Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Highlights

Computer Vision and Image Understanding xxx (xxxx) xxx

Grow-push-prune: Aligning deep discriminants for effective structural network compression

Qing Tian[*], Tal Arbel, James J. Clark

- Proactive pruning can be more effective than after-the-fact pruning.
- Maximizing, unraveling, and pushing discriminants into neurons prepare for pruning.
- Growing the pruning base can boost accuracy and offer more sub-architecture choices.
- Inception architectures can beat residual ones at similar complexities on ImageNet.

YCVIU: 103682

ARTICLE IN PRESS

Computer Vision and Image Understanding xxx (xxxx) xxx

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Grow-push-prune: Aligning deep discriminants for effective structural network compression

Qing Tian [a,b,*], Tal Arbel [b], James J. Clark [b]

[a] Department of Computer Science, Bowling Green State University, Hayes Hall, Bowling Green, OH 43403, USA
[b] Centre for Intelligent Machines, McGill University, 3480 University Street, Montreal, QC H3A 0E9, Canada

ARTICLE INFO

Communicated by Nikos Paragios

MSC:
41A05
41A10
65D05
65D17

Keywords:
Deep neural network pruning
Deep discriminant analysis
Deep representation learning

ABSTRACT

Most of today's popular deep architectures are hand-engineered to be generalists. However, this design procedure usually leads to massive redundant, useless, or even harmful features for specific tasks. Unnecessarily high complexities render deep nets impractical for many real-world applications, especially those without powerful GPU support. In this paper, we attempt to derive task-dependent compact models from a deep discriminant analysis perspective. We propose an iterative and proactive approach for classification tasks which alternates between (1) a pushing step, with an objective to simultaneously maximize class separation, penalize co-variances, and push deep discriminants into alignment with a compact set of neurons, and (2) a pruning step, which discards less useful or even interfering neurons. Deconvolution is adopted to reverse 'unimportant' filters' effects and recover useful contributing sources. A simple network growing strategy based on the basic Inception module is proposed for challenging tasks requiring larger capacity than what the base net can offer. Experiments on the MNIST, CIFAR10, and ImageNet datasets demonstrate our approach's efficacy. On ImageNet, by pushing and pruning our grown Inception-88 model, we achieve more accurate models than Inception nets generated during growing, residual nets, and popular compact nets at similar sizes. We also show that our grown Inception nets (without hard-coded dimension alignment) clearly outperform residual nets of similar complexities.

1. Introduction

Compact yet capable neural network architectures are desirable for many real-world problems, such as HCI, autonomous driving perception, and video analytics. Many network pruning approaches proposed so far pay little attention to whether the complexity decrease follows a task-optimal direction, such as those based on weight magnitudes (Han et al., 2015b). Moreover, most of them are ex post facto, i.e., useful and useless components are already mixed and it is too late to trim one without influencing the other. Aside from pruning, a compact structure design practice is to utilize a random number of 1×1 filters, usually at module ends to reduce feature map dimension (He et al., 2015; Szegedy et al., 2015; Iandola et al., 2016; Howard et al., 2017). Nevertheless, an ad-hoc filter number may lead to irrecoverable information loss or redundancy/overfitting/interference.

In this paper, we propose to derive task-suitable compact networks through deep discriminant analysis in the feature space. Instead of counting on an optimally pre-trained model, the proposed approach follows a two-step procedure in iterations. (1) through learning, it proactively unravels useful twisted threads of deep variation and pushes them into alignment with a compact substructure that can be easily decoupled from the rest. (2) with important features being held separated from the rest, the second pruning step simply throws away the inactive, useless, or even harmful features over the layers. Cross-layer dependency is tracked by deconvolution-based utility reconstruction. We push and prune in a progressive and gradual manner since it helps improve and expedite the convergence at each iteration. We will show, through solving a generalized eigenvalue problem, that the first step can be achieved by simultaneously including deep LDA and covariance penalty terms to the optimization objective. The LDA and covariance loss terms are calculated per batch at the easily disentangled end (final latent space), but exert influence over the layers. For scenarios where the desired capacity is larger than what the base structure can offer, a simple network growing/expansion strategy is proposed. In contrast to fixed network architectures, our grow-push-prune pipeline provides an approach capable of generating a range of task-suitable models for different needs and constraints.

It is worth mentioning that this work is fundamentally different from our previous work (Tian et al., 2021). Tian et al. (2021) is a passive after-the-fact pruning approach that targets a pre-trained model. There is no active alignment of discriminants with easily pruned

* Corresponding author at: Department of Computer Science, Bowling Green State University, Hayes Hall, Bowling Green, OH 43403, USA.
E-mail addresses: qtian@bgsu.edu, qing.tian@mail.mcgill.ca (Q. Tian).

https://doi.org/10.1016/j.cviu.2023.103682
Received 2 October 2021; Received in revised form 11 March 2023; Accepted 14 March 2023
Available online xxxx

structures. Tian et al. (2021) cannot prune much if useful and useless features are already intertwined in the base. On the other hand, the proposed framework here proactively maximizes, condenses, and separates useful information flows over the network that contribute to the final class separation. Also, we propose a useful base net growing strategy while Tian et al. (2021) cannot do much if the base net capacity is not large enough (one-way top-down search).

In our experiments on the MNIST, CIFAR10, and ImageNet datasets, more efficient models with comparable/better accuracies to the base can be derived. On ImageNet, our series of grown deep Inception nets beat residual structures at similar complexities without any hard-coded dimension alignment. One of our grown deep Inception net, Inception-88, beats ResNet-50 (slightly larger) after training with the conventional cross-entropy and L_2 losses. Deep LDA pushing not only pushes utility into alignment with a compact set of latent neuron dimensions but also further increases the accuracy by 0.2%. The pruning step based on the 'pushed' model leads to a series of compact models with accuracies even higher than our grown deep Inception nets and resnets at similar sizes. At a pruning rate of approximately 6%, a pruned model achieves accuracy 0.4% higher than the original unpruned Inception-88. In addition, on ImageNet with MobileNetv2 as the base, we demonstrate our push-and-prune method's superiority over many state-of-the-art approaches.

2. Related work

2.1. Neural networks pruning

Early pruning approaches targeted at shallow nets date back to the late 1980s (Pratt, 1989; LeCun et al., 1989; Hassibi and Stork, 1992). Reed (1993) offers a review of such early researches that target shallow nets.

Aimed at deep nets, Han et al. (2015b) abandon weights of small magnitude by setting them to zero. Similar approaches that sparsify networks by setting zeros include (Srinivas and Babu, 2015; Mariet and Sra, 2016; Jin et al., 2016; Guo et al., 2016; Hu et al., 2016; Sze et al., 2017). Frankle and Carbin (2019) hypothesize that a large neural network contains a smaller subnetwork (winning ticket) which, when trained separately, can achieve similar accuracy. The top-down manner of search is necessary.

Weights based pruning usually leads to unstructured sparsity. As a solution to this issue, filter/channel pruning has gained popularity. Instead of setting zeros in weights matrices, they remove rows, columns, depths in weight/convolution matrices. Thus, the resulting architectures are more hardware friendly. They not only require less storage space and transportation/update bandwidth, but also result in less computation and carbon emissions. Moreover, with fewer intermediate feature maps produced and consumed, the number of slow and energy-consuming memory accesses is decreased. Some early works in this direction include (Polyak and Wolf, 2015; Anwar et al., 2015; Wen et al., 2016; Li et al., 2016; Tian et al., 2017; He et al., 2017; Zhuang et al., 2018).

Instead of regularizing the weights in a group-wise manner as Wen et al. (2016) does, Liu et al. (2017) and Ye et al. (2018) try to sparsify the scaling parameters of batch normalization. That said, features from locally 'unimportant' filters can gain importance over the layers. Guo et al. (2021) propose Gates with Differentiable Polarization (GDP) that are generally applicable with normal convolutional layers (even without batch normalization). The gate function used, namely smoothed L_0 formulation, can drive the gate values to either 0 or 1 during training, simplifying the later pruning.

Following a similar direction as LeCun et al. (1989), Molchanov et al. (2019) define neuron importance as the within-filter sum of weight importances (i.e., Taylor expansion of squared error induced without a weight). For efficiency concerns, they only conduct the first-order Taylor expansion and prune greedily based on every few batches during training. Peng et al. (2019) also try to approximate the loss function using Taylor expansion based on the pre-trained model parameters. Like LeCun et al. (1989), they attempt to approximate the loss using up to the second order of the Taylor expansion. Instead of assuming a diagonal Hessian matrix, Peng et al. (2019) take into consideration inter-channel dependency and approximate the Hessian matrix with first-order derivatives only (under a few assumptions and approximations). In Li et al. (2020), the authors reveal the importance of the Hessian/covariance structure for pruning. Compared to the 'brittle' magnitude-based pruning, Hessian-based pruning is more robust to problem structure and less sensitive to feature normalization. Due to the complexity involved, they only focus on linear models and shallow networks. Yu et al. (2022) also base their pruning on Hessian sensitivity (average Hessian trace). They prune insensitive components and keep sensitive ones. When it comes to 'moderately' sensitive components, they replace them with low rank neural implant, instead of completely removing them. Their approach is subject to the diagonal assumption as well.

In addition, many pruning works focus on redundancy. In He et al. (2019a), the authors point out two requirements of norm-based pruning (i.e., large norm deviation and small minimum norm). They then propose filter pruning via geometric median related redundancy (of filter norms in a layer) rather than importance. Suzuki et al. (2020) propose spectral pruning with minimizing within-layer redundancy as its primary goal. In each layer, they quantify the intrinsic dimensionality using the eigenvalues of the nodes' covariance matrix. A group of nodes' utility depends on how well the group can replace other nodes in the layer without significantly changing the layer's output. Wang et al. (2021) present a redundancy-based channel pruning approach and show that pruning from high redundant layers can outperform some importance-based approaches. They use quotient space size and ℓ-covering number on an undirected graph to measure the redundancy of each layer. Following the same line of thought, Joo et al. (2021) propose Linearly Replaceable Filter (LRF) pruning. They prune filters that can be approximated by a linear combination of others. To compensate for the output difference caused by pruning, they also present a technique called Weights Compensation, which wraps 1×1 convolutions (initialized as identity matrices) around the filters to be pruned. Singh et al. (2020b) use filter correlation (Pearson correlation coefficient) to measure the redundancy between filter pairs. The approach iteratively looks for filter pairs of greatest correlations, maximizes these pair correlations, and discards one filter from each pair. Singh et al. (2020a) base their pruning on a min–max game between two modules: an adaptive filter pruning (AFP) module and a pruning rate controller (PRC) module. Orthogonality constraints are applied during training to reduce redundancy and encourage diversity between filters.

He et al. (2020) develop a differentiable pruning criteria sampler named LFPC, which samples different criteria for different layers from a pool of candidates (e.g., L_1-norm, L_2-norm, geometric median related redundancy of filter norms). In Chin et al. (2020), the authors use L_2 norm to compare the filters' importance within each layer and propose to learn layer-wise affine transformations to compare filter importance across layers. Guo et al. (2020) model channel pruning as a Markov process where each state represents keeping the corresponding channel and transitions between states stand for the pruning process.

