Slimming Neural Networks Using Adaptive Connectivity Scores

Madan Ravi Ganesh, Dawsin Blanchard, Jason J. Corso, *Senior Member, IEEE,* and Salimeh Yasaei Sekeh, *Member, IEEE*

***Abstract*—In general, deep neural network (DNN) pruning methods fall into two categories: 1) Weight-based deterministic constraints, and 2) Probabilistic frameworks. While each approach has its merits and limitations there are a set of common practical issues such as, trial-and-error to analyze sensitivity and hyper-parameters to prune DNNs, which plague them both. In this work, we propose a new single-shot, fully automated pruning algorithm called *Slimming Neural networks using Adaptive Connectivity Scores* (SNACS). Our proposed approach combines a probabilistic pruning framework with constraints on the underlying weight matrices, via a novel connectivity measure, at multiple levels to capitalize on the strengths of both approaches while solving their deficiencies. In SNACS, we propose a fast hash-based estimator of *Adaptive Conditional Mutual Information* (ACMI), that uses a weight-based scaling criterion, to evaluate the connectivity between filters and prune unimportant ones. To automatically determine the limit up to which a layer can be pruned, we propose a set of operating constraints that jointly define the upper pruning percentage limits across all the layers in a deep network. Finally, we define a novel *sensitivity* criterion for filters that measures the strength of their contributions to the succeeding layer and highlights critical filters that need to be completely protected from pruning. Through our experimental validation we show that SNACS is faster by over 17x the nearest comparable method and is the state of the art single-shot pruning method across three standard Dataset-DNN pruning benchmarks: CIFAR10-VGG16, CIFAR10-ResNet56 and ILSVRC2012-ResNet50.**

***Index Terms*—Neural Network Compression, Pruning, Mutual Information, Multivariate Dependency Measure, Sensitivity.**

I. Introduction

Critical real-world applications like autonomous vehicle navigation [1], [2] and simultaneous machine translation [3], [4] demand real-time response [5] without any compromise in performance. Proposed solutions in these application domains are implicitly bound to constraints brought forward by restricted space and memory availability on custom hardware implementations. These factors are at odds with the general research design goal of high performance in deep neural networks (DNN), which is often achieved by increasing the overall size and capacity of the DNN. The trade-off between these constraints has brought increased attention to the field of DNN pruning [6], [7], whose main objective is to maintain an adequate level of performance, often within a few percent of the original DNN, while only using a fraction of its memory or FLOPs. Of course, the adequacy of any level of performance depends on the specific application, but the general goal nevertheless remains critical.

There are two main approaches to pruning: 1) deterministic constraints on weight matrices [8]–[10] and 2) probabilistic frameworks [11]–[13]. Methods based on deterministic constraints on weight matrices are straightforward to implement and do 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 modelling the sensitivity of filters at an individual level. In a sense, the two types of method are converses: one's weakness is remedied by the other. Yet, to the best of our knowledge, there has been no recent work that combines both approaches and improves upon them. Further, there are many unresolved practical issues among both approaches, e.g., the labor intensive process of analyzing the sensitivity of different layers to pruning or imposing an upper limit on pruning percentage for each layer and the amount of resources and time spent in iteratively pruning DNNs.

To that end, we are able to unify the benefits of both methods while mitigating their respective drawbacks: we propose *Slimming Neural networks using Adaptive Connectivity Scores* (SNACS) as a hybrid single-shot pruning approach. In SNACS, we introduce the *Adaptive Conditional Mutual Information* (ACMI) measure, which incorporates weights as a scaling function within the framework of conditional mutual information [14], [15]. The ACMI measure evaluates the connectivity between pairs of filters across adjacent layers and prunes unimportant filters. In this work, we explore weight and activation-based scaling functions.

To remove the manual effort involved in setting the upper pruning percentage limit of layers, we define a set of operating constraints to automatically evaluate them. The constraints are based on the degradation in quality of activations at various levels of compression. Additionally, we encapsulate the importance of a filter using our 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. We highlight all the main components of SNACS in Fig. 1.

Madan Ravi Ganesh is with the Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI, USA.

Dawsin Blanchard and Salimeh Yasaei Sekeh are with the School of Computing and Information Science, University of Maine, ME, USA.

Jason J. Corso was previously with the University of Michigan and is currently with the Stevens Institute for Artificial Intelligence, Stevens Institute of Technology, NJ, USA

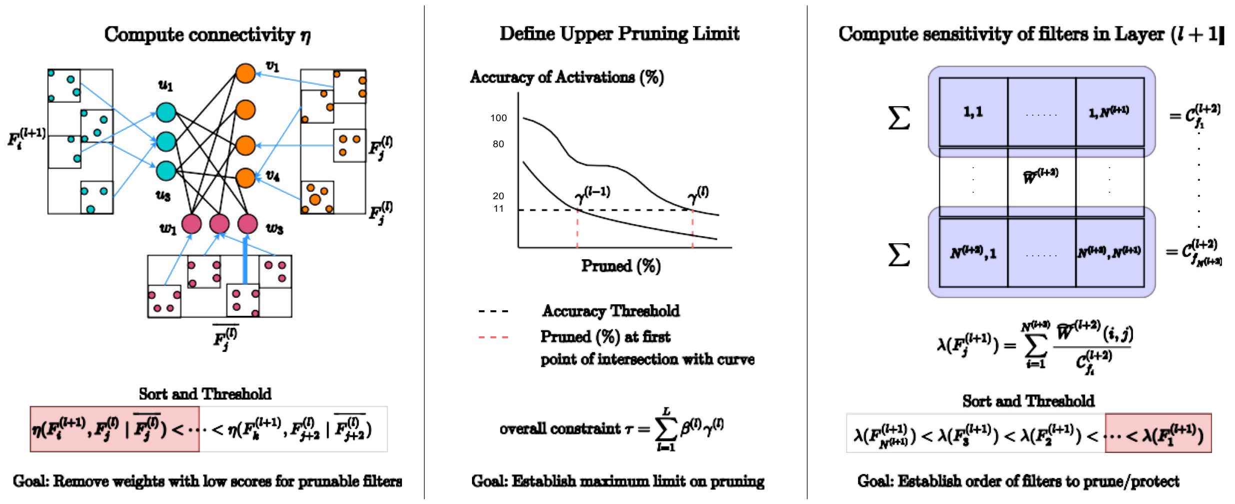

Compute connectivity η
Sort and Threshold
Goal: Remove weights with low scores for prunable filters
Define Upper Pruning Limit
Accuracy of Activations (%)
Pruned (%)
Accuracy Threshold
Pruned (%) at first point of intersection with curve
overall constraint $\tau = \sum_{l=1}^{L} \beta^{(l)}\gamma^{(l)}$
Goal: Establish maximum limit on pruning
Compute sensitivity of filters in Layer $(l+1)$
Sort and Threshold
Goal: Establish order of filters to prune/protect

Fig. 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 need to be pruned. Next, to protect the network from being irregularly/excessively pruned, we use 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 which should be protected from pruning. Combining the information from all three components we prune layer $l+1$.

Overall, we summarize our contributions in this work below,

- We propose a hybrid single-shot pruning approach, SNACS, which takes advantage of both a probabilistic pruning framework and simple weight-based constraints.
- In SNACS, we propose the use of Adaptive Conditional Mutual Information (ACMI) as a way to measure the connectivity between filters and derive its hash-table-based implementation,
- In the interest of simplifying the process of defining upper pruning percentage limits of layers in a DNN, we propose a set of operating constraints to help automate their definition, and
- We apply a custom notion of *Sensitivity* in filters, using their contribution to succeeding layers, to prioritize the pruning of largely insensitive filters while protecting highly sensitive ones.

By incorporating our contributions within the SNACS framework, we improve the overall run-time of the pruning algorithm by upwards of $17\times$, increase the accuracy of the estimator, create an entirely automated pruning pipeline while offering state of the art performance in single-shot pruning of DNNs.

II. Related Works

In the following subsections, we discuss prior works in pruning and mutual information (MI) estimators, as well as methods at their intersection. Among pruning approaches there are two broad categories: 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. Within the first category of methods, there is a subset that enforces sparsity by modifying the objective function while the remaining directly apply constraints on the weight matrices.

A. Deterministic Constraints on Weight Matrix

Direct Constraint on Weight Matrices: Some of the earliest works in pruning used the second-order relationship between the objective function and weights of the network to evaluate and remove unimportant values [16], [17]. Since then, several advancements in the form of directly thresholding weights [10], [18] or using the $||\cdot||_1$ constraint to define the importance of filters [19] have been proposed. A more recent subset of methods have adopted data-driven logic to derive the importance of filter weights. Two such methods are ThiNet [9] and NISP [20], where the reconstruction of outcomes with the removal of weights is posed as a post-training objective. By virtue of how direct constraints are placed on weight matrices, 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 the combination of a weight-based scaling function and filter connectivity within a probabilistic framework to maintain the flow of information between layers and overcome these issues.

