Adjoined Networks: A Training Paradigm With Applications to Network Compression

Utkarsh Nath[1], Shrinu Kushagra[2] and Yingzhen Yang[1]

[1]School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ 85281, USA
[2]University of Waterloo, Waterloo, ON N2L 3G1, Canada

Abstract

Compressing deep neural networks while maintaining accuracy is important when we want to deploy large, powerful models in production and/or edge devices. One common technique used to achieve this goal is knowledge distillation. Typically, the output of a static pre-defined teacher (a large base network) is used as soft labels to train and transfer information to a student (or smaller) network. In this paper, we introduce *Adjoined Networks*, or AN, a learning paradigm that trains both the original base network and the smaller compressed network together. In our training approach, the parameters of the smaller network are shared across both the base and the compressed networks. Using our training paradigm, we can simultaneously compress (the student network) and regularize (the teacher network) any architecture. In this paper, we focus on popular CNN-based architectures used for computer vision tasks. We conduct an extensive experimental evaluation of our training paradigm on various large-scale datasets. Using ResNet-50 as the base network, AN achieves 71.8% top-1 accuracy with only 1.8M parameters and 1.6 GFLOPs on the ImageNet data-set. We further propose Differentiable Adjoined Networks (DANs), a training paradigm that augments AN by using neural architecture search to jointly learn both the width and the weights for each layer of the smaller network. DAN achieves ResNet-50 level accuracy on ImageNet with 3.8× fewer parameters and 2.2× fewer FLOPs.

Keywords

Knowledge Distillation, Differentiable Adjoined Networks, Neural Architecture Search

1. Introduction

Deep Neural Networks (DNNs) have achieved state-of-the-art performance on many tasks such as classification, object detection and image segmentation. However, the large number of parameters often required to achieve the performance makes it difficult to deploy them at the edge (like on mobile phones, IoT and embedded devices, etc). Unlike cloud servers, these edge devices are constrained in terms of memory, compute, and energy resources. A large network performs a lot of computations, consumes more energy, and is difficult to transport and update. A large network also has a high prediction time per image. This is a constraint when real-time inference is needed. Thus, compressing neural networks while maintaining accuracy and improving inference time has received significant attention in the last few years. Popular techniques for network compression include pruning and knowledge distillation.

In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen (Eds.), Proceedings of the AAAI 2022 Spring Symposium on Machine Learning and Knowledge Engineering for Hybrid Intelligence (AAAI-MAKE 2022), Stanford University, Palo Alto, California, USA, March 21–23, 2022.

unath@asu.edu (U. Nath); skushagr@uwaterloo.ca (S. Kushagra); yingzhen.yang@asu.edu (Y. Yang)

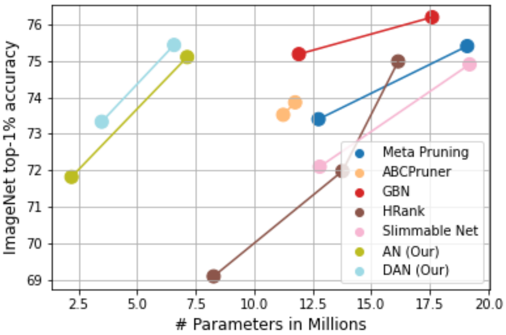

Figure 1: Top-1 accuracy of various structured pruning methods (by compressing the ResNet-50 or ResNet-100 architecture) on the ImageNet dataset plotted against the number of parameters in the model. Our methods, Adjoined Networks (AN), and Differentiable Adjoined Networks (DANs) achieve similar accuracy as compared against current SOTA pruning methods but with fewer (in many cases 2x fewer) parameters.

Pruning methods remove parameters (or weights) of overparameterized DNNs based on some pre-defined criteria. For example, [1] removes weights whose absolute value is smaller than a threshold. While weight pruning methods are successful at reducing the number of parameters of the network, they often work by creating spares tensors that may require special hardware [2] or special software [3] to provide inference time speed-ups. These methods are also known as *unstructured pruning* and has been extensively studied in [1, 4, 5, 6, 7]. To overcome this limitation, channel pruning [8] and filter pruning [9] techniques are used. These *structured pruning* methods work by removing entire convolution channels or sometimes even filters based on some pre-defined criteria and can often provide significant improvement in inference times. In this paper, we show that our algorithm, Adjoined Networks or AN, achieves accuracy similar to the current state-of-the-art structured pruning methods but uses a significantly lower number of parameters and FLOPs (Fig 1).

The AN training paradigm works as follows. A given input image X is processed by two networks, the larger network (or the base network) and the smaller network (or the compressed network). The base network outputs a probability vector p and the compressed network outputs a probability vector q. This setup is similar to the student-teacher training used in Knowledge Distillation [10] where the base network (or the teacher) is used to train the compressed network (or the student). However, there are two very important distinctions. (1) In knowledge distillation, the parameters of the base (or larger or teacher) network are fixed and the output of the base network is used as a "soft label" to train the compressed (or smaller or student) network. In the paradigm of the adjoined network, both the base and the compressed network are trained together. The output of the base network influences the compressed network and vice-versa. (2) The parameters of the compressed network are shared across both the smaller and larger networks (Fig. 2). We train the two networks using a novel time-dependent loss function called *adjoined loss*. An additional benefit of training the two networks together is that the smaller network can have a regularizing effect on the larger network. In our experiments (Section 6), we see that on many datasets and for many architectures, the base network trained in the adjoined fashion has greater prediction accuracy than the standard situation when the base network was trained alone. We also provide theoretical justification for this observation in

