Knowledge Evolution in Neural Networks

Ahmed Taha Abhinav Shrivastava Larry Davis
University of Maryland, College Park

Abstract

Deep learning relies on the availability of a large corpus of data (labeled or unlabeled). Thus, one challenging unsettled question is: how to train a deep network on a relatively small dataset? To tackle this question, we propose an evolution-inspired training approach to boost performance on relatively small datasets. The knowledge evolution (KE) approach splits a deep network into two hypotheses: the fit-hypothesis and the reset-hypothesis. We iteratively evolve the knowledge inside the fit-hypothesis by perturbing the reset-hypothesis for multiple generations. This approach not only boosts performance, but also learns a slim network with a smaller inference cost. KE integrates seamlessly with both vanilla and residual convolutional networks. KE reduces both overfitting and the burden for data collection.

We evaluate KE on various network architectures and loss functions. We evaluate KE using relatively small datasets (e.g., CUB-200) and randomly initialized deep networks. KE achieves an absolute 21% improvement margin on a state-of-the-art baseline. This performance improvement is accompanied by a relative 73% reduction in inference cost. KE achieves state-of-the-art results on classification and metric learning benchmarks. Code available at http://bit.ly/3uLgwYb

1. Introduction

Gene transfer is the transfer of genetic information from a parent to its offspring. Genes encode genetic instructions (knowledge) from ancestors to descendants. The ancestors do not necessarily have better knowledge; yet, the evolution of knowledge across generations promotes a better learning curve for the descendants. In this paper, we strive to replicate this process for deep networks. We encapsulate a deep network's knowledge inside a subnetwork, dubbed the fit-hypothesis $H^{\triangle}$. Then, we pass the fit-hypothesis's knowledge from a parent network to its offspring (next deep network generation). We repeat this process iteratively and demonstrate a significant performance improvement in the descendant networks as shown in Fig. 1.

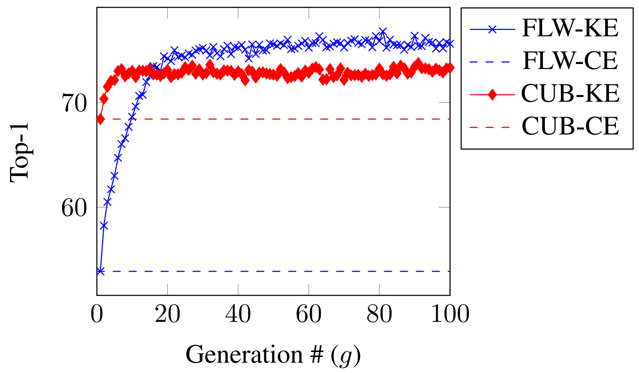


Figure 1. Classification performance on Flower-102 (FLW) and CUB-200 (CUB) datasets trained on a randomly initialized ResNet18. The horizontal dashed-lines denote a SOTA cross-entropy (CE) baseline [58]. The marked-curves show our approach (KE) performance across generations. The 100th generation KE-N_{100} achieves absolute 21% and 5% improvement margins over the Flower-102 and CUB-200 baselines, respectively.

The lottery ticket literature [8, 62, 34, 42, 10] regards a dense network as a set of hypotheses (subnetworks). Zhou *et al*. [62] propose a sampling-based approach, while Ramanujan *et al*. [42] propose an optimization-based approach, to identify the best *randomly-initialized* hypothesis. This hypothesis may be called the lottery ticket, but it is still limited by its random initialization. In this paper, we pick a random hypothesis, with inferior performance, and iteratively evolve its knowledge.

The main contribution of this paper is an evolution-inspired training approach. To evolve knowledge inside a deep network, we split the network into two hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset hypothesis $H^{\triangledown}$ as shown in Fig. 2. We evolve the knowledge inside $H^{\triangle}$ by re-training the network for multiple generations. For every new generation, we perturb the weights inside $H^{\triangledown}$ to encourage the $H^{\triangle}$ to learn an independent representation. This knowledge evolution approach boosts performance on relatively small datasets and promotes a better learning curve for descendant networks. Our intuitions are presented in Sec. 3.3 and empirically validated in Sec. 5.

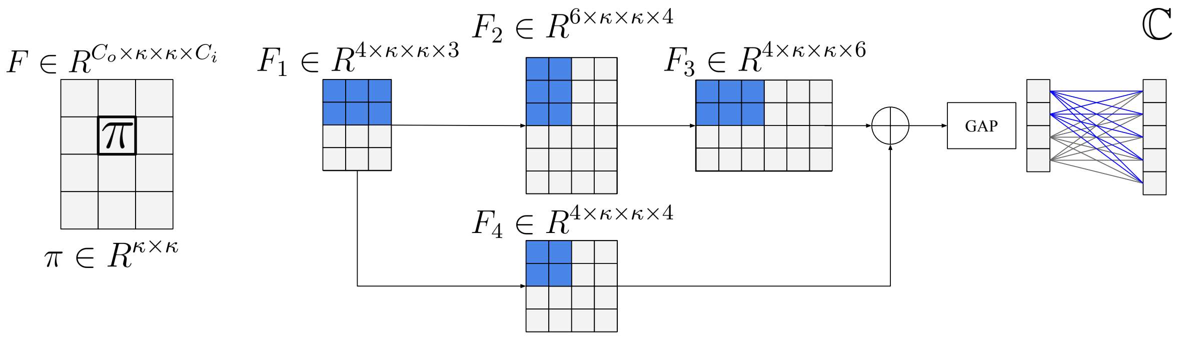

$F \in R^{C_o \times \kappa \times \kappa \times C_i}$
π
$\pi \in R^{\kappa \times \kappa}$
$F_1 \in R^{4 \times \kappa \times \kappa \times 3}$
$F_2 \in R^{6 \times \kappa \times \kappa \times 4}$
$F_3 \in R^{4 \times \kappa \times \kappa \times 6}$
$F_4 \in R^{4 \times \kappa \times \kappa \times 4}$
GAP
$\mathbb{C}$