Modification of Objective Function: Inducing sparsity in weight matrices by modifying the objective function involves imposing a strong constraint on how weights develop during training. Constraints range from simple methods, such as single or multiple l_n norms [21] on channel outputs, simple patterned masks to regulate group sparsity [22], and optimization over group-lasso-based objective functions [8], [23], to more complicated ideas like balancing individual vs. group sparsity constraints [24], [25] and adding discrimination-aware losses at intermediate layers to enhance and easily identify important channels [26].

More recent methods combine the idea of modifying the objective function with more abstract concepts like meta-learning [27], where sparsity inducing regularizers are used to learn latent vectors that help decide on the weight values directly or GANs, where an adversarial pruned network generator optimizes a loss based on the features derived from the original network [28]. To provide a controlled set up to study and compare the effects of pruning a network against its original counter-part, we avoid strong comparisons against methods that modify the objective function. Apart from optimizing over a fundamentally different objective function, which are harder to optimize, these methods require multiple iterations of pruning and fine-tuning built-in to their setup, while we use only a single pruning and retraining step while targeting a simple objective function.

B. Probabilistic Frameworks

Pruning approaches that use a probabilistic frameworks can be divided into bayesian and non-bayesian methods. Bayesian methods apply a variational bayesian inference perspective to pruning, with a focus on estimating the posterior distribution of weights using ELBO [29], [30]. While they offer a theoretically sound perspective to pruning, they require a strong assumption on the prior distribution of weights which induces sparsity across the network. Further, their performances on large-scale datasets have more room to grow.

The non-bayesian approach to pruning focuses on using information theoretic measures, with minimal assumptions and widespread applicability when compared to the bayesian methods. These include, Luo and Wu [13], in which entropy of activations is used as a measure of importance of a filter, VIBNet [12], where the information bottleneck principle is used to minimize the redundancy between adjacent layers and MINT [11], in which geometric conditional MI is used to determine the dependencies between filter pairs in adjacent layers. While they are adept at reducing redundancy and maintaining the flow of information between layers, they are slightly slow and inefficient at modelling the sensitivity of individual filters to pruning. In SNACS, we propose the use of ACMI which improves the speed of dependency computations for MI as well as the accuracy of the estimates. Further, by highlighting sensitive filters that need to remain un-pruned and jointly defining the upper pruning percentage limit of layers we obtain additional gains when pruning a DNN.

C. Multivariate Dependency Measures

Approaches for estimating multivariate dependencies using MI can be broadly classified into two categories: plugin and direct estimation. Plugin estimators like Kernel Density Estimators (KDEs) [31], KNN estimators [32], and others [33], [34] 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 un-scalable. To overcome these issues, direct estimators for Renyi-entropy and MI [33], [34], and Henze-Penrose divergence measure [35] 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 [36], which uses the k-NN graph to estimate MI, and the minimum spanning tree used to estimate the GMI [37]. These graph-based approaches help make the evaluation of MI computationally tractable. While most methods fall into either plugin or direct categories, recent work has focused on the development of a hybrid approach [38]. This approach combines the fast run-time implementation of hash-tables with an error convergence rate akin to plugin methods, thus merging the advantages of both the estimation approaches.

III. Algorithm and Component Description

In the following subsections, we outline SNACS's algorithm. This is followed by details of the ACMI measure and the set of constraints that automatically define the upper pruning percentage limits of layers in a DNN. The final sub-section explains our notion of sensitivity, which identifies and protects important filters from being pruned.

A. Notation

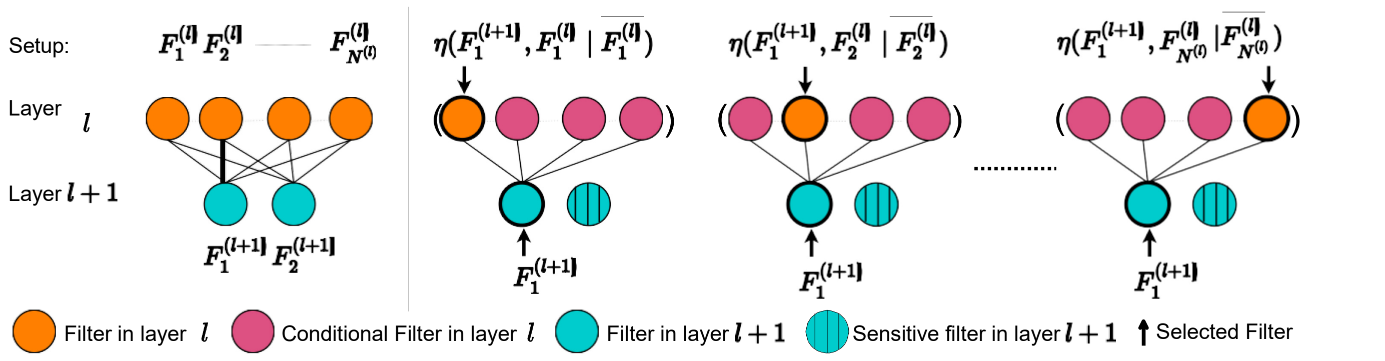
We assume that a given DNN has a total of L layers where,

- Sensitive filters() : Function that returns the indices of a subset of filters that need to be protected from pruning, computed using sensitivity (Section III-E).
- $F_i^{(l+1)}$: 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 filter indices whose values are pruned from the weight vector.
- $\eta()$: Connectivity score between two filters computed using ACMI (Section III-C).
- $F_j^{(l)}$: The set of all filters excluding $F_j^{(l)}$ in layer l.
- 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 III-D.

B. Algorithm

The overall goal of our algorithm 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. We apply SNACS between every pair of adjacent layers in 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 on the activations from each set of filters. An example of this is shown in Fig. 2. The connectivity score evaluates the strength of the relationship between



Setup:
$F_1^{(l)}$ $F_2^{(l)}$ $F_{N^{(l)}}^{(l)}$
Layer l
Layer $l+1$
$F_1^{(l+1)}$ $F_2^{(l+1)}$
$\eta(F_1^{(l+1)}, F_1^{(l)} \mid \overline{F_1^{(l)}})$
$\eta(F_1^{(l+1)}, F_2^{(l)} \mid \overline{F_2^{(l)}})$
$\eta(F_1^{(l+1)}, F_{N^{(l)}}^{(l)} \mid \overline{F_{N^{(l)}}^{(l)}})$
$F_1^{(l+1)}$
Filter in layer l
Conditional Filter in layer l
Filter in layer $l+1$
Sensitive filter in layer $l+1$
Selected Filter

Fig. 2: 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.

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 do not exceed the pre-determined upper limit, denoted by $\gamma^{(l+1)}$, we add the index of the filter to $S_{F^{(l+1)}}$. The weights for retained and protected filters/neurons are untouched while the weights for the entire kernel/elements are zeroed out for pruned filters/neurons.

In the practical implementation of Alg. 1 the value of δ is determined by thresholding η values from a chosen layer so as to remove sufficient weights and match the predetermined γ^l. 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. 1, 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.

