,260 / 60,120,160	150,250 /90,180,260	150,225 / 150,225	90,180,260 /90,180,260
Opt.	SGD / SGD	SGD / SGD	SGD / SGD	SGD / SGD
Decay	0.0005 / 0.0005	0.00004 / 0.0001	0.0001 / 0.0001	0.0002 / 0.0002
Mult.	0.2 / 0.2	0.1 / 0.2	0.1 / 0.1	0.1 / 0.1
Mtm.	True / True	False / True	False / False	True / True

Table 6.1: Training setups for mini-batch SGD (Baseline) on CIFAR-10 / CIFAR-100 respectively. Here, MobileNet uses cosine LR scheduling for CIFAR-100.

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300 / 300	450 / 200	450 / 300	1000 / 300
Batch	32 / 64	64 / 128	64 / 64	128 / 128
Lr	0.01 / 0.01	0.1 / 0.1	0.1 / 0.1	0.1 / 0.1
Sched.	200 / 90,180,260	300,400 /90,180,260	300,400 / 150,225	300,400,600,800 /90,180,260
Opt.	SGD / SGD	SGD / SGD	SGD / SGD	SGD / SGD
Decay	0.0005 / 0.0005	0.0005 / 0.0001	0.0005 / 0.0001	0.0005 / 0.0002
Mult.	0.1 / 0.2	0.2 / 0.2	0.2 / 0.1	0.2 / 0.1
Mtm.	True / True	True / True	False / False	True / True

Table 6.2: Training setups for mini-batch SGD (Baseline) on STL-10 / miniImagenet respectively.

- For the miniImagenet experiments, $\gamma = 50, 125, 125, 5$ and $\epsilon = 0.1, 0.7, 0.3, 0.1$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- Finally, for the ILSVRC2012 experiment, $\gamma = 11700$ and $\epsilon = 0.3$.

6.3.2.2 Ablation: Window Functions

In studying the effects of a variety of window functions, we observe an improvement in overall γ as well as the final testing Accuracy ($\%$). We list the number of filters, post sensitivity, and the ϵ used to compute the final performance for each DNN.

- For VGG16, we use a subset of 17 filters and $\epsilon = 0.7$.
- For MobileNet, we use a subset of 16 filters and $\epsilon = 0.5$.

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300 / 300	350 / 300	300 / 300	300 / 300
Bandit Alg.	EXP3 / EXP3	EXP3 / EXP3	EXP3 / EXP3	EXP3 / EXP3
Mean Teacher	True / True	True / True	True / True	True / True
Loss Fb.	True / True	True / True	True / True	True / True
Batch Size	128 / 128	128 / 128	128 / 128	128 / 128

Table 6.3: Training setups for DIHCL on CIFAR-10 / CIFAR-100 respectively. MobileNet uses a schedule of [0 5 10 15 20 30 40 60 90 140 210 300 350].

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300 / 300	350 / 300	300 / 300	300 / 300
Bandit Alg.	UCB / TS	UCB / TS	UCB / TS	UCB / TS
Mean Teacher	True / True	True / True	True / True	True / True
Loss Fb.	False / False	False / False	False / False	False / False
Batch Size	128 / 128	128 / 128	64 / 64	128 / 128

Table 6.4: Training setups for DIHCL on STL-10 / miniImagenet respectively.

- For DenseNet, we use a subset of 12 filters and $\epsilon = 0.0$.
- For ResNet50, we use a subset of 12 filters and $\epsilon = 0.3$. We list the optimal results from $\tau = 100$ for $\alpha = \mathbb{1}_{\frac{L}{2}:L}$ and $\alpha = \mathcal{N}(0, 1)$.

6.3.2.3 Adversarial Robustness

The adversarial training algorithms we use were cloned from `https://github.com/locuslab/fast_adversarial`. Most of the adversarial attacks were cloned from `https://github.com/Harry24k/adversarial-attacks-pytorch`. PGD20 and CW loss-based attacks were ported from `https://github.com/zjfheart/Friendly-Adversarial-Training`.

6.3.2.4 Adversarial Attacks

In general, we use the default settings provided for all the adversarial attacks throughout our experiments. We highlight some of the specifications (variable names) for each attack below,

- MIFGSM: $\epsilon = 8/255.$, $\alpha = 2/255.$, decay=1.0, iterations=5.
- FFGSM: $\epsilon = 8/255.$, $\alpha = 10/255.$.

	PreActResNet18
Epochs	200
Batch	128
LR schedule	piecewise
LR max	0.1
LR one drop	0.01
LR one drop epoch	100
Attack	PGD
Epsilon	8
Attack iters	10
restarts	1
PGD-alpha	2

Table 6.5: Training setup for [131] on CIFAR-10.

- DI2FGSM: $\epsilon = 8/255.$, $\alpha = 2/255.$, decay=0.0, steps=20, resize_rate=0.9, diversity_prob =0.5, random_state=False.
- APGD: $\epsilon = 8/255.$, steps=100.
- CWLoss: steps=30,$\epsilon = 0.031$, step_size=$0.031/4$, category='Madry', rand _init=True.
- PGD20: steps=20, $\epsilon = 0.031$, step_size=$0.031/4$, category='Madry', rand _init = True.

6.3.2.5 Standard Adversarial Training

In this section, we list the hyper-parameters used to train Rice *et al.* [131] and Cui *et al.* [132] in Tables 6.5 and 6.6. For CAPER, we re-use the hyper-parameters from the original algorithms alongside our selection of γ and ϵ while setting $\tau = 35$ and 15 respectively. Specifically,

- For the CAPER+[131], $\gamma = 245$ and $\epsilon = 0.1$.
- For the CAPER+[132], $\gamma = 350$ and $\epsilon = 0.3$.

6.3.2.6 Efficient Adversarial Training

We list the hyper-parameters used to train Wong *et al.* [139] and Shafahi *et al.* [138] in Tables 6.7 and 6.8. For CAPER, we re-use the hyper-parameters from the original algorithms alongside our selection of γ and ϵ while setting $\tau = 50$. Specifically,

	ResNet18
Epochs	100
Batch	128
Decay	0.0002
LR	0.1
Mtm.	0.9
Epsilon	0.031
Steps	10
Step size	0.007

Table 6.6: Training setup for [132] on CIFAR-100.

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	350	300	300
Batch	128	128	64	128
LR min	0.0	0.0	0.0	0.0
LR max	0.1	0.1	0.1	0.1
Sched.	Cyclic	Cyclic	Cyclic	Cyclic
Opt.r	SGD	SGD	SGD	SGD
Decay	0.0005	0.00004	0.0001	0.0002
epsilon	8	8	8	8
alpha	10	10	10	10
delta-init	Random	Random	Random	Random
Mtm.	True	True	True	True

Table 6.7: Training setup for [139] on CIFAR-10.

- For the CAPER+[139], $\gamma = 125, 250, 25, 12$ and $\epsilon = 0.1, 0.5, 0.2, 0.3$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.
- For the CAPER+[138], $\gamma = 12, 125, 5, 25$ and $\epsilon = 0.5, 0.3, 0.1, 0.7$ for VGG16, MobileNet, DenseNet and ResNet50 respectively.

6.3.3 Curriculum Comparison

In this experiment, our main goal is to compare the performance of CAPER against mini-batch SGD training and highlight how we can improve performance while only retaining a subset of our training data. Additionally, we compare against the state-of-the-art curriculum learning method DIHCL [48], which prioritizes the removal of samples throughout the training process. We extend

	VGG16	MobileNet	DenseNet	ResNet50
Epochs	300	350	300	300
Batch	128	128	64	128
LR min	0.0	0.0	0.0	0.0
LR max	0.1	0.1	0.1	0.1
Sched.	Cyclic	Cyclic	Cyclic	Cyclic
Opt.r	SGD	SGD	SGD	SGD
Decay	0.0005	0.00004	0.0001	0.0002
epsilon	8	8	8	8
Mtm.	True	True	True	True

Table 6.8: Training setup for [138] on CIFAR-10.

DNN	Algorithm	CIFAR-10	CIFAR-100	STL-10	miniImagenet
VGG16	Baseline	94.04	74.23	82.75	70.95
	Random	93.19	71.63	80.38	67.57
	DIHCL	94.03	72.89	79.71	66.07
	CAPER	**94.47** ($\gamma = 125$)	**75.06** ($\gamma = 250$)	**83.01** ($\gamma = 12$)	**71.61** ($\gamma = 50$)
MobileNet	Baseline	93.50	72.75	77.95	64.62
	Random	92.31	71.15	73.86	62.11
	DIHCL	88.97	61.58	75.40	49.37
	CAPER	**93.62** ($\gamma = 125$)	**74.97** ($\gamma = 50$)	**80.04** ($\gamma = 12$)	**66.92** ($\gamma = 125$)
DenseNet	Baseline	95.13	76.95	85.55	73.78
	Random	93.88	74.18	82.39	71.23
	DIHCL	94.72	76.03	85.82	64.34
	CAPER	**95.16** ($\gamma = 50$)	**77.74** ($\gamma = 50$)	**85.83** ($\gamma = 25$)	**75.97** ($\gamma = 125$)
ResNet50	Baseline	95.63	79.27	72.77	68.76
	Random	95.27	76.71	69.29	64.69
	DIHCL	**95.83**	79.71	**73.58**	66.86
	CAPER	95.75 ($\gamma = 5$)	**79.78** ($\gamma = 50$)	73.40 ($\gamma = 12$)	**69.77** ($\gamma = 5$)

Table 6.9: Across most datasets CAPER achieves the best performance when compared against mini-batch SGD, DIHCL, and Random baselines. Here, **bold** refers to the best performance while underline refers to the second best method.

their code to accommodate our datasets and DNN architectures while maintaining their training protocols.

From Table 6.9, across all combinations of datasets and DNN architectures, we observe that our

DNN	Algorithm	ILSVRC2012
ResNet50	Baseline	76.32
	DIHCL	<u>76.33</u>*
	CAPER (Ours)	**76.62** ($\gamma = 11700$)

Table 6.10: CAPER achieves the best performance even after removing 11700 samples across 10 classes. Here, **bold** refers to the best performance while <u>underline</u> refers to the second best method.* indicates numbers cited by authors. ILSVRC2012 results are across one trial.

algorithm easily outperforms the baseline mini-batch SGD setup, even with the removal of a subset of the training data. To ensure fair comparison, we use the same hyper-parameter setups across mini-batch SGD and our method.