Figure 2. A split network illustration using a toy residual network. (Left) A convolutional filter F with $C_i = 3$ input, $C_o = 4$ output channels, and 2D kernels (*e.g.*, $\pi \in R^{3\times3}$). (Center-Right) A toy residual network N with a three-channel input (*e.g.*, RGB image) and a five-logit output ($\mathbb{C} = 5$). GAP denotes a global average pooling layer while $\bigoplus$ denotes the add operation. We split N into a fit-hypothesis $H^{\triangle}$ (dark-blue) and a reset-hypothesis $H^{\triangledown}$ (light-gray). The fit-hypothesis $H^{\triangle}$ is a slim network that can be extracted from the dense network N to perform inference efficiently. The paper appendix shows the dimensions of a fit-hypothesis in the ResNet18 architecture.

The knowledge evolution (KE) approach requires network-splitting. If we split the weights of a neural network into two hypotheses ($H^{\triangle}$ and $H^{\triangledown}$) *randomly*, KE will boost performance. This emphasizes the generality of our approach. Furthermore, we propose a **ke**rnel-**l**evel convolutional-aware **s**plitting (KELS) technique to reduce inference cost. KELS is a splitting technique tailored for convolutional neural networks (CNNs). KELS splits a CNN such that the fit-hypothesis $H^{\triangle}$ is a slim independent network with a smaller inference cost as shown in Fig. 2. The KELS technique supports both vanilla CNNs (AlexNet and VGG) and modern residual networks.

KE supports various network architectures and loss functions. KE integrates seamlessly with other regularization techniques (*e.g.*, label smoothing). While KE increases the training time, the KELS technique reduces the inference cost significantly. Most importantly, KE mitigates overfitting on relatively small datasets, which in turn reduces the burden for data collection. Our community takes natural images for granted because they are available publicly. However, for certain applications, such as autonomous navigation and medical imaging, the data collection process is expensive even when labeling is not required.

In summary, the key contributions of this paper are:

1. A training approach, knowledge evolution (KE), that boosts the performance of deep networks on relatively small datasets (Sec. 3.1). We evaluate KE using both classification (Sec. 4.1) and metric learning (Sec. 4.2) tasks. KE achieves SOTA results.
2. A network splitting technique, KELS, which learns a slim network automatically while training a deep network (Sec. 3.2). KELS supports a large spectrum of CNNs and introduces neither hyperparameters nor regularization terms. Our ablation studies (Sec. 5) demonstrate how KELS reduces inference cost significantly.

2. Related Work

This section compares knowledge evolution (KE) with two prominent training approaches: Born-Again Networks (BANs) [9] and Dense-Sparse-Dense (DSD) [14]. In the paper appendix, we compare KELS with the pruning literature [26, 16, 15, 13, 27, 54, 61, 31, 29, 57, 21]

DSD [14] starts with a dense-phase to learn connections' weights and importance. Then, the sparse-phase prunes the unimportant connections and resumes training given a sparsity constraint. The final dense-phase removes the sparsity constraint, re-initializes the pruned connections, and trains the entire dense network. KE differs from DSD in multiple ways: (1) DSD masks (prunes) individual weights, while KE masks complete convolution kernels. Thus, DSD delivers dense networks, while KE delivers both dense and slim networks. (2) KE introduces the idea of a fit-hypothesis to encapsulate a network's knowledge and to evolve this knowledge across generations.

BANs [9] is a knowledge-distillation based approach. Similar to KE, BANs trains the same architecture iteratively. However, to transfer knowledge between successive networks, BANs uses the class-logits distribution, while KE uses the networks' weights. This explains why BANs uses the teacher-student terminology while KE uses the parent-sibling terminology. This difference is important because (1) training a teacher network, which teaches future students, requires a large corpus of data (labeled or not). In contrast, KE acknowledges the deficiency of a parent network trained on a small dataset; (2) BANs randomly initializes student networks while KE leverages the knowledge of a parent network to initialize the next generation.

We distance our work from neural architecture search (NAS) literature [63, 28] such as Neural Rejuvenation [40] and MorphNet [11]. We assume the network's connections and the number of parameters are fixed.

3. Knowledge Evolution

In this section, we present (1) the knowledge evolution (KE) approach (Sec. 3.1), (2) various network-splitting techniques (Sec. 3.2), (3) intuitions behind KE (Sec. 3.3), and (4) how we evaluate KE (Sec. 3.4).

3.1. The Knowledge Evolution Training Approach

We first introduce our notation. We assume a deep network N with L layers. The network N has convolutional filters F, batch norm Z, and fully connected layers with weight W, bias B terms.

The Knowledge evolution (KE) approach starts by *conceptually* splitting the deep network N into two exclusive hypotheses (subnetworks): the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$ as shown in Fig. 2. These hypotheses are outlined by a binary mask M; 1 for $H^{\triangle}$ and 0 for $H^{\triangledown}$, *i.e.*, $H^{\triangle} = MN$ and $H^{\triangledown} = (1-M)N$. We present various splitting techniques in Sec. 3.2. After outlining the hypotheses, the network N is initialized randomly, *i.e.*, both $H^{\triangle}$ and $H^{\triangledown}$ are initialized randomly. We train N for e epochs and refer to the trained network as the first generation N_1, where $H_1^{\triangle} = MN_1$ and $H_1^{\triangledown} = (1-M)N_1$.