Algorithm 1: 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$(\{1, 2, \ldots N^{(l+1)}\})$ do
        Initialize $S_{F_i^{(l+1)}} = \emptyset$;
        for $F_j^{(l)}, j \in 1, 2, \ldots N^{(l)}$ do
            Compute $\eta(F_i^{(l+1)}, F_j^{(l)} \mid \overline{F_j^{(l)}})$;
            if $\eta(F_i^{(l+1)}, F_j^{(l)} \mid \overline{F_j^{(l)}}) \leq \delta$ and $\left(\sum_i |S_{F_i^{(l+1)}}| / (N^{(l+1)} N^{(l)}) < \gamma^{(l+1)}\right)$ then
                $S_{F_i^{(l+1)}} = S_{F_i^{(l+1)}} \cup \text{index}(F_j^{(l)})$
            end
        end
    end
end
```

C. Adaptive Conditional Mutual Information

In the following subsection, we introduce Adaptive Mutual Information, a non-linear dependency measure that is based on the f-divergence measure [15], [39], 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 [14], for given function $(x, y) \in \mathcal{X} \times \mathcal{Y} \mapsto \phi(x, y) \geq 0$, the Adaptive Mutual Information (AMI), denoted by $I_\phi(X; Y)$, is defined as,

$$I_\phi(X; Y) = \mathbb{E}_{P_X P_Y}\left[\phi(X, Y)\, g\left(\frac{dP_{XY}}{dP_X P_Y}\right)\right], \quad (1)$$

where $\frac{dP_{XY}}{dP_X P_Y}$ is the Radon-Nikodym derivative, and $g : (0, \infty) \to \mathbb{R}$ is a convex function and $g(1) = 0$. Note that when $\frac{dP_{XY}}{dP_X P_Y} \to 1$ then $I_\phi \to 0$. The overall bounds on the AMI measure are given by,

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

An explanation of how we arrive at these bounds is provided in Appendix A.

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 [14], the Adaptive Conditional Mutual

Information (ACMI), denoted by $I_\phi(X; Y|Z)$, is defined as,

$$I_\phi(X; Y|Z) = \mathbb{E}_{P_Z P_{X|Z} P_{Y|Z}} \left[\phi(X, Y, Z) g\left(\frac{dP_{XY|Z}}{dP_{X|Z} P_{Y|Z}} \right) \right]. \quad (3)$$

In this paper, we focus on the particular case of $g(t) = \frac{(t-1)^2}{2(t+1)}$ so $I_\phi \in [0, 1]$. Note that when $\phi = 1$, the ACMI in (3) becomes the conditional geometric MI measure proposed in [40]. 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 $\{(X_i, Y_i, Z_i)\}_{i=1}^N$ 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 [38], 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,

$$N_{ijk} = \#\{(X_t, Y_t, Z_t) \ s.t. \ H(X_t) = i, \quad (4)$$
$$H(Y_t) = j, H(Z_t) = k\},$$

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,

$$r_{ijk} := \frac{N_{ijk}}{N}, \ r_{ik} := \frac{N_{ik}}{N}, \ r_{jk} := \frac{N_{jk}}{N}, \ r_k := \frac{N_k}{N}. \quad (5)$$

Finally, using the above ratios we propose the following hash-based estimator of the ACMI measure (3):

$$I_\phi(X; Y|Z) = \sum_{e_{ijk} \in E_G} \phi(i, j, k) \frac{r_{ik} r_{jk}}{r_k} g\left(\frac{r_{ijk} r_k}{r_{ik} r_{jk}} \right), \quad (6)$$

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, [41]. For fixed d_X, d_Y, and d_Y, as $N \to \infty$ we have*

$$I_\phi(X; Y|Z) \longrightarrow I_\phi(X; Y|Z), \quad a.s. \quad (7)$$

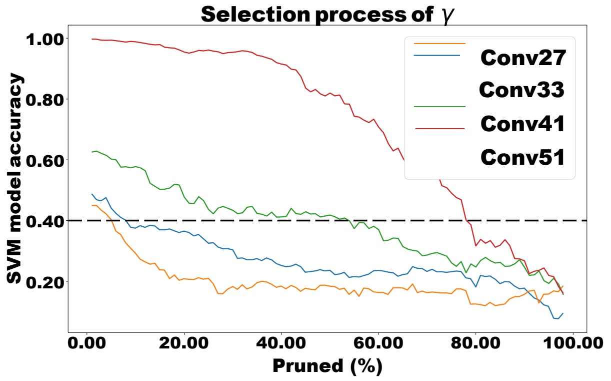


Fig. 3: The selection process for upper pruning percentage limits for each layer of the DNN is based on using a fixed threshold (dotted line) over the SVM model's performance such that the weighted sum of Pruned(%) allocated to each layer, the x coordinate where the threshold intersects the curve latest, matches the overall sparsity τ.

The proof of Theorem 1 is available in the Appendices.

Implementation: Overall, X, Y, 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 (6). The flexibility in defining function ϕ offers a way to connect the probabilistic framework of MI to existing weight-based pruning approaches. In Section IV, 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 in our experiments.

D. Definition of Upper Pruning Percentage 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 is based on trends in the degradation of the quality of activations when a layer is pruned to varying extents. At each layer, we collect the performances of an SVM model with an RBF kernel $(\alpha^{(l)})_c$ 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, the ground-truth labels from the dataset are used to train the SVM model.

Once we have the performance of SVM models across all layers, we cycle through performances between $100 - 0\%$ 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 individual 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, lower is 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,

$$\tau = \sum_{l=1}^{L} \beta^{(l)}\gamma^{(l)}, \tag{8}$$

where $\beta^{(l)}$ is ratio of number of parameters in layer l to the total number of parameters across the entire DNN and τ denotes the desired pruning percentage across the entire DNN. Fig. 3 illustrates this process using an example of 4 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 when compared 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 contribution in this work is a systematic approach to decide 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 of all layers for each DNN architecture used in the experiments in our supplementary materials.

E. 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 approaches like MINT [11] aim to maintain the flow of information between a pair of layers but they consider all filters to be 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. 1. 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, 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,

$$\lambda(F_i^{l+1}) = \sum_{f_c=1}^{N^{(l+2)}} \bar{W}^{(l+2)}(f_c, i) \Big/ C^{(l+2)}(f_c), \tag{9}$$

$$\text{where } C^{(l+2)}(f_c) = \sum_{f_p=1}^{N^{(l+1)}} \bar{W}^{(l+2)}(f_c, f_p). \tag{10}$$

Here, $C^{(l+2)}$ is the normalization constant used to relate the weights of filters from $l+1$ contributing to the same filter in $l+2$ and $\bar{W}^{(l+2)}$ 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 un-pruned, after empirically comparing the improvement in performance at similar pruning levels with and without protecting sensitive filters. This is critical to ensure that only sensitive filters, which contribute a majority of the information downstream, remain untouched. This in turn helps improve the overall compression performance since less sensitive filters can be pruned more without compromising the quality of information flowing between layers too much. After empirically comparing the degradation in performance of the SVM model used to define the upper pruning percentage limits for layers, between the case when all the filters are pruned and when we protect a variable percentage of sensitive filters, we determine the set of highly sensitive filters to protect and return their indices to Alg. 1.

IV. Experimental Results

We divide our results into three subsections, formatted as an ablative study. Section IV-A focuses on the evaluation of run-time and choice of ϕ, to highlight the impact of using our ACMI estimator in place of the MST-based estimator used in [11]. 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 IV-B, we detail the results of applying SNACS (ACMI + Automated upper pruning percentage limits) across three 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 IV-C we discuss the impact of adding our sensitivity measure as a way to prioritize and fully protect important filters from being pruned.