More interestingly, when we observe the performance of DIHCL adapted to our selection of dataset-DNN pairs, we see that it consistently exhibits strong performances on the ResNet architectures. This, in conjunction with DIHCL's propensity to perform significantly worse than randomly removing the same number of samples as in CAPER (marked in Table as Random) across the other tested architectures points toward the strong affinity of the training setup used in DIHCL to residual architectures. Despite this, the performance of CAPER in conjunction with the starkly different training setup used in DIHCL (which includes cyclic learning rate schedules, a teacher-like copy of the DNN, etc., and no explicit fine-tuning of DIHCL's hyper-parameters) still improves upon DIHCL in most cases. This improvement is further highlighted when applying CAPER to the ILSVRC2012 dataset (Table 6.10), where we can significantly outperform DIHCL and standard mini-batch SGD, even with the removal of 11700 samples. For more context, we remove the 11700 samples within the span of 10 classes, which is similar to removing nine entire classes with approximately 1300 images each.

6.3.3.1 Ablation: Window Functions

Across all the results presented in Tables 6.9 and 6.10, we use $\alpha = \mathbb{1}_L$, which results in the collection of features from the last convolutional layer. In this section, we compare and contrast four different window functions to identify the impact of comparing features across multiple layers of a DNN and its potential benefits. Based on Table 6.11, there are two main observations. First, the use of additional layers in assessing the susceptibility of samples to noise often allows for an increase in γ when compared to the case of $\mathbb{1}_L$, with minor trade-offs in performance. Second, in conjunction with the first observation, $\alpha = \mathbb{1}_{1:\frac{L}{2}}$ shows the best Accuracy(%) across our restricted set of window functions. These results highlight the regularization affect our method imposes on the DNN, regardless of the location at which we ascertain the distance between features. Further, by

Window Function	VGG16	MobileNet	DenseNet	ResNet50
Baseline	94.04	93.50	95.13	95.63
$\alpha = \mathbb{1}_L$	94.47 ($\gamma = 125$)	93.62 ($\gamma = 125$)	95.16 ($\gamma = 50$)	95.75 ($\gamma = 25$)
$\alpha = \mathbb{1}_{1:\frac{L}{2}}$	**94.49** ($\gamma = 300$)	**93.66** ($\gamma = 150$)	**95.28** ($\gamma = 50$)	**95.78** ($\gamma = 50$)
$\alpha = \mathbb{1}_{\frac{L}{2}:L}$	94.41 ($\gamma = 300$)	93.59 ($\gamma = 150$)	95.22 ($\gamma = 50$)	95.72 ($\gamma = 50$)
$\alpha = \mathcal{N}(0, 1)$	94.43 ($\gamma = 250$)	93.61 ($\gamma = 125$)	95.16 ($\gamma = 300$)	95.74 ($\gamma = 100$)

Table 6.11: Assessing the susceptibility of samples to noise using multiple layers boosts γ as well as Accuracy (%). $\alpha = \mathbb{1}_{1:\frac{L}{2}}$ provides the best performance. The optimal result for each CIFAR10-DNN-α combination is given in the table.

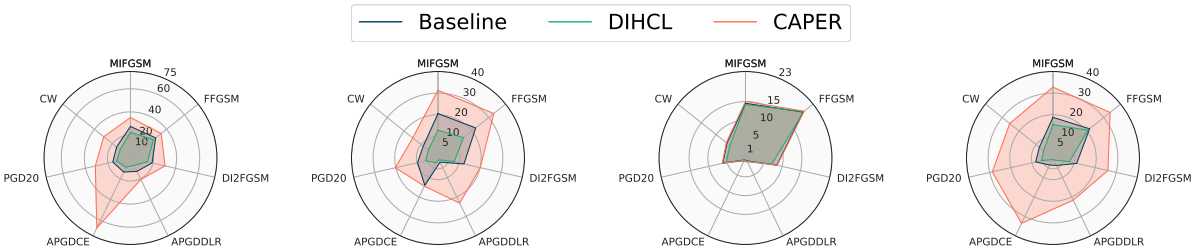
assessing distances across layers other than the final one in the DNN, we can reduce the relationship between specific task-oriented information and how we assess noisy samples, allowing CAPER to be more extensible to alternative tasks.

6.3.4 Adversarial Robustness

In this subsection, we compare and contrast the adversarial robustness of CAPER-based training against a variety of adversarial attacks on CIFAR-10/100. Specifically, we measure adversarial robustness to attacks where the source and target models are the same as well as the case when multiple source models are used to generate the attacks for a single target. To ensure parity and high efficiency, we avoid comparing results across methods with and without adversarial training and restrict CAPER to use $\alpha = \mathbb{1}_L$.

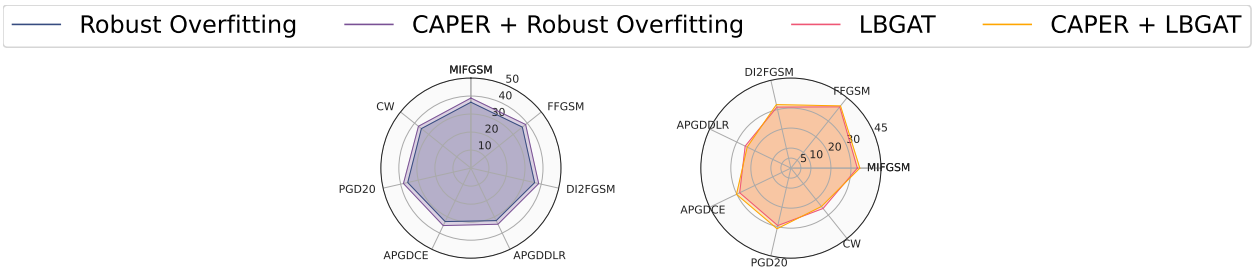
6.3.4.1 Adversarial Source = Target

Curriculum-based Comparison Using Fig. 6.1, we establish two main observations, 1) in multiple instances, DIHCL reduces the robustness of DNNs, when compared to mini-batch SGD training, and more importantly, 2) CAPER significantly improves the robustness of DNNs to multiple adversarial attacks. We hypothesize two possible reasons why DIHCL reduces the adversarial robustness of a variety of DNNs. First, the repeated sampling with replacement and steady decline in the number of available samples does not allow for a stable learning environment to help address adversarial robustness. Secondly, the use of gradients/loss/prediction values or their change has a direct impact on the set of samples removed and, therefore, the final adversarial robustness. A deeper dive into the correlation between the selection procedure and the final outcomes could help provide more insight.



(a) VGG ($\gamma = 125$) (b) MobileNet ($\gamma = 125$) (c) DenseNet ($\gamma = 50$) (d) ResNet ($\gamma = 25$)

Figure 6.1: Curriculum-based Comparison: Across all DNN architectures, CAPER matches and often significantly improves upon the adversarial robustness of mini-batch SGD training and DIHCL. Methods with the largest area of plot are preferred.

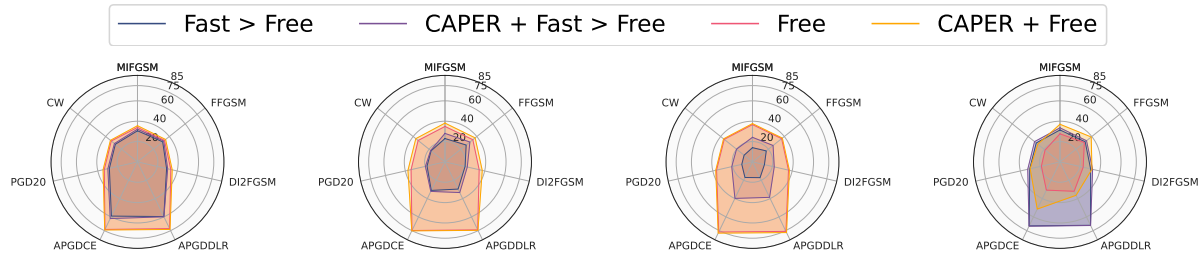


(a) PreActResNet18 (b) ResNet18

Figure 6.2: Standard Adversarial Training Comparison: Largest improvements from adversarial training are observed for MIFGSM, FFGSM, DI2FGSM, APGDCE, and PGD attacks. More generally, the addition of CAPER atop adversarial training methods improves their robustness. Results are provided across one trial. γ values are 245 and 350 for CAPER+[131] and CAPER+[132] respectively.

Standard Adversarial Training Comparison Adversarial training approaches impart robustness to DNNs by exposing them to multiple examples of adversarial input during the training phase. A key component of such an approach is the retention and modification of the entire training dataset to create different adversarial examples. We use Rice *et al.* [131], with settings corresponding to their validation-based early stopping setup on CIFAR-10, and Cui *et al.* [132], with settings corresponding to ResNet18 for both natural and robust models on CIFAR-100, as representatives for standard adversarial training.

When using CAPER alongside standard adversarial training, we observe an improvement in performance over the original adversarial training methods across most adversarial attacks, as shown in Fig. 6.2. We emphasize that these improvements are in addition to an increase in Accuracy(%), from 82.66% to 83.14% for PreActResNet18 and from 69.22% to 69.61% for ResNet18. While the original methods emphasize a balanced improvement in Robust Accuracy(%) and Accuracy(%), the addition of CAPER atop these methods allows us to maintain their original benefits while further



(a) VGG ($\gamma = 125/12$) (b) MobileNet ($\gamma = 125$) (c) DenseNet ($\gamma = 50/5$) (d) ResNet ($\gamma = 12/25$)

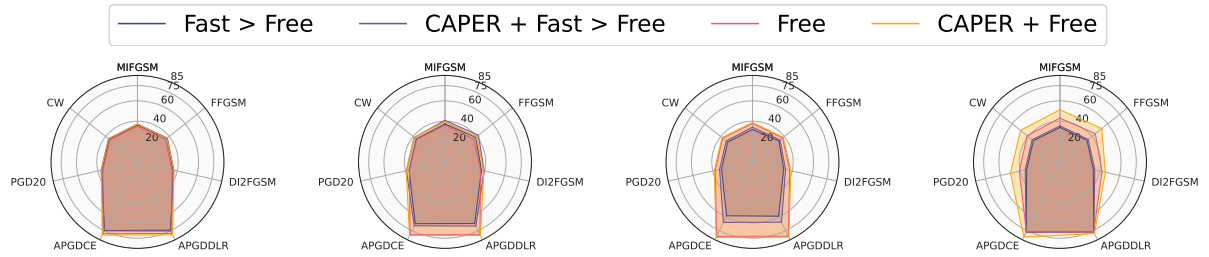
Figure 6.3: Efficient Adversarial Training Comparison: Consistently, the largest improvements from efficient adversarial training is observed for various types of FGSM attacks. However, in general, the addition of CAPER atop efficient adversarial training methods improves the robustness of the DNN. Results provided for methods using [138] are across one trial. γ values are 125/12, 250/125, 25/5, and 12/25 for CAPER+Wong *et al.* [139] and CAPER+Shafahi *et al.* [138], respectively, across VGG16, MobileNet, DenseNet, and ResNet50.

improving on their efficiency and adversarial robustness.