Although promising compression rates have been achieved, most pruning works possess some of the following drawbacks: (1) hard-coded or ad hoc utilities are computed locally and not directly related to final classification, such as magnitude and variance of weights and activation. (2) they usually depend on a pre-trained or passively learned model, i.e., the training is not pruning or separation aware. It may be too late to prune after the fact that useful and useless/harmful components are already intertwined together. At that time, it is hard to discard useless/redundant structures safely without hurting useful ones. (3) some filter-based approaches (e.g., Zhuang et al. (2018), Molchanov et al. (2019), Peng et al. (2019) and Chin et al. (2020))

rely on an implicit weight-level independence assumption (e.g. L_p-norms). (4) Redundancy-only pruning approaches (e.g., Suzuki et al. (2020) and Wang et al. (2021)) usually ignore the fact that useful redundancies are more desirable than useless ones, e.g., they can contribute to model robustness. Also, we argue that pruning according to an accurate importance measure can get rid of useless redundancies automatically and that unimportant features should be discarded even though they are unique. In this paper, we propose a deep discriminant analysis based importance measure that takes into consideration the relationships between weights, filters, and across layers. Unlike most existing pruning importance measures, our measure is directly related to the final class separation power. We proactively align latent neuron dimensions with discriminant directions during network training before discarding useless, redundant, or interfering dimensions.

Aside from pruning, approaches like bitwise reduction (Rastegari et al., 2016; Han et al., 2015a; Sun and Lin, 2016; Gupta et al., 2015; Gong et al., 2014; Courbariaux et al., 2015), filter decomposition (Denton et al., 2014; Jaderberg et al., 2014; Zhang et al., 2016), knowledge distillation (Hinton et al., 2015), and depth-wise separable convolution (Chollet, 2017; Howard et al., 2017) can further reduce model complexity. That said, they are not the focus of this paper.

Growing a network before pruning can sometimes be beneficial (Yuan et al., 2020; Mixter and Akoglu, 2020; Huang et al., 2005; Narasimha et al., 2008). Wang et al. (2017) demonstrate that more layers can lead to better clustered concepts. Belilovsky et al. (2019) show progressive linear separability with the depth increase.

2.2. Efficient neural architecture search

Most AutoML or Neural Architecture Search (NAS) approaches fall into one of the two categories: reinforcement learning (policy gradient) based (Baker et al., 2016; Zoph and Le, 2016; Zoph et al., 2017; Zhong et al., 2017) and evolutionary or genetic algorithms based (Stanley and Miikkulainen, 2002; Xie and Yuille, 2017; Miikkulainen et al., 2017; Real et al., 2017, 2018). Strict constraints are usually applied to reduce the search space. That said, each sampled architecture still needs to be trained separately. Given the large number of possible architecture samples, the procedure is very computationally expensive. For example, the search processes in Zoph and Le (2016) and Real et al. (2017, 2018) took the authors 28 days on 800 GPUs and one week on 450 GPUs, respectively. Most such works are done on the small CIFAR10 dataset. When it comes to larger datasets, resulting structures from small datasets are usually stacked up. Rather than design the entire network, some start with a macro architecture and fill in different substructure samples into each cell (micro search). ENAS (Pham et al., 2018) makes a strong assumption that common structures share the same weights. Similarly, instead of fully training all architecture samples, PNAS (Liu et al., 2018) 'predicts' the accuracy based on the differences between the new and parent samples. Bottom-up search in infinite spaces could possibly miss an 'optimal' structure in the early stage and never come back to it. He et al. (2018) propose AutoML to search compact models where they train a reinforcement learning agent to predict layerwise channel shrinking actions. To gain efficiency, the reward is roughly estimated based on the model accuracy prior to finetuning. Given the large number of architectures sampled, it is no surprise that the best achieves high accuracy. Liu et al. (2019b) tackle the problem in a differentiable manner and present a weight-sharing method for NAS named DARTS. However, it suffers from performance collapse caused by an inevitable aggregation of skip connections.

3. Proactive deep LDA dimension reduction

Existing AutoML works are generally expensive while passive pruning approaches rely heavily on the base model. In this paper, we propose a proactive deep discriminant analysis based approach that tracks down task-desirable compact architectures by exploring the deep feature space. Our approach iterates between two steps: (1) maximizing and pushing class separation utility to easily pruned substructures (e.g., neurons) and (2) pruning away less useful substructures. These two steps are illustrated in Algorithm 1, and the details will be given in Sections 3.1 and 3.2.

Algorithm 1: Proactive deep discriminant analysis based pushing and pruning

```
Input: base model (a popular net or one grown as in Sec. 4),
       acceptable accuracy $t_{acc}$
Output: task-suitable compact models
while True do
    Step 1 → Pushing
        Train the net with the deep LDA pushing objectives
        (Equation (11) in Sec.3.1) added. The process is also
        illustrated in Fig. 1;
        if accuracy ≤ $t_{acc}$ then break;
    Step 2 → Pruning
        Prune less useful components based on deconv source
        recovery;
return compact models derived
```

3.1. Pushing step

The room for complexity reduction in a deep network mainly comes from the useless and the redundant structures. Unlike after-the-fact pruning approaches, we explicitly embed these considerations into the loss function. We leverage LDA to boost class separation and utilize covariance losses to penalize redundancies. As we will show later, these terms simultaneously sort out and maximize useful information flow transferred over the network and push discriminating power into a small set of decision-making latent space neurons. The pushing step is demonstrated as Fig. 1. The LDA and covariance penalty terms (colored in red in Fig. 1) are computed in the last latent space (after ReLU) because: (1) it is directly related to decision making and it accepts information from all other layers, (2) the linear assumption of LDA is reasonable or easily enforced for over-parameterized networks. After all, there is only one linear FC layer left before decision making. Post-decision softmax, if any, is only a monotonic normalization and it cannot change the decision. (3) utility can be unraveled with ease from this disentangled or loosely twisted end. That said, these terms, as part of the training objective function, exert influence over the entire network.

Apart from cross-entropy, we explicitly and proactively apply linear discriminant analysis (LDA) in the final latent space to maximize class separation. The goal of the LDA term is to transform data from a noisy and complicated space to one where different categories can be linearly separated (there is only one final FC layer left). It is aligned with the training goal to reduce classification error. Latent features learned are expected to pick up class separating statistics in the input. Inspired by Fisher (1936) and Rao (1948), we define our deep LDA utility in the final latent space as Eq. (1), which will take advantage of both deep feature extraction and simple linear separability analysis:

$$S_{W,\theta} = \frac{|\, W^T \Sigma_{b,\theta} W \,|}{|\, W^T \Sigma_{w,\theta} W \,|}\,, \tag{1}$$

where

$$\Sigma_{w,\theta} = \sum_i \tilde{X}_{\theta,i}^T \tilde{X}_{\theta,i}\,, \tag{2}$$

$$\Sigma_{b,\theta} = \Sigma_{a,\theta} - \Sigma_{w,\theta}\,, \tag{3}$$

$$\Sigma_{a,\theta} = \tilde{X}_\theta{}^T \tilde{X}_\theta\,, \tag{4}$$

with X_θ being the set of observations obtained in the final latent space given model parameter setting θ. $X_{\theta,i}$ is the subset for category i. A

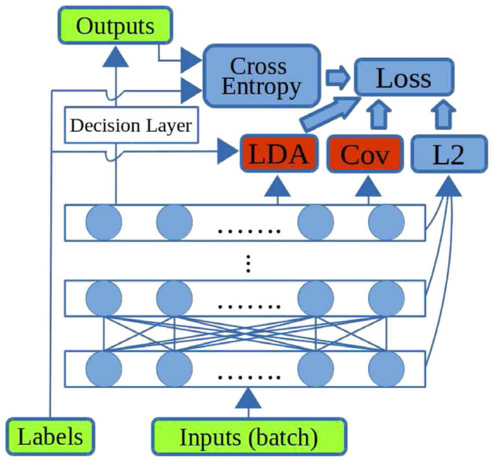


Fig. 1. Pushing step. Our deep LDA push objectives are colored in red. They maximize, unravel, and condense useful information flow transferred over the network and bring discriminants into alignment with latent space neurons. L_2 regularization is also applied to the decision layer, but is not shown for clarity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

pair of single vertical bars denote matrix determinant. The tilde sign ($\tilde{}$) represents a centering operation; for data X this means:

$$\tilde{X} = (I_n - n^{-1} 1_n 1_n^T) X \, , \tag{5}$$

where n is the number of observations in X, 1_n denotes an n × 1 vector of ones. The training objective of deep LDA is to maximize the final latent space class separation (Eq. (1)), which comes down to solving the following generalized eigenvalue problem:

$$\Sigma_{b,\theta} \vec{e}_j = v_j \Sigma_{w,\theta} \vec{e}_j \, , \tag{6}$$

where $(\vec{e}_j, v_j)$ represents a generalized eigenpair of the matrix pencil $(\Sigma_{b,\theta}, \Sigma_{w,\theta})$ with $\vec{e}_j$ as a W column. The LDA objective of maximizing Eq. (1) can be achieved by maximizing the average of v_js. Thus, we define the LDA-related loss term as its reciprocal:

$$\ell_{lda} = \frac{N}{\sum_j^N v_j} \, . \tag{7}$$

In practice, we set N as the number of neurons with non-negligible variances (dormant dimensions are not considered). Simultaneously, to penalize co-adapted structures and reduce redundancy in the network, we inject covariance penalty into the latent space. The corresponding loss is:

$$\ell_{cov} = \left\| \Sigma_{a,\theta} - diag(\Sigma_{a,\theta}) \right\|_1 \, , \tag{8}$$

where $\|.\|_1$ indicates entrywise 1-norm. This term agrees with the intuition that, unlike lower layers' common primitive features, higher layers of a well-trained deep net capture a wide variety of high-level, global, and easily disentangled abstractions (Bengio et al., 2013; Zeiler and Fergus, 2014). Generally speaking, the odds of various high-level patterns firing together should be low. ℓ_{cov} has a side effect of forcing useless dimensions to zero[1] and thus alleviates over-fitting (similar to dropout, but in a non-random and activation-based way).