Dataset-DNN: We use three standard Dataset-DNN combinations to evaluate and compare our approach to standard baselines. They are, CIFAR10 [42]-VGG16 [43], CIFAR10-ResNet56 [44] and ILSVRC2012 [45]-ResNet50. A detailed breakdown of each dataset and the experimental setup used in each experiment is included in the supplementary materials.

Metric: We use the following metrics to compare performances,

- 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 accuracy on the testing set, after re-training for pruned networks,
- Memory (Mb): The amount of memory consumed to store the weight matrices in "CSR" format.
- FLOPs Reduced (%): The percentage of FLOPs reduced when compared to the un-pruned DNN.

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

Φ function	Pruned (%)
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
$\exp(-\frac{\|\text{weights}\|_2^2 \|\text{act}\|_2^2}{2})$	76.99

Apart from the above metrics, we also use *run-time* to compare speed of estimators. A high quality method must have high compression performance while maintaining a test accuracy relatively close to the baseline.

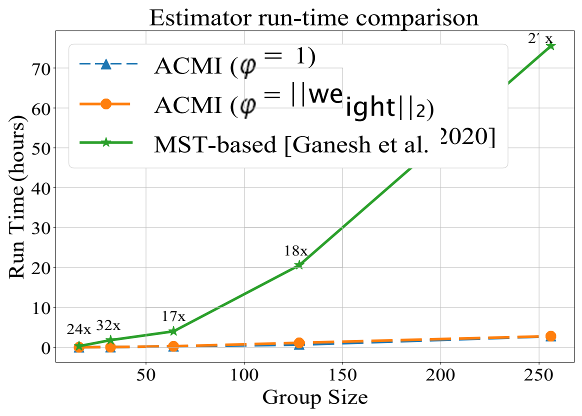
A. Evaluation of Estimator

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 [11]. For this experiment, we use three distinct estimators, the MST-based estimator from MINT, our ACMI estimator with $\phi = 1$ and $\phi = \|\text{weight}\|_2$. 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 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 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.

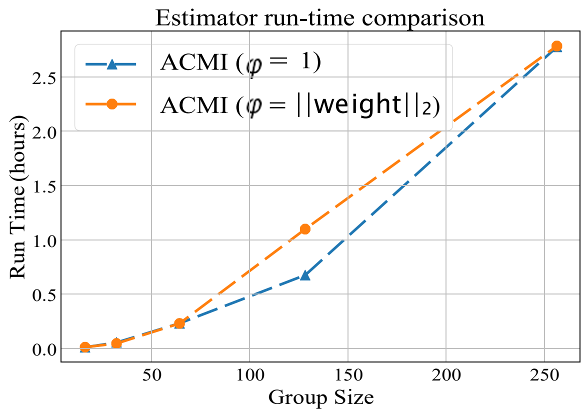
Selection of ϕ: There are number of potential functions we can associate with ϕ. In Table I, we illustrate a variety of functions and their performance, w.r.t. the Pruning (%) while maintaining an accuracy $\geq 93.43\%$ in the VGG16-CIFAR10 setup. Between Section IV-A and MINT [11] the main differences are the inclusion of ACMI and the manual definition of upper pruning percentage limits using artificially capped SVM model accuracies (0.8). From Table I, we observe that most variants of ϕ outperform MINT, including $\phi = 1$. Furthermore, we find that $\phi = \exp(\frac{-\text{weights}^2}{2})$ 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.

B. Large-scale Comparison

When compared to existing single-shot pruning methods, from Table II 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 I) and the joint definition of upper pruning percentage limits for each layer in the DNN are the main contributors to our high performance.



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



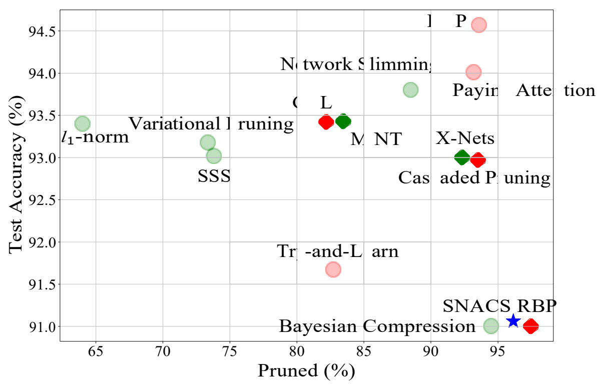
(b) Run-time comparison across various ϕ

Fig. 4: (4a) When comparing run-times between the MST-based estimator used in MINT [11] and our hash-based ACMI estimator, our estimator provides up to $27\times$ speedup in run-time. (4b) Across different selections of the scaling function in our estimator, the run-times scale similarly as the number of groups increase.

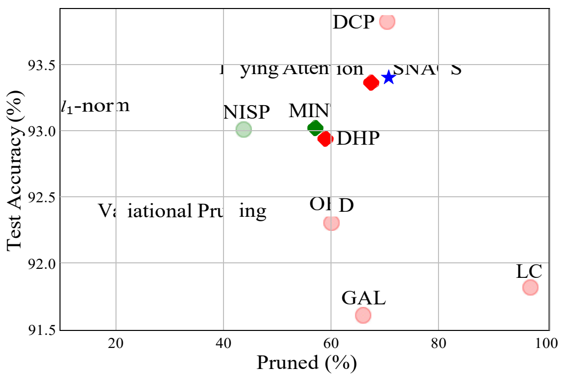
Fig. 5 helps put SNACS's performance in perspective of pruning approaches that use either sparsity inducing objective functions or iterative re-training setups. In general, we expect a decrease in performance with an increase in the number of parameters pruned. Often, iterative approaches achieve the highest compression while suffering minimal drop in testing accuracy, with methods 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 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 large pruning

TABLE II: 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 (%)

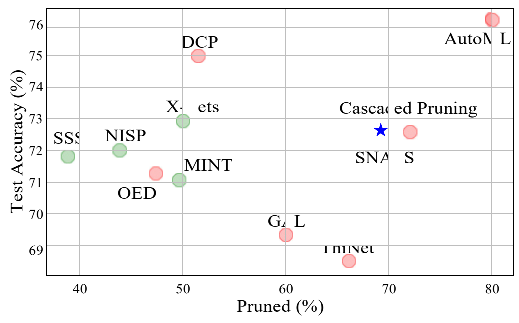
	Method	Pruning (%)	Test Accuracy (%)	FLOPs Reduced (%)
CIFAR-10 VGG16	Baseline	N.A.	93.98	N.A.
	l_1-norm [19]	64.00	93.40	34.18
	Variational Pruning [29]	73.34	93.18	39.29
	SSS [23]	73.80	93.02	41.60
	MINT [11]	83.46	93.43	N.A.
	Network Slimming [21]	88.52	93.80	50.94
	X-Nets [46]	92.33	93.00	N.A.
	Bayesian Compression [30]	94.50	91.00	N.A.
	SNACS	96.16	91.06	67.85
CIFAR-10 ResNet56	Baseline	N.A.	92.55	N.A.
	l_1-norm [19]	13.70	93.06	27.28
	Variational Pruning [29]	20.49	92.26	20.17
	NISP [20]	42.60	93.01	43.61
	FSDP [47]	50.00	92.64	N.A.
	MINT [11]	57.01	93.02	N.A.
	SNACS	68.59	93.38	37.61
ILSVRC2012 ResNet50	Baseline	N.A.	76.13	N.A.
	SSS [23]	38.82	71.82	43.04
	NISP [20]	43.82	71.99	44.01
	MINT [11]	49.62	71.05	N.A.
	X-Nets [46]	50.00	72.85	50.00
	SNACS	55.10	74.65	41.73
	SNACS	59.61	73.60	46.63
	SNACS	64.26	72.90	51.65
	SNACS	68.80	72.36	56.79



(a) CIFAR10-VGG16



(b) CIFAR10-ResNet56



(c) ILSVRC2012-ResNet50

Fig. 5: 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.

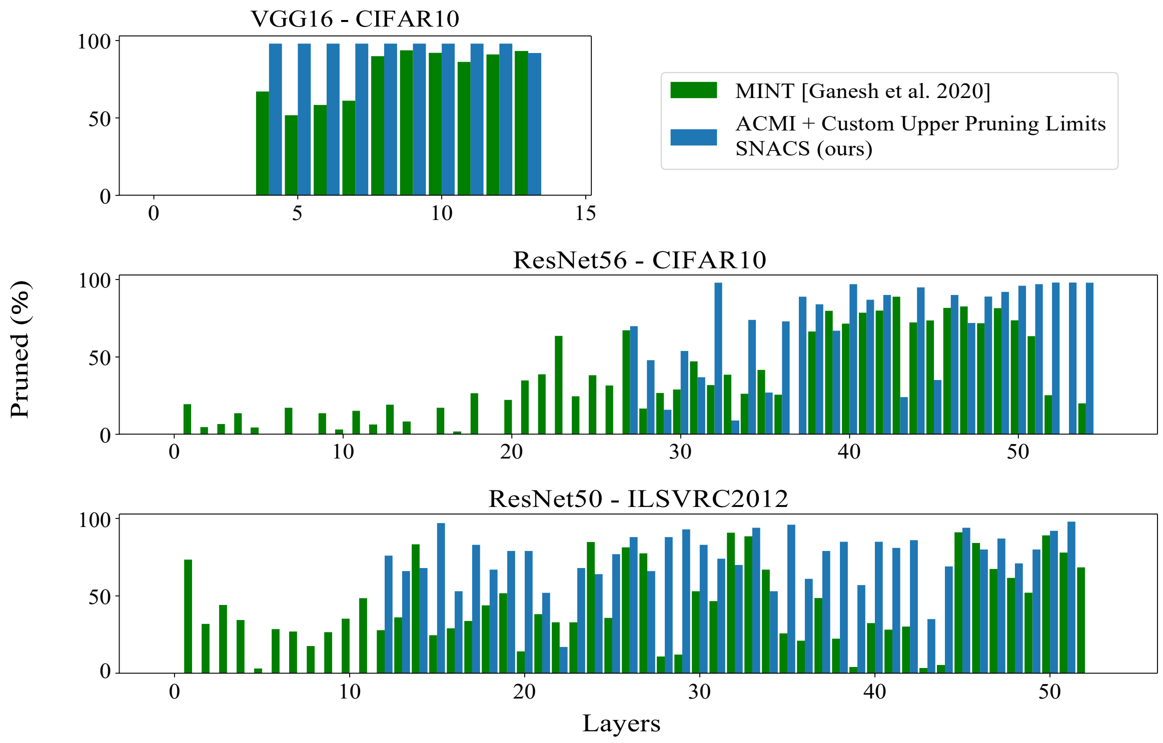


Fig. 6: On observing the compression performance per layer in the ILSVRC2012-ResNet50 experiment, SNACS is able to achieve high Pruning (%) while focusing only on the middle and latter layers and avoiding the early layers. Interestingly, the pattern of pruning in MINT and SNACS is extremely different.

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

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

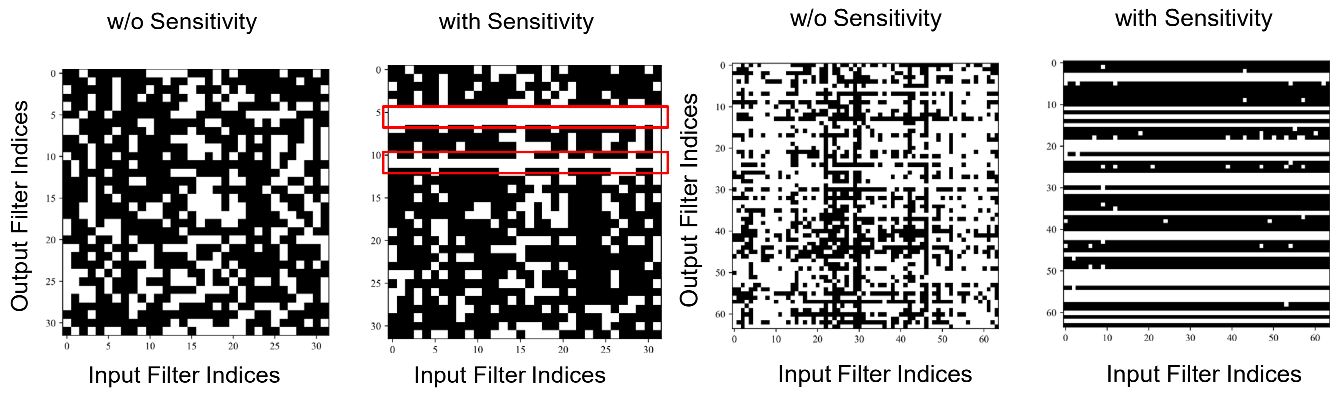
levels despite using a **single** prune-retrain step.

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. 6. Given that a large portion of FLOPs are concentrated in the early portion of the network, the percentage of FLOPs reduced by our SNACS is slightly lower when compared to methods like X-Nets, which preemptively prunes the network before training, or SSS, which optimizes a different objective function altogether. Interestingly, on closer inspection of Fig. 6, we observe 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 set 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 were co-opted from prior works where the focus 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 when using SNACS DNNs are more forgiving when pruning layers closer to the output 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 [47] and the median oracle ranking statistics per layer from [48]. However, these observations are in direct contrast to previous works which identify that portions of the network closer to the input are often pruned first [23], [28]. We hypothesize that their outcomes stem from the modification of the objective function and subsequent training of baseline networks whereas our approach and those in [47], [48] focus on removing filters based on a pre-defined criterion without the modification of the loss function.

C. *Sensitivity-based Pruning*

Experiments in Sections IV-A and IV-B assumed that all filters contributed equally to the information flow downstream



(a) Convolution 35 (b) Convolution 46

Fig. 7: Illustrations of filters retained (white) and pruned (black) w/o and with sensitivity based pruning. When protecting important filters from pruning, all its associate connections are maintained (red highlight). An interesting impact of sensitivity is that the connections pruned can be completely modified compared to its counterpart w/o pruning. This is illustrated by the pruning mask of convolution 46.