Efficient Adversarial Training Comparison Efficient adversarial training approaches focus on optimizing the quality of adversarial input during the training phase to impart adversarial robustness quickly. These methods align more closely with our efficiency and robustness goals, while standard adversarial training methods emphasize the trade-off between robustness and performance. We use Shafahi *et al.* [138] and Wong *et al.* [139], with modifications to the training hyper-parameters to match our baselines, as representatives for efficient adversarial training.

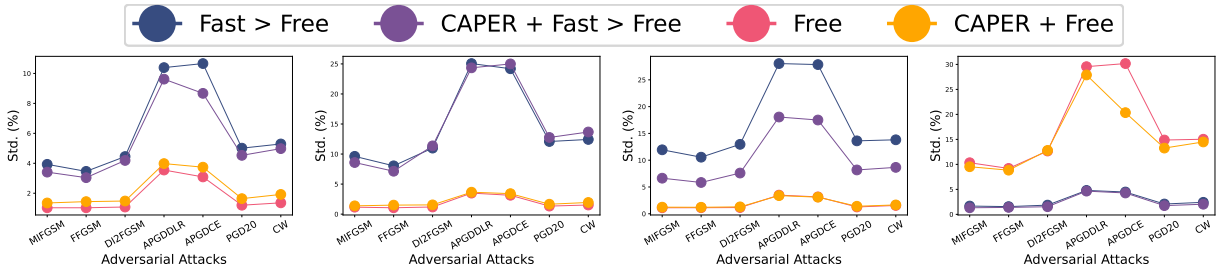
Our first observation based on Fig. 6.3 is the high level of robustness shown by all DNNs to APGDDLR and APGDCE attacks across both efficient adversarial training and CAPER-based training. In addition, when using CAPER-based adversarial training, DI2FGSM, MIFGSM and FFGSM consistently shows the largest magnitude of improvement. Finally, similar to the previous scenario's results on standard adversarial training, adding CAPER atop common efficient adversarial training approaches further boosts their performance against all adversarial attacks.

General Takeaways The increase in Robust Accuracy(%) across multiple types of adversarial training approaches support our hypothesis that CAPER is complementary to adversarial training. While CAPER removes noisy samples from the training set to make it more cohesive, it does not harm the robustness offered by exposing DNNs to various examples of adversarial input. These improvements are in addition to a strong increase in the standard Accuracy(%) metric as well. An interesting and important takeaway from these results is that they support the notion of an optimal training subset [177] during both adversarial and normal mini-batch SGD training.



(a) VGG16-CIFAR10 (b) MobileNet-CIFAR10 (c) DenseNet-CIFAR10 (d) ResNet50-CIFAR10

Figure 6.4: CAPER-based training boosts the mean Robustness Accuracy(%) across multiple sources of adversaries. In our experiments, we use all four possible DNN architectures to generate attacks. γ values are 125/12, 250/125, 25/5 and 12/25 for CAPER+Wong *et al.* [139] and CAPER +Shafahi *et al.* [138] respectively across VGG16, MobileNet, DenseNet and ResNet50.



(a) VGG16-CIFAR10 (b) MobileNet-CIFAR10 (c) DenseNet-CIFAR10 (d) ResNet50-CIFAR10

Figure 6.5: CAPER-based adversarial training algorithms consistently have some of the lowest deviation in performance, thus ensuring in-Transferability.

6.3.4.2 Adversarial Sources $\neq$ Target

We define in-Transferability as the robustness of DNNs to attacks designed on a variety of DNN backbones. To measure in-Transferability, we use the mean and standard deviation of Robust Accuracy (%) when a selected model is attacked using adversaries generated from all four of the DNN architectures used in our experiments. We specifically demand that standard deviation in performance is minimized, in addition to high average performance, since a high deviation indicates that the robustness is dependent on the type of DNN backbone used to generate adversaries.

In Figs. 6.4 and 6.5, we highlight the mean and standard deviation of Robust Accuracy(%) when we take into account adversaries generated across all four possible DNN architectures. Across almost all DNN-adversarial attack combinations, the average Robust Accuracy(%) for CAPER-based approaches improves on the original adversarial training approach. The only exception is on VGG16 for APGDCE and APGDDLR attacks, which fall within one standard deviation of the original approach's performance. From the standard deviation point of view, Shafahi *et al.* [138]-based training has lower values when compared to Wong *et al.* [139]-based training across

DNN	TFLOPs Reduced			
	CIFAR-10	CIFAR-100	STL-10	miniImagenet
VGG16	9.83	11.8	8.85	36.9
MobileNet	3.55	0.711	0.276	0.104
DenseNet	1.45	16.1	20.4	63.8
ResNet50	6.5	16.3	8.61	23.6

Table 6.12: Illustration of the improvement in efficiency offered by CAPER during the training phase. The baseline number of FLOPs is calculated by assuming a simple forward pass through the DNN. For reference an Nvidia Jetson Nano offers 0.5 TFLOPs.

Algorithm	TFLOPs Reduced			
	VGG16	MobileNet	DenseNet	ResNet50
CAPER + Wong *et al.* [139]	9.83	3.55	1.61	3.9
CAPER + Shafahi *et al.* [138]	9.43	3.55	0.161	8.13

Table 6.13: The addition of CAPER atop existing efficient adversarial training methods improves upon the overall FLOPs reduced during the training phase. The baseline number of FLOPs is calculated by assuming a simple forward pass through the DNN. Results for CAPER+[138] are across one trial. For reference an Nvidia Jetson Nano offers 0.5 TFLOPs.

all DNNs except ResNet50, both with and without CAPER. Overall, our results closely resemble the standard deviation of existing efficient adversarial training approaches while improving the in-Transferability of DNNs to a variety of adversarial attacks.

A broader takeaway from our results is the distinctly visible pattern of performances, average and standard deviation, between [139] and [138], with some degree of architectural specificity (Residual vs. the rest). A further study into their relationship could help highlight factors essential in designing adversarial training approaches to have intended consequences. Overall, the common trend of improved mean Robust Accuracy(%) with low standard deviations highlights our CAPER-based adversarial training as a definite way to ensure in-Transferability across several DNN architectures.

6.3.5 Time Efficiency Comparison

The third of our targets deals with efficiency, specifically decreasing the amount of computations performed during the training phase. To understand the impact of CAPER on efficiency, we observe the number of FLOPs reduced by CAPER when compared against the number of FLOPs computed by an appropriate baseline method, measured as a single forward pass over a DNN scaled over the entire training phase. We compute the number of FLOPs for each Algorithm-Dataset-DNN triplet

using their respective hyper-parameter settings listed under the experimental setup (Section 6.3.2).

From Table 6.12, we observe a strong increase in the number of FLOPs reduced across a variety of Dataset-DNN combinations when compared to standard mini-batch training. When we extend this comparison to the ILSVRC2012 dataset, we save 4.08 PFLOPs, which comes close to almost an entire epoch of training. Since our approach uses hard sampling to remove noisy samples permanently from the training set, the computation and comparison of FLOPs are easier than DIHCL, which uses sampling with replacement. Practically, we permanently remove the noisy samples from the dataloader and do not use excess memory to access them throughout the remainder of the training phase.

As a complementary piece to standard adversarial training, we have already demonstrated the improvement in Robust Accuracy($\%$) as well as improved in-Transferability of adversarial attacks. These improvements are supplemented by a reduction in the number of FLOPs computed during training, as shown in Table 6.13. Regardless of the adversarial training algorithm in question, be it [131] or [132], where we observe a maximum reduction of 22.5 TFLOPs or efficient adversarial training algorithms shown in Table 6.13, where we have a maximum reduction of 9.83 PFLOPs, there is a significant gain in efficiency when using CAPER. Overall, when combining the benefits of performance, efficiency, and robustness, CAPER manages to successfully deliver on all of our targets

6.4 Discussion and Limitations

Adversarial Response When comparing the performance of CAPER, with and without the addition of other adversarial training regimes, we find their performance across FFGSM, MIFGSM, and DI2FGSM extremely similar, often within 5% of each other. The importance of this observation is further highlighted by the fact that we do not expose the model to any adversarial input during training. This outcome suggests that our approach could provide an inexpensive alternative to boosting performance across FGSM-based attacks while complementing existing adversarial training approaches.

Performance vs. Adversarial Robustness The trade-off between performance and adversarial robustness is a commonly known and accepted fact. A number of works like Rice *et al.* [131] and others try to resolve this by aiming to improve on both fronts simultaneously. From our results on adversarial robustness, we observe a sharp drop in Accuracy($\%$) when we apply any form of adversarial training, regardless of the underlying DNN architecture. Only CAPER+Shafahi *et al.* [138] on ResNet50-CIFAR10 comes within 4% of the highest performance we achieve in curriculum-based comparison (Table 6.14). While there are differences in the settings and hyper-

Algorithm	Accuracy(%)			
	VGG16	MobileNet	DenseNet	ResNet50
Best from Table 6.9	**94.47**	**93.62**	**95.16**	**95.83**
Wong *et al.* [139]	76.58	80.66	74.10	79.19
CAPER+Wong *et al.* [139]	79.75	83.29	75.96	79.28
Shafahi *et al.* [138]	80.86	82.17	83.69	91.64
CAPER+Shafahi *et al.* [138]	82.62	82.97	84.73	92.55

Table 6.14: There is a still a significant gap between the improved Accuracy(%) achieved by adversarial training and results from Table 6.9 which suggests there is still room for improvement in this domain. Results for [138]-based experiments are across 1 trial.

parameters suggested by the original authors of the adversarial training works, we find that there is still room for improvement when it comes to bridging the gap between improving Accuracy(%) and Robust Accuracy(%) simultaneously.

DenseNet Performance Throughout our experiments on adversarial robustness, specifically in curriculum-based comparisons, DenseNet has offered the smallest magnitude of robustness and improvement in performance. While there could be many contributing factors, we hypothesize that DenseNet's foundational building block of continuously retaining and combining features from previous blocks is one of the main reasons why standard mini-batch, DIHCL, and even CAPER-based training does not provide a strong improvement in adversarial robustness. A more comprehensive combination of the features across the entire DNN might help overcome the current issue and improve its overall adversarial robustness.