Furthermore, to safely prune on the neuron level without much information loss, we need to align the above-mentioned LDA utility (v_js) with neuron dimensions. For this purpose, we try to align W columns with standard basis directions, e.g., (1,0, …,0), and let the network learn an optimal θ that leads to large class separation. This will also save us from using an actual W transformation/rotation in addition to the neural net. Given that duplicate neurons have been discouraged by ℓ_{cov} and inactive neurons are not considered here, Eq. (6) can be rewritten as:

$$(\Sigma_{w,\theta}^{-1} \Sigma_{b,\theta}) \vec{e}_j = v_j \vec{e}_j \, . \tag{9}$$

As we can see, W column $\vec{e}_j$s are the eigenvectors of $\Sigma_{w,\theta}^{-1} \Sigma_{b,\theta}$. Thus, forcing the direction alignment of LDA utilities and neuron dimensions is equivalent to forcing $\Sigma_{w,\theta}^{-1} \Sigma_{b,\theta}$ to be a diagonal matrix (eigenvectors of a diagonal matrix form a standard basis). We incorporate this constraint by including the following term to the loss function:

$$\ell_{align} = \left\| \Sigma_{w,\theta}^{-1} \Sigma_{b,\theta} - diag(\Sigma_{w,\theta}^{-1} \Sigma_{b,\theta}) \right\|_1 \, , \tag{10}$$

where, similar to Eq. (8), entrywise 1-norm is used instead of entrywise 2-norm (a.k.a. Frobenius norm) because our aim is to put as many off-diagonal elements to zero as possible. Combining all the three terms, we get our pushing objective as follows. Its three components jointly maximize class separation, squeeze and push classification utility into a compact set of neurons for later pruning:

$$\ell_{push} = \gamma \ell_{lda} + \lambda \ell_{cov} + \beta \ell_{align} \, , \tag{11}$$

where λ, β, and γ are weighting hyperparameters. Many advanced loss weighting strategies are available (e.g., Kendall et al. (2018)). Here, we set them so that (1) LDA utilities and neuron dimensions are aligned and (2) high accuracy is maintained. In our experiments, through parameter θ learning in the (large enough) base networks, the two goals can be met simultaneously without much hyperparameter tweaking. With the pushing terms above, we actually obtain higher accuracy on all the datasets explored (Section 5). In addition to class separation utility boost, the extra constraints can add some structure/regularization to the original overfitted deep space with very high degree of freedom. These terms help constrain useful information within or near more compact manifolds. ℓ_{lda} is sometimes numerically unstable. Inspired by Friedman (1989), we add a constant to the diagonal elements of the within-scatter matrix. When the category number is large, we cannot include all categories in one forward pass and the scatter matrices at a certain batch are calculated for a random subset of classes. Each class has the same or similar number of samples ($\geqslant$8). When latent space dimension d is large (e.g., in the first iteration), the ℓ_{align} constraint which includes an expensive $d \times d$ matrix inverse operation can be omitted. The reason is that in the context of over-parameterized network and high dimensional latent space, neuron activation is sparse: only a limited number of neurons tend to fire for a class and each high-level neuron motif corresponds to only one or few classes. In this scenario, positive within-class correlation indicates positive total correlation, and minimizing ℓ_{cov} has an effect of minimizing ℓ_{align}. Through training with the pushing objectives added, the network learns to organize itself for easy pruning. W columns that lead to large class separation (Eq. (1)) are aligned with some latent neuron dimensions. This pushing step lays the foundation for neuron/filter level pruning across all layers.

3.2. Pruning step

We treat pruning as a dimensionality reduction problem in the deep feature space. After the pushing step, the final class separation power is maximized and discriminants are simultaneously pushed into alignment with some top-layer neurons. It follows that the direct abandonment of less useful neurons and their dependencies on previous layers is safe. The discriminant power along the jth neuron dimension v_j is the corresponding diagonal value of $\Sigma w^{-1} \Sigma_b$:

$$v_j = diag(\Sigma w^{-1} \Sigma_b)_j \, , \tag{12}$$

where Σw and Σ_b are within-class and between-class scatter matrices of the final features based on the parameters trained. When pruning,

[1] For k category classification, there are at most $k-1$ uncorrelated linear discriminants (Hou and Riley, 2015). l_{cov} reduces redundancy across the dimensions and forces the activations (ideally) only along the ($\leqslant$) $k-1$ directions.

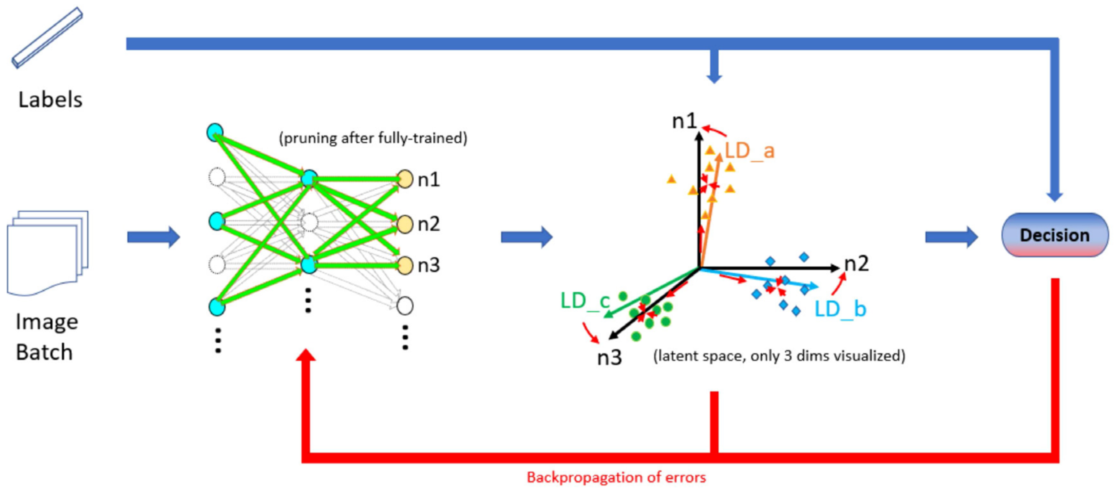

Labels
Image Batch
(pruning after fully-trained)
n1
n2
n3
LD_a
LD_b
LD_c
(latent space, only 3 dims visualized)
Decision
Backpropagation of errors

Fig. 2. Overview of the proposed push-and-prune pipeline. The blue arrows indicate forward information flow while the large red arrows represent backward propagation of errors. In the latent space (only 3 dimensions are visualized), the small red arrows illustrate some effects of our pushing terms (e.g., class separation, neuron-discriminant alignment, and decorrelation of neuron dimensions). Pruning occurs only after a model is fully trained according to our objectives. White components in the network are of limited utility and can thus be pruned. n indicates final latent space neuron dimensions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

we discard neuron dimension js of small v_j along with its cross-layer contributing sources in the 'pushed' model where useful components have been separated from others. To this end, we need to quantitatively measure the utility/contribution of all neurons to final utility before abandoning useless ones. In this paper, we utilize deconvolution/deconv (Zeiler and Fergus, 2014; Tian et al., 2021) to trace the task utility unraveled from final latent space backwards across all layers. In the final layer, only the responses of the most discriminative dimensions are preserved (other dimensions with small v_j are set to 0) before deconv starts. It is worth mentioning that Zeiler and Fergus (2014) use 'deconvolution' for visualization purposes in the image space while we focus on reconstructing contributing sources over the layers. Also, by tracing dependency exclusively from the useful final latent dimensions, our proposed method only back-propagates useful final variations. Irrelevant and interfering features of various kinds are 'filtered out' in the backward pass. The unit deconv procedure performs convolution with the same filters transposed. It can be considered as inversion of convolution with an assumption of orthogonal convolution matrix. More detail on this part can be found in our previous work (Tian et al., 2021). For modular structures, we need to sum many-to-one dependencies in a module. Specifically, the reconstructed utility maps from all module branches need to be summed/averaged at the module beginning before the utility can be further traced backwards to previous modules (the procedure is similar to reverse mode automatic differentiation). With all neurons'/filters' contribution to final discriminability known, pruning simply becomes discarding those of low utility/contribution. The utility threshold is directly related to the pruning rate. Since feature maps (neuron outputs) correspond to next-layer filter depths (neuron weights), our pruning leads to filter-wise and channel-wise savings simultaneously. After pruning at each iteration, retraining with surviving parameters is needed. Putting the pieces together, we provide a high-level summary of the proposed push-and-prune pipeline for deep dimension reduction in Fig. 2.

4. Base net growing strategy

One limitation with pruning is that the top-down, one-way search is bounded above by the base net's capacity. For datasets requiring larger capacities than the base net can offer, a growing step before iterative push-and-prune is necessary to first encompass/contain enough sub-architecture candidates. In the language of the famous 'lottery ticket hypotheses' (Frankle and Carbin, 2019), the growing step is like 'buying more lottery tickets'. According to Wang et al. (2017) and Belilovsky et al. (2019), growing can also lead to better-clustered concepts and progressive linear separability, which are critical to our deep LDA pushing and pruning. The growing/expansion step will add extra adjustment/wiggle room for the next push step to disentangle, compress, and re-organize utility in the network. We grow deep networks by greedily and iteratively adding more modules according to Algorithm 2, which can be viewed as a trial-and-error evolutionary process.

Algorithm 2: Greedy base net growing strategy

```
Input: $net = \{s_0, s_1, ..., s_i, ...\}$, $s_i = \{m_{i0}, m_{i1}, ..., m_{ij}, ...\}$, where $s$: stage, $m$:
       module, $net$: starting base. $N$: number of extra modules to add
Output: net with $N$ extra modules added
$n = 1; acc_{max} = 0; net_{opt};$
while $n \leqslant N$ do
    for $stage$ in $net$ do
        $net' = extend(net, stage);$
        train $net'$ and predict, get val accuracy $acc$;
        if $acc > acc_{max}$ then
            $acc_{max} = acc;$
            $net_{opt} = net';$
    $net = net_{opt}$, save if necessary;
    $n = n + 1;$
return $net$
```

At each iteration, we try to add one module to one of the network stages. Here, a stage consists of several modules with the same output feature map dimension before a pooling layer. The newly added module has the same architecture as the module underneath. We quickly train all the possible options (e.g., 3 for the Inception net) and keep only the one that achieves the highest accuracy. The process is repeated for N iterations until reaching a complexity bound or no noticeable accuracy gain can be observed after two consecutive iterations. By this growing strategy, a superset of abundant deep features can be obtained, from which our deep LDA pushing and pruning can derive task-desirable ones.

As an example, in this paper, we explore Inception structures for network growing/expansion. We prefer the Inception module over ResNets' residual modules as the building block because the latter requires hard-coded alignment of dimension, which is known to greatly limit the freedom of pruning (Li et al., 2016). To be more specific, the skip/residual dimension has to agree with the main trunk dimension for summation. However, after pruning according to any importance measure (including ours), they do not necessarily agree unless we force them to. Given that each ResNet module has only 2–3 layers, such a hard-coded constraint at the module end would greatly limit the freedom of pruning. Also, compared to residual models, inception modules offer us a variety of filter types. Our deep LDA pruning can make use of this and select both the numbers and types of filters on different abstraction levels. Compared to ResNets with up to hundreds of layers, current Inception models are relatively shallow and they only have a dozen or so modules. It has been proven that deep networks are able to approximate the accuracy of shallow networks with an exponentially fewer number of parameters, at least for some classes of functions (Telgarsky, 2016; Eldan and Shamir, 2016; Mhaskar et al., 2016; Safran and Shamir, 2017; Poggio et al., 2017). In this paper, we explore to grow from the basic inception net (Szegedy et al., 2015) by greedily adding more unit modules and see whether the resulting deep Inception nets can achieve ResNet-comparable accuracy. We use the initial Inception net (a.k.a. GoogLeNet) as the starting point but with two modifications inspired by Ioffe and Szegedy (2015). The first is to approximate the function of 5 × 5 filters with two consecutive 3 × 3 filters, and the second is to add batchnorm after each conv layer. In the rest of the paper, when we talk about the Inception net/module, we refer to this variant. Later inception modules (V2–V4) include more architecture fiddling and usually require higher resolution data (e.g. 299 × 299). We do not incorporate those changes since we want to perform fair comparisons between our grown deep Inception nets, ResNets, and some other popular networks taking 224 × 224 input. Also, this keeps human expert knowledge involved as minimum as possible. Ideally, we aim to replace such human knowledge with learning and pruning. Like the initial GoogLeNet, when training a grown Inception architecture, we add two auxiliary classifiers to the second stage (one after the first module and the other before the last module). We find the auxiliary classifiers very useful when the depth becomes large. Interestingly, despite its simplicity, no works have explored simply growing the original Inception net to gain accuracy. In this paper, we grow the net before deep LDA based pushing and pruning. The results on ImageNet will be shown in Section 5.3.