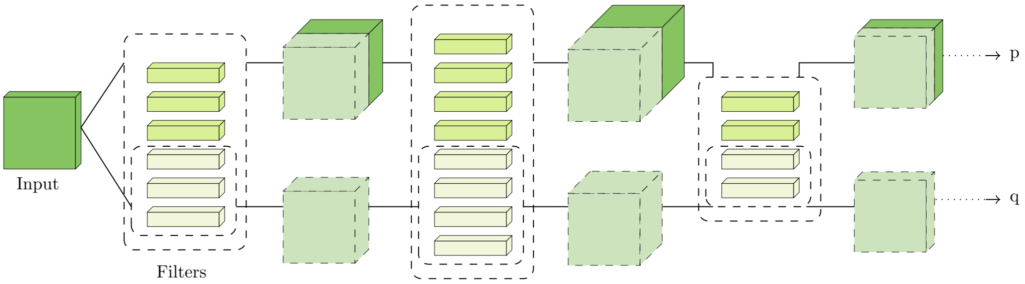

Figure 2: Training paradigm based on adjoined networks. The original and the compressed version of the network are trained together with the parameters of the smaller network shared across both. The network outputs two probability vectors p (original network) and q (smaller network).

the appendix materials. The details of our design, the loss function and how it supports fast inference are discussed in Section 3.

As discussed in the previous paragraph, in the AN training paradigm, all the parameters of the smaller network are shared across both the smaller and larger network. Our compression architecture design involves selecting and tuning a hyper-parameter α, the size (or the number of parameters in each convolution layer) of the smaller network as compared against the larger base network. In our experiments (Section 6) with the AN paradigm, we found that choosing a value of $\alpha = 2$ *or* 4 as a global (same across all the layers of the network) constant typically worked well. To get more boost in compression performance, we propose the framework of Differentiable Adjoined Network (DAN). DAN uses techniques from Neural Architecture Search (NAS) to further optimize and choose the right value of α at each layer of our compressed model. The details of DAN are discussed in Section 5.

Below are the main contributions of this work.

1. We propose a novel training paradigm based on *Adjoined Networks* or AN, that can compress any CNN based neural architecture. This involves adjoined training where the original network and the smaller network are trained together. This has twin benefits of compression and regularization whereby the larger network (or teacher) transfers information and helps compress the smaller network while the smaller network helps regularize the larger teacher network.

2. We further propose *Differentiable Adjoined Networks*, or DAN, that adjointly learns some of the hyper-parameters of the smaller network including the number of filters in each layer of the smaller network.

3. We conducted an exhaustive experimental evaluation of our method and compared it against several state-of-the-art methods on datasets such as ImageNet [11], CIFAR-10 and CIFAR-100 [12]. We consider different architectures such as ResNet-18,-50,-20,-32,-44,-56,-110 and DenseNet-121. On ImageNet, using adjoined training paradigm, we can compress ResNet-50 by $4\times$ with $2\times$ FLOPs reduction while achieving 75.1% accuracy. Moreover, the base network gains 0.7% in accuracy when compared against the same network trained in the standard (non-adjoined) fashion. We further increase the accuracy of the compressed model to 75.7% by augmenting our approach with architecture search (DAN), clearly showing that it is *better* to train the networks *together*. Furthermore, we compare our approach against several state-of-the-art knowledge distillation methods on CIFAR-10 on various architectures

like Resnet-20,-32,-44,-56, and -110. On each of these architectures the student trained using the adjoined method outperforms those trained using other methods (Table 2).

The paper is organized as follows. In Section 2, we discuss some of the other methods that are related to the discussions in the paper. In Section 3, we provide details of the architecture for adjoined networks and the loss function. In Section 4, we show how training both the base and compressed network together provides compression (for the smaller network) as well as regularization (for the larger network). In Section 5, we combine AN with neural architecture search and introduce Differentiable Adjoined Networks (or DANs). In Section 6, we provide the details of our experimental results. In Section A of the appendix, we provide strong theoretical guarantees on the regularization behaviour of adjoined training.

2. Related Work

In this section, we discuss various techniques used to design efficient neural networks in terms of size and FLOPs. We also compare our approach to other similar approaches and ideas in the literature.

Knowledge Distillation is the transfer of knowledge from a cumbersome model to a small model. [10] proposed teacher student model, where soft targets from the teacher are used to train the student model. This forces the student to generalize in the same manner as the teacher. Various knowledge transfer methods have been proposed recently. [13] used intermediate layer's information from teacher model to train thinner and deeper student model. [14] proposes to use instance level correlation congruence instead of just using instance congruence between the teacher and student. [15] tried to maximize the mutual information between teacher and student models using variational information maximization. [16] aims at transferring structural knowledge from teacher to student. [17] argues that directly transferring a teacher's knowledge to a student is difficult due to inherent differences in structure, layers, channels, etc., therefore, they paraphrase the output of the teacher in an unsupervised manner making it easier for the student to understand. Most of these methods use a trained teacher model to train a student model. In contrast in this work, we train both the teacher and the student together from scratch. In recent work, [18], rather than using a teacher to train a student, they let a cohort of students train together using a distillation loss function. In this paper, we consider a teacher and a student together rather than using a pre-trained teacher. We also use a novel time-dependent loss function. Moreover, we also provide theoretical guarantees on the efficacy of our approach. We have compared our AN with various knowledge distillation methods in the experiments section.