Limitations In introducing the notion of noise injection to identify and remove samples, we limited the scope of the kind of noise used to perturb the samples to only gaussian. The correlation between noisy samples and the different kinds and levels of noise is a key emphasis of our future work in this domain. An important restriction to our methodology is the emphasis on a relatively large sweep on the number of samples removed from the training set. The outcome of the number of samples removed can span extremely small or large values. As part of our ongoing effort to adapt our work to multiple domains, we plan to formulate this as an optimization constraint. Finally, we acknowledge that our results are based on a relatively small set of parameter sweeps, including the use of only one α function, and there are some alternative hyper-parameter combinations still unexplored.

Potential Negative Impacts Since we reduce the total amount of training data provided to the model, we risk losing some of the representational depth and complexity in the learned features. This is especially important when considering the impact of weaker pretraining on downstream tasks. In addition, our core idea revolves around removing data points that have a high proclivity of being ambiguous. One possible implication of the removal of such data points could be a reduction in the fairness of the overall model since such data points could be part of an underrepresented set of data. From an adversarial robustness perspective, when using the l_2 metric distance as a sensitivity measure, we risk exposing our feature embeddings to alternative forms of adversarial attack.

6.5 Key Takeaways

Overall, we establish CAPER as an algorithm that simultaneously tackles improvements in performance, efficiency and adversarial robustness. The use of noise-injection in CAPER to identify and remove noisy samples helps modify the feature embedding leaned by DNNs in a favourable manner. In doing so, there is a strong improvement in classification accuracy achieved via a more efficient training process. We also establish high adversarial robustness and in-Transferability by incorporating CAPER like a plug-and-play module atop existing adversarial training methods. Our goal is to jointly target PER in an effort to develop more cost and resource efficient training protocols, with a view to reducing the environmental impact of developing DNNs.

CHAPTER 7

Future Directions and Conclusion

In this dissertation, we have proposed techniques that reduce the number of FLOPs during inference and training, as well as improve the sparsity of DNN architectures and adversarial robustness while maintaining a high level of accuracy. The central idea to reducing the number of parameters and FLOPs of a DNN architecture was the use of a one-shot DNN pruning framework which was based on mutual information. At a process level, the use of a single train-prune-retrain setup avoids iterative pruning or retraining steps and condenses the architecture development and training phase into a tighter loop. In Chapter 3, we introduce the use of conditional geometric mutual information in a one-shot pruning framework as a method that takes into account the uncertainty in the relationships between filters. In SNACS (Chapter 4), we propose the adaptive conditional mutual information measure as an approach to balance the contributions from activations with those from the weights. The final product of our contributions to pruning is a slim and resource efficient DNN that contains lesser number of parameters and FLOPs, thereby reducing the required storage memory and inference time.

When aiming for efficiency in training, we need to be well aware of a large number of external factors that affect the underlying process and outcome. As a step towards understanding the relationship between dataset and DNN from the data's perspective, we used a label-based curriculum in LILAC (Chapter 5), with no changes to the preprocessing or experimental hyper-parameters, to elevate the performance of a DNN. Combining our takeaways from LILAC and the adversarial susceptibility of DNNs, pruned or otherwise, we pushed the boundary on concurrently tackling performance, decrease computations and adversarial robustness with a minimal plug-and-play algorithm, CAPER (Chapter 6). CAPER explores the use of feature-differences across the DNN to identify and remove samples that strongly affect the evolution of a DNN's weights. Overall, our contributions to curriculum learning have yielded techniques that break the mould of sample-based curricula and instead provide simple modules that not only elevate the performance and security of DNNs but allow users to quickly achieve it.

Despite the strides and advancements made in this dissertation, there are many open challenges in both Architecture Prototyping and Training that offer new avenues for research. Our work on

neural network pruning ensured a single train-prune-retrain step was sufficient to generate slim DNNs. However, there is a gap in the level of sparsity that can be achieved by a one-shot pruning approach in comparison to iterative pruning approaches. One of the main reasons for this gap in achievable sparsity is the concept of *layer collapse* [63]. By integrating novel uncertainty-based measures into the training phase, we can push the limits of the sparsity achieved by probabilistic pruning. Another possible advantage to integrating pruning into the training phase would be the large number of iterations with which algorithms can compensate for the reduction in capacity when pruning. Further, Tanaka *et al.* [63]) have already shown the benefit of iterative setups in avoiding layer collapse. In addition, this integration should able to remove the individual pruning and retraining steps, further reducing the time taken to develop DNNs.

The evolution of weights in DNNs through the course of training is a complex interaction between dataset, DNNs and the optimization algorithm used. While we have addressed the challenge of simultaneously tackling performance, efficiency and robustness, through the use of feature differences, the theoretical relationship between a true optimal dataset and DNN remains unaddressed. Understanding their relationship can help shed more light on how to collect and prepare new datasets, reduce the amount of resources and time spent on it, and the issues plaguing existing datasets. By posing the development of DNNs as a multifaceted optimization objective with several constraints that emphasize real-world applicability, we believe that researchers can identify a smarter and more resource efficient way to develop DNNs, with significantly lesser environment impact.

APPENDIX A

SNACS: Proof of Theorem 1

A.1 Proof of Theorem 1

Recall our estimator in 4.7,

$$\widehat{I}_{\varphi}(X;Y|Z) = \sum_{e_{ijk}\in E_G} \varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right), \tag{A.1}$$

where $\alpha_{ijk} = \dfrac{r_{ik}\ r_{jk}}{r_k}$. The expectation of $\widehat{I}_{\varphi}$ is derived as,

$$\mathbb{E}\left[\sum_{e_{ijk}\in E_G} \varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\big|E_G\right] = \sum_{e_{ijk}\in E_G} \mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\big|E_{ijk}\right], \tag{A.2}$$

where E_{ijk} is the event that there is an edge between the vertices v_i, u_j, and ω_k in the dependency graph $G(X,Y,Z)$. Let hash function H_1 map the N i.i.d points X_k, Y_k, and Z_k to $\tilde{X}_k$, $\tilde{Y}_k$, and $\tilde{Z}_k$. Following the notations used in Morteza *et al.* [88], we denote $E_i^{=1}$ be the event that there is exactly one vector from $\tilde{X}_i$ that maps to v_i using H_2. Similarly, we define $E_j^{=1}$ and $E_k^{=1}$. We denote $E_{ijk}^{=1} := E_i^{=1} \cap E_j^{=1} \cap E_k^{=1}$ and let $\overline{E_{ijk}^{=1}}$ be the complement set of $E_{ijk}^{=1}$.

We simplify Eqn. A.2 by splitting it into two parts: without collision and due to collision. Based on the law of total expectation we have,

$$\begin{aligned} &= \sum_{e_{ijk}\in E_G} P(E_{ijk}^{=1}|E_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk} g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\big|E_{ijk}^{=1}, E_{ijk}\right] \\ &\quad + \sum_{e_{ijk}\in E_G} P(\overline{E_{ijk}^{=1}}|E_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\big|\overline{E_{ijk}^{=1}}, E_{ijk}\right]. \end{aligned} \tag{A.3}$$

Step 1 Bias on w/o collision: Similar to Lemma 7.3 in Morteza *et al.* [88], we derive,

$$P(E_{ijk}^{=1}|E_{ijk}) = 1 - O\left(\frac{1}{\epsilon^d N}\right), \quad d = d_X + d_Y + d_Z. \tag{A.4}$$

This is because all three $|V|$, $|U$, and $|W|$ are upper bounded by $O(\epsilon^{-d})$. Note that ϵ is a function of N. Additionally from [88] we infer the following results:

$$\mathbb{E}[\alpha_{ijk}] = \frac{\mathbb{E}[r_{ik}]\ \mathbb{E}[r_{jk}]}{\mathbb{E}[r_k]} + O\left(\sqrt{\frac{1}{N}}\right). \tag{A.5}$$

Note that (A.5) is implied based on the fact that $\mathbb{V}(\alpha_{ijk}) \leq O(1/N)$ which is proved by applying Efron-Stein inequality under assumptions **(A1)** and **(A3)**, similar to arguments in Lemma 7.10 from [88]. In addition, we have

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\right] = \frac{\mathbb{E}[r_{ijk}]}{\mathbb{E}[\alpha_{ijk}]} + O\left(\sqrt{\frac{1}{N}}\right), \tag{A.6}$$

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\right] = P(E_{ijk}^{\leq 1})\ \mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}|E_{ijk}^{\leq 1}\right] + P(E_{ijk}^{>1})\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}|E_{ijk}^{>1}\right], \tag{A.7}$$

where by using similar arguments as in Eqn. 56 from [88], we have $P(E_{ijk}^{\leq 1}) = 1 - O(\sqrt{1/(\epsilon^d N)})$. Therefore, $P(E_{ijk}^{>1}) = O(\sqrt{1/(\epsilon^d N)})$. Further the second term in Eqn. A.7 is the bias because of collision of H, which will be proved in the following section, that is upper bounded by $O(\sqrt{1/(\epsilon^d N)})$.

Let x_D and x_C respectively denote the discrete and continuous components of the vector x, with dimensions d_D and d_C . Also let $f_{X_C}(x_C)$ and $p_{X_D}(x_D)$ respectively denote density and pmf functions of these components associated with the probability measure P_X. Let X have d_C and d_D, Y have d_C',d_D', and Z have d_C'', d_D'' as their continuous and discrete components, respectively. Then it can be shown that,

$$\begin{aligned}\mathbb{E}[r_{ijk}|E_{ijk}^{\leq 1}] =& P(X_D = x_D, Y_D = y_D, Z_D = z_D)\epsilon^{d_C + d_C' + d_C''}(f(x_C, y_C, z_C|x_D, y_D, Z_D) \\ &+ \Delta(\epsilon, q, \gamma)),\end{aligned} \tag{A.8}$$

where densities have bounded derivatives up to the order $q \geq 0$ and belong to the Hölder continuous class with smoothness parameter γ. Note that $\Delta(\epsilon, q, \gamma) \to 0$ as $N \to \infty$. Now from Eqns. 50, 51, and 53 in [88] and from Eqn. A.5, A.6 above, under assumptions **(A1)** and **(A3)**, we derive

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}|E_{ijk}^{\leq 1}\right] = \frac{dP_{XYZ}\ P_Z}{dP_{XZ}\ P_{YZ}} + \widetilde{\Delta}(\epsilon, q, \gamma) + O\left(\sqrt{\frac{1}{N}}\right), \tag{A.9}$$

where $H(x) = i$, $H(y) = j$, $H(z) = k$, and as $N \to \infty$, $\widetilde{\Delta}(\epsilon, q, \gamma) \longrightarrow 0$.