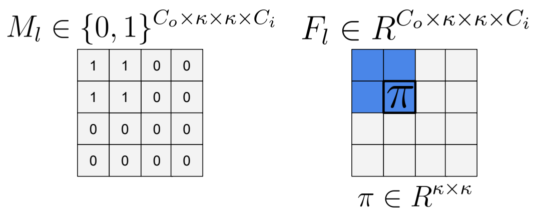
To learn a better network (the next generation), we (1) **re-initialize** the network N using $H_1^{\triangle}$, then (2) **re-train** N to learn N_2. First, the network N is **re-initialized** using the convolutional filters F and weights W in the fit-hypothesis $H_1^{\triangle}$ from N_1, while the rest of the network ($H^{\triangledown}$) is initialized randomly. Formally, we re-initialize each layer l, using Hadamard product, as follows

$$F_l = M_l F_l + (1 - M_l)\, F_l^r, \tag{1}$$

where F_l is a convolutional filter at layer l, M_l is the corresponding binary mask and F_l^r is a randomly initialized tensor. These three tensors (F_l, F_l^r, and M_l) have the same size ($\in R^{C_o \times \kappa \times \kappa \times C_i}$). F_l^r is initialized using the default initialization distribution. For example, PyTorch uses Kaiming uniform [18] for convolution layers.

Similarly, we re-initialize the weight W_l and bias B_l through their corresponding binary masks. Modern architectures have bias terms in the single last fully connected layer only ($B \in R^{\mathbb{C}}$). Thus, for these architectures, all bias terms belong to the fit-hypothesis, *i.e.*, $B \subset H^{\triangle}$. We transfer the learned batch norm Z across generations without randomization.

After re-initialization, we **re-train** N for e epochs to learn the second generation N_2. To learn better networks, we repeatedly **re-initialize** and **re-train** N for g generations. Basically, we transfer knowledge (convolutional filters and weights) from one generation to the next through the fit-hypothesis $H^{\triangle}$. It is important to note that (1) the contribution of a network-generation ends immediately-after initializing the next generation, *i.e.*, each generation is trained independently, (2) After training a new generation, the weights inside both hypotheses change, *i.e.*, $H_1^{\triangle} \neq H_2^{\triangle}$ and $H_1^{\triangledown} \neq H_2^{\triangledown}$, and (3) all network generations are trained using the exact hyperparameters, *i.e.*, same number of epochs, optimizer, learning rate scheduler, etc.



$M_l \in \{0,1\}^{C_o \times \kappa \times \kappa \times C_i}$
$F_l \in R^{C_o \times \kappa \times \kappa \times C_i}$
1 1 0 0
1 1 0 0
0 0 0 0
0 0 0 0
π
$\pi \in R^{\kappa \times \kappa}$

Figure 3. The KELS technique for CNNs. Given a split-rate s_r and a convolutional filter F_l at a layer l, the binary split-mask M_l outlines the first $\lceil s_r \times C_i \rceil$ kernels inside the first $\lceil s_r \times C_o \rceil$ filters. In this example, $C_o = C_i = 4$ and $s_r = 0.5$. Through KELS, the binary mask M outlines the fit-hypothesis $H^{\triangle}$ such that it is a slim network inside a dense network. The slim network $H^{\triangle}$ is equivalent to a dense network with $(1 - s_r^2)$ sparsity.

3.2. Split-Networks

KE requires network-splitting. We support KE with two splitting techniques: (1) a simple technique to highlight the generality of KE, and (2) an efficient technique for CNNs.

The simple technique is the ***weight-level*** **splitting** (WELS) technique. For every layer l, a binary mask M_l splits l into two exclusive parts: the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$. Given a split-rate $0 < s_r < 1$, we *randomly* split the weights $W_l \in R^{|W_l|}$ using the mask $M_l \in \{0,1\}^{|W_l|}$, where $|W_l|$ is the number of weights inside layer l and $\text{sum}(M_l) = s_r \times |W_l|$. The WELS technique supports a large spectrum of layers – fully connected, convolution, recurrent, and graph convolution. This highlights the generality of KE.

Through WELS, KE boosts the network performance across generations. However, WELS does not benefit from the connectivity of CNNs. Thus, we propose a splitting technique that not only boosts performance but also reduces inference cost for relatively small datasets. We leverage the CNNs' connectivity and outline the fit-hypothesis $H^{\triangle}$ such that it is a slim (pruned) network as shown in Fig. 2. Instead of masking individual weights, we mask kernels, *i.e.*, ***kernel-level convolutional-aware*** **splitting** (KELS) technique. Given a split-rate s_r and a convolutional filter $F_l \in R^{C_o \times \kappa \times \kappa \times C_i}$, KELS outlines the fit-hypothesis to include the first $\lceil s_r \times C_i \rceil$ kernels inside the first $\lceil s_r \times C_o \rceil$ filters as shown in Fig. 3. KELS guarantees matching dimensions between consequence convolutional filters. Thus, KELS integrates seamlessly in both vanilla CNNs (AlexNet and VGG) and modern architectures with residual links.

For relatively small datasets, the performance of the slim fit-hypothesis $H^{\triangle}$ reaches the performance of the dense