5. Experiments and results

This section tests our proactive deep discriminant analysis based pruning on the MNIST, CIFAR10, and ImageNet datasets. For all the datasets, we first train a network (fixed or grown) in the conventional way to report the baseline accuracy before applying the pushing step (with l_{push}, defined in Section 3.1, added to the objective). There are various non-architectural tricks that can possibly help increase a network's accuracy, such as extra training data, bounding box info, multi-cropping, various input resolutions, label smoothing regularization, mixup training, and distillation (He et al., 2019b). In our experiments, we refrain from employing extra non-architectural tweakings. Instead of trying to achieve the highest accuracy with all possibly beneficial additions, our focus here is to fairly compare different architectures with few factors of interference.

5.1. A toy experiment on MNIST

We use the MNIST dataset to illustrate deep LDA pushing's influence on the latent space. MNIST details can be found in LeCun et al. (1998). We leave out the first 1000 images in each category of the training set for validation. With a simple five hidden layer fully-connected network (1024-1024-1024-1024-32), we will show deep LDA pushing's efficacy. In this toy experiment, the last hidden layer is set to have 32 neurons for illustration clarity.

As mentioned previously, the main purpose of proactive LDA pushing (Section 3.1) is to push deep discriminants or class separation power into alignment with latent space neuron dimensions so that

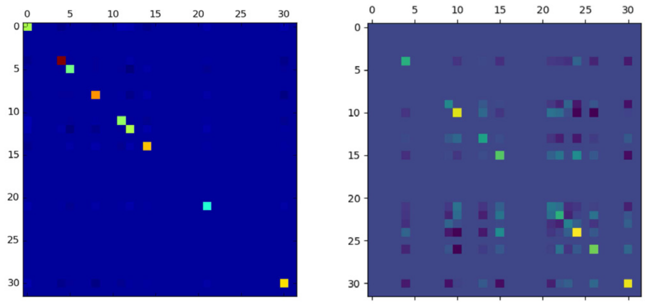
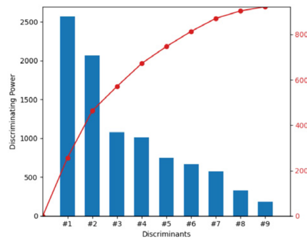

(a) with pushing objective (b) without pushing objective

Fig. 3. Variance-covariance matrices of the latent space neuron output after training (a) with and (b) without the pushing objective (Section 3.1) on the MNIST dataset using a toy FC architecture (hidden dimensions: 1024-1024-1024-1024-32). The values are color coded using the default bgr color map of the Matplotlib pyplot matshow function (Hunter, 2007). From small to large, the color transits from blue to green and finally to red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

filter-level pruning is safe. Although the pushing influence is across the layers, here via this toy example, we only illustrate how the final latent space is changed as other layers' changes influence the final decision via this space. Fig. 3 visualizes the variance–covariance matrix of latent space neuron output after training with and without the pushing objective. From Fig. 3, we can see that our proposed deep LDA pushing objective is effective and successfully pushes useful final decision-making variances to a subset of latent dimensions. Compared to Fig. 3(b), training with the pushing objective better decorrelates useful variances (Fig. 3(a)). As aforementioned, this contributes to the alignment of deep discriminants with latent dimensions. Also, more useless neuron dimensions are put to dormant in Fig. 3(a) than in Fig. 3(b). More importantly, the accuracy does not change much by including the pushing objective. In fact, the accuracy even improves slightly. The conventional cross-entropy with L_2 leads to a validation accuracy of 97.9%. This number increases to 98.3% with the addition of the deep LDA pushing objective. Fig. 4 shows the top nine discriminants after training with and without our pushing objective. As expected, the discriminating power (v_j in Eq. (9)), is improved with our deep LDA pushing by two orders of magnitude. Also, the distribution after pushing is more spiky and, in terms of proportion, more discriminating power is pushed to the large discriminants on the left. This can also be seen from the red accumulative discriminating power curve in Figs. 4(a) and 4(b). The first two discriminants count for about one third of the total discriminating power in the no-push case (Fig. 4(b)) while this number increases to 50% for the case with our pushing objective (Fig. 4(a)). It means that, with pushing, we can separate out more useful information with the same number of dimensions, and can thus afford to throw away more neuron dimensions in the later pruning stage (while still maintaining enough discriminating power). In this simple example, all neurons other than the top nine are put to dormant (with 0 discriminating power) after pushing while there are more active neurons in the no-push case (which are not shown in Fig. 4(b)). In the latter case, more useful information resides in those extra dimensions, intertwined with useless trivia. In our experiments, after reducing the original network size from 4.0M parameters to only 38.6K parameters, we can still maintain comparable test accuracy to the original.


(a) with pushing objective

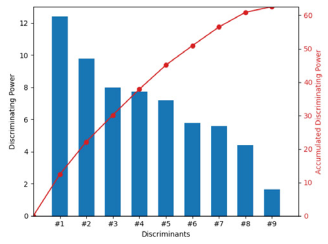

(b) without pushing objective

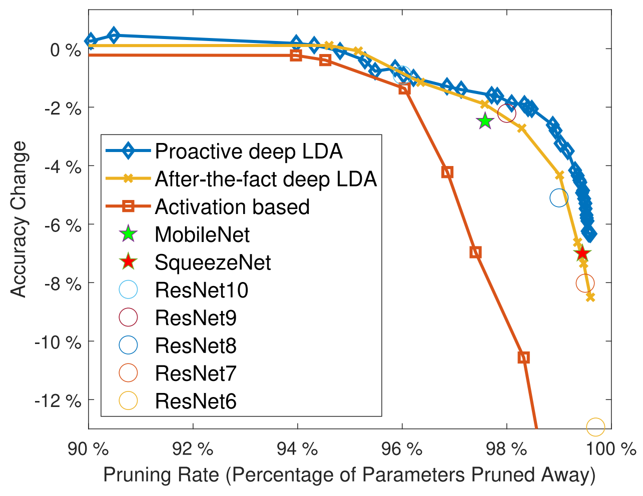
Fig. 4. Top nine discriminants after training (a) with and (b) without our pushing objective. The horizontal axis represents the nine top discriminants and the left vertical axis indicates their corresponding discriminating power (v_j in Eqs. (9) and (12)). The right vertical axis and the curve in red denote the accumulated discriminating power.

5.2. Experimental results of VGG-16 on CIFAR10

In this section, we will show the relationship of accuracy change v.s. pruned parameters on the CIFAR10 dataset (Krizhevsky and Hinton, 2009)[2] with VGG-16 as the base. The number of parameters is an important measure of complexity. Model size directly determines where the model/features reside, whether they can fit to different levels of caches and memories. It is memory accesses (rather than the raw number of operations) that dominantly influence energy and latency (Horowitz, 2014). In this experiment, we start with a VGG-16 model pre-trained on ImageNet. Cross-entropy loss with L_2 regularization leads to a validation accuracy of 95.19% on CIFAR10. In addition to aligning discriminants with neuron dimensions, our deep LDA pushing objective helps improve the accuracy to 95.72% without pruning. Fig. 5 illustrates the change of accuracy with respect to parameter pruning rate (the percentage of parameters discarded from the original model). We focus on high pruning rates where the accuracy changes fast with the decrease of parameters. That said, it is worth noting that among the few small pruning rates investigated, a pruned model with 118M parameters enjoys an even better accuracy (96.01%) than both the original model and the pushed one. For comparison, we add after-the-fact deep LDA pruning (Tian et al., 2021) and activation-based filter pruning (as mentioned in Molchanov et al. (2016), which treats filter importance as average activation magnitudes/variances within a filter). Also, we compare our method with some popular compact fixed nets, i.e., MobileNet (Howard et al., 2017), SqueezeNet (Iandola et al., 2016), and tiny ResNets. Here, tiny ResNets refer to residual nets with shallow depths. In this experiment, we test ResNet6–ResNet10. Their detailed configurations can be found in the Appendix.

As we can see from the results, our proactive-deep-LDA pruning, generally speaking, enjoys higher accuracy than the other two pruning approaches and the compact nets at similar complexities. The gaps are more obvious at high pruning rates, especially between activation-based pruning and our proactive deep LDA pruning. This performance difference implies that strong activation does not necessarily indicate high final classification utility. It is possible that some strong yet irrelevant activation skews or misleads the data analysis at the top of the network. Compared to after-the-fact deep LDA pruning, the proactive deep LDA pruning proposed in this paper enjoys better performance. The reason is that although after-the-fact deep LDA is capable of capturing final class separation utility, useful and useless components may already be mixed in the given pre-trained model, and it is hard to trim one without influencing the other. The performance differences are small at low pruning rates, perhaps because even when 'useful' feature components are discarded, the network can recover such or similar features through re-training when pruning rates are low. This 'learning to repair' ability via re-training gradually declines when the network capacity becomes small. Furthermore, even though ResNet is currently one of the most successful deep nets, stacking a few residual modules with random numbers of filters only leads to suboptimal performance compared to the proposed proactive deep LDA pruning. In Fig. 5, our deep LDA-pushed-and-pruned models beat tiny ResNets at most similar complexities. This indicates the necessity of informed pruning over architecture hand-engineering with human expertise.

[2] We set aside the first 10,000 images in the training set of CIFAR10 for validation.



CIFAR10, VGG-16 base accuracy: 95.19%, after pushing: 95.72%

Fig. 5. Accuracy change v.s. parameter pruning rate on CIFAR10. In addition to our proactive deep LDA pruning, we add after-the-fact deep LDA pruning (Tian et al., 2021), activation-based pruning (as in Molchanov et al. (2016)), MobileNet (Howard et al., 2017), SqueezeNet (Iandola et al., 2016), and tiny ResNets (details in Table 3) for comparison. Small pruning rates are skipped where accuracy changes little. The original base and competing fixed models are pre-trained on ImageNet. Each network is trained three times and the average accuracy is reported (non-percentage standard deviation < 0.001).