Step 2 Bias because of collision: Let $\widetilde{\mathbf{X}} = \left\{\widetilde{X}_i\right\}_{i=1}^{L_X}$, $\widetilde{\mathbf{Y}} = \left\{\widetilde{Y}_i\right\}_{i=1}^{L_Y}$, $\widetilde{\mathbf{Z}} = \left\{\widetilde{Z}_i\right\}_{i=1}^{L_Z}$ respectively denote distinct outputs of H_1 with the N i.i.d points X_k, Y_k, Z_k as inputs. We denote $L_{XYZ} := |\widetilde{\mathbf{X}} \cup \widetilde{\mathbf{Y}} \cup \widetilde{\mathbf{Z}}|$, $L_{XZ} := |\widetilde{\mathbf{X}} \cup \widetilde{\mathbf{Z}}|$, and $L_{YZ} := |\widetilde{\mathbf{Y}} \cup \widetilde{\mathbf{Z}}|$.

$$\begin{aligned}\mathbb{B}_\varphi :&= \sum_{e_{ijk}\in E_G} P(\overline{E^{=1}_{ijk}}|E_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\Big|\overline{E^{=1}_{ijk}}, E_{ijk}\right] \\ &\leq \sum_{i,j,k\in\mathcal{F}} P(E^{>1}_{ijk})\mathbb{E}\left[\mathbb{1}_{E_{ijk}}\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\Big|E^{>1}_{ijk}\right], \end{aligned} \tag{A.10}$$

where $E^{>1}_{ijk} = E^{>1}_i \cap E^{>1}_j \cap E^{>1}_k$, and $E^{>1}_i$ is the event that there are at least two vectors from $\tilde{X}_i$ that map to v_i using H_2. Once again, using the law of total expectation, then the RHS of Eqn. A.10 becomes,

$$\begin{aligned}&= \sum_{i,j,k\in\mathcal{F}} P(E^{>1}_{ijk})\Bigg(P(E_{ijk}|E^{>1}_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\Big|E^{>1}_{ijk}, E_{ijk}\right] \\ &\qquad + P(\overline{E_{ijk}}|E^{>1}_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\Big|E^{>1}_{ijk}, \overline{E_{ijk}}\right]\Bigg) \\ &= \sum_{i,j,k\in\mathcal{F}} P(E_{ijk})P(E^{>1}_{ijk}|E_{ijk})\mathbb{E}\left[\varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\Big|E^{>1}_{ijk}, E_{ijk}\right]. \end{aligned} \tag{A.11}$$

The equality in Eqn. A.11 is obtained based on Bayes error and $g = 0$ on the event $\overline{E_{ijk}}$. Now recalling Eqn. A.4, using Eqn. 4.3 we bound the last line in Eqn. A.11 by,

$$O\left(\frac{1}{\epsilon^d N}\right)\sum_{i,j,k\in\mathcal{F}} P(E_{ijk})\mathbb{E}\left[\varphi(i,j,k)\left(r_{ijk}+\alpha_{ijk}\right)\Big|E^{>1}_{ijk}, E_{ijk}\right]. \tag{A.12}$$

This implies that,

$$\begin{aligned}\mathbb{B}_\varphi &\leq O\left(\frac{1}{\epsilon^d N}\right)\sum_{i,j,k\in\mathcal{F}} P(E_{ijk})\left(\mathbb{E}\left[\varphi(i,j,k)r_{ijk}\Big|E^{>1}_{ijk}, E_{ijk}\right] + \mathbb{E}\left[\varphi(i,j,k)\alpha_{ijk}\Big|E^{>1}_{ijk}, E_{ijk}\right]\right) \\ &= O\left(\frac{1}{\epsilon^d N^2}\right)\sum_{i,j,k\in\mathcal{F}} P(E_{ijk})\Bigg(\mathbb{E}\left[\varphi(i,j,k)\ N_{ijk}\Big|E^{>1}_{ijk}, E_{ijk}\right] \\ &\qquad + \mathbb{E}\left[\varphi(i,j,k)\frac{N_{ik}N_{jk}}{N_k}\Big|E^{>1}_{ijk}, E_{ijk}\right]\Bigg). \end{aligned} \tag{A.13}$$

If we extend our discussion to all the possible mappings from H_1 we obtain,

$$= O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in\mathcal{F}} P(E_{ijk})$$
$$\left(\mathbb{E}\left[\varphi(i,j,k)\ N_{ijk}\middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}},\ \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]\right.$$
$$\left.+\,\mathbb{E}\left[\varphi(i,j,k)\frac{N_{ik}N_{jk}}{N_k}\middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}},\ \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]\right).$$

Let us define,

$$\begin{aligned}
\mathcal{A}_{ijk} &:= \left\{r : H_2(\tilde{X}_r)=i, H_2(\tilde{Y}_r)=j, H_2(\tilde{Z}_r)=k\right\},\\
\mathcal{A}_{k} &:= \left\{r : H_2(\tilde{Z}_r)=k\right\},\\
\mathcal{A}_{ik} &:= \left\{r : H_2(\tilde{X}_r)=i, H_2(\tilde{Z}_r)=k\right\},\\
\mathcal{A}_{jk} &:= \left\{r : H_2(\tilde{Y}_r)=j, H_2(\tilde{Z}_r)=k\right\}.
\end{aligned} \tag{A.14}$$

Let M_r, be the number of the input points $(\mathbf{X},\mathbf{Y},\mathbf{Z})$ mapped to $(\tilde{X}_r,\tilde{Y}_r,\tilde{Z}_r)$. Therefore for i,j,k we can rewrite N_{ijk} as

$$N_{ijk} = \sum_{r=1}^{L_{XYZ}} \mathbb{1}_{A_{ijk}}(r) M_r. \tag{A.15}$$

Similarly M'_r, $\widetilde{M}_s$, and $\overline{M}_t$ are defined the number of the input points mapped to $(\tilde{X}_r,\tilde{Z}_r)$, $(\tilde{Y}_s,\tilde{Z}_s)$, and $\tilde{Z}_t$, respectively and we can write

$$N_{ik} = \sum_{r=1}^{L_{XZ}} \mathbb{1}_{A_{ik}}(r) M'_r, \quad N_{jk} = \sum_{s=1}^{L_{YZ}} \mathbb{1}_{A_{jk}}(s) \widetilde{M}_s, \quad N_k = \sum_{t=1}^{L_Z} \mathbb{1}_{A_k}(t) \overline{M}_t. \tag{A.16}$$

Under the assumption that φ is bounded, we have

$$\begin{aligned}\mathbb{B}_{\varphi} \leq O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in\mathcal{F}} P(E_{ijk}) \\ \Bigg(\sum_{r=1}^{L_{XYZ}} P\left(r \in \mathcal{A}_{ijk} \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \right. \\ \left. \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \mathbb{E}\left[M_r \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right] \\ + \sum_{r=1}^{L_{XZ}} \sum_{s=1}^{L_{YZ}} \sum_{t=1}^{L_Z} P\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_k \middle| E_{ijk}^{>1}, \right. \\ \left. E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\ \mathbb{E}\left[\frac{M_r' \widetilde{M}_s}{\overline{M}_t} \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right] \Bigg). \end{aligned} \tag{A.17}$$

Next we find the probability terms:

$$P\left(r \in \mathcal{A}_{ijk} \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) = \frac{P\left(r \in \mathcal{A}_{ijk}, E_{ijk}^{>1} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right)}{P\left(E_{ijk}^{>1} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right)}. \tag{A.18}$$

We first find the denominator of Eqn. A.18 first. We define $a = 1$ when $i = j = k$ and $a = 3$ for the case $i \neq j \neq k$:

$$\begin{aligned} & P\left(E_{ijk}^{>1} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\ & = 1 - P\left(E_{ijk}^{=0} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) - P\left(E_{ijk}^{=1} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\ & = 1 - \left(\frac{F-a}{F}\right)^{L_{XYZ}} - \left(\frac{L_{XYZ}}{F^a}\left(\frac{F-a}{F}\right)^{L_{XYZ}-a}\right) \\ & = O\left(\frac{L_{XYZ}^2}{F^{a+1}}\right). \end{aligned} \tag{A.19}$$

Further,

$$\begin{aligned} & P\left(r \in \mathcal{A}_{ijk}, E_{ijk}^{>1} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\ & = P\left(r \in \mathcal{A}_{ijk} | E_{ijk}^{>1}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) P\left(r \in \mathcal{A}_{ijk} | \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\ & = \left(1 - \left(\frac{F-a}{F}\right)^{L_{XYZ}-a}\right)\left(\frac{1}{F}\right)^a = O\left(\frac{L_{XYZ}}{F^{a+1}}\right). \end{aligned} \tag{A.20}$$

Combining Eqn. A.19 and A.20 yields

$$P\left(r \in \mathcal{A}_{ijk} \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) = O\left(\frac{1}{L_{XYZ}}\right).$$

Now we simplify the following term:

$$P\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_k \middle| E_{ijk}^{>1}, E_{ijk},\ \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right). \tag{A.21}$$

First we assume that $\tilde{X}_v \neq \tilde{Y}_v \neq \tilde{Z}_v$ for $v = r, s, t$. Then

$$\begin{aligned}
&P\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_k \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}},\ \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\
&\leq P\left(r \in \mathcal{A}_{ik} \middle| E_{ik}^{>1}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) P\left(s \in \mathcal{A}_{jk} \middle| E_{jk}^{>1}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\
&\quad P\left(t \in \mathcal{A}_k \middle| E_k^{>1}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) \\
&= O\left(\frac{1}{L_{XZ} L_{YZ} L_Z}\right).
\end{aligned} \tag{A.22}$$

Next assume that $\tilde{X}_v = \tilde{Y}_v = \tilde{Z}_v$ for $v = r, s, t$, therefore $H_2(\tilde{X}_v) = H_2(\tilde{Y}_v) = H_2(\tilde{Z}_v)$, for $v = r, s, t$. Then,

$$P\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_k \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}},\ \tilde{\mathbf{Z}} = \tilde{\mathbf{z}}\right) = \delta_{ijk} O\left(\frac{1}{L_{XYZ}}\right). \tag{A.23}$$