Algorithm 1: The KE training approach splits a dense network N, with L layers, into fit and reset hypotheses using a split-rate s_r and a binary mask M. Then, KE trains N for g generations. The network N has convolutional filters F, weight W, bias B, and batch norm Z. We assume a single fully connected layer for simplicity.

```
Result: Both a dense network $N_g$ and a slim
        network $H_g^{\triangle}$ outlined by the split mask $M$
/* Set the split masks $M$ for conv and FC layers
   once and for all.                                        */
for layer $l$ to $L$ do
    if is_conv($l$) then
        $C_o, \kappa, \_, C_i = F_l$.shape;
        $M_l$ = zeros(($C_o, \kappa, \kappa, C_i$));
        if $C_i$ == 3 then
            $M_l[: C_o \times s_r, :, :, :] = 1$ ; // First conv
        else
            $M_l[: C_o \times s_r, :, :, : C_i \times s_r] = 1$;
        end
    else if is_fc($l$) then
        $C_o, C_i = W_l$.shape;                       // $C_o$=ℂ
        $M_l$ = zeros(($C_o, C_i$));
        $M_l[:, : C_i \times s_r] = 1$;
    end
end
$W, B, Z, F$ are initialized randomly;
for generation $i$ to $g$ do
    $N_i \leftarrow$ Train $N$ for $e$ epochs;   // Learn $W,B,Z,F$
    for layer $l$ to $L$ do
        if is_conv($l$) then
            $F_l^r = \text{rand}(F_l.\text{shape})$;
            $F_l = M_l F_l + (1 - M_l) F_l^r$;
        else if is_fc($l$) then
            $W_l^r = \text{rand}(W_l.\text{shape})$;
            $W_l = M_l W_l + (1 - M_l) W_l^r$;
        end
    end
end
```

network N. In these cases, $H^{\triangle}$ not only delivers the dense network's performance but also reduces the inference cost. Through KELS, the slim $H^{\triangle}$ runs on general purpose hardware, *i.e.*, neither sparse BLAS libraries nor specialized hardware [12] is required. Given a split rate s_r, KELS delivers a slim $H^{\triangle}$ that is equivalent to a dense network N with *approximately* $(1 - s_r^2)$ sparsity. It is *approximate* because the network's end-points have s_r sparsity. The first convolutional layer operates on all input channels (*e.g.*, RGB) and fully connected layers have s_r sparsity. Algorithm 1 summarizes KE while applying the KELS technique.

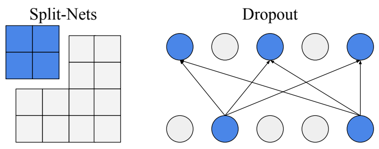


Figure 4. Split-Nets vs Dropout: The reset-hypothesis $H^{\triangledown}$ and dead neurons are highlighted in gray, while the fit-hypothesis $H^{\triangle}$ and "alive" neurons are highlighted in blue.

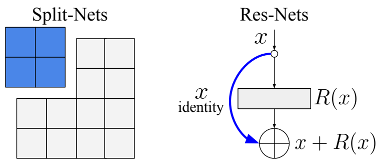


Figure 5. Split-Nets vs Res-Nets: Res-Nets split a network into an identity shortcut (blue) and a residual subnetwork $R(x)$. Split-Nets split a network into a fit-hypothesis $H^{\triangle}$ (blue) and a reset-hypothesis $H^{\triangledown}$. By splitting a network into two branches, Res-Net and Split-Net enable a zero-mapping in one of these branches ($R(x)$ and $H^{\triangledown}$) while keeping the network's depth intact.

3.3. Knowledge Evolution Intuitions

To understand KE, we give two complementary intuitions. These intuitions do not require the KELS technique. We use KELS for visualization purpose only (*e.g.*, Fig. 4). We empirically validate these intuitions in Sec. 5.

Intuition #1: Dropout

Dropout [45] randomly drops neurons during training as shown in Fig. 4. This encourages neurons to rely less on each other and to learn independent representations [5]. In KE, we drop the reset-hypothesis $H^{\triangledown}$ during re-initialization by randomly initializing $H^{\triangledown}$ before every generation. This encourages $H^{\triangle}$ to rely less on $H^{\triangledown}$ and to learn an independent representation. We validate this intuition by evaluating the performance of the slim $H^{\triangle}$ across generations. We observe that the performance of $H^{\triangle}$ increases as the number of generations increases.

Intuition #2: Residual Network

Res-Nets set the default mapping, between consecutive layers, to the identity as shown in Fig. 5. Yet, from a different perspective, Res-Nets enable a zero-mapping in some subnetworks (residual links) without limiting the network's capacity [51, 55]. Similarly, KE enables a zero-mapping in the reset-hypothesis $H^{\triangledown}$ by re-using the fit-hypothesis $H^{\triangle}$ across generations. After the first generation N_1, $H^{\triangle}$ is always closer to convergence compared to $H^{\triangledown}$ that contains random values. Thus, KE encourages new generations to evolve the previous-generations' knowledge inside the fit-hypothesis $H^{\triangle}$ and suppress $H^{\triangledown}$.

We validate this intuition by measuring the mean absolute value inside both hypotheses. We observe that $H^{\triangle}$ and $H^{\triangledown}$ have comparable mean values at the first generation N_1.

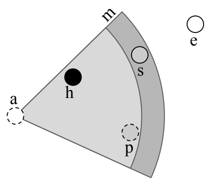


Figure 6. Triplet loss tuple (anchor, positive, negative) and margin m. The (h)ard, (s)emi-hard, and (e)asy negatives are highlighted in black, gray, and white, respectively.

However, as the number of generations increases, the mean absolute value inside $H^{\triangle}$ increases and $H^{\triangledown}$ decreases. This supports our claim that KE promotes a zero-mapping inside the reset-hypothesis $H^{\triangledown}$.

Please note that Split-Nets have one degree of freedom that Res-Nets omit. Through the split-rate s_r, we control the size of the fit and reset hypotheses ($H^{\triangle}$ and $H^{\triangledown}$). If the training data is abundant, a large split-rate is better where a Split-Net reverts into a dense Res-Net. However, for relatively small datasets, a small split-rate s_r significantly reduces the inference cost while improving performance. In the paper appendix, we elaborate more on both intuitions.

3.4. Evaluation Tasks

We evaluate KE using two supervised tasks: (1) classification and (2) metric learning. The performance of deep networks on small datasets is studied extensively using the classification task [48, 38, 7, 3, 35, 56, 59, 60, 58]. Thus, the classification task provides a rigorous performance benchmark. The metric learning evaluation highlights the flexibility of our approach and shows the generality of KE beyond mainstream literature that requires class logits.