Upon further analysis of the smallest pruned model with comparable accuracy, we find that most parameters have been discarded in the layers except for the early layers that capture commonly useful patterns. Detailed layerwise complexity can be found in Appendix.

5.3. Experimental results of inception nets on ImageNet

In this subsection, we demonstrate our 'grow-push-prune' pipeline's efficacy on the ImageNet (ILSVRC12) dataset (Russakovsky et al., 2015). The dataset is widely used for benchmarking algorithms in computer vision and machine learning. In our experiment, all the images are pre-resized to 256 × 256. During training, the images are randomly cropped to 224 × 224 and randomly mirrored about the vertical axis. Most of the model training are conducted on two Nvidia Tesla V100 GPUs. Like many previous works, we report accuracy change on the validation set (no test set labels are publicly available) and only the center crop is used.

5.3.1. Base net growing and pushing

Table 1 shows some models encountered in the growing step using the basic Inception module on ImageNet. The accuracy in Table 1 is Top-1 accuracy using only one center crop. The name Inception-X indicates that the net is X-layer deep (only conv and fully-connected layers are considered).

As can be seen from the results, we can obtain more accuracy by simply stacking more modules to the basic InceptionNet stages (Algorithm 2). Particularly, we would like to introduce Inception-88

Table 1
Deep Inception net examples encountered in the base net growing process on ImageNet. The accuracy here indicates Top-1 accuracy using only one center crop. The name Inception-X means the net is X-layer deep (only conv and fully-connected layers are considered). The stage size column shows module numbers across the three stages. M = 10^6, B = 10^9.

Name	Modules	Stage size	Parameters	FLOPs	Accuracy
InceptionV1	9	(2,5,2)	6.7M	3.0B	70.64%
Inception-34	10	(3,5,2)	7.1M	3.7B	71.12%
Inception-37	11	(3,5,3)	8.6M	3.8B	71.75%
Inception-40	12	(4,5,3)	9.0M	4.5B	71.97%
Inception-43	13	(4,6,3)	10.0M	4.9B	72.03%
Inception-46	14	(4,7,3)	11.0M	5.3B	73.45%
Inception-49	15	(4,7,4)	12.5M	5.4B	73.51%
Inception-52	16	(4,7,5)	14.0M	5.6B	73.69%
Inception-55	17	(4,8,5)	15.0M	6.0B	73.91%
Inception-58	18	(5,8,5)	15.5M	6.6B	74.27%
Inception-61	19	(6,8,5)	15.9M	7.3B	74.20%
Inception-64	20	(7,8,5)	16.3M	8.0B	74.42%
Inception-67	21	(7,8,6)	17.8M	8.1B	74.58%
Inception-70	22	(7,8,7)	19.3M	8.3B	74.54%
Inception-73	23	(8,8,7)	19.8M	8.9B	74.64%
Inception-76	24	(8,8,8)	21.3M	9.1B	74.60%
Inception-79	25	(8,8,9)	22.8M	9.2B	74.59%
Inception-82	26	(9,8,9)	23.2M	9.9B	74.77%
Inception-85	27	(9,8,10)	24.7M	10.1B	74.60%
Inception-88	**28**	**(10,8,10)**	**25.1M**	**10.7B**	**75.01%**
Inception-91	29	(10,8,11)	26.6M	10.9B	74.71%

that achieves comparable accuracy (75.01%) to ResNet-50 (74.96%[3]) at a slightly smaller complexity. Even better, there are no hard-coded dimension alignment by human experts, which would limit the room for pruning. The depths of the three stages in Inception-88 are respectively 30, 24, and 30. Beyond Inception-88, accuracy first drops slightly before increasing slowly with increasing module number. This is also similar to the ResNet-50 case where 19M more parameters (ResNet-101) only increase the accuracy by less than 1% (He et al., 2016). Apart from increasing capacity and accuracy, this growing step offers more wiggle/adjustment room for the net to re-organize utility during the pushing step. Our pushing step increases Inception-88's accuracy to 75.2%, in addition to compressing and aligning utility with latent neuron dimensions.

5.3.2. *Accuracy change vs. pruning rate*

In the final pruning step, we strip off the separated unnecessary complexities. In this grow-push-prune pipeline, bottom-up search and top-down search are combined.

Fig. 6 demonstrates the influence of our pruning on accuracy. As can be seen, comparable accuracy can be maintained even after about 70% parameters are discarded. At the pruning rate of approximately 6%, a pruned model achieves an accuracy of 75.36% which is higher than that of both unpruned versions.

For comparison, we add to Fig. 6 the results of the deep Inception nets derived from the growing step, a range of ResNets, models from network trimming (Hu et al., 2016), and popular compact nets, including SqueezeNet (Iandola et al., 2016), MobileNet (Howard et al., 2017), NASNet-B (Zoph et al., 2017), BN-GoogLeNet[4] (Ioffe and Szegedy, 2015). We also include the results of training some pruned architectures from scratch (no weights are inherited). The detailed configurations of the ResNets used for comparison can be found in Appendix.

According to Fig. 6, we can see that our compact models pushed-and-pruned from Inception-88 outperform smaller deep Inception nets grown, the resnets, nets trimmed using Hu et al. (2016), and the fixed compact nets. The pruned models, by Hu et al. (2016) and our approach, achieve better performance compared to training the same architectures from scratch. This highlights the value of the knowledge acquired by and transferred from the larger grown-pushed base model in the form of weights. That said, even when trained from scratch, our pruned nets still attain satisfactory accuracy and beat many others, including those produced by network trimming (Hu et al., 2016) and then trained from scratch. It means that, besides the weights, there is some value in our derived architectures themselves.

Also, Fig. 6 reveals that our grown series of deep Inception nets outperform the residual structures at similar complexities. As far as we know, this is the first time that a range of basic Inception structures are fairly compared against residual structures on the same input, at least in the complexity range we investigated. Another advantage of these deep Inception nets over the residual structures is that the former does not need to enforce the output dimensions of a module's branches to be the same. To our best knowledge, there is no theoretical justification for why different branches need to have exactly the same dimension (except for some efficiency concerns). It is possible that there is more information lying on the 3×3 scale than others. As a result, in a ResNet module, the output dimensions will most likely not agree after a filter-level pruning based on importance (unless the agreement is enforced at the expense of more complexity). In this sense, deep Inception nets are more conducive to pruning approaches in general.

Compared to the three fixed nets shown as five-pointed stars in Fig. 6, the proposed pipeline not only achieves better accuracy at similar complexities but also offers a wide range of compact models for different accuracy and complexity requirements.

In contrast to expensive NAS approaches that train numerous architecture samples separately or based on ad hoc relations, our top-down search only needs to sample architectures along the direction aligned with task utility (e.g., 10^4 samples in NASNets Zoph et al., 2017 vs. 10^1 in ours). Therefore, unlike methods that 'predict' post-retraining performance, we can afford to fully retrain sampled architectures. Additionally, useful parameters inherited from the previous base make the sample architecture retraining process converge very fast. Usually, it only takes a few epochs to achieve accuracy within 5% from that of the fully trained.

With extra training data, more computing budget (e.g., more pruning iterations), and some tricks previously mentioned, the accuracy reported may be further improved. That said, achieving the best accuracy possible with non-architectural tricks is not the focus of this paper and is deferred to future work. A more detailed layerwise complexity analysis of one of our pruned models reveals that most parameters and computations over the layers can be pruned away, and different types of filters are pruned differently depending on the abstraction level and the scales where more task utility lies. More layerwise complexity detail is available in Appendix.

5.4. *Experimental results of MobileNetv2 on ImageNet*

To show our method's wide applicability, we also test our proactive pruning approach on MobileNetv2 (Sandler et al., 2018) on the ImageNet dataset. Unlike Inception nets, MobileNetv2 makes heavy use of both depthwise separable convolutions and residual connections. In particular, the depthwise separable convolutions greatly bring down the size and computation of MobileNetv2. In our implementation, to maintain the dimension consistency of adding layers in residual modules, we group together the layers bounded by the same constraint and prune them according to their average channel-wise utility. Our base model has a top-1 accuracy of 72.01% and our deep LDA pushing slightly increases the accuracy to 72.12%. Fig. 7 shows the relationship between accuracy and pruned FLOPs of the pruned models. As comparison, we also include in the figure many state-of-the-art approaches, e.g., MetaPruning (Liu et al., 2019a), DMCP (Guo et al., 2020), GFS (Ye et al., 2020), LeGR (Chin et al., 2020), AMC (He et al., 2018), and Tu

[3] Unlike the ResNet-50 achieving 76% in Tensorflow, no bounding box info is used in any of our models. Only 1-center crop is used for validation.

[4] It is not just GoogLeNet + batchnorm. There are more architectural changes to GoogLeNet which we do not include in our grown deep Inception nets.

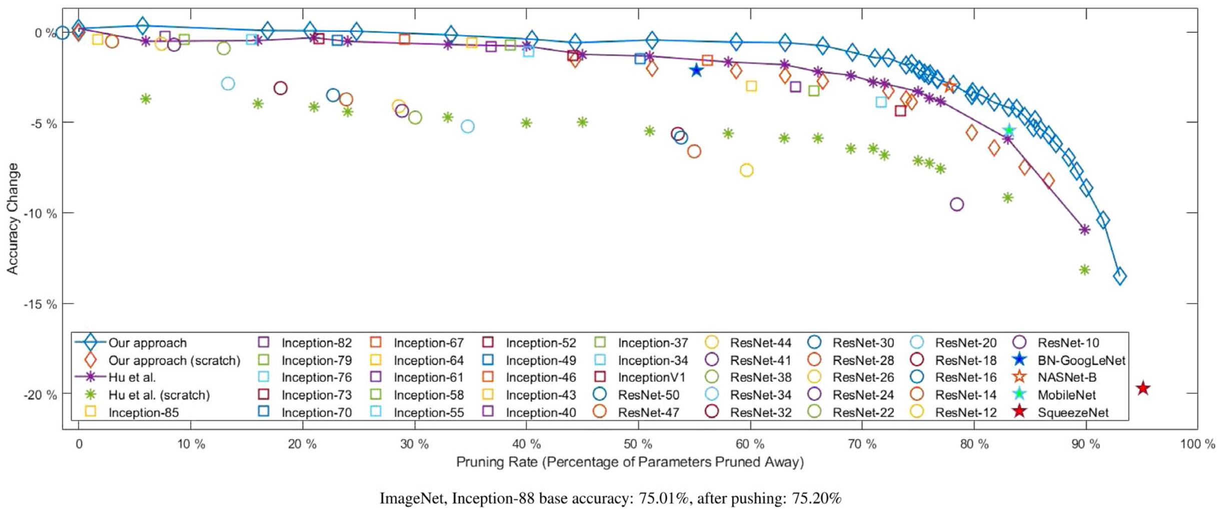


Fig. 6. Accuracy change vs. parameter pruning rate on ImageNet. In addition to our deep LDA push-and-prune method (blue), we add our grown deep Inception nets, various ResNets, network slimming (Hu et al., 2016), and popular fixed compact nets for comparison. There are two accuracies at 0 pruning rate. The upper one represents Inception-88 trained with our deep LDA push objective added. The negative pruning rate of ResNet-50 indicates its larger size than Inception-88. Our derived nets trained from scratch (red diamonds) mark the beginning of each iteration for our approach. NASNet-B architecture is from Zoph et al. (2017) and trained from scratch in the same experimental setting as ours. Each network is trained three times and the average accuracy is reported (non-percentage standard deviation < 0.001). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