Pruning techniques aim to achieve network compression by removing parameters or weights from a network while still maintaining accuracy. These techniques can be broadly classified into two categories; unstructured and structured. Unstructured pruning methods are generic and do not take network architecture (channel, filters) into account. These methods induce sparsity based on some pre-defined criteria and often achieve a state-of-the-art reduction in the number of parameters. However, one drawback of these methods is that they are often unable to provide inference time speed-ups on commodity hardware due to their unstructured nature. Unstructured sparsity has been extensively studied in [1, 4, 5, 6, 7]. Structured pruning aims to address the issue of inference time speed-up by taking network architecture into account. As an example, for CNN architectures, these methods try to remove entire channels or filters, or blocks. This ensures that the reduction in the number of parameters also translates to a reduction in inference time on commodity hardware. For example, ABCPruner [19] decides the convolution filters to be removed in each layer using an artificial bee colony algorithm. [20] prunes filters with low-rank feature maps. [21] uses Taylor expansion to estimate the change in the loss function by removing a particular filter, and finally removes the filters with max change. The AN compression technique proposed in this paper can also be thought of as a structured pruning method where the architecture choice at the start of training fixes the convolution filters to be pruned and the amount of pruning at each layer. Another related work is of Slimmable Networks [22]. Here different networks (or architectures) are *switched on* one at a time and trained using the standard cross-entropy loss function. By contrast, in this work, both the networks are trained together at the same time using a novel loss function (adjoined-loss). We have compared our work with Slimmable Networks in Table 1.

Neural Architecture Search (NAS) is a technique that automatically designs neural architecture without human intervention. The best architecture could be found by training all architectures in the given search space from scratch to convergence but this is computationally impractical. Earlier studies in NAS were based on RL [23, 24] and EA [25], however, they required lots of computation resources. Most recent studies [26, 27, 28] encode architectures as a weight sharing a super-net and optimize the weights using gradient descent. A recent study Meta Pruning [29] searches over the number of channels in each layer. It generates weights for all candidates and then selects the architecture with the highest validation accuracy. A lot of these techniques focus on designing compact architecture from scratch. In this paper, we use architecture search to help guide the choice of architecture for compression, that is, the fraction of filters which should be removed from each layer.

Small architectures - Another research direction that is orthogonal to ours is to design smaller architectures that can be deployed on edge devices, such as SqueezeNet [30], MobileNet [31] and EfficientNet [32]. In this paper, our focus is to compress existing architectures while ensuring inference time speedups as well as maintaining prediction accuracy.

3. Adjoined networks

In our training paradigm, the original (larger) and the smaller network are trained together. The motivation for this kind of training comes from the principle that *good teachers are lifelong learners*. Hence, the larger network which serves as a teacher for the smaller network should not be frozen (as in standard teacher-student architecture designs [10]). Rather both should learn together in a "combined learning environment", that is, adjoined networks. By learning together both the networks can be better together.

We are now ready to describe our approach and discuss the design of adjoined networks. Before that, let's take a re-look at the standard convolution operator. Let $\mathbf{x} \in \mathbf{R}^{h \times w \times c_{in}}$ be the input to a convolution layer with weights $\mathbf{W} \in \mathbf{R}^{c_{out} \times k \times k \times c_{in}}$ where c_{in}, c_{out} denotes the number of input and output channels, k the kernel size and h, w the height and width of the

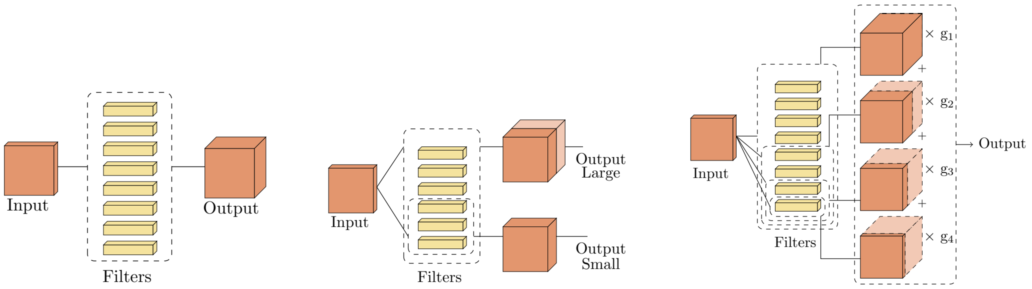

Figure 3: (Left) Standard layer of CNN. (Center) Layer in Adjoined Network. (Right) Layer in DAN.

image. Then, the output of the convolution $\mathbf{z}$ is given by

$$\mathbf{z} = conv(\mathbf{x}, \mathbf{W})$$

In the adjoined paradigm, a convolution layer with weight matrix $\mathbf{W}$ and a binary mask matrix $M \in \{0,1\}^{c_{out} \times k \times k \times c_{in}}$ receives two inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ of size $h \times w \times c_{in}$ and outputs two vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ as defined below.

$$\mathbf{z}_1 = conv(\mathbf{x}_1, \mathbf{W}) \qquad\qquad \mathbf{z}_2 = conv(\mathbf{x}_2, \mathbf{W} * M) \tag{1}$$

Here M is of the same shape as $\mathbf{W}$ and $*$ represents an element-wise multiplication. Note that the parameters of the matrix M are fixed before training and not learned. The vector $\mathbf{x}_1$ represents an input to the original (bigger) network while the vector $\mathbf{x}_2$ is the input to the smaller, compressed network. For the first convolution layer of the network $\mathbf{x}_1 = \mathbf{x}_2$ but the two vectors are not necessarily equal for the deeper convolution layers (Fig. 2). The mask matrix M serves to zero-out some of the parameters of the convolution layer thereby enabling network compression. In this paper, we consider matrices M of the following form.

$$M := a_\alpha = \text{ matrix such that the first } \alpha \text{ filters } \text{ are all 1 and the rest 0} \tag{2}$$

In Section 6, we run experiments with $M := a_\alpha$ for $\alpha \in \{2, 4, 8, 16\}$. Putting this all together, we see that any CNN-based architecture can be converted and trained in an adjoined fashion by replacing the standard convolution operation by the adjoined convolution operation (Eqn. 1). Since the first layer receives a single input (Fig. 2), two copies are created which are passed to the adjoined network. The network finally gives two outputs $\mathbf{p}$ corresponding to the original (bigger or unmasked) network and $\mathbf{q}$ corresponding to the smaller (compressed) network, where each convolution operation is done using a subset of the parameters described by the mask matrix M (or M_α). We train the network using a novel time-dependent loss function which forces $\mathbf{p}$ and $\mathbf{q}$ to be close to one another (Defn. 1).