We benchmark KE using both the cross-entropy and the triplet loss. We use these loss functions because most supervised tasks employ one of them.

Cross-Entropy (CE) Loss: We denote $x \in X$ as an input and $y \in Y = \{1, ..., \mathbb{C}\}$ as its ground-truth label. For a classification network N, CE is defined as follows

$$\text{CE}_{(x,y)} = -\log \frac{\exp\left(N(x;y)\right)}{\sum_{i=1}^{\mathbb{C}} \exp\left(N(x;i)\right)}, \tag{2}$$

where $N(x;y)$ denotes the output logit for class y given x.

Triplet Loss: A metric learning network learns an embedding where samples from the same class are close together, while samples from different classes are far apart. To train a metric learning network, we leverage triplet loss for its simplicity and efficiency. Triplet loss is defined as follows

$$\text{TL}_{(a,p,n)\in T} = \left[(D_{a,p} - D_{a,n} + m)\right]_+, \tag{3}$$

where $[\bullet]_+ = \max(0, \bullet)$, m is the margin between classes. $D_{x_1,x_2} = D(N(x_1), N(x_2))$; $N(\bullet)$ and $D(,)$ are the network's output-embedding and Euclidean distance, respectively. In Eq. 3, a, p, and n are the anchor, positive, and negative images in a triplet (a, p, n) from the triplets set T.

Table 1. Statistics of five classification datasets and their corresponding train, validation, and test splits.

	$\mathbb{C}$	Trn	Val	Tst	Total
Flower-102 [36]	102	1020	1020	6149	8189
CUB-200 [52]	200	5994	N/A	5794	11788
Aircraft [33]	100	3334	3333	3333	10000
MIT67 [41]	67	5360	N/A	1340	6700
Stanford-Dogs [24]	120	12000	N/A	8580	20580

The performance of triplet loss relies heavily on the sampling strategy. Since we train randomly initialized networks, we leverage the semi-hard sampling strategy for its stability [43, 49]. In semi-hard negative sampling, instead of picking the hardest positive-negative samples, all anchor-positive pairs and their corresponding semi-hard negatives are considered. Semi-hard negatives are further away from the anchor than the positive exemplar yet within the banned margin m as shown in Fig. 6. Semi-hard negatives (n) satisfy Eq. 4

$$D_{a,p} < D_{a,n} < D_{a,p} + m. \tag{4}$$

4. Experiments

In this section, we evaluate KE using classification and metric learning tasks.

4.1. Knowledge Evolution on Classification

Datasets: We evaluate KE using five datasets: Flower-102 [36], CUB-200-2011 [52], FGVC-Aircraft [33], MIT67 [41], and Stanford-Dogs [24]. Table 1 summarizes the datasets' statistics.

Technical Details: We evaluate KE using two architectures: ResNet18 [19, 20] and DenseNet169 [22]. These architectures demonstrate the efficiency of KE on modern architectures. All networks are initialized randomly and optimized by stochastic gradient descent (SGD) with momentum 0.9 and weight decay 1e-4. We use cosine learning rate decay [30] with an initial learning rate $lr = 0.256$. We use batch size $b = 32$ and train N for $e = 200$ epochs. We use the standard data augmentation technique, *i.e.*, flipping and random cropping. For simplicity, we use the same training settings (lr, b, e) for all generations. We report the network accuracy at the last training epoch, *i.e.*, no early stopping.

Baselines: We benchmark KE using the cross-entropy (CE), label-smoothing (Smth) regularizer [35, 48], RePr [39], CS-KD [58], AdaCos [60], Dense-Sparse-Dense (DSD) [14], and Born Again Networks (BANs) [9] introduced in Sec. 2:

- **DSD** determines the duration of each training phase (# epochs) using the loss-convergence criterion. For small datasets, the loss converges rapidly to zero and

Table 2. Quantitative classification evaluation (Top-1 ↑) using ResNet18 with KELS. N_g denotes the performance of the g^{th} network generation. The first generation N_1 is both a baseline and a starting point for KE. As the number of generations increases, KE boosts performance.

Method	Flower	CUB	Aircraft	MIT	Dog
CE + AdaCos	**55.45**	**62.48**	**57.06**	56.25	**65.34**
CE + RePr	41.90	42.88	39.43	46.94	50.39
CE + DSD	51.39	53.00	57.24	53.21	63.58
CE + BANs-N_{10}	48.53	53.71	53.19	55.65	64.16
CE (N_1)	48.48	53.57	51.28	55.28	63.83
CE + KE-N_3 **(ours)**	52.53	56.73	52.53	57.44	64.28
CE + KE-N_{10} **(ours)**	56.15	58.11	53.21	**58.33**	64.56
Smth (N_1)	50.97	59.75	55.00	57.74	65.95
Smth + KE-N_3 **(ours)**	56.87	62.88	57.47	58.78	66.91
Smth + KE-N_{10} **(ours)**	**62.56**	**66.85**	**60.03**	**60.42**	**67.06**
CS-KD (N_1)	55.10	67.71	58.15	57.37	69.60
CS-KD + KE-N_3 **(ours)**	61.74	71.63	**59.97**	**58.41**	70.62
CS-KD + KE-N_{10} **(ours)**	**69.88**	**73.39**	59.08	57.96	**70.81**

some datasets do not have validation splits (see Table 1). So, we use $e = 200$, $e = 100$, and $e = 100$ epochs for the dense, sparse, dense phases, respectively. We prune each layer to the default 30% sparsity.