By using Eqns. A.23, A.22, and A.21 in Eqn. A.17 we obtain an upper bound on bias with collision:

$$\begin{aligned}
\mathbb{B}_\varphi \leq\ & O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}, \tilde{\mathbf{Z}}}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}) \sum_{i,j,k \in \mathcal{F}} P(E_{ijk}) \\
&\left(O\left(\frac{1}{L_{XYZ}}\right) \sum_{r=1}^{L_{XYZ}} \mathbb{E}\left[M_r \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}},\ \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}} \right] \right. \\
&+ \left(O\left(\frac{1}{L_{XZ} L_{YZ} L_Z}\right) + \delta_{ijk} O\left(\frac{1}{L_{XYZ}}\right) \right) \\
&\left. \sum_{r=1}^{L_{XZ}} \sum_{s=1}^{L_{YZ}} \sum_{t=1}^{L_Z} \mathbb{E}\left[\frac{M_r' \widetilde{M}_s}{\overline{M}_t} \middle| E_{ijk}^{>1}, E_{ijk}, \tilde{\mathbf{X}} = \tilde{\mathbf{x}},\ \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}} \right] \right) \\
=\ & O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}, \tilde{\mathbf{Z}}}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}) \sum_{i,j,k \in \mathcal{F}} P(E_{ijk}) \\
&\left(O\left(\frac{N}{L_{XYZ}}\right) + \left(O\left(\frac{N}{L_{XZ} L_{YZ} L_Z}\right) + \delta_{ijk} O\left(\frac{N}{L_{XYZ}}\right) \right) \right).
\end{aligned} \tag{A.24}$$

Re-arranging the expectation term we get,

$$
\begin{aligned}
&= O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \Bigg(O\left(\frac{N}{L_{XYZ}}\right) + \\
&\quad \left(O\left(\frac{N}{L_{XZ}L_{YZ}L_Z}\right) + O\left(\frac{1}{L_{XYZ}}\right)\right)\Bigg) \mathbb{E}\left[\sum_{i,j,k\in\mathcal{F}} \mathbb{1}_{E_{ijk}}\right] \\
&\leq O\left(\frac{1}{\epsilon^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \Bigg(O\left(\frac{N}{L_{XYZ}}\right) + \\
&\quad \left(O\left(\frac{N}{L_{XZ}L_{YZ}L_Z}\right) + O\left(\frac{1}{L_{XYZ}}\right)\right)\Bigg) L_{XYZ} \\
&\leq O\left(\frac{1}{\epsilon^d N}\right).
\end{aligned} \tag{A.25}
$$

Hence as $N \longrightarrow \infty$, the bias estimator due to collision tends to zero i.e. $\mathbb{B}_\varphi \longrightarrow 0$.

Step 3 Combine Results: Let us denote N'_{ijk}, N'_{ik}, N'_{jk}, and N'_k respectively as the number of the input points $(\mathbf{X},\mathbf{Y},\mathbf{Z})$, $(\mathbf{X},\mathbf{Z})$, $(\mathbf{Y},\mathbf{Z})$, and $\mathbf{Z}$ mapped to the bins $(\tilde{X}_i,\tilde{Y}_j,\tilde{Z}_k)$, $(\tilde{X}_i,\tilde{Z}_k)$, $(\tilde{Y}_j,\tilde{Z}_k)$, and $\tilde{Z}_k$ using H_1. We define the notations $r(i) = H_2^{-1}(i)$ for $i \in \mathcal{F}$ and $s(x) := H_1(x)$ for $x \in \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$. Then from Eqn. A.9, we have

$$
\mathbb{E}\left[\frac{N'_{s(X)s(Y)s(Z)}N'_{s(Z)}}{N'_{s(X)s(Z)}N'_{s(Y)s(Z)}}\right] = \frac{dP_{XYZ}\ P_Z}{dP_{XZ}\ P_{YZ}} + \widetilde{\Delta}(\epsilon,q,\gamma) + O\left(\sqrt{\frac{1}{N}}\right). \tag{A.26}
$$

We simplify the first term in Eqn. A.3 as,

$$
\begin{aligned}
&\sum_{i,j,k\in\mathcal{F}} P(E_{ijk}^{\leq 1}) \mathbb{E}\left[\mathbb{1}_{E_{ijk}} \varphi(i,j,k)\ \alpha_{ijk}\ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \big| E_{ijk}^{\leq 1}\right] \\
&= \left(1 - O\left(\frac{1}{\epsilon^d N}\right)\right) \sum_{i,j,k\in\mathcal{F}} \mathbb{E}\left[\mathbb{1}_{E_{ijk}} \varphi(i,j,k)\ \alpha_{ijk}\ \ g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \big| E_{ijk}^{\leq 1}\right] \\
&= \sum_{i,j,k\in\mathcal{F}} \mathbb{E}\left[\mathbb{1}_{E_{ijk}} \varphi(i,j,k)\ \frac{N_{ik}N_{jk}}{N_k N}\ g\left(\frac{N_{ijk}N_k}{N_{ik}N_{jk}}\right) \big| E_{ijk}^{\leq 1}\right] + O\left(\frac{1}{\epsilon^d N}\right) \\
&= \sum_{i,j,k\in\mathcal{F}} \mathbb{E}\Bigg[\mathbb{1}_{E_{ijk}} \varphi(r(i),r(j),r(k))\ \frac{N'_{r(i)r(k)}N'_{r(j)r(k)}}{N'_{r(k)}N} \\
&\qquad\qquad g\left(\frac{N'_{r(i)r(j)r(k)}N'_{r(k)}}{N'_{r(i)r(k)}N'_{r(j)r(k)}}\right)\Bigg] + O\left(\frac{1}{\epsilon^d N}\right).
\end{aligned} \tag{A.27}
$$

Lets denote

$$\beta(r(i), r(j), r(k)) = \frac{N'_{r(i)r(j)r(k)} N'_{r(k)}}{N'_{r(i)r(k)} N'_{r(j)r(k)}}.$$

Therefore the last line in Eqn. A.27 is equal to

$$= \frac{1}{N} \sum_{i,j,k \in \mathcal{F}} \mathbb{E}\left[\varphi(r(i), r(j), r(k)) \; \frac{N'_{r(i)r(j)r(k)}}{\beta(r(i), r(j), r(k))} \; g\Big(\beta(r(i), r(j), r(k))\Big)\right] + O\left(\frac{1}{\epsilon^d N}\right)$$

$$= \frac{1}{N} \mathbb{E}\left[\sum_{i=1}^{N} \frac{\varphi(s(X), s(Y), s(Z))}{\beta(s(X), s(Y), s(Z))} \; g\Big(\beta(s(X), s(Y), s(Z))\Big)\right] + O\left(\frac{1}{\epsilon^d N}\right), \tag{A.28}$$

where,

$$\beta(s(X), s(Y), s(Z)) = \frac{N'_{s(X)s(Y)s(Z)} N'_{s(Z)}}{N'_{s(X)s(Z)} N'_{s(Y)s(Z)}}.$$

The expression in Eqn. A.28 equals:

$$= \mathbb{E}_{P_{XYZ}}\left[\mathbb{E}\left[\frac{\varphi(s(X), s(Y), s(Z))}{\beta(s(X), s(Y), s(Z))} \; g\Big(\beta(s(X), s(Y) \;\; , s(Z))\Big)\Big|\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z}\right]\right]$$
$$+ O\left(\frac{1}{\epsilon^d N}\right)$$

$$= \mathbb{E}_{P_{XYZ}}\left[\varphi(X, Y, Z) h\Big(\frac{dP_{XYZ} \; P_Z}{dP_{XZ} \; P_{YZ}}\Big)\right] + \widetilde{\Delta}(\epsilon, q, \gamma) + O\left(\sqrt{\frac{1}{N}}\right) + O\left(\frac{1}{\epsilon^d N}\right), \tag{A.29}$$

where $h(t) = g(t)/t$ and Eqn. A.29 is derived by borrowing Lemma 7.9 from [88]. Hence from Eqn. A.29 and Eqn. A.3, and the fact that $\widetilde{\Delta}(\epsilon, q, \gamma) \longrightarrow 0$ as $N \to \infty$, we conclude

$$\mathbb{E}\left[\widehat{I}_{\varphi}(X; Y|Z)\right] \longrightarrow \mathbb{E}_{P_{XYZ}}\left[\varphi(X, Y, Z) h\Big(\frac{dP_{XYZ} \; P_Z}{dP_{XZ} \; P_{YZ}}\Big)\right], \text{as} \quad N \to \infty. \tag{A.30}$$

This completes the proof.

BIBLIOGRAPHY

[1] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, "Comparing deep neural networks against humans: Object recognition when the signal gets weaker," *CoRR*, vol. abs/1706.06969, 2017. arXiv: 1706.06969. [Online]. Available: http://arxiv.org/abs/1706.06969.

[2] R. De Man, G. J. Gang, X. Li, and G. Wang, "Comparison of deep learning and human observer performance for detection and characterization of simulated lesions," *Journal of Medical Imaging*, vol. 6, no. 2, 2019.

[3] C. Lu and X. Tang, "Surpassing human-level face verification performance on LFW with gaussianface," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, B. Bonet and S. Koenig, Eds., AAAI Press, 2015, pp. 3811–3819. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9845.

[4] P. J. Phillips, A. N. Yates, Y. Hu, C. A. Hahn, E. Noyes, K. Jackson, J. G. Cavazos, G. Jeckeln, R. Ranjan, S. Sankaranarayanan, *et al.*, "Face recognition accuracy of forensic examiners, superrecognizers, and face recognition algorithms," *Proceedings of the National Academy of Sciences*, vol. 115, no. 24, pp. 6171–6176, 2018.

[5] T. Weyand, I. Kostrikov, and J. Philbin, "Planet - photo geolocation with convolutional neural networks," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., ser. Lecture Notes in Computer Science, vol. 9912, Springer, 2016, pp. 37–55. DOI: 10.1007/978-3-319-46484-8_3. [Online]. Available: https://doi.org/10.1007/978-3-319-46484-8%5C_3.

[6] M. Jethanandani and D. Tang, "Adversarial attacks against lipnet: End-to-end sentence level lipreading," in *2020 IEEE Security and Privacy Workshops, SP Workshops, San Francisco, CA, USA, May 21, 2020*, IEEE, 2020, pp. 15–19. DOI: 10.1109/SPW50608.2020.00020. [Online]. Available: https://doi.org/10.1109/SPW50608.2020.00020.

[7] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto, *et al.*, "The ai index 2021 annual report," *arXiv preprint arXiv:2103.06312*, 2021.

[8] K. Darrah, B. Mehta, and A. Mousavizadeh, *Ai boom time*, 2021. [Online]. Available: https://www.tortoisemedia.com/2021/12/02/ai-boom-time/.

[9] A. Cam, M. Chui, and B. Hall, "Global ai survey: Ai proves its worth, but few scale impact," 2019.

[10] K. He, R. Girshick, and P. Dollár, "Rethinking imagenet pre-training," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4918–4927.