4. Regularization and Compression

In the previous section, we looked at the design on adjoined networks. For one input $(\mathbf{X}, \mathbf{y}) \in \mathbf{R}^{h \times w \times c_{in}} \times [0,1]^{n_c}$, the network outputs two vectors $\mathbf{p}$ and $\mathbf{q} \in [0,1]^{n_c}$ where n_c denotes the number of classes and c_{in} denotes the number of input channels (equals 3 for RGB images).

Definition 1 (Adjoined loss)**.** *Let y be the ground-truth one-hot encoded vector and p and q be output probabilities by the adjoined network. Then*

$$\mathcal{L}(y,p,q) \;=\; -y\log p \;+\; \lambda(t)\,KL(p,q) \tag{3}$$

where $KL(p,q) = \sum_i p_i \log \frac{p_i}{q_i}$ is the measure of difference between two probability measures [33]. The regularization term $\lambda : [0,1] \to \mathbf{R}$ is a function which changes with the number of epochs during training. Here $t = \frac{\textit{current epoch}}{\textit{Total number of epochs}}$ equals zero at the start of training and equals one at the end.

In our definition of the loss function, the first term is the standard cross-entropy loss function which trains the bigger network. To train the smaller network, we use the predictions from the bigger network as a soft ground-truth signal. We use KL-divergence to measure how far the output of the smaller network is from the bigger network. This also has a regularizing effect as it forces the network to learn from a smaller set of parameters. Note that, in our implementations, we use $KL(p,q) = \sum p_i \log \frac{p_i+\epsilon}{q_i+\epsilon}$ to avoid rounding and division by zero errors where $\epsilon = 10^{-6}$.

At the start of training, p is not a reliable indicator of the ground-truth labels. To compensate for this, the regularization term λ changes with time. In our experiments, we used $\lambda(t) = \min\{4t^2, 1\}$. Thus, the contribution of the second term in the loss is zero at the beginning and steadily grows to one at 50% training.

5. DAN: Differentiable Adjoined Networks

In Sections 3 and 4, we described the framework of adjoined networks and the corresponding loss function. An important parameter in the design of these networks is the choice of parameter α. Currently, the choice of α is global, that is, we choose the same value of α for all the layers of our network. However, choosing α independently for each layer would add more flexibility and possibly improve performance of the current framework. To solve this problem, we propose the framework of Differentiable Adjoined Networks (or DANs).

Consider the following example of a convolution network with 1 layer with the following choices for $\alpha \in A = \{1,2,4\}$ that outputs a vector p_α. Finding the optimal network structure is equivalent to solving $\arg\max_{\alpha\in A} L(p_\alpha)$ where L is some loss function. For a one layer network, we can solve this problem by computing $L(p_\alpha)$ for all the different values and then computing the max. However, this becomes intractable as the number of layers increase; for a 50-layer network, the search space has size 3^{50}.

Definition 2 (Gumbel-softmax ([34]))**.** *Given vector $v = [v_1, \ldots, v_n]$ and a constant τ. The gumbel-softmax function is defined as $g(v) = [g_1, \ldots, g_n]$ where*

$$g_i = \frac{\exp[(v_i+\epsilon_i)/\tau]}{\sum_i \exp[(v_i+\epsilon_i)/\tau]} \tag{4}$$

and $\epsilon_i \sim N(0,1)$ is uniform random noise (also referred to as gumbel noise). Note that as $\tau \to 0$, gumbel-softmax tends to the $\arg\max$ *function.*

Gumbel-softmax is a “re-parametrization trick" that can be viewed as a differentiable approximation to the $\arg\max$ function. Returning back to the one-layer example, the optimization objective now becomes $\sum_{\alpha\in A} g_\alpha L(p_\alpha)$ where g_α represents the gumbel weights corresponding to the particular α. This objective is now differentiable and can be solved using standard techniques like back-propagation.

With this insight, we propose the DAN architecture (Fig 3) where the standard convolution operation is replaced by a DAN convolution operation. As before, let $\mathbf{x} \in R^{h\times w\times c_{in}}$ be the input to the DAN convolution layer with weights $W \in R^{c_{out}\times k\times k\times c_{in}}$ where c_{in}, c_{out} denotes the number of input and output channels, k the kernel size and h, w the height and width of the image. Let $A = \{\alpha_1, \ldots, \alpha_m\}$ be the range of values of α for the layer. Then, the output $\mathbf{z}$ of the DAN convolution layer is given by

$$z(\eta) = \sum_{i=1}^{m} g(\eta)_i \;\; z_i \tag{5}$$

where $\eta = [\eta_1, \ldots, \eta_m]$ denotes the mixing weights corresponding to the different α's, g is the gumbel-softmax function and $z_i = conv(\mathbf{x}, W * M_i)$ where M_i is the mask matrix corresponding to α_i (as in Eqn. 2). Thus, each layer of the DAN convolution layer combines its outputs according to the gumbel weights. Choosing the hyper-parameter α now corresponds to learning the values of the parameter η for each layer of our DAN conv network. Note that as before, our network outputs two probability vectors $\mathbf{p}$ and $\mathbf{q}$. But these vectors now also depend upon the weights vector η at each layer. We are now ready to define our main loss function.