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

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

and hence, the connectivity scores were 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. 7a and 7b, 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. 6 Convolution 32, 34, and many others) we observe a complete restructure in the way filters are pruned. Fig. 7b 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 III.

Across the results presented in Table III, the percentage of filters protected from pruning are 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 the case when no filters are protected. The performance comparison is restricted to SVM model only and no re-training is necessary. When we relax this constraint (Table IV), 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 in order to obtain the optimal combination of high sparsity with accuracy.

V. Conclusion

Overall, we propose a novel DNN pruning algorithm called SNACS which uses ACMI to measure the connectivity between filters, a simple set of operating constraints to automate the definition of upper pruning percentage limits of layers in a DNN and a sensitivity criterion that helps protect a subset of critical filters from pruning. SNACS provides a faster overall run-time and improves accuracy in the estimation process, offers state-of-the-art levels of compression using a single train-prune-retrain cycle while the sensitivity criterion can be used to further boost the compression performance. An important direction of future work is to extend this algorithm to an iterative approach and incorporate it into the training phase. Doing so would help reduce the overall training time while achieving extreme levels of sparsity. Additionally, characterizing the pruned networks using a multitude of events like adversarial attacks, calibration error and many others could shed light on how close such networks are to being deployed in the real-world.

Appendix A
Bounds on AMI

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

$$I_\phi(X;Y) = \frac{1}{2}\mathbb{E}_{P_XP_Y}\left[\phi(X,Y)\left(\frac{dP_{XY}}{dP_XP_Y}+1\right)\right] \tag{11}$$

$$-2\,\mathbb{E}_{P_XP_Y}\left[\phi(X,Y)\,h\left(\frac{dP_{XY}}{dP_XP_Y}\right)\right], \tag{12}$$

where $h(t) = \frac{t}{t+1}$. When $\frac{dP_{XY}}{dP_XP_Y} = 1$, then the minimum value of I_ϕ is zero. Further, when P_{XY} and P_XP_Y have no overlapping space then the second term in (11) becomes zero. Therefore, bounds on I_ϕ is given as,

$$0 \le I_\phi(X,Y) \le \frac{1}{2}\mathbb{E}_{P_XP_Y}\left[\phi(X,Y)\left(\frac{dP_{XY}}{dP_XP_Y}+1\right)\right]. \tag{13}$$

Appendix B
Proof of Theorem 1

Recall our estimator in Section III-C,

$$\hat{I}_\phi(X;Y|Z) = \sum_{e_{ijk}\in E_G}\phi(i,j,k)\,\alpha_{ijk}\,g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right), \tag{14}$$

where $\alpha_{ijk} = \frac{r_{ik}r_{jk}}{r_k}$. The expectation of $\hat{I}_\phi$ is derived as

$$\mathbb{E}\left[\sum_{e_{ijk}\in E_G}\phi(i,j,k)\,\alpha_{ijk}\,g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\mathbb{1}_{E_G}\right] \tag{15}$$

$$= \sum_{e_{ijk}\in E_G}\mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\,g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\mathbb{1}_{E_{ijk}}\right], \tag{16}$$

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, Y, and Z to $\tilde{X}_k$, $\tilde{Y}_k$, and $\tilde{Z}_k$. Following the notations used in [38], 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. 16 by splitting it into two parts: without collision and due to collision. Based on the law of total expectation we have,

$$= \sum_{e_{ijk}\in E_G} P(E_{ijk}^{=1}|E_{ijk})\,\mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\,g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\mathbb{1}_{E_{ijk}^{=1},E_{ijk}}\right]$$
$$+ \sum_{e_{ijk}\in E_G} P(\overline{E_{ijk}^{=1}}|E_{ijk})\,\mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\,g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right)\mathbb{1}_{\overline{E_{ijk}^{=1}},E_{ijk}}\right]. \tag{17}$$

Step 1 Bias on w/o collision: Similar to Lemma 7.3 in [38], we derive,

$$P(E_{ijk}^{=1}|E_{ijk}) = 1 - O\left(\frac{1}{E^dN}\right),\quad d = d_X + d_Y + d_Z. \tag{18}$$

This is because all three $|V|$, $|U|$, and $|W|$ are upper bounded by $O(E^{-d})$. Note that E is a function of N. Additionally from [38] we infer the following results:

$$\mathbb{E}[\alpha_{ijk}] = \frac{\mathbb{E}[r_{ik}]\,\mathbb{E}[r_{jk}]}{\mathbb{E}[r_k]} + O\left(\frac{1}{\sqrt{N}}\right). \tag{19}$$

Note that (19) is implied based on the fact that $\mathbb{V}(\alpha_{ijk}) \le O(1/N)$ which is proved by applying Efron-Stein inequality under assumptions **(A1)** and **(A3)**, similar to arguments in Lemma 7.10 from [38]. In addition, we have

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\right] = \frac{\mathbb{E}[r_{ijk}]}{\mathbb{E}[\alpha_{ijk}]} + O\left(\frac{1}{\sqrt{N}}\right), \tag{20}$$

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\right] = P(E_{ijk}^{\le 1})\,\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\Big|E_{ijk}^{\le 1}\right] + P(E_{ijk}^{>1})\,\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\Big|E_{ijk}^{>1}\right], \tag{21}$$

where by using similar arguments as in Eqn. 56 from [38], we have $P(E_{ijk}^{\le 1}) = 1 - O(\sqrt{1/(E^dN)})$. Therefore, $P(E_{ijk}^{>1}) = O(\sqrt{1/(E^dN)})$. Further the second term in Eqn. 21 is the bias because of collision of H, which will be proved in the following section, that is upper bounded by $O(\sqrt{1/(E^dN)})$.

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,

$$\mathbb{E}[r_{ijk}|E_{ijk}^{\le 1}] = P(X_D = x_D, Y_D = y_D, Z_D = z_D)\; E^{d_C+d'_C+d''_C}\left(f(x_C,y_C,z_C|x_D,y_D,Z_D) + \Delta(E,q,\gamma)\right), \tag{22}$$

where densities have bounded derivatives up to the order $q \ge 0$ and belong to the Hölder continuous class with smoothness parameter γ. Note that $\Delta(E,q,\gamma) \to 0$ as $N \to \infty$. Now from Eqns. 50, 51, and 53 in [38] and from Eqn. 19, 20 above, under assumptions **(A1)** and **(A3)**, we derive

$$\mathbb{E}\left[\frac{r_{ijk}}{\alpha_{ijk}}\Big|E_{ijk}^{\le 1}\right] = \frac{dP_{XYZ}\,P_Z}{dP_{XZ}\,P_{YZ}} + \Delta(E,q,\gamma) + O\left(\frac{1}{\sqrt{N}}\right), \tag{23}$$

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

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

$$B_\phi := \sum_{e_{ijk}\in E_G} P(\overline{E^{=1}_{ijk}}|E_{ijk}) \mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mid E^{=1}_{ijk}, E_{ijk}\right]$$
$$\leq \sum_{i,j,k\in F} P(E^{>1}_{ijk}) \mathbb{E}\left[1_{E_{ijk}}\phi(i,j,k)\,\alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mid E^{>1}_{ijk}\right], \quad (24)$$

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 X_i that map to v_i using H_2. Once again, using the law of total expectation, then the RHS of Eqn. 24 becomes

$$= \sum_{i,j,k\in F} P(E^{>1}_{ijk}) \Big(P(E_{ijk}|E^{>1}_{ijk}) \mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mid E^{>1}_{ijk}, E_{ijk}\right]$$