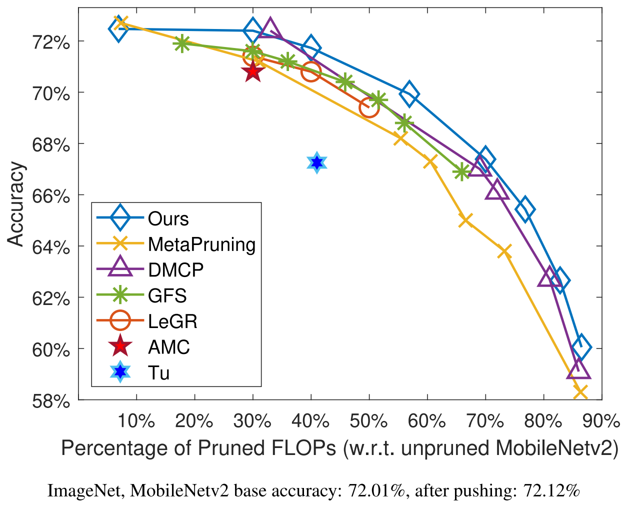


Fig. 7. Accuracy v.s. percentage of pruned FLOPs on ImageNet. In addition to our proactive deep LDA pruning, we add MetaPruning (Liu et al., 2019a), DMCP (Guo et al., 2020), GFS (Ye et al., 2020), LeGR (Chin et al., 2020), AMC (He et al., 2018), and Tu et al. (2020) for comparison. The MobileNetv2 base has 314M multiply-adds. MetaPruning and DMCP prune from MobileNetv2 2× and 1.5× super nets, respectively. Each of our networks is trained three times and the average accuracy is reported (non-percentage standard deviation < 0.001).

et al. (2020). It is worth mentioning that most neural architecture search models, such as NASNet models (A–C) (Zoph and Le, 2016), are even heavier than the unpruned MobileNetv2. Thus, they are not included in the figure.

As we can see from the figure, our proactive deep LDA pruning also works for the already compact MobileNetv2 model with residual connections and depthwise separable convolutions. It beats the state-of-the-art models by clear margins at similar complexities.

Table 2
Model inference time. The inference times are measured on a machine with an Intel i7-9750H CPU and a Nvidia RTX 2080 Max-Q GPU. Each latency number is the average of 100 runs. Synchronization between the GPU and the CPU is enforced.

Model	Parameters	FLOPs	GPU latency	CPU latency
Original	25.1M	10.7B	53.5 ms	432.4 ms
Pruned	8.4M	5.1B	39.8 ms	189.0 ms
Ratio	2.99×	2.10×	1.34×	2.29×

5.5. *Inference latency analysis*

In this section, we illustrate how parameter/FLOPs savings from pruning translate to actual inference latency reduction. Alongside parameters and FLOPs savings, we report the inference latency of our Inception-88 model and its smallest pruned model with comparable accuracy. It is worth noting that direct timing depends heavily on hardware specifics and environmental factors (e.g., base frequency, memory/cache structure and scheduling, asynchronous execution, and warm-up of a GPU). In our case, we perform the inference locally on a personal computer with an Intel i7-9750H CPU and a Nvidia RTX 2080 Max-Q GPU. The latency results of the pruned and unpruned models on both the GPU and the CPU are shown in Table 2, along with the speedup ratios.

According to Table 2, the pruned model, about one third of the original model's size, achieves a 2.29 times speedup on the CPU. On the GPU, the speedup ratio decreases to 1.34. The reason is that the unpruned model can make better use of the massive parallelism of the GPU (3072 cores) than the pruned model.

6. Discussion and future directions

Most of today's popular deep architectures are handcrafted/designed following a generalist trend to solve as many tasks as possible, which results in unnecessarily cumbersome and power-hungry models that are infeasible for embedded applications (e.g., autonomous driving). Our deep discriminant analysis (Tian et al., 2021), a non-linear generalization of LDA, is able to pick up high-order moments/statistics embedded in the complex raw data space with the help of deeply learned transformation. In this paper, targeting classification tasks, we propose proactive deep discriminant analysis pruning. During (pre-)training, our pushing terms align deep discriminants with latent neurons/filters, which makes it feasible to separate useful and useless information on the neuron/filter level without hurting the final performance. The terms also help make more room for later pruning.

As a future direction, we plan to extend the idea of proactive deep discriminant analysis and reduction to visual detection tasks. Visual detection involves both class separation and localization, which we believe are closely related. Accurate localization is based on correctly identifying distinguishing features for each class (Xie et al., 2017). We will design a location-and-class-aware variant of deep LDA to quantify the detection utility and maximize it during training. Correlation penalty is also needed as part of the training objective to reduce redundancy and help align discriminants. When calculating the LDA utility for detection tasks, we will only extract and pool the activations/features from relevant object locations and surroundings. The importance of an intermediate layer neuron/filter will be calculated based on both the magnitude and location of the corresponding reconstructed useful activations (e.g., the object location information can be utilized to filter out more irrelevant neurons/filters). Our utility tracing is expected to offer insight into what aspects of, where exactly in the image, and to what extent the intermediate layer features contribute to the visual detection.

Also, it would be of great interest to investigate our deep-discriminant-based pruning's influence on model robustness. Our hypothesis is that, in addition to slowing down the inference process, unnecessary and overfitted components can open more doors for malicious attacks. To be more specific, we hypothesize that attacks can trigger or take advantage of interfering features that are not aligned with task demands and throw off the prediction. The more such task-irrelevant features a model has, the higher the chance it will be hit by adversarial attacks and noises. By discarding irrelevant and interfering structures, our pruning can potentially decrease the chance of the model falling victim to irrelevant factors in the image space and thus mitigate the model's vulnerability. We will also explore discriminant-aware adversarial training to better align the adversarial gradients derived from the classification and localization losses for more effective adversarial attack and defense.

Finally, it is worth mentioning that the proposed growing strategy in this paper is not restricted to a particular module type. The Inception module just serves as an example building block that has standard convolutions and various filter types. In our future work, we plan to experiment with more building block types when there is a need to expand/grow a base network. Furthermore, the proposed growing strategy can potentially offer more plasticity for transfer learning and domain adaptation tasks, which will also be investigated in our future work.

7. Conclusion

In this paper, instead of relying on a pre-trained model, we have proposed a proactive pruning approach for compact architecture search based on deep discriminant analysis. The approach follows a two-step procedure in iterations: (1) through learning, it proactively maximizes and unravels twisted threads of deep discriminants, condenses and pushes them into alignment with a subset of neurons; (2) after useful features are separated from others, the second pruning step simply throws away the useless or even harmful components over the layers based on deconv tracing. In addition, we explore growing the base model for tasks requiring larger capacity. We demonstrate our methods' efficacy and superiority to many state-of-the-art pruned/fixed models on the MNIST, CIFAR10, and ImageNet datasets. By growing from the basic InceptionV1 to an 88-layer-deep Inception variant, we show that deep Inception nets, without any hard-coded agreement of dimension, can beat ResNets of similar sizes on ImageNet.

Table 3
Tiny ResNets used as comparison in our experiments on CIFAR10. The dash sign '-' separates different stages. As defined in He et al. (2015), there are two types of residual modules, i.e., identity module and convolutional module where 1×1 filters are employed on the shortcut path to match dimension. Only depth-2 modules are used here. In this table, 'i' stands for depth-2 identity block and 'c' represents depth-2 convolutional block. The number follows 'i' or 'c' indicates the number of filters within each conv layer in that module. We adopt the same stem layers as in He et al. (2015).

Name	Configuration
ResNet6	i64-c128
ResNet7	i64-c128-1c256
ResNet8	i64-c128-c256
ResNet9	i64-c128-c256-1c512
ResNet10	c64-c128-c256-c512

CRediT authorship contribution statement

Qing Tian: Methodology, Implementation, Writing. **Tal Arbel:** Supervision, Writing – review & editing. **James J. Clark:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

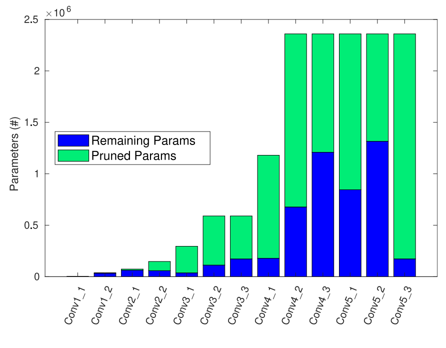
This research was partially supported by McGill Engineering Doctoral Awards and the National Science Foundation (under Award No. 2153404). This work would not have been possible without the computing resources provided by Compute Canada (www.computecanada.ca) and the Ohio Supercomputer Center (www.osc.edu).

Appendix A. Detailed configurations of ResNet 6–10 used for comparison in Section 5.2

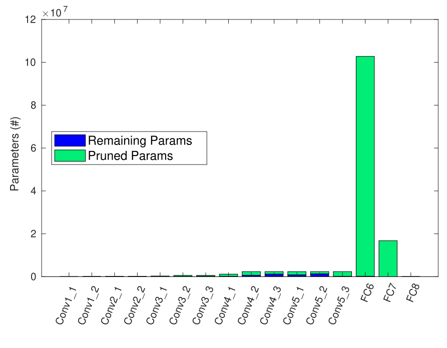
Table 3 demonstrates the detailed structures of ResNet 6–10 used in our Experiments (Fig. 5, Section 5.2).

Appendix B. Layerwise complexity of the VGG-16 pruned on CIFAR10

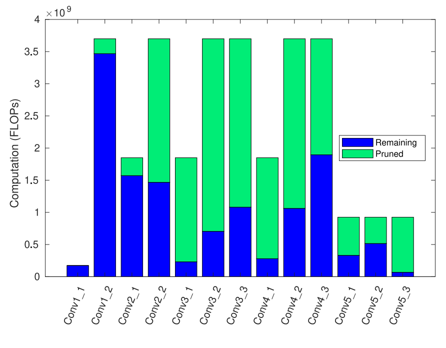
Fig. 8 demonstrates the layerwise complexity of our smallest pruned model that maintains comparable accuracy to the original VGG-16 on CIFAR-10 (Section 5.2). The FC layers dominate the original net size, while almost all computation comes from the conv layers. According to the results, most parameters and computations have been thrown away in the layers except for the first three layers that capture commonly useful patterns.