Definition 3 (Differentiable Adjoined loss). *Let the search space be $A = \{\alpha_1, \ldots, \alpha_m\}$Let y be the ground-truth one-hot encoded vector and p and q be output probabilities of the adjoined network. Then*

$$\mathcal{L}(y, p, q) = -y \log p + \lambda(t)(\, KL(p, q) + \gamma n_f(H)) \tag{6}$$

where $KL(p, q), \lambda(t)$ are the same as used in Eqn. 1. $H = [\eta_1, \ldots, \eta_l]$ where η_i is the mixing weight vector for the i^{th} convolution layer. n_f represents the gumbel weighted FLOPs or floating point operations for the given network. That is,

$$n_f(H) = \sum_{\eta_i\in H} \sum_{j=1}^{m} g(\eta_i)_j \;\; FLOPs(i, \alpha_j)$$

where $flops(i, \alpha_j)$ measures the number of floating point operations at the i^{th} convolution layer corresponding to the hyper-parameter α_j. Also, note that γ in Eqn. 6 is a normalization constant.

Differentiable Adjoined Loss is similar to Adjoined Loss defined in Eqn. 3. However, the key difference is the n_f term. First note that, larger architectures tend to have higher accuracies. Hence, DAN learning tends to prefer a network with low alpha (large network) against that with high alpha (small network). Thus, the n_f term acts as a regularization penalty against DAN preferring large architectures. Another point to note is that for a large network say Resnet-50, the number of flops corresponding to any setting of the mixing weights can be very large. Gamma normalizes it so that all the terms in the loss function are in the same scale.

6. Experiments

We are now ready to describe our experiments in detail. We run experiments on three different datasets. (1) *ImageNet* - an image classification dataset [11] with 1000 classes and about $1.2M$ images . (2) *CIFAR-10* - a collection of $60k$ images in 10 classes. (3) *CIFAR-100* - same as CIFAR-10 but with 100 classes [12]. For each of these datasets, we use standard data augmentation techniques such as random-resize cropping, random flipping.

We train different architectures such as ResNet-100, ResNet-50, ResNet-18, ResNet-110, ResNet-56, DenseNet-121 on all of the above datasets. On each dataset, we first train these architectures in the standard non-adjoined fashion using the cross-entropy loss function. We will refer to it by the name *Standard*. Next, we train the adjoined network, obtained by replacing the standard convolution operation with the adjoined convolution operation, using the adjoined loss function. In the second step, we obtain two different networks. In this section, we refer to them by *AN-X-Full* a_α and the *AN-X-Small* a_α networks where X represents the number of layers and a_α denotes the mask matrix as defined in 2. For example, *AN-50-Full* a_2, *AN-50-Small* a_2 represents larger and smaller networks obtained on adjoinedly training ResNet-50 with $\alpha = 2$. *AN-121-Full* a_4, *AN-121-Small* a_2 represents models obtained on adjoinedly training DenseNet-121 with $\alpha = 4$. We compare the performance of the AN-X-Full a_α and AN-X-Small a_α networks against the standard network. One point to note is that we do not replace the convolutions in the stem layers but only those in the residual blocks. Since most of the weights are in the later layers, this leads to significant space and time savings while retaining competitive accuracy. DAN describes the performace of adjoined network on architectures found by Differentiable Adjoined Network. DAN-50 has the same number of blocks as ResNet-50 whereas DAN-100 has twice the number of blocks of ResNet-50.

We ran our experiments on GPU enabled machine using Pytorch. We have also open-sourced our implementation [1]. Hyperparameters for the experiments are mentioned on our github page.

In Section 6.1, we compare our compression results against other structured pruning methods. In Section 6.2, we compare AN with various types of knowledge distillation methods. In Section 6.3, we describe our results for compression and performance of architectures found by DAN. In Section 6.4, we show the strong regularizing effect of AN training.

6.1. Comparison against other Structured Pruning works

Table 1 and Figure 1 compare the performance of various structured compression schemes on the ImageNet dataset for the ResNet-50 architecture. Note, these methods provide inference speed-up without special hardware or software. We see that the adjoined training regime can achieve compression that is significantly better than other methods considered in the literature. In Figure 1, models trained using our paradigm are explicitly on the left side of the graph while other methods are clustered on to the right side. Other methods obtain compression ratios in the range $2 - 3\times$, compared to which our method achieves up to $12\times$ compression in size. Similarly, GFLOPS for our method is amongst the highest as compared to the other state-of-the-art works, while suffering a small accuracy drop as compared against the base ResNet-50 model. Figure 4 compares the performance of AN against various pruning methods on CIFAR-10 dataset for

[1] The code can be found at https://github.com/utkarshnath/Adjoint-Network.git

Model Compression Results			
Method	# Params	GFLOPs	Accuracy
ABCPruner-0.8 [19]	11.75	1.89	73.86
ABCPruner-0.7 [19]	11.24	1.8	73.52
GBN-50 [21]	11.91	1.9	75.18
GBN-60 [21]	17.6	2.43	76.19
DCP [35]	12.3	1.82	74.95
HRank [20]	16.15	2.3	74.98
HRank [20]	13.77	1.55	71.98
HRank [20]	8.27	0.98	69.1
MetaPruning [29]	19.1	2	75.4
MetaPruning [29]	12.7	1	73.4
Slimmable Net [22]	19.2	2.3	74.9
Slimmable Net [22]	12.8	1.1	72.1
AN-50-Small a_4 (our)	2.2	1.6	71.82
DAN-50 (our)	3.49	1.7	73.33
DAN-100 (our)	6.58	2.15	**75.43**
AN-50-Small a_2 (our)	7.14	2.2	75.1

Table 1

The table shows the performance of various structured pruning methods when trained on the ImageNet dataset. a_α in AN-50-Small denotes the mask matrix as defined in Eqn. 2.