- **AdaCos** maximizes the inter-class angular margin by dynamically scaling the cosine similarities between training samples and their corresponding class center. Thus, AdaCos is a hyperparameter-free feature embedding regularizer.
- **CS-KD** is a knowledge distillation inspired approach that achieves state-of-the-art performance on small datasets. It distills the logits distribution between different samples from the same class. Thus, it mitigates overconfident predictions and reduces intra-class variations. We set CS-KD's hyperparameters $T = 4$ and $\lambda_{\text{cls}} = 3$ in all experiments.
- **RePr** is similar to DSD, but instead of pruning weights, RePr prunes *redundant* convolutional filters. Prakash *et al.* [39] recommend repeating the dense-sparse-dense phases three times. Since we train N for $e = 200$ epochs, we set RePr's hyperparameters $S1 = 50$ and $S2 = 10$. We use the default sparsity rate (prune rate) $p = 30\%$.

Results: Tables 2 and 3 present quantitative classification evaluation using ResNet18 and DenseNet169, respectively. For ResNet18, we use a split-rate $s_r = 0.8$ and KELS, *i.e.*, $\approx 36\%$ sparsity. For DenseNet169, we use $s_r = 0.7$ and WELS, *i.e.*, 30% sparsity. We report the performance of the dense network N because all baselines learn dense networks. In Sec. 5, we report the slim fit-hypothesis $H^{\triangle}$ performance and inference cost. Tables 2 and 3 present the performance of the first generation (N_1) as a baseline, the third generation (N_3) as the short-term benefit, and the tenth-generation (N_{10}) as the long-term benefit of KE.

A deeper network achieves higher accuracy when presented with enough training data. However, if the training data is scarce, a deeper network becomes vulnerable to overfitting. This explains why regularization techniques (*e.g.*, AdaCos) deliver competitive performance on the small ResNet18, but degrade on the large DenseNet169. Interestingly, KE remains resilient on the large DenseNet169 and delivers similar, if not superior, performance.

Table 3. Quantitative evaluation using DenseNet169 with WELS.

Method	Flower	CUB	Aircraft	MIT	Dog
CE + AdaCos	49.96	**62.20**	56.15	50.89	65.33
CE + RePr	39.75	47.01	36.04	49.77	55.63
CE + DSD	48.85	56.11	53.66	58.31	65.76
CE + BANs-N_{10}	44.92	57.30	52.56	57.66	65.49
CE (N_1)	45.85	55.16	51.73	56.62	64.82
CE + KE-N_3 **(ours)**	52.44	57.75	56.70	**59.67**	67.06
CE + KE-N_{10} **(ours)**	**60.15**	58.01	**59.73**	58.71	**67.75**
Smth (N_1)	46.34	59.93	57.74	57.81	65.12
Smth + KE-N_3 **(ours)**	55.46	**62.53**	62.86	**60.27**	**68.21**
Smth + KE-N_{10} **(ours)**	**64.18**	61.34	**65.86**	59.75	67.46
CS-KD (N_1)	46.97	67.32	58.87	56.62	69.83
CS-KD + KE-N_3 **(ours)**	59.36	69.77	59.91	**59.00**	**71.70**
CS-KD + KE-N_{10} **(ours)**	**65.27**	**70.36**	**61.22**	57.44	70.72

We applied KE on top of (1) the cross-entropy loss, (2) the label smoothing (Smth) regularizer with its hyperparameter [35] $\alpha = 0.1$, and (3) the CS-KD regularizer. KE is flexible and boosts performance on each baseline. N_3 outperforms N_1 on all datasets. After reaching a peak, KE's performance fluctuates. Thus, if N_3 outperforms N_{10} marginally, this indicates that KE reached its peak. In Fig. 1, KE reached its peak on CUB-200 after 20 generations, then KE fluctuates for 80 generations without degrading.

Even though RePr seems similar to KE, the following caveat explains RePr's inferior performance. RePr ranks the *redundant* filters across the entire network, *i.e.*, no per-layer ranking. Prakash *et al.* [39] report pruning more filters from deeper layers when training on large datasets. Yet, RePr prunes many filters from earlier layers when training on small datasets. The earlier layers get a small gradient compared to deeper layers; and with small datasets, the earlier filters remain close to their initialization, *i.e.*, no significant difference between earlier filters. Pruning earlier filters cripples the optimization process and achieves an inferior performance.

Another important difference between KE and RePr is how filters are re-initialized. KE re-initializes the reset-hypothesis randomly. Thus, KE makes no assumptions about the network architecture. In contrast, RePr is designed specifically for CNNs. RePr re-initializes the pruned filters to be orthogonal to both their values before being dropped and the current value of non-pruned filters. RePr uses the QR decomposition on the weights of the filters from the same layer to find the null-space, that is used to find an orthogonal initialization point. Basically, RePr stores the pruned filters to use them for re-initialization. This makes RePr more complex compared to KE. In the paper appendix, we highlight other differences.

Table 4. Quantitative retrieval evaluation using standard metric learning datasets and architectures.

	ResNet50			GoogLeNet		
Datasets	NMI	R@1	R@4	NMI	R@1	R@4
CUB (N_1)	0.396	13.01	30.37	0.396	10.16	25.71
CUB + KE-N_3 **(ours)**	0.424	17.22	36.14	0.418	13.94	33.78
CUB + KE-N_{10} **(ours)**	**0.429**	**18.25**	**39.40**	**0.419**	**15.34**	**34.30**
Cars (N_1)	0.374	11.63	28.66	0.319	5.29	17.94
Cars + KE-N_3 **(ours)**	0.514	34.28	60.25	0.476	24.98	50.06
Cars + KE-N_{10} **(ours)**	**0.523**	**42.36**	**68.11**	**0.495**	**32.63**	**58.84**

Similar to KE, The BANs training approach trains a network for multiple generations. However, BANs transfers knowledge through the class-logits distribution. For small datasets, a teacher's logits distribution resembles the ground-truth labels (one-hot vector) when the loss converges to zero. Thus, BANs achieves regular cross-entropy performance even after training for 10 generations.