$$+ P(\overline{E_{ijk}}|E^{>1}_{ijk}) \mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mid E^{>1}_{ijk}, \overline{E_{ijk}}\right]\Big)$$
$$= \sum_{i,j,k\in F} P(E_{ijk}) P(E^{>1}_{ijk}|E_{ijk}) \mathbb{E}\left[\phi(i,j,k)\,\alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mid E^{>1}_{ijk}, E_{ijk}\right]. \quad (25)$$

The equality in Eqn. 25 is obtained based on Bayes error and $g = 0$ on the event $\overline{E_{ijk}}$. Now recalling Eqn. 18, using Eqn. 13 we bound the last line in Eqn. 25 by,

$$O\left(\frac{1}{E^d N}\right) \sum_{i,j,k\in F} P(E_{ijk}) \mathbb{E}\left[\phi(i,j,k)\,(r_{ijk}+\alpha_{ijk}) \mid E^{>1}_{ijk}, E_{ijk}\right] \quad (26)$$

This implies that

$$B_\phi \leq O\left(\frac{1}{E^d N}\right) \sum_{i,j,k\in F} P(E_{ijk}) \Big(\mathbb{E}\left[\phi(i,j,k) r_{ijk} \mid E^{>1}_{ijk}, E_{ijk}\right] + \mathbb{E}\left[\phi(i,j,k)\alpha_{ijk} \mid E^{>1}_{ijk}, E_{ijk}\right]\Big)$$
$$= O\left(\frac{1}{E^d N^2}\right) \sum_{i,j,k\in F} P(E_{ijk}) \Big(\mathbb{E}\left[\phi(i,j,k) N_{ijk} \mid E^{>1}_{ijk}, E_{ijk}\right] + \mathbb{E}\left[\phi(i,j,k) \frac{N_{ik}N_{jk}}{N_{k}} \mid E^{>1}_{ijk}, E_{ijk}\right]\Big)$$

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

$$= O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in F} P(E_{ijk}) \Big(\mathbb{E}\left[\phi(i,j,k) N_{ijk} \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]$$
$$+ \mathbb{E}\left[\phi(i,j,k) \frac{N_{ik}N_{jk}}{N_k} \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]\Big). \quad (28)$$

Let us define,

$$A_{ijk} := \left\{r : H_2(\tilde{X}_r) = i, H_2(\tilde{Y}_r) = j, H_2(\tilde{Z}_r) = k\right\},$$
$$A_{k} := \left\{r : H_2(\tilde{Z}_r) = k\right\},$$
$$A_{ik} := \left\{r : H_2(\tilde{X}_r) = i, H_2(\tilde{Z}_r) = k\right\},$$
$$A_{jk} := \left\{r : H_2(\tilde{Y}_r) = j, H_2(\tilde{Z}_r) = k\right\}. \quad (29)$$

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

$$N_{ijk} = \sum_{r=1}^{L_{XYZ}} 1_{A_{ijk}}(r) M_r. \quad (30)$$

Similarly M'_r, $\overline{M}_s$, and 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

$$N_{ik} = \sum_{r=1}^{L_{XZ}} 1_{A_{ik}}(r) M'_r, \quad N_{jk} = \sum_{s=1}^{L_{YZ}} 1_{A_{jk}}(s) \overline{M}_s, \quad (31)$$

$$N_k = \sum_{t=1}^{L_Z} 1_{A_k}(t) M_t. \quad (32)$$

Under the assumption that ϕ is bounded, we have

$$B_\phi \leq O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in F} P(E_{ijk})$$
$$\Big(\sum_{r=1}^{L_{XYZ}} P\left(r \in A_{ijk} \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \mathbb{E}\left[M_r \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]$$
$$+ \sum_{r=1}^{L_{XZ}}\sum_{s=1}^{L_{YZ}}\sum_{t=1}^{L_Z} P\left(r \in A_{ik}, s \in A_{jk}, t \in A_k \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right)$$
$$\mathbb{E}\left[\frac{M'_r \overline{M}_s}{M_t} \mid E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right]\Big)$$

$$\mathsf{E} \quad M_r^{\mathsf{i}}\overline{M}_s \, {}^1E^{>1}, E \quad , \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}}, \tilde{\mathbf{Z}} = \tilde{\mathbf{z}} \quad \mathsf{J} \quad . \tag{33}$$

$$N_k \quad {}^{ijk} \quad {}^{ijk}\,\mathsf{IJ} \quad . \quad +\mathsf{E} \quad \phi(i, j, k) \quad {}_{ijk} \, {}^1E \quad , E \tag{27}$$

Next we find the probability terms:

$$P\left(r \in \mathcal{A}_{ijk} \mid \mathbf{1}_{E^{>1}_{ijk}}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) = \frac{P\left(r \in \mathcal{A}_{ijk}, E^{>1}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right)}{P\left(E^{>1}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right)}. \tag{34}$$

We first find the denominator of Eqn. 34 first. We define $a=1$ when $i=j=k$ and $a=3$ for the case $i \neq j \neq k$:

$$\begin{aligned}
P\left(E^{>1}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right)
&= 1 - P\left(E^{=0}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&\quad - P\left(E^{=1}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&= 1 - \left(\frac{F-a}{F}\right)^{L_{XYZ}} - \left(\frac{L_{XYZ}}{F^a}\right)\left(\frac{F-a}{F}\right)^{L_{XYZ}-a} \\
&= O\left(\frac{L^2_{XYZ}}{F^{a+1}}\right).
\end{aligned} \tag{35}$$

Further,

$$\begin{aligned}
P\left(r \in \mathcal{A}_{ijk}, E^{>1}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right)
&= P\left(r \in \mathcal{A}_{ijk} \mid E^{>1}_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&\quad P\left(r \in \mathcal{A}_{ijk} \mid \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&= \left(1 - \left(\frac{F-a}{F}\right)^{L_{XYZ}-a}\right)\left(\frac{1}{F}\right)^{a} = O\left(\frac{L_{XYZ}}{F^{a+1}}\right).
\end{aligned} \tag{36}$$

Combining Eqn. 35 and 36 yields

$$P\left(r \in \mathcal{A}_{ijk} \mid \mathbf{1}_{E^{>1}_{ijk}}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) = O\left(\frac{1}{L_{XYZ}}\right). \tag{37}$$

Now we simplify the following term:

$$\Pr\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_{k} \mid \mathbf{1}_{E^{>1}_{ijk}}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right). \tag{38}$$

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

$$\begin{aligned}
&P\left(r \in \mathcal{A}_{ik}, s \in \mathcal{A}_{jk}, t \in \mathcal{A}_{k} \mid \mathbf{1}_{E^{>1}_{ijk}}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&\le P\left(r \in \mathcal{A}_{ik} \mid \mathbf{1}_{E^{>1}_{ik}}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&\quad P\left(s \in \mathcal{A}_{jk} \mid \mathbf{1}_{E^{>1}_{jk}}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&\quad P\left(t \in \mathcal{A}_{k} \mid \mathbf{1}_{E^{>1}_{k}}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}\right) \\
&= O\left(\frac{1}{L_{XZ}L_{YZ}L_Z}\right).
\end{aligned} \tag{39}$$

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

By using Eqns. 40, 39, and 37 in Eqn. 33 we obtain an upper bound on bias with collision:

$$\begin{aligned}
B_\phi &\le O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in F} P\left(E_{ijk}\right) \\
&\quad \Bigg(O\left(\left(\frac{1}{L_{XYZ}}\right) \sum_{r=1}^{L_{XYZ}} \mathbb{E}\left[M_r \mathbf{1}_{E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}}\right]\right) \\
&\quad + O\left(\frac{1}{L_{XZ}L_{YZ}L_Z}\right) + \delta_{ijk} O\left(\frac{1}{L_{XYZ}}\right) \\
&\quad \sum_{r=1}^{L_{XZ}} \sum_{s=1}^{L_{YZ}} \sum_{t=1}^{L_Z} \mathbb{E}\left[\frac{M_r M_s}{M_t} \mathbf{1}_{E^{>1}_{ijk}, E_{ijk}, \tilde{\mathbf{X}}=\tilde{\mathbf{x}}, \tilde{\mathbf{Y}}=\tilde{\mathbf{y}}, \tilde{\mathbf{Z}}=\tilde{\mathbf{z}}}\right]\Bigg) \\
&= O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \sum_{i,j,k\in F} P\left(E_{ijk}\right) \\
&\quad \left(O\left(\frac{N}{L_{XYZ}}\right) + O\left(\frac{N}{L_{XZ}L_{YZ}L_Z}\right) + \delta_{ijk} O\left(\frac{N}{L_{XYZ}}\right)\right).
\end{aligned} \tag{41}$$

Re-arranging the expectation term we get,

$$\begin{aligned}
&= O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \left(O\left(\frac{N}{L_{XYZ}}\right) + O\left(\frac{N}{L_{XZ}L_{YZ}L_Z}\right) + O\left(\frac{1}{L_{XYZ}}\right)\right) \\
&\quad \mathbb{E}\left[\sum_{i,j,k\in F} \mathbf{1}_{E_{ijk}}\right] \\