[11] M. Raghu, C. Zhang, J. M. Kleinberg, and S. Bengio, "Transfusion: Understanding transfer learning for medical imaging," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 3342–3352. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/eb1e78328c46506b46a4ac4a1e378b91-Abstract.html.

[12] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. Le, "Rethinking pre-training and self-training," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/27e9661e033a73a6ad8cefcde965c54d-Abstract.html.

[13] *The state of development and operations of ai applications 2019*. [Online]. Available: https://dotscience.com/assets/downloads/Dotscience_Survey-Report-2019.pdf.

[14] *Enterprise trends in machine learning - 2021*. [Online]. Available: https://info.algorithmia.com/hubfs/2020/Reports/2021-Trends-in-ML/Algorithmia_2021_enterprise_ML_trends.pdf?hsLang=en-us.

[15] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, Eds., BMVA Press, 2016. [Online]. Available: http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, IEEE Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90.

[17] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[18] J. Zhao and C. G. Snoek, "Dance with flow: Two-in-one stream action detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9935–9944.

[19] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[20] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, IEEE Computer Society, 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243. [Online]. Available: https://doi.org/10.1109/CVPR.2017.243.

[21] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 6450–6459. DOI: 10.1109/CVPR.2018.00675. [Online]. Available: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Tran%5C_A%5C_Closer%5C_Look%5C_CVPR%5C_2018%5C_paper.html.

[22] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX.

[23] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg.

[24] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," *CoRR*, vol. abs/1806.10282, 2018. arXiv: 1806.10282. [Online]. Available: http://arxiv.org/abs/1806.10282.

[25] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ByME42AqK7.

[26] D. User, *An overview of state of the art (sota) deep neural networks (dnns)*, 2022. [Online]. Available: https://deci.ai/blog/sota-dnns-overview/.

[27] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[28] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, IEEE, 2017, pp. 39–57.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[30] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. Singh and S. Markovitch, Eds., AAAI Press, 2017, pp. 4278–4284. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806.

[31] D.-B. Wang, L. Feng, and M.-L. Zhang, "Rethinking calibration of deep neural networks: Do not be afraid of overconfidence," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 11 809–11 820. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/61f3a6dbc9120ea78ef75544826c814e-Abstract.html.

[32] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 1849–1857. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/6b180037abbebea991d8b1232f8a8ca9-Abstract.html.

[33] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. arXiv: 1212.5701. [Online]. Available: http://arxiv.org/abs/1212.5701.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[35] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2021068.

[36] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2020, pp. 1–16.

[37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*, 2019.

[38] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*, PMLR, 2015, pp. 1737–1746.

[39] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[40] L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, T. Pham, Ed., International Society for Optics and Photonics, vol. 11006, SPIE, 2019, pp. 369–386. DOI: 10.1117/12.2520589. [Online]. Available: https://doi.org/10.1117/12.2520589.

[41] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Workshop and Conference Proceedings, vol. 28, JMLR.org, 2013, pp. 1139–1147. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html.

[42] S. Fox, J. Faraone, D. Boland, K. A. Vissers, and P. H. W. Leong, "Training deep neural networks in low-precision with high accuracy using fpgas," in *International Conference on Field-Programmable Technology, FPT 2019, Tianjin, China, December 9-13, 2019*, IEEE, 2019, pp. 1–9. DOI: 10.1109/ICFPT47387.2019.00009. [Online]. Available: https://doi.org/10.1109/ICFPT47387.2019.00009.

[43] A. Woodie, *Lowering precision does not mean lower accuracy*, 2020. [Online]. Available: https://www.datanami.com/2020/11/09/lowering-precision-does-not-mean-lower-accuracy/.

[44] J. Avrahami, Y. Kareev, Y. Bogot, R. Caspi, S. Dunaevsky, and S. Lerner, "Teaching by examples: Implications for the process of category acquisition," *The Quarterly Journal of Experimental Psychology Section A*, vol. 50, no. 3, pp. 586–606, 1997.

[45] B. F. Skinner, "Reinforcement today.," *American Psychologist*, vol. 13, no. 3, p. 94, 1958.

[46] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x. [Online]. Available: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[47] J. G. Kreer, "A question of terminology," *IRE Trans. Inf. Theory*, vol. 3, no. 3, p. 208, 1957. DOI: 10.1109/TIT.1957.1057418. [Online]. Available: https://doi.org/10.1109/TIT.1957.1057418.

[48] T. Zhou, S. Wang, and J. A. Bilmes, "Curriculum learning by dynamic instance hardness," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[49] T. Zhou and J. Bilmes, "Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity," in *International Conference on Learning Representations*, 2018.

[50] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proceedings of the British Machine Vision Conference 2014*, British Machine Vision Association, 2014. DOI: 10.5244/c.28.88. [Online]. Available: https://doi.org/10.5244%2Fc.28.88.

[51] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[52] L. Lu, M. Guo, and S. Renals, "Knowledge distillation for small-footprint highway networks," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2017. DOI: 10.1109/icassp.2017.7953072. [Online]. Available: https://doi.org/10.1109%2Ficassp.2017.7953072.

[53] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[54] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[55] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2790–2799.

[56] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990.

[57] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in neural information processing systems*, 1993.

[58] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances in neural information processing systems*, 2016.

[59] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *5th International Conference on Learning Representations, ICLR*, 2017.

[60] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJGCiw5gl.

[61] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[62] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *CoRR*, vol. abs/1607.03250, 2016. arXiv: 1607.03250. [Online]. Available: http://arxiv.org/abs/1607.03250.

[63] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/46a4378f835dc8040c8057beb6a2da52-Abstract.html.

[64] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=B1VZqjAcYX.

[65] C. Wang, G. Zhang, and R. B. Grosse, "Picking winning tickets before training by preserving gradient flow," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=SkgsACVKPH.

[66] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Pruning neural networks at initialization: Why are we missing the mark?" In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=Ig-VyQc-MLK.

[67] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *IEEE International Conference on Computer Vision*, 2017.

[68] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 2074–2082. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/41bfd20a38bb1b0bec75acf0845530a7-Abstract.html.

[69] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[70] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016.

[71] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *European conference on computer vision*, 2018.

[72] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun, "Oicsr: Out-in-channel sparsity regularization for compact deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[73] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *IEEE International Conference on Computer Vision*, 2017.

[74] J. Yoon and S. J. Hwang, "Combined group and exclusive sparsity for deep neural networks," in *International Conference on Machine Learning-Volume 70*, 2017.

[75] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018.

[76] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," *arXiv preprint arXiv:2003.13683*, 2020.

[77] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[78] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Advances in neural information processing systems*, 2017.

[79] J.-H. Luo and J. Wu, "An entropy-based pruning method for cnn compression," *arXiv preprint arXiv:1706.05791*, 2017.

[80] B. Dai, C. Zhu, B. Guo, and D. Wipf, "Compressing neural networks using the variational information bottleneck," in *International Conference on Machine Learning*, 2018.

[81] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, 2004.

[82] K. R. Moon, K. Sricharan, and A. O. Hero, "Ensemble estimation of mutual information," in *2017 IEEE International Symposium on Information Theory*, 2017.

[83] J. C. Principe, D. Xu, J. Fisher, and S. Haykin, "Information theoretic learning," *Unsupervised adaptive filtering*, 2000.

[84] H. H. Yang and J. Moody, "Data visualization and feature selection: New algorithms for nongaussian data," in *Advances in neural information processing systems*, 2000.

[85] N. Leonenko, L. Pronzato, V. Savani, *et al.*, "A class of rényi information estimators for multidimensional densities," *The Annals of Statistics*, 2008.

[86] M. Noshad, K. R. Moon, S. Y. Sekeh, and A. O. Hero, "Direct estimation of information divergence using nearest neighbor ratios," in *2017 IEEE International Symposium on Information Theory*, 2017.

[87] S. Yasaei Sekeh and A. O. Hero, "Geometric estimation of multivariate dependency," *Entropy*, 2019.

[88] M. Noshad, Y. Zeng, and A. O. Hero, "Scalable mutual information estimation using dependence graphs," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019.

[89] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[90] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[91] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *international conference on machine learning*, PMLR, 2017, pp. 1311–1320.

[92] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on robot learning*, PMLR, 2017, pp. 482–495.

[93] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky, "From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing," in *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, The Association for Computational Linguistics, 2010, pp. 751–759. [Online]. Available: https://aclanthology.org/N10-1116/.

[94] A. Pentina, V. Sharmanska, and C. H. Lampert, "Curriculum learning of multiple tasks," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, IEEE Computer Society, 2015, pp. 5492–5500. DOI: 10.1109/CVPR.2015.7299188. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7299188.

[95] D. Weinshall, G. Cohen, and D. Amir, "Curriculum learning by transfer learning: Theory and experiments with deep networks," in *International Conference on Machine Learning*, PMLR, 2018, pp. 5238–5246.

[96] C. Gong, D. Tao, S. J. Maybank, W. Liu, G. Kang, and J. Yang, "Multi-modal curriculum learning for semi-supervised image classification," *IEEE Trans. Image Process.*, vol. 25, no. 7, pp. 3249–3260, 2016. DOI: 10.1109/TIP.2016.2563981. [Online]. Available: https://doi.org/10.1109/TIP.2016.2563981.

[97] P. Soviany, C. Ardei, R. T. Ionescu, and M. Leordeanu, "Image difficulty curriculum for generative adversarial networks (cugan)," in *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, IEEE, 2020, pp. 3452–3461. DOI: 10.1109/WACV45572.2020.9093408. [Online]. Available: https://doi.org/10.1109/WACV45572.2020.9093408.

[98] D. Zhang, D. Meng, C. Li, L. Jiang, Q. Zhao, and J. Han, "A self-paced multiple-instance learning framework for co-saliency detection," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, IEEE Computer Society, 2015, pp. 594–602. DOI: 10.1109/ICCV.2015.75. [Online]. Available: https://doi.org/10.1109/ICCV.2015.75.

[99] Y. Fan, R. He, J. Liang, and B.-G. Hu, "Self-paced learning: An implicit regularization perspective," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, S. Singh and S. Markovitch, Eds., AAAI Press, 2017, pp. 1877–1883. [Online]. Available: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14374.

[100] L. Jiang, D. Meng, S.-I. Yu, Z.-Z. Lan, S. Shan, and A. G. Hauptmann, "Self-paced learning with diversity," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 2078–2086. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/c60d060b946d6dd6145dcbad5c4ccf6f-Abstract.html.

[101] G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 2535–2544.

[102] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *International Conference on Machine Learning*, PMLR, 2018, pp. 2304–2313.