4.2. Knowledge Evolution on Metric Learning

Datasets: We evaluate KE using two standard metric learning datasets: CUB-200-2011 [52], Stanford Cars196 [25].
Evaluation Metrics: For quantitative evaluation, we use the Recall@K metric and Normalized Mutual Info (NMI) on the test split.
Technical Details: We use the same hyperparameters (e, lr scheduler) and optimizer used in the classification experiments. However, the feature embedding $\in R^{d=128}$ is normalized to the unit circle and we use a batch size $b = 125$. Each mini-batch contains 25 different classes and 5 samples per class. We use a small learning rate $lr = 0.0256$ to avoid large fluctuations in the feature embedding during training.
Results: Table 4 presents a quantitative retrieval evaluation using two standard metric learning architectures: ResNet50 [19, 20] and GoogLeNet [47]. We use a split-rate $s_r = 0.8$ and KELS with both architectures (See the paper appendix on how KELS handles concatenation operations inside GoogLeNet). As the number of generations increases, the retrieval performance of the dense network increases. Through this experiment, we highlight how KE supports a large spectrum of network architectures and loss functions. Equipped with WELS, we expect KE to spread beyond CNNs. It is straight forward to tweak WELS and impose a regular sparsity, as in KELS, but for non CNNs.

5. Ablation Study

This section presents three ablation studies: We (1) validate the dropout and Res-Net intuitions (from Sec. 3.3), (2) compare WELS and KELS techniques, (3) present the tradeoffs of the split-rate s_r.
(1) Dropout and Res-Net intuitions' validation
To validate the dropout and Res-Net intuitions, we monitor the fit and reset hypotheses across generations. According

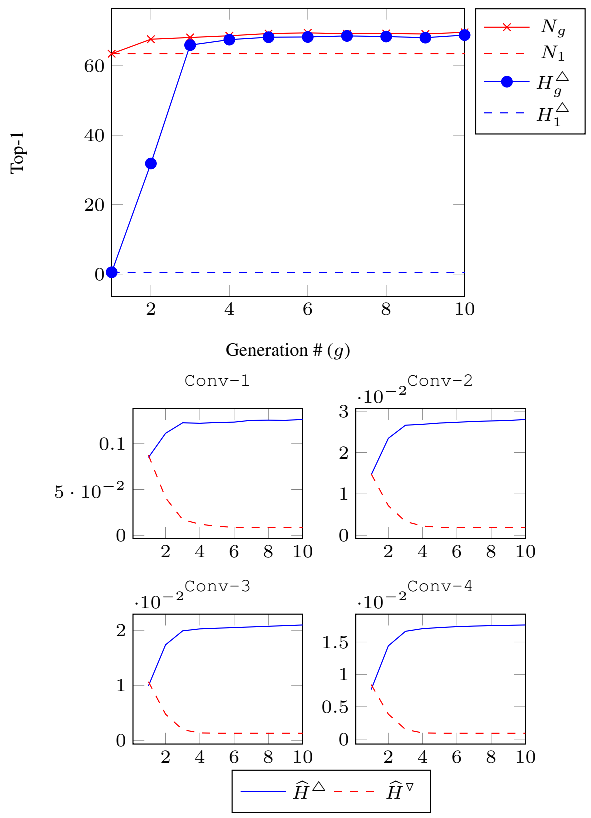


Figure 7. Quantitative classification evaluation using CUB-200 on VGG11_bn. The x-axis denotes the number of generations. The fit-hypothesis $H^{\triangle}$ achieves an inferior performance at $g = 1$, but its performance increases as the number of generations increases. $\widehat{H}^{\triangle}$ and $\widehat{H}^{\triangledown}$ denote the mean absolute value inside $H^{\triangle}$ and $H^{\triangledown}$.

to the dropout intuition, the fit-hypothesis should learn an independent representation. The KELS technique enables measuring the fit-hypothesis's performance. In this study, we use the CUB-200 dataset, VGG11_bn [44], and a split-rate $s_r = 0.5$. Fig. 7 (Top) shows the performance of the dense network N and the slim fit-hypothesis $H^{\triangle}$ for 10 generations. The horizontal dashed lines denote the performance of the first generation (N_1 and $H_1^{\triangle}$). At the first generation, the fit-hypothesis's performance is inferior. Yet, as the number of generations increases, the fit-hypothesis performance increases. Table 5 (Top section) presents both the performance and inference cost of both N and $H^{\triangle}$.

According to the Res-Net intuition, the reset-hypothesis should converge to a zero-mapping because, after the first generation (N_1), the fit-hypothesis is always closer to convergence. Fig. 7 shows the mean absolute values ($\widehat{H}^{\triangle}$ and $\widehat{H}^{\triangledown}$) inside the fit and reset hypotheses. We present these values inside the first four convolution layers of VGG11_bn (See paper appendix for all eight conv layers). $\widehat{H}_1^{\triangle}$ and $\widehat{H}_1^{\triangledown}$ are comparable at N_1. However, as the number of generations increases, $\widehat{H}^{\triangle}$ increases while $\widehat{H}^{\triangledown}$ decreases.

Table 5. Quantitative evaluation for KELS using the number of both operations (G-Ops) and parameters (millions). Acc$_g$ denotes the classification accuracy at the g^{th} generation. $\blacktriangle_{\text{ops}}$ denotes the relative reduction in the number of operations. $\blacktriangle_{\text{acc}}$ denotes the absolute accuracy improvement on top of the dense baseline N_1.