(a) Conv param



(b) Param savings



(c) FLOP savings

Fig. 8. Layerwise complexity reductions (CIFAR10, VGG16). Green: pruned, blue: remaining. We add a separate parameter analysis for conv layers because FC layers dominate the model size. Since almost all computations (above 99%) are in the conv layers, only conv layer FLOPs are demonstrated. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
ResNets used as comparison in our experiments on ImageNet. The dash sign '-' separates different stages. As defined in He et al. (2015), there are two types of residual modules, i.e., identity module and convolutional module where 1×1 filters are employed on the shortcut path to match dimension. Here, 'i' stands for depth-2 identity block, 'c' represents depth-2 convolutional block, 'I' stands for depth-3 identity block, and 'C' represents depth-3 convolutional block. The number follows 'i', 'c', 'I', or 'C' indicates the number of filters within each conv layer in that module. Parentheses are used to group multiple modules in a stage. We adopt the same stem layers as in He et al. (2015).

Name	Configuration
ResNet6	i64-c128
ResNet7	i64-c128-1c256
ResNet8	i64-c128-c256
ResNet9	i64-c128-c256-1c512
ResNet10	c64-c128-c256-c512
ResNet12	(c64, i64)-c128-c256-c512
ResNet18	(c64, i64)-(c128, i128)-(c256, i256)-(c512, i512)
ResNet20	(c64, i64)-(c128, i128)-(c256, i256)-(c512, i512, i512)
ResNet22	(c64, i64)-(c128, i128)-(c256, i256, i256)-(c512, i512, i512)
ResNet24	(c64, i64)-(c128, i128, i128)-(c256, i256, i256)-(c512, i512, i512)
ResNet26	(c64, i64, i64)-(c128, i128, i128)-(c256, i256, i256)-(c512, i512, i512)
ResNet28	(c64, i64, i64)-(c128, i128, i128)-(c256, i256, i256, i256)-(c512, i512, i512)
ResNet30	(c64, i64, i64)-(c128, i128, i128, i128)-(c256, i256, i256, i256)-(c512, i512, i512)
ResNet32	(c64, i64, i64)-(c128, i128, i128, i128)-(c256, i256, i256, i256, i256)-(c512, i512, i512)
ResNet34	(c64, i64, i64)-(c128, i128, i128, i128)-(c256, i256, i256, i256, i256, i256)-(c512, i512, i512)
ResNet38	(C64, I64, I64)-(C128, I128, I128)-(C256, I256, I256)-(C512, I512, I512)
ResNet41	(C64, I64, I64)-(C128, I128, I128)-(C256, I256, I256, I256)-(C512, I512, I512)
ResNet44	(C64, I64, I64)-(C128, I128, I128, I128)-(C256, I256, I256, I256)-(C512, I512, I512)
ResNet47	(C64, I64, I64)-(C128, I128, I128, I128)-(C256, I256, I256, I256, I256)-(C512, I512, I512)
ResNet50	(C64, I64, I64)-(C128, I128, I128, I128)-(C256, I256, I256, I256, I256, I256)-(C512, I512, I512)

Appendix C. Detailed configurations of ResNets used for comparison in Section 5.3

The detailed configurations of the ResNets used for comparison in Section 5.3 are shown in Table 4.

Starting from ResNet-50, a module is iteratively removed from the longest stage to produce smaller ResNets. When two stages have the same number of modules, we follow a bottom-to-top order to choose which module to remove (until ResNet18). From ResNet-50 to ResNet-38, the residual modules are of depth 3. From ResNet-34 downwards, each module has a maximum depth of 2. The depth-2 and depth-3 residual modules are defined the same as in He et al. (2015).

Appendix D. Layerwise complexity of the Inception-88 pruned on ImageNet

Figs. 9 and 10 visualize layerwise parameter and FLOPs complexity reduction results of our pruning on our 'grown-pushed' Inception-88 model (with comparable accuracy maintained, Section 5.3). From left to right, the conv layers within an Inception module are (1×1), (1×1, 3×3), ($1 \times 1, 3 \times 3$a, 3×3b), (1×1 after pooling) layers. According to Figs. 9 and 10, most parameters and computations over the layers are pruned away, and different types of filters are pruned differently depending on the abstraction level and the scales where more task utility lies. As anticipated, the pruning rates of the first few layers, which capture commonly useful primitive patterns, are low. Almost all of the parameters and FLOPs are pruned away in the last two modules, which indicates that the depth is large enough (at least locally). This is in agreement with our observation at the growing step that adding one or two more modules to the Inception-88 net does not help much. Interestingly, while the deep Inception net was greedily grown to achieve the highest accuracy locally, there are still massive redundant and useless structures over the layers. That is to say, at the growing step, each time we stacked one more module in an attempt to gain more accuracy, we simultaneously added more useless structures due to the ad hoc filter numbers used. Those useless structures cannot be effectively aligned with task utility even after training and can thus be discarded. The large pruning rates over the layers highlight our approach's advantage over architecture hand-engineering with ad-hoc filter numbers.

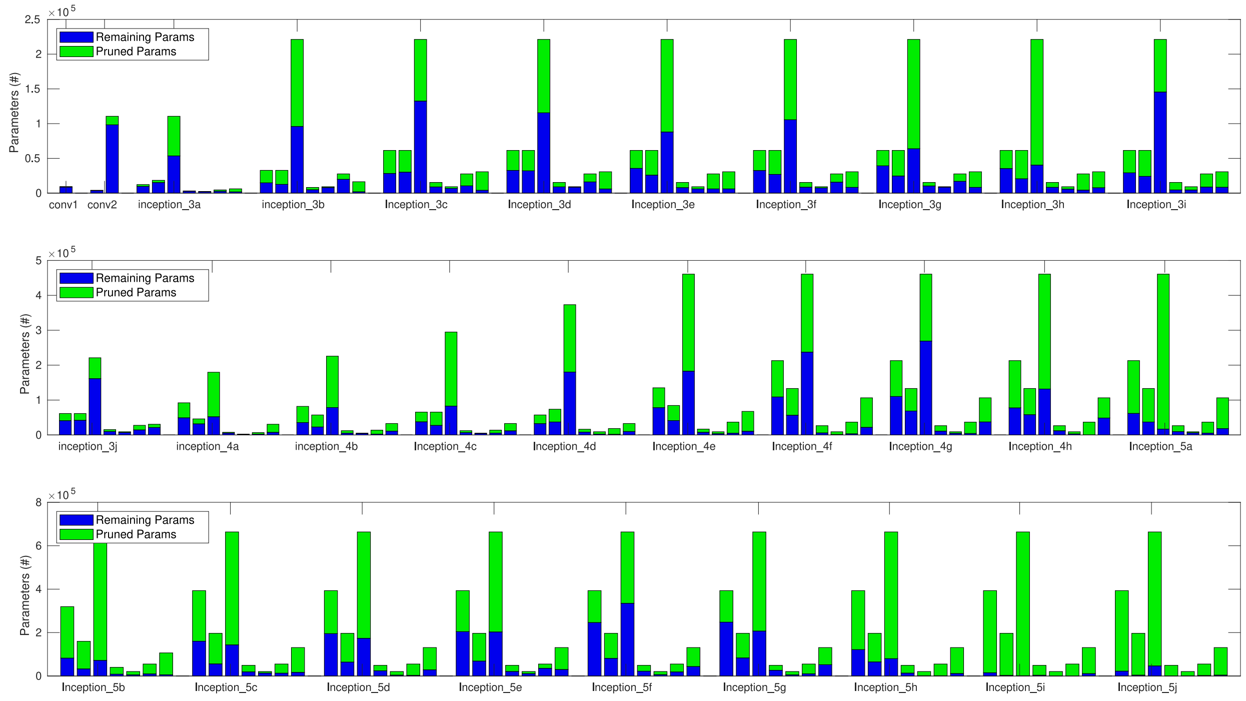


Fig. 9. Layerwise parameter reductions of the grown Inception-88 on ImageNet. From left to right, the conv layers in an Inception module are (1 × 1), (1 × 1, 3 × 3), (1 × 1, 3 × 3 a, 3 × 3 b), (1 × 1 after pooling). Green: pruned, blue: remaining. Due to the large network depth, the layer-wise parameter complexity figure is displayed in three rows. conv2 includes a dimension-reducing layer in front (notation skipped because of space limit). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

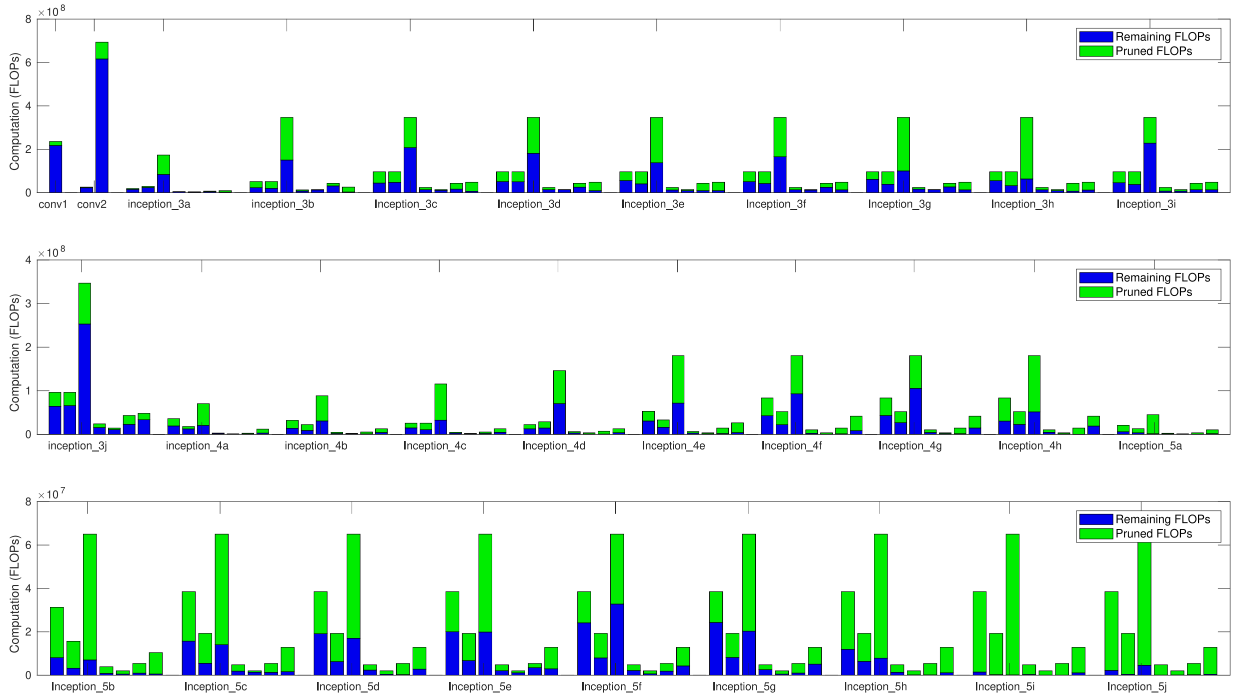


Fig. 10. Layerwise FLOPs reductions of the grown Inception-88 on ImageNet. From left to right, the conv layers in an Inception module are (1 × 1), (1 × 1, 3 × 3), (1 × 1, 3 × 3 a, 3 × 3 b), (1 × 1 after pooling). Green: pruned, blue: remaining. Due to the large network depth, the layer-wise FLOPs complexity figure is displayed in three rows. conv2 includes a dimension-reducing layer in front (notation skipped because of space limit). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

References

Anwar, S., Hwang, K., Sung, W., 2015. Structured pruning of deep convolutional neural networks. arXiv preprint arXiv:1512.08571.