[103] T. Zhou, S. Wang, and J. Bilmes, "Curriculum learning by optimizing learning dynamics," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 433–441.

[104] Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu, "Learning to teach," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=HJewuJWCZ.

[105] M. Toneva, A. Sordoni, R. T. des Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," in *International Conference on Learning Representations*, 2018.

[106] M. R. Smith, T. Martinez, and C. Giraud-Carrier, "An instance level analysis of data complexity," *Machine learning*, vol. 95, no. 2, pp. 225–256, 2014.

[107] I. Loshchilov and F. Hutter, "Online batch selection for faster training of neural networks," *arXiv preprint arXiv:1511.06343*, 2015.

[108] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian, "Disturblabel: Regularizing cnn on the loss layer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4753–4762.

[109] G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. E. Hinton, "Regularizing neural networks by penalizing confident output distributions," *CoRR*, 2017. arXiv: 1701.06548.

[110] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[111] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi, "Label refinery: Improving imagenet classification through label progression," *arXiv preprint arXiv:1805.02641*, 2018.

[112] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii, "Distributional smoothing by virtual adversarial examples," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[113] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[114] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "Icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[115] F. M. Castro, M. J. Marın-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 233–248.

[116] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *International Conference on Machine Learning*, 2018, pp. 4535–4544.

[117] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 348–358.

[118] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6467–6476.

[119] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-GEM," in *International Conference on Learning Representations*, 2019.

[120] M. Bucher, S. Herbin, and F. Jurie, "Hard negative mining for metric learning based zero-shot classification," in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds., Cham: Springer International Publishing, 2016, pp. 524–531, ISBN: 978-3-319-49409-8.

[121] X. Li, C. M. Snoek, M. Worring, D. Koelma, and A. W. Smeulders, "Bootstrapping visual categorization with relevant negatives," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 933–945, 2013.

[122] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2794–2802.

[123] A. Shrivastava, A. Gupta, and R. B. Girshick, "Training region-based object detectors with online hard example mining," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, IEEE Computer Society, 2016, pp. 761–769. DOI: 10.1109/CVPR.2016.89. [Online]. Available: https://doi.org/10.1109/CVPR.2016.89.

[124] O. Canévet and F. Fleuret, "Efficient sample mining for object detection," in *Proceedings of the Sixth Asian Conference on Machine Learning, ACML 2014, Nha Trang City, Vietnam, November 26-28, 2014*, D. Q. Phung and H. Li, Eds., ser. JMLR Workshop and Conference Proceedings, vol. 39, JMLR.org, 2014. [Online]. Available: http://proceedings.mlr.press/v39/canevet14a.html.

[125] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *International Conference on Machine Learning*, 2019.

[126] ——, "Theoretically principled trade-off between robustness and accuracy," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 7472–7482. [Online]. Available: http://proceedings.mlr.press/v97/zhang19p.html.

[127] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of supervised models through robust optimization," *Neurocomputing*, vol. 307, pp. 195–204, 2018.

[128] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 2206–2216. [Online]. Available: http://proceedings.mlr.press/v119/croce20b.html.

[129] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, B. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, Eds., ACM, 2017, pp. 27–38. DOI: 10.1145/3128572.3140451. [Online]. Available: https://doi.org/10.1145/3128572.3140451.

[130] Y. Zhang, W. Ruan, F. Wang, and X. Huang, "Generalizing universal adversarial attacks beyond additive perturbations," in *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, C. Plant, H. Wang, A. Cuzzocrea, C. Zaniolo, and X. Wu, Eds., IEEE, 2020, pp. 1412–1417. DOI: 10.1109/ICDM50108.2020.00186. [Online]. Available: https://doi.org/10.1109/ICDM50108.2020.00186.

[131] L. Rice, E. Wong, and Z. Kolter, "Overfitting in adversarially robust deep learning," in *International Conference on Machine Learning*, PMLR, 2020, pp. 8093–8104.

[132] J. Cui, S. Liu, L. Wang, and J. Jia, "Learnable boundary guided adversarial training," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 721–15 730.

[133] Q.-Z. Cai, C. Liu, and D. Song, "Curriculum adversarial training," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 3740–3747.

[134] J. Zhang, X. Xu, B. Han, G. Niu, L. Cui, M. Sugiyama, and M. Kankanhalli, "Attacks which do not kill training make adversarial learning stronger," in *International Conference on Machine Learning*, PMLR, 2020, pp. 11 278–11 287.

[135] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu, "On the convergence and robustness of adversarial training," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6586–6595.

[136] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent advances in adversarial training for adversarial robustness," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, Z.-H. Zhou, Ed., ijcai.org, 2021, pp. 4312–4321. DOI: 10.24963/ijcai.2021/591. [Online]. Available: https://doi.org/10.24963/ijcai.2021/591.

[137] F. W. andDBLP:journals/corr/abs-2103-03076 Yanghao Zhang, Y. Zheng, and W. Ruan, "Gradient-guided dynamic efficient adversarial training," *CoRR*, vol. abs/2103.03076, 2021. arXiv: 2103.03076. [Online]. Available: https://arxiv.org/abs/2103.03076.

[138] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!" *Advances in Neural Information Processing Systems*, vol. 32, pp. 3358–3369, 2019.

[139] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=BJx040EFvH.

[140] J. H. Friedman, L. C. Rafsky, *et al.*, "Graph-theoretic measures of multivariate association and prediction," *The Annals of Statistics*, vol. 11, no. 2, pp. 377–391, 1983.

[141] R. Sen, A. T. Suresh, K. Shanmugam, A. G. Dimakis, and S. Shakkottai, "Model-powered conditional independence test," in *NIPS'17 Proceedings of the 31th International Conference on Neural Information Processing Systems*, 2017.

[142] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[143] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[144] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[145] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[146] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016. DOI: 10.1109/cvpr.2016.90. [Online]. Available: https://doi.org/10.1109%2Fcvpr.2016.90.

[147] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[148] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[149] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[150] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[151] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[152] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[153] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 1321–1330.

[154] Y. Suhov, I. Stuhl, S. Yasaei Sekeh, and M. Kelbert, "Basic inequalities for weighted entropies," *Aequationes mathematicae*, 2016.

[155] T. Cover and J. A. Thomas, *Elements of information theory*. Chichester: 1st edn. John Wiley & Sons, 1991.

[156] I. Csiszár and P. C. Shields, "Information theory and statistics: A tutorial," *J. Royal Statist. Soc. Ser. B (Methodology.)*, 2004.

[157] V. Berisha and A. O. Hero, "Empirical non-parametric estimation of the fisher information," *IEEE Signal Processing Letters*, vol. 22, no. 7, pp. 988–992, 2014.

[158] V. Berisha, A. Wisler, A. O. Hero, and A. Spanias, "Empirically estimable classification bounds based on a nonparametric divergence measure," *IEEE Transactions on Signal Processing*, vol. 64, no. 3, pp. 580–591, 2015.

[159] S. Yasaei Sekeh and A. O. Hero, "Geometric estimation of multivariate dependency," *Entropy (Women in Information Theory)*, 2018.

[160] W. Härdle, *Applied Nonparametric Regression*. Cambridge University Press, 1990.

[161] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[162] M. R. Ganesh, J. J. Corso, and S. Y. Sekeh, "MINT: deep network compression via mutual information-based neuron trimming," in *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, IEEE, 2020, pp. 8251–8258. DOI: 10.1109/ICPR48806.2021.9412590. [Online]. Available: https://doi.org/10.1109/ICPR48806.2021.9412590.

[163] A. Prabhu, G. Varma, and A. Namboodiri, "Deep expander networks: Efficient deep networks from graph theory," in *European Conference on Computer Vision*, 2018.

[164] N. Gkalelis and V. Mezaris, "Fractional step discriminant pruning: A filter pruning framework for deep convolutional neural networks," in *IEEE International Conference on Multimedia & Expo Workshops*, 2020.

[165] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, and C. Xu, "Scop: Scientific control for reliable neural network pruning," *arXiv preprint arXiv:2010.10732*, 2020.

[166] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2019.

[167] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[168] P. Rodrıguez, M. A. Bautista, J. Gonzalez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and Vision Computing*, vol. 75, pp. 21–31, 2018.

[169] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[170] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *British Machine Vision Conference 2016*, British Machine Vision Association, 2016.

[171] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: Multilevel residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1303–1314, 2017.

[172] B. Graham, "Fractional max-pooling (2014)," *arXiv preprint arXiv:1412.6071*, 2014.

[173] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[174] S. Liang, Y. Kwoo, and H. Yang, "Drop-activation: Implicit parameter reduction and harmonic regularization," *arXiv preprint arXiv:1811.05850*, 2018.

[175] Y. Yamada, M. Iwamura, T. Akiba, and K. Kise, "Shakedrop regularization for deep residual learning," *arXiv preprint arXiv:1802.02375*, 2018.

[176] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[177] À. Lapedriza, H. Pirsiavash, Z. Bylinskii, and A. Torralba, "Are all training examples equally valuable?" *CoRR*, vol. abs/1311.6510, 2013. arXiv: 1311.6510. [Online]. Available: http://arxiv.org/abs/1311.6510.

[178] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *International Conference on Learning Representations*, 2018.

[179] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, pp. 3630–3638, 2016.

[180] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556.

[181] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474. [Online]. Available: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Sandler%5C_MobileNetV2%5C_Inverted%5C_Residuals%5C_CVPR%5C_2018%5C_paper.html.

[182] G. Huang, Z. Liu, G. Pleiss, L. Van Der Maaten, and K. Weinberger, "Convolutional networks with dense connectivity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[183] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 9185–9193. DOI: 10.1109/CVPR.2018.00957. [Online]. Available: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Dong%5C_Boosting%5C_Adversarial%5C_Attacks%5C_CVPR%5C_2018%5C_paper.html.

[184] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille, "Improving transferability of adversarial examples with input diversity," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, Computer Vision Foundation / IEEE, 2019, pp. 2730–2739. DOI: 10.1109/CVPR.2019.00284. [Online]. Available: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Xie%5C_Improving%5C_Transferability%5C_of%5C_Adversarial%5C_Examples%5C_With%5C_Input%5C_Diversity%5C_CVPR%5C_2019%5C_paper.html.

[185] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, IEEE Computer Society, 2017, pp. 39–57. DOI: 10.1109/SP.2017.49. [Online]. Available: https://doi.org/10.1109/SP.2017.49.

[186] H. Kim, "Torchattacks: A pytorch repository for adversarial attacks," *arXiv preprint arXiv:2010.01950*, 2020.