&\le O\left(\frac{1}{E^d N^2}\right) \sum_{\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}} p_{\tilde{\mathbf{X}},\tilde{\mathbf{Y}},\tilde{\mathbf{Z}}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \left(O\left(\frac{N}{L_{XYZ}}\right) + O\left(\frac{N}{L_{XZ}L_{YZ}L_Z}\right) + O\left(\frac{1}{L_{XYZ}}\right)\right) L_{XYZ} \\
&\le O\left(\frac{1}{E^d N}\right).
\end{aligned} \tag{42}$$

Hence as $N \longrightarrow \infty$, the bias estimator due to collision tends to zero i.e. $B_\phi \longrightarrow 0$.

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 $i \in F$ and $s(x) := H_1(x)$ for $x \in X \cup Y \cup Z$.

Then from Eqn. 23, we have

$$\frac{M_{s(X)s(Y)s(Z)} M_{s(Z)}}{M_{s(X)s(Z)} M_{s(Y)s(Z)}}$$

$P \; r \in \mathcal{A}, s \in \mathcal{A}, t \in \mathcal{A} \; 1 E^{>1}, E, \tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \tilde{\mathbf{Y}} = \tilde{\mathbf{y}},$

$\tilde{\mathbf{Z}} = \tilde{\mathbf{z}} = \delta$

$O \; L_{XYZ} 1 .$ (40)

$\mathbb{E}_{N} \; N$

$= \; dP_{XYZ} P_Z \; dP_{XZ} P_{YZ} + \Delta(E, q, \gamma) + O \; \frac{1}{N} .$ (43)

We simplify the first term in Eqn. 17 as,

$$P\left(E^{\le 1}_{ijk}\right) \sum_{i,j,k\in \mathbf{F}} \mathbb{E}_{E_{ijk}}\left[\phi(i,j,k)\, \alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mathbf{1}_{E^{\le 1}_{ijk}}\right]$$

$$= \left(1 - O\left(\frac{1}{E^{d}N}\right)\right) \sum_{i,j,k\in \mathbf{F}} \mathbb{E}\left[\mathbf{1}_{E_{ijk}} \phi(i,j,k)\, \alpha_{ijk}\, g\left(\frac{r_{ijk}}{\alpha_{ijk}}\right) \mathbf{1}_{E^{\le 1}}\right]$$

$$= \sum_{i,j,k\in \mathbf{F}} \mathbb{E}_{E_{ijk}} \phi(i,j,k) \frac{N_{ik}N_{jk}}{N_k N}\, g\left(\frac{N_{ijk}N}{N_{ik}N_{jk}}\right) \mathbf{1}_{E^{\le 1}_{ijk}} + O\left(\frac{1}{E^{d}N}\right)$$

$$= \sum_{i,j,k\in \mathbf{F}} \mathbb{E}\, \mathbf{1}_{E_{ijk}} \phi(r(i), r(j), r(k)) \frac{N'_{r(i)r(k)}N'_{r(j)r(k)}}{N'_{r(k)}N}\, g\left(\frac{N'_{r(i)r(j)r(k)}N'_{r(k)}}{N'_{r(i)r(k)}N'_{r(j)r(k)}}\right) + O\left(\frac{1}{E^{d}N}\right). \tag{44}$$

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. 44 is equal to

$$= \frac{1}{N} \sum_{i,j,k\in \mathbf{F}} \phi(r(i), r(j), r(k)) \frac{N'_{r(i)r(j)r(k)}}{\beta(r(i), r(j), r(k))}\, g\left(\beta(r(i), r(j), r(k))\right) + O\left(\frac{1}{E^{d}N}\right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{\phi(s(X), s(Y), s(Z))}{\beta(s(X), s(Y), s(Z))}\, g\left(\beta(s(X), s(Y), s(Z))\right) + O\left(\frac{1}{E^{d}N}\right), \tag{45}$$

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. 45 equals:

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

$$= \mathbb{E}_{P_{XYZ}} \phi(X, Y, Z)\, h\left(\frac{dP_{XYZ}\, P_Z}{dP_{XZ}\, P_{YZ}}\right) + \Delta(E, q, \gamma) + O\left(\frac{1}{N}\right) + O\left(\frac{1}{E^{d}N}\right), \tag{46}$$

where $h(t) = g(t)/t$ and Eqn. 46 is derived by borrowing Lemma 7.9 from [38]. Hence from Eqn. 46 and Eqn. 17, and the fact that $\Delta(E, q, \gamma) \longrightarrow 0$ as $N \to \infty$, we conclude

$$\mathbb{E}\, I_{\phi}(X; Y|Z) \longrightarrow \mathbb{E}_{P_{XYZ}} \phi(X, Y, Z)\, h\left(\frac{dP_{XYZ}\, P_Z}{dP_{XZ}\, P_{YZ}}\right),$$

as $N \to \infty$. (47)

This completes the proof.

Appendix C
Complexity of SNACS

We breakdown the discussion on the computational complexity of SNACS into two parts, 1) the complexity of the hash-based ACMI estimator, and 2) the complexity of Algorithm 1 in the main paper.

A. Complexity of hash-based estimator

By extending the discussion provided in [38], 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.

B. Complexity of Algorithm 1 (Main Paper)

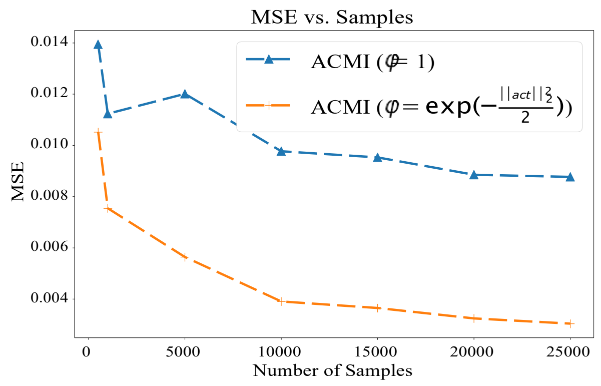
There are 2 primary factors which affect the complexity of Algorithm 1 in the main paper, 1) the number of groups associated with each layer l and $l+1$, and 2) the total number of layers in the DNN. The internal double **FOR** loop has an upper bound of $O(N^{(l)}N^{(l+1)})$, if the number of groups 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.

Appendix D
Validating the Estimator

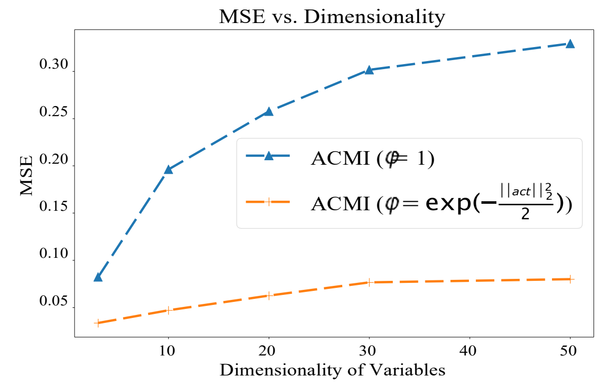
In this section we validate the MSE performance of the ACMI estimator across various dimensionalities and total number of samples to asses the trends in estimation accuracy.

A. Setup

To observe the performance of the estimator when the number of samples are varied, we set the dimensionality of X, Y to one and Z to two. This setup is used to mimic the dimensionality 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, 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 as the identity function and μ is zero.



(a) MSE vs. No. of Samples



(b) MSE vs. Dimensionality

Fig. 8: (**Fig. 8a**) An increase in the number of samples while dimensionality of input variables are held constant shows steadily decreasing MSE. (**Fig. 8b**) 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.

B. Results

Fig. 8 shows the results of our experiments where in Fig. 8a, we observe the steady decrease in MSE as the number of samples are 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. 8b illustrates the steady increase in MSE when the number of samples are held constant but the dimensionality of the input variables grows larger. Further, the trends from secondary curves with $\phi = \exp(-\frac{||act||_2^2}{2})$ show that the inclusion of a scaling term improves the overall performance. Thus, our observations match the expected trends from a valid estimator.

Appendix E
Dataset and Preprocessing

A. CIFAR10

This 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, the images are only normalized and provided to the DNN.

B. ILSVRC2012

This 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.

Appendix F
Experimental Setup

Throughout our experiments we use three major Dataset-DNN combinations, CIFAR10-VGG16, CIFAR10-ResNet56 and ILSVRC2012-ResNet50. Table V lists the main hyper-parameters used to train the VGG16 and ResNet56 networks and obtain their baseline performances. Pre-trained weights for ILSVRC2012-ResNet50 are used to compute ACMI values. Table VI list the basic hyper-parameters used to retrain the VGG16, ResNet56 and ResNet50 networks and obtain their final performance.

TABLE V: Training setups used to obtain pre-trained network weights

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

TABLE VI: Base retraining setup used to obtain final performance listed in Table 1 of main paper

	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	0.9

TABLE VII: 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 %

	1	$\|weights\|_2$	$\|weights\|_2^2$	$\exp(-\frac{\|weights\|_2^2}{2})$	$\|act\|_2^2$	$\|weights\|_2 \|act\|_2$	$\exp(-\frac{\|weights\|_2^2 \|act\|_2^2}{2})$
δ	0.9865	0.9925	0.9925	0.988	0.995	0.880	0.919
$\gamma^{(1)}$	00.00	00.00	00.00	00.00	00.00	00.00	00.00
$\gamma^{(2)}$	00.00	00.00	00.00	00.00	00.00	00.00	00.00
$\gamma^{(3)}$	21.02	21.02	21.02	21.02	00.00	41.01	36.03
$\gamma^{(4)}$	51.02	51.02	51.02	51.02	96.02	56.03	61.03
$\gamma^{(5)}$	61.03	51.02	51.02	71.02	51.02	61.03	56.03
$\gamma^{(6)}$	86.03	91.01	91.01	86.03	96.02	81.03	86.03
$\gamma^{(7)}$	91.01	91.01	91.01	91.01	86.03	86.03	96.02
$\gamma^{(8)}$	91.01	91.01	91.01	91.01	91.01	91.01	86.03
$\gamma^{(9)}$	96.02	96.02	96.02	96.02	96.02	96.02	91.01