Baker, B., Gupta, O., Naik, N., Raskar, R., 2016. Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167.

Belilovsky, E., Eickenberg, M., Oyallon, E., 2019. Greedy layerwise learning can scale to imagenet. In: Proceedings of the International Conference on Machine Learning. PMLR, pp. 583–593.

Bengio, Y., Mesnil, G., Dauphin, Y., Rifai, S., 2013. Better mixing via deep representations. In: Proceedings of the International Conference on Machine Learning. pp. 552–560.

Chin, T.-W., Ding, R., Zhang, C., Marculescu, D., 2020. Towards efficient model compression via learned global ranking. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1518–1528.
Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1251–1258.
Courbariaux, M., Bengio, Y., David, J.-P., 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. pp. 3123–3131.
Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R., 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems. pp. 1269–1277.
Eldan, R., Shamir, O., 2016. The power of depth for feedforward neural networks. In: Conference on Learning Theory. pp. 907–940.
Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. Ann. Eugen. 7 (2), 179–188.
Frankle, J., Carbin, M., 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: Proceedings of the International Conference on Learning Representations. ICLR.
Friedman, J.H., 1989. Regularized discriminant analysis. J. Amer. Statist. Assoc. 84 (405), 165–175.
Gong, Y., Liu, L., Yang, M., Bourdev, L., 2014. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115.
Guo, S., Wang, Y., Li, Q., Yan, J., 2020. Dmcp: Differentiable markov channel pruning for neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1539–1547.
Guo, Y., Yao, A., Chen, Y., 2016. Dynamic network surgery for efficient dnns. In: Advances in Neural Information Processing Systems. pp. 1379–1387.
Guo, Y., Yuan, H., Tan, J., Wang, Z., Yang, S., Liu, J., 2021. Gdp: Stabilized neural network pruning via gates with differentiable polarization. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5239–5250.
Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P., 2015. Deep learning with limited numerical precision. In: Proceedings of the International Conference on Machine Learning. pp. 1737–1746.
Han, S., Mao, H., Dally, W.J., 2015a. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. p. 2, CoRR abs/1510.00149.
Han, S., Pool, J., Tran, J., Dally, W., 2015b. Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. pp. 1135–1143.
Hassibi, B., Stork, D.G., 1992. Second order derivatives for network pruning: optimal brain surgeon. In: Advances in Neural Information Processing Systems. pp. 164–171.
He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., Yang, Y., 2020. Learning filter pruning criteria for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2009–2018.
He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., Han, S., 2018. Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision. ECCV, pp. 784–800.
He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y., 2019a. Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349.
He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.
He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778.
He, Y., Zhang, X., Sun, J., 2017. Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1389–1397.
He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Li, M., 2019b. Bag of tricks for image classification with convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 558–567.
Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
Horowitz, M., 2014. 1.1 Computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers. ISSCC, IEEE, pp. 10–14.
Hou, S., Riley, C., 2015. Is uncorrelated linear discriminant analysis really a new method? Chemometr. Intell. Lab. Syst. 142, 49–53.
Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
Hu, H., Peng, R., Tai, Y.-W., Tang, C.-K., 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250.
Huang, G.-B., Saratchandran, P., Sundararajan, N., 2005. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Trans. Neural Netw. 16 (1), 57–67.
Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. Comput. Sci. Eng. 9 (3), 90–95.
Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv preprint arXiv:1602.07360.
Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
Jaderberg, M., Vedaldi, A., Zisserman, A., 2014. Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866.
Jin, X., Yuan, X., Feng, J., Yan, S., 2016. Training skinny deep neural networks with iterative hard thresholding methods. arXiv preprint arXiv:1607.05423.
Joo, D., Yi, E., Baek, S., Kim, J., 2021. Linearly replaceable filters for deep network channel pruning. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35. pp. 8021–8029.
Kendall, A., Gal, Y., Cipolla, R., 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7482–7491.
Krizhevsky, A., Hinton, G., 2009. Learning Multiple Layers of Features from Tiny Images. Technical Report.
LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2324.
LeCun, Y., Denker, J.S., Solla, S.A., Howard, R.E., Jackel, L.D., 1989. Optimal brain damage. In: NIPs. pp. 598–605.
Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2016. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.
Li, M., Sattar, Y., Thrampoulidis, C., Oymak, S., 2020. Exploring weight importance and hessian bias in model pruning. arXiv preprint arXiv:2006.10903.
Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C., 2017. Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744.
Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., Sun, J., 2019a. Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3296–3305.
Liu, H., Simonyan, K., Yang, Y., 2019b. DARTS: Differentiable architecture search. In: Proceedings of the International Conference on Learning Representations. ICLR.
Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K., 2018. Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision. ECCV, pp. 19–34.
Mariet, Z., Sra, S., 2016. Diversity networks. In: Proceedings of the International Conference on Learning Representations. ICLR.
Mhaskar, H., Liao, Q., Poggio, T., 2016. Learning functions: when is deep better than shallow. arXiv preprint arXiv:1603.00988.
Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., Hodjat, B., 2017. Evolving deep neural networks. arXiv preprint arXiv:1703.00548.
Mixter, J., Akoglu, A., 2020. Growing artificial neural networks. arXiv preprint arXiv:2006.06629.
Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J., 2019. Importance estimation for neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11264–11272.
Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J., 2016. Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440.
Narasimha, P.L., Delashmit, W.H., Manry, M.T., Li, J., Maldonado, F., 2008. An integrated growing-pruning method for feedforward network training. Neurocomputing 71 (13–15), 2831–2847.
Peng, H., Wu, J., Chen, S., Huang, J., 2019. Collaborative channel pruning for deep networks. In: Proceedings of the International Conference on Machine Learning. PMLR, pp. 5113–5122.
Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J., 2018. Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268.
Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q., 2017. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. Int. J. Autom. Comput. 14 (5), 503–519.
Polyak, A., Wolf, L., 2015. Channel-level acceleration of deep face representations. IEEE Access 3, 2163–2175.
Pratt, L.Y., 1989. Comparing biases for minimal network construction with back-propagation. In: Advances in Neural Information Processing Systems. 1, pp. 875–886.
Rao, C.R., 1948. The utilization of multiple measurements in problems of biological classification. J. R. Stat. Soc. Ser. B Stat. Methodol. 10 (2), 159–203.
Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A., 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In: Proceedings of the European Conference on Computer Vision. Springer, pp. 525–542.
Real, E., Aggarwal, A., Huang, Y., Le, Q.V., 2018. Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548.
Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q., Kurakin, A., 2017. Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041.
Reed, R., 1993. Pruning algorithms-a survey. IEEE Trans. Neural Netw. 4 (5), 740–747.
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vis. (IJCV) 115 (3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Safran, I., Shamir, O., 2017. Depth-width tradeoffs in approximating natural functions with neural networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, pp. 2979–2987.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520.

Singh, P., Verma, V.K., Rai, P., Namboodiri, V.P., 2020a. Acceleration of deep convolutional neural networks using adaptive filter pruning. IEEE J. Sel. Top. Sign. Proces. 14 (4), 838–847.

Singh, P., Verma, V.K., Rai, P., Namboodiri, V., 2020b. Leveraging filter correlations for deep model compression. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 835–844.

Srinivas, S., Babu, R.V., 2015. Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149.

Stanley, K.O., Miikkulainen, R., 2002. Evolving neural networks through augmenting topologies. Evol. Comput. 10 (2), 99–127.

Sun, F., Lin, J., 2016. Memory efficient nonuniform quantization for deep convolutional neural network. arXiv preprint arXiv:1607.02720.

Suzuki, T., Abe, H., Murata, T., Horiuchi, S., Ito, K., Wachi, T., Hirai, S., Yukishima, M., Nishimura, T., 2020. Spectral pruning: Compressing deep neural networks via spectral analysis and its generalization error. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. pp. 2839–2846.

Sze, V., Yang, T.-J., Chen, Y.-H., 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5687–5695.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–9.

Telgarsky, M., 2016. Benefits of depth in neural networks. In: Conference on Learning Theory. pp. 1517–1539.

Tian, Q., Arbel, T., Clark, J.J., 2017. Deep LDA-Pruned nets for efficient facial gender classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 512–521.

Tian, Q., Arbel, T., Clark, J.J., 2021. Task dependent deep LDA pruning of neural networks. Comput. Vis. Image Underst. 203, 103154. http://dx.doi.org/10.1016/j.cviu.2020.103154.

Tu, C.-H., Lee, J.-H., Chan, Y.-M., Chen, C.-S., 2020. Pruning depthwise separable convolutions for mobilenet compression. In: Proceedings of the IEEE International Joint Conference on Neural Networks. pp. 1–8.

Wang, Z., Li, C., Wang, X., 2021. Convolutional neural network pruning with structural redundancy reduction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14913–14922.

Wang, Y.-X., Ramanan, D., Hebert, M., 2017. Growing a brain: Fine-tuning by increasing model capacity. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2471–2480.

Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H., 2016. Learning structured sparsity in deep neural networks. Adv. Neural Inf. Process. Syst. 29.

Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., Yuille, A., 2017. Adversarial examples for semantic segmentation and object detection. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1369–1378.

Xie, L., Yuille, A., 2017. Genetic cnn. arXiv preprint arXiv:1703.01513.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., Liu, Q., 2020. Good subnetworks provably exist: Pruning via greedy forward selection. In: Proceedings of the International Conference on Machine Learning. PMLR, pp. 10820–10830.

Ye, J., Lu, X., Lin, Z., Wang, J.Z., 2018. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In: Proceedings of the International Conference on Learning Representations. ICLR.

Yu, S., Yao, Z., Gholami, A., Dong, Z., Kim, S., Mahoney, M.W., Keutzer, K., 2022. Hessian-aware pruning and optimal neural implant. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 3880–3891.

Yuan, X., Savarese, P., Maire, M., 2020. Growing efficient deep networks by structured continuous sparsification. arXiv preprint arXiv:2007.15353.

Zeiler, M.D., Fergus, R., 2014. Visualizing and understanding convolutional networks. In: Proceedings of the European Conference on Computer Vision. Springer, pp. 818–833.

Zhang, X., Zou, J., He, K., Sun, J., 2016. Accelerating very deep convolutional networks for classification and detection. IEEE Trans. Pattern Anal. Mach. Intell. 38 (10), 1943–1955.

Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.-L., 2017. Practical block-wise neural network architecture generation. arXiv preprint arXiv:1708.05552.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J., 2018. Discrimination-aware channel pruning for deep neural networks. In: Advances in Neural Information Processing Systems. pp. 875–886.

Zoph, B., Le, Q.V., 2016. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V., 2017. Learning transferable architectures for scalable image recognition. arXiv preprint arXiv:1707.07012.