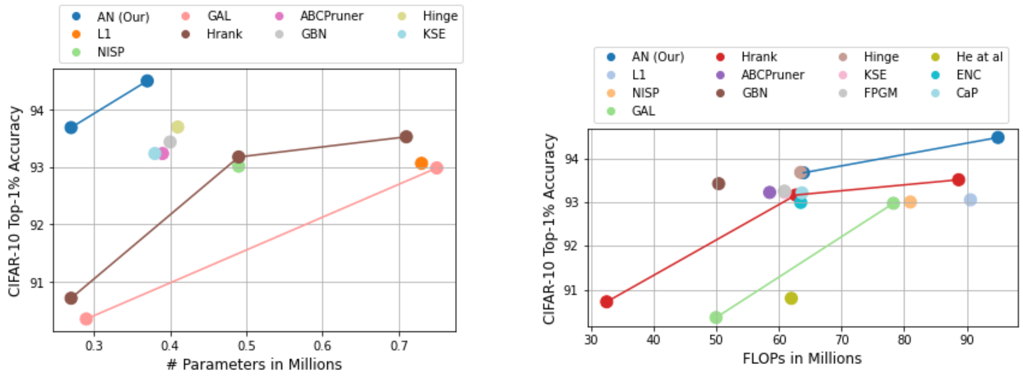

Figure 4: Top-1% accuracy of various pruning methods (by compressing ResNet-56 architecture) on CIFAR-10 dataset plotted against number of parameters (Left) and FLOPs (Right). Pruning methods - Gal([36]), Hrank([20]) ,He at al([36]), ENC([37]),NISP([38]), L1([39]), ABC-Prunner([19]), CaP([40]), KSE([41]), FPGM([42]), GBN([21]) and Hinge([43])

ResNet-56 architecture. Models trained using AN paradigm achieve highest accuracy with fewest number of parameters on CIFAR-10. AN exceeds the next best model (Hinge [43]) by 0.8% while being smaller than 9 of the 11 models. The smallest AN model achieves accuracy similar to hinge but with 35% fewer parameter. We see similar results for FLOPs.

6.2. Comparison against other Knowledge Distillation Works

In this section, we discuss the effectiveness of weight sharing and training two networks together. We compare AN against the various state-of-the-art variants of knowledge distillation. In Table

Knowledge Distillation Variants					
Method	AN-110-Small	AN-56-Small	AN-44-Small	AN-32-Small	AN-20-Small
RKD [16]	94.11	93.72	93.41	92.5	92.15
VID [15]	93.6	93.23	92.86	92.41	91.6
FT [17]	93.76	93.33	93.17	92.49	91.2
CC [14]	93.6	93.22	92.95	92.46	91.5
DML [18]	93.5	93.3	93.2	92.59	91.35
KR [44]	94.08	93.85	93.51	93.45	92.3
KDCR [45]	94.26	93.7	93.4	92.9	91.85
AN (Our)	**95**	**94.49**	**94.01**	**93.45**	**92.45**

Table 2
AN compared to various state-of-the-art KD methods on CIFAR-10. All AN-X-Small models refers to models with $\alpha = 2$.

2, we compare accuracy (Top-1%) of AN-X-Small against the same architecture trained using various KD variants on CIFAR-10 dataset. The corresponding pre-trained ResNet architecture was used as the teacher model for KD variants. Teacher models were trained on CIFAR-10 using standard training paradigm. We see that all models trained using AN paradigm significantly outperforms the models trained using various teacher-student paradigm showing the effectiveness of training a subset of weights together.

6.3. Ablation study: Compression

Compression using AN paradigm			
Network	#Params	GFLOPs	Accuracy
CIFAR-10			
ResNet-20	0.27	40	92.5
AN-20-Small a_2	0.07	21	92.45
ResNet-32	0.46	69	93.1
AN-32-Small a_2	0.13	35	**93.45**
ResNet-44	0.65	97	93.5
AN-44-Small a_2	0.19	49	**94.01**
ResNet-56	0.84	127	93.9
AN-56-Small a_2	0.24	63	**94.49**
ResNet-110	1.72	253	94.3
AN-110-Small a_2	0.49	127	**95.0**
ImageNet			
ResNet-50	25.5	4774	76.1
AN-50-Small a_2	7.14	2202	75.1
AN-50-Small a_4	2.2	1619	71.84
DAN-50	3.49	1745	73.33
ResNet-100	46.99	8473	77.3
AN-100-Small a_4	3.86	2681	74.51
DAN-100	6.58	2153	**75.43**

Table 3
a_α denote the masking matrix (defined in Eqn. 2).

In this section, we evaluate the performance of models compressed by Adjoined training paradigm. Table 3 compares the performance (top 1% accuracy) of the models compressed using AN against the performance of standard network. For AN, we use the a_α as the masking matrix (defined in Eqn. 2). The mask is such that the last $(1 - \frac{1}{\alpha})$ filters are zero. Hence, these can be pruned away to support fast inference. For CIFAR-10, 4 out of 5 models compressed using AN paradigm exceed it's base architecture by 0.5%-0.8%. These models achieves 3.5-4× reduction in parameters and 2× reduction in FLOPs.

We also observe that ResNet-50 is a bigger network and can be compressed more. Also, different datasets can be compressed by different amounts. For example, on CIFAR-100 dataset, the network can be compressed by factors $\sim 35\times$ while for other datasets it ranges from $2\times$ to $12\times$. DAN is able to search compressed architecture with minimum loss in accuracy as compared to base architecture. For ImageNet, DAN architectures were searched on Imagewoof (a proxy dataset with 10 different dog breeds from ImageNet [46, 47]). γ as defined in Defn. 3 is e^{-13}, e^{-19} for DAN-50 and DAN-100 respectively. During architecture search, temperature τ in gumbel softmax was initialized to 15 and exponentially annealed by $e^{-0.045}$ every epoch.