$\gamma^{(10)}$	91.01	91.01	91.01	91.01	91.01	96.02	96.02
$\gamma^{(11)}$	91.01	91.01	91.01	91.01	91.01	91.01	81.03
$\gamma^{(12)}$	66.01	66.01	66.01	66.01	61.03	61.03	71.02
$\gamma^{(13)}$	91.01	91.01	91.01	91.01	91.01	91.01	86.03
$\gamma^{(14)}$	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Pruned (%)	84.02	84.12	84.17	**84.46**	76.13	82.59	76.99

A. 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 $c \in \{1, 2, \ldots, 99\}$.

B. Evaluation of Estimator

a) 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 [11] 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 $\phi = 1$ as well as $\|weights\|_2$ where weights are scaled 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 samples per class which results in a total of 2000 samples of activations used by the estimators.

b) 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 obtain the final test accuracy are provided in Tables VI, and VII. We maintain $G = 64$ throughout these experiments. The retraining performances are based on the highest Pruned (%) at which the model has a test accuracy that matches or exceeds 93.43% (from MINT).

C. Large Scale Comparison

The basic setup to obtain the final results presented in Table 2 of the main paper is listed under Table VI. 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 [49]. Below, we list the γ values obtained through our set of operating constraints used to define the upper pruning percentage limit for all layers in the DNN.

For VGG16, γ values from convolution layer 1 to the final linear layer are 0, 0, 0.8599, 0.9799, 0.9799, 0.9799, 0.9799, 0.9799, 0.9799, 0.9699, 0.9499, 0.8399, 0.9099, 0.

For ResNet56, γ values from convolution layer 1 to the final linear layer are 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.699, 0.4794, 0.1591, 0.539, 0.3691, 0.9794, 0.089, 0.7392, 0.2695, 0.7294, 0.8896, 0.8398, 0.6699, 0.9699, 0.8698, 0.899, 0.2399, 0.9499, 0.3498, 0.899, 0.7199, 0.8898, 0.9199, 0.9599, 0.9699, 0.9799, 0.9799, 0.9799, 0.

For ResNet50, γ values from convolution layer 1 to the final linear layer are 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 75.99, 65.99, 67.99, 96.99, 52.99, 82.99, 66.99, 78.99, 78.99, 51.99, 16.99, 67.99, 63.99, 76.99, 87.99, 66.00, 87.99, 92.99, 82.99, 73.99, 69.99, 93.99, 52.99, 95.99, 60.99, 78.99, 84.99, 57.00, 84.99, 80.99, 85.99, 34.99, 68.99, 94.00, 80.00, 87.00, 70.99, 79.99, 91.99, 98.00, 0.0, 0.0.

D. Sensitivity-based Pruning

When using sensitivity-based pruning for ResNet56, we observe both an increase and decrease in final γ values used to achieve higher Pruned (%) when compared to the case without sensitivity. In Table VIII we highlight the difference in γ values achieved in each case. It is important to note that while γ values represent the limit up to which layers should be pruned, in our implementation we obtain this point by selecting η value just below the point which triggers the fail-safe. Hence, layers with a skew in the distribution of η values tend to be pruned more.

Acknowledgment

This work has been partially supported (Madan Ravi Ganesh and Jason J. Corso) by a Google Faculty Research Award, and

TABLE VIII: Comparison of γ values in CIFAR10-ResNet56 when sensitive filters are protected

	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

NIST 60NANB17D191 and (Salimeh Yasaei Sekeh) by NSF 1920908; the findings are those of the authors only and do not represent any position of these funding bodies.

References

[1] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2018.

[2] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, A. Patsekin, J. Kindelsberger, L. Ding, S. Seaman, A. Mehler, A. Sipperley, A. Pettinato, B. D. Seppelt, L. Angell, B. Mehler, and B. Reimer, "Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, 2019.

[3] J. Gu, G. Neubig, K. Cho, and V. O. Li, "Learning to translate in real-time with neural machine translation," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, 2017.

[4] Y. Jia, R. J. Weiss, F. Biadsy, W. Macherey, M. Johnson, Z. Chen, and Y. Wu, "Direct speech-to-speech translation with a sequence-to-sequence model," *Proc. Interspeech 2019*, 2019.

[5] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[6] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[7] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2018.

[8] 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.

[9] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *IEEE international conference on computer vision*, 2017.

[10] 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.

[11] M. R. Ganesh, J. J. Corso, and S. Y. Sekeh, "Mint: Deep network compression via mutual information-based neuron trimming," in *IEEE International Conference on Pattern Recognition*, 2020.

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

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

[14] Y. Suhov, I. Stuhl, S. Y. Sekeh, and M. Kelbert, "Basic inequalities for weighted entropies," *Aequationes mathematicae*, vol. 90, no. 4, pp. 817–848, 2016.

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

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

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

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

[26] 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.

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

[28] 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 *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[29] 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.

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

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

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

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

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

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

[36] 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.
[37] S. Yasaei Sekeh and A. O. Hero, “Geometric estimation of multivariate dependency,” *Entropy*, 2019.
[38] 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.
[39] I. Csiszár and P. C. Shields, “Information theory and statistics: A tutorial,” *J. Royal Statist. Soc. Ser. B (Methodology.)*, 2004.
[40] S. Yasaei Sekeh and A. O. Hero, “Geometric estimation of multivariate dependency,” *Entropy (Women in Information Theory)*, 2018.
[41] W. Härdle, *Applied Nonparametric Regression*. Cambridge University Press, 1990.
[42] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.
[43] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR*, 2015.
[44] 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*, 2016.
[45] 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*, 2015.
[46] A. Prabhu, G. Varma, and A. Namboodiri, “Deep expander networks: Efficient deep networks from graph theory,” in *European Conference on Computer Vision*, 2018.
[47] 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.
[48] 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.
[49] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *IEEE conference on computer vision and pattern recognition*, 2016.