6.4. Ablation study: Regularization

AN-Full vs Standard			
Network	α	AN-Full	Standard
CIFAR-10			
ResNet-20	a_2	**93.51**	92.5
ResNet-32	a_2	**94.39**	93.1
ResNet-44	a_2	**94.64**	93.5
ResNet-56	a_2	**95.01**	93.9
ResNet-110	a_2	**95.4**	94.3
CIFAR-100			
ResNet-50	a_8	77.36	76.8
ResNet-18	a_2	74.8	74.3
DenseNet-121	a_4	**80.8**	79.0
ImageNet			
ResNet-50	a_2	**76.87**	76.1
ResNet-50	a_4	75.84	76.1

Table 4
a_α denote the masking matrix (defined in Eqn. 2).

In this section, we study the regularization effect of Adjoined training paradigm on AN-Full network. Table 4 compares the performance of the base network trained in adjoined fashion (AN-Full) to the same network trained in Standard fashion. We see a consistent trend that the network trained adjoinedly outperforms the same network trained in the standard way. We see maximum gains on CIFAR-100, exceeding accuracy by as much as 1.8%. Even on ImageNet, we see a gain of about 0.77%.

7. Conclusion

In this work, we introduced the paradigm of Adjoined Network training where both the larger teacher (or base) network and the smaller student network are trained together. We showed how this approach to training neural networks can allow us to reduce the number of parameters of large networks like ResNet-50 by $12\times$, (even going up to $35\times$ on some datasets) without significant loss in classification accuracy with 2-$3\times$ reduction in the number of FLOPs. We showed (both theoretically and experimentally) that adjoining a large and a small network together has a regularizing effect on the larger network. We also introduced DAN, a search strategy that automatically selects the best architecture for the smaller student network. Augmenting adjoined training with DAN, the smaller network achieves accuracy that is close to that of the base teacher network.

References

[1] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv preprint arXiv:1510.00149 (2015).

[2] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, W. J. Dally, Eie: Efficient inference engine on compressed deep neural network, 2016. arXiv:1602.01528.

[3] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, P. Dubey, Faster cnns with direct sparse convolutions and guided pruning, 2017. arXiv:1608.01409.

[4] M. Zhu, S. Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017. arXiv:1710.01878.

[5] T. Gale, E. Elsen, S. Hooker, The state of sparsity in deep neural networks, 2019. arXiv:1902.09574.

[6] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, A. Farhadi, Soft threshold weight reparameterization for learnable sparsity, 2020. arXiv:2002.03231.

[7] U. Evci, T. Gale, J. Menick, P. S. Castro, E. Elsen, Rigging the lottery: Making all tickets winners, 2021. arXiv:1911.11134.

[8] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744.

[9] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, arXiv preprint arXiv:1608.08710 (2016).

[10] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015).

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV) 115 (2015) 211–252. doi:10.1007/s11263-015-0816-y.

[12] A. Krizhevsky, V. Nair, G. Hinton, Cifar-10 and cifar-100 datasets, URl: https://www. cs. toronto. edu/kriz/cifar. html 6 (2009).

[13] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, 2015. arXiv:1412.6550.

[14] B. Peng, X. Jin, J. Liu, S. Zhou, Y. Wu, Y. Liu, D. Li, Z. Zhang, Correlation congruence for knowledge distillation, 2019. arXiv:1904.01802.

[15] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, Z. Dai, Variational information distillation for knowledge transfer, 2019. arXiv:1904.05835.

[16] W. Park, D. Kim, Y. Lu, M. Cho, Relational knowledge distillation, 2019. arXiv:1904.05068.

[17] J. Kim, S. Park, N. Kwak, Paraphrasing complex network: Network compression

via factor transfer, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 31, Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/6d9cb7de5e8ac30bd5e8734bc96a35c1-Paper.pdf.

[18] Y. Zhang, T. Xiang, T. M. Hospedales, H. Lu, Deep mutual learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4320–4328.

[19] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel pruning via automatic structure search, arXiv preprint arXiv:2001.08565 (2020).

[20] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, Hrank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1529–1538.

[21] Z. You, K. Yan, J. Ye, M. Ma, P. Wang, Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks, arXiv preprint arXiv:1909.08174 (2019).

[22] J. Yu, L. Yang, N. Xu, J. Yang, T. Huang, Slimmable neural networks, arXiv preprint arXiv:1812.08928 (2018).

[23] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, 2017. arXiv:1611.01578.

[24] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le, Mnasnet: Platform-aware neural architecture search for mobile, 2019. arXiv:1807.11626.

[25] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, A. Kurakin, Large-scale evolution of image classifiers, 2017. arXiv:1703.01041.

[26] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, 2019. arXiv:1806.09055.

[27] H. Cai, L. Zhu, S. Han, Proxylessnas: Direct neural architecture search on target task and hardware, 2019. arXiv:1812.00332.

[28] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, 2019. arXiv:1812.03443.

[29] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, J. Sun, Metapruning: Meta learning for automatic neural network channel pruning, 2019. arXiv:1903.10258.

[30] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size, arXiv preprint arXiv:1602.07360 (2016).

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, 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.

[32] M. Tan, Q. V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, arXiv preprint arXiv:1905.11946 (2019).

[33] S. Kullback, R. A. Leibler, On information and sufficiency, The annals of mathematical statistics 22 (1951) 79–86.

[34] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, P. Vajda, J. E. Gonzalez, Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions, in: CVPR, IEEE, 2020, pp. 12962–12971.

[35] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, J. Zhu, Discrimination-aware

channel pruning for deep neural networks, arXiv preprint arXiv:1810.11809 (2018).

[36] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, Towards optimal structured cnn pruning via generative adversarial learning, 2019. arXiv:1903.09291.

[37] H. Kim, M. U. K. Khan, C.-M. Kyung, Efficient neural network compression, 2019. arXiv:1811.12781.

[38] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, L. S. Davis, Nisp: Pruning networks using neuron importance score propagation, 2018. arXiv:1711.05908.

[39] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, 2017. arXiv:1608.08710.

[40] B. Minnehan, A. Savakis, Cascaded projection: End-to-end network compression and acceleration, 2019. arXiv:1903.04988.

[41] Y. Li, S. Lin, B. Zhang, J. Liu, D. Doermann, Y. Wu, F. Huang, R. Ji, Exploiting kernel sparsity and entropy for interpretable cnn compression, 2019. arXiv:1812.04368.

[42] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, 2019. arXiv:1811.00250.

[43] Y. Li, S. Gu, C. Mayer, L. V. Gool, R. Timofte, Group sparsity: The hinge between filter pruning and decomposition for network compression, 2020. arXiv:2003.08935.

[44] H. Z. Pengguang Chen, Shu Liu, J. Jia, Distilling knowledge via knowledge review, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[45] Q. Guo, X. Wang, Y. Wu, Z. Yu, D. Liang, X. Hu, P. Luo, Online knowledge distillation via collaborative learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[46] J. Howard, Imagenette, URL: Github repository with links todataset. https://github.com/fastai/imagenette (2019).

[47] S. Shleifer, E. Prokop, Using small proxy datasets to accelerate hyperparameter search, arXiv preprint arXiv:1906.04887 (2019).

A. Regularization theory

Theorem A.1. *Given a deep neural network $\mathcal{A}$ which consists of only convolution and linear layers. Let the network use one of $f(x) = \min\{x, 0\}$ (relu) or $f(x) = x$ (linear) as the activation function. Let the network be trained using the adjoined loss function as defined in Eqn. 3. Let $\mathbf{X}$ be the set of parameters of the network $\mathcal{A}$ which is shared across both the smaller and bigger networks. Let $\mathbf{Y}$ be the set of parameters of the bigger network not shared with the smaller network. Let $\mathbf{p}$ be the output of the larger network and let $\mathbf{q}$ be the output of the smaller network where $\mathbf{p}_i, \mathbf{q}_i$ represents their $\mathbf{i}^{th}$ component. Then, the adjoined loss function induces a data-dependent regularizer with the following properties.*

- *For all $x \in X$, the induced L_2 penalty is given by $\sum_i \mathbf{p}_i \big(\log' \mathbf{p}_i - \log' \mathbf{q}_i\big)^2$*
- *For all $y \in Y$, the induced L_2 penalty is given by $\sum_i \mathbf{p}_i \big(\log' \mathbf{p}_i\big)^2$*

Proof. We are interested in analyzing the regularizing behavior of the following loss function. $-y \log p + KL(p, q)$ y is the ground truth label, p is the output probability vector of the bigger network and q is the output probability vector of the smaller network. Recall that the parameters of smaller network are shared across both. We will look at the second order taylor expansion for the kl-divergence term. This will give us insights into regularization behavior of the loss function.

Let x be a parameter which is common across both the networks and y be a parameter in the bigger network but not the smaller one.

$$D(x) = \sum_i p_i(x)\big(\log p_i(x) - \log q_i(x)\big) \text{ and } D(y) = \sum_i p_i(y)\big(\log p_i(y) - \log q_i\big)$$

For the parameter y, q_i is a constant. Now, computing the first order derivative, we get that

$$D'(x) = \sum_i p_i'(x)\big(\log p_i(x) - \log q_i(x)\big) + p_i'(x) - \frac{q_i'(x)p_i(x)}{q_i(x)}$$

$$D'(y) = \sum_i p_i'(y)\big(\log p_i(y) - \log q_i\big) + p_i'(y)$$

Now, computing the second derivative for both the types of parameters, we get that

$$D''(x) = \sum_i p_i''(x)\big(\log p_i(x) - \log q_i(x)\big) + p_i'(x)\left(\frac{p_i'(x)}{p_i(x)} - \frac{q_i'(x)}{q_i(x)}\right) + p_i''(x)$$
$$- \frac{q_i(x)q_i'(x)p_i'(x) + q_i(x)q_i''(x)p_i(x) - q_i'(x)q_i'(x)p_i(x)}{q_i^2(x)}$$

$$D''(y) = \sum_i p_i''(y)\big(\log p_i(y) - \log q_i\big) + \frac{p_i'(y)p_i'(y)}{p_i(y)} + p_i''(y)$$

$$D''(x) = \sum_i \frac{p_i'(x)p_i'(x)}{p_i(x)} - \frac{2p_i'(x)q_i'(x)}{q_i(x)} + \frac{q_i'(x)q_i'(x)p_i(x)}{q_i^2(x)}$$

$$= \sum_i p_i(x)\left(\frac{p_i'(x)}{p_i(x)} - \frac{q_i'(x)}{q_i(x)}\right)^2 = \sum_i p_i(\log' p_i - \log' q_i)^2 \tag{7}$$

Similarly, for the parameters only in the bigger network, we get that

$$D''(y) = \sum_i \frac{p_i'(y)p_i'(y)}{p_i(y)} = \sum_i p_i(\log' p_i)^2 \tag{8}$$

Note that y represents the over-parameterized weights of the model. The equations above show that the regularization imposed by the KL-divergence term on these parameters is such that if these parameters change a lot (on the log scale) then the penalty imposed on such parameters is more. Thus, the kl-divergence term encourages such parameters not to change by a lot. □