CUB on VGG11_bn							
	s_r	Acc$_1$	Acc$_{10}$	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.5	63.47	69.65	6.1%	15.22	-	259.16
$H^{\triangle}_g$		0.52	68.84	5.3%	3.85	74.7%	65.20
FLW on ResNet18							
	s_r	Acc$_1$	Acc$_{100}$	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.8	53.87	75.62	21.7%	3.63	-	22.44
$H^{\triangle}_g$		6.41	75.62	21.7%	2.39	34.1%	14.43
N_g	0.5	52.62	74.60	21.9%	3.63	-	22.44
$H^{\triangle}_g$		0.37	74.60	21.9%	0.96	73.5%	5.64
CUB on GoogLeNet							
	s_r	Acc$_1$	Acc$_{10}$	$\blacktriangle_{\text{acc}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
N_g	0.8	64.76	72.93	8.1%	3.00		11.59
$H^{\triangle}_g$		0.64	71.67	6.9%	1.98	34.0%	7.54
N_g	0.5	65.18	72.44	7.2%	3.00		11.59
$H^{\triangle}_g$		0.50	57.23	-7.9%	0.81	73.0%	3.00

(2) WELS vs KELS techniques

KE requires a network-splitting technique. WELS delivers a dense network N only. Thus, we compare WELS and KELS using N. Fig. 8 (Left) compares WELS and KELS using Flower-102, cross-entropy with the CS-KD regularizer [58], ResNet18, and two split-rates ($s_r = \{0.5, 0.8\}$). KELS and WELS achieve comparable performance. This is promising because WELS can be applied to any neural network. Fig. 8 (Right) re-assures that KELS delivers high performance while reducing inference cost as shown in Table 5 (middle section). The performance of $H^{\triangle}_{100}$ matches N_{100} because $H^{\triangle}$ has enough capacity for the small Flower-102. With $s_r = 0.5$, KELS achieves an absolute 21% improvement margin while reducing inference cost by 73%.

(3) The split-rate s_r tradeoffs

The split-rate s_r controls the size of the fit-hypothesis; a small s_r reduces the inference cost. Yet, a small s_r reduces the capacity of $H^{\triangle}$. Fig. 9 (Left) compares two split-rate ($s_r = \{0.5, 0.8\}$) using CUB-200 and GoogLeNet for 10 generations. Both split-rates achieve significant improvement margins on the dense network N. However, Fig. 9 (Right) shows that the large split-rate $s_r = 0.8$ helps the fit-hypothesis $H^{\triangle}$ to converge faster and to achieve better performance. Table 5 (third section) highlights this performance and inference-cost tradeoff. For a large dataset, a large split-rate is required to deliver a slim fit-hypothesis $H^{\triangle}$ with competitive performance.

5.1. Discussion

ImageNet [6] will eventually become a toy dataset given the increasing size of deep networks [4, 46, 32, 1] (*e.g.*, GPT-3). To train these large networks, unsupervised [23, 2] and self-supervised [53, 37, 17, 50] learning mitigate the burden of data annotation. However, these learning approaches still require storing and maintaining a large corpus of data. This is (1) expensive even if neither labeling nor curating is required, (2) impractical for applications with privacy concerns like medical imaging. KE tackles the problem of training deep networks on relatively small datasets. KE's main limitation is the training time. It takes ≈ 8 hours to train 100 generations, 200 epochs each, on Flower-102 using GTX1080Ti GPU. This long training time can be reduced by monitoring the performance on a validation split.

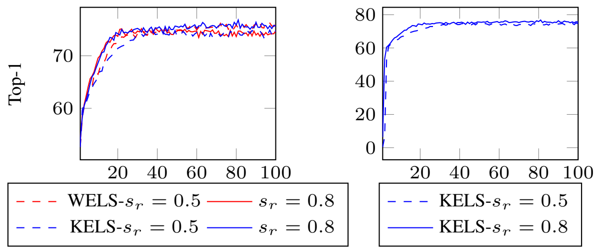


Figure 8. Quantitative evaluation for both KELS and WELS using Flower-102 on ResNet18 for 100 generations. The x and y axes denote the number of generations and the top-1 accuracy, respectively. (Left) The classification performance of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

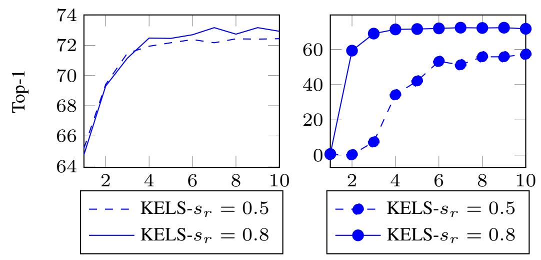


Figure 9. Quantitative evaluation for different split-rates using CUB-200 on GoogLeNet for 10 generations. (Left) The classification performance of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

6. Conclusion

We have proposed knowledge evolution (KE) to train deep networks on relatively small datasets. KE picks a random subnetwork (fit-hypothesis), with inferior performance, and evolves its knowledge. We have equipped KE with a kernel-level convolution-aware splitting (KELS) technique to learn a slim network automatically while training a dense network. Through KELS, KE reduces the inference cost while boosting performance. Through the weight-level splitting (WELS) technique, KE supports a large spectrum of architectures. We evaluated KE using classification and metric learning tasks. KE achieves SOTA results.

Acknowledgments: This work was partially funded by independent grants from Facebook AI and DARPA SAIL-ON program (W911NF2020009).

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[3] Binghui Chen, Weihong Deng, and Haifeng Shen. Virtual class enhanced discriminative embedding learning. In *NeurIPS*, 2018.

[4] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *ICML*, 2013.

[5] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] Abhimanyu Dubey, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Maximum-entropy fine grained classification. In *NeurIPS*, 2018.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

[10] Sharath Girish, Shishira R Maiya, Kamal Gupta, Hao Chen, Larry Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. *arXiv preprint arXiv:2012.04643*, 2020.

[11] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

[12] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 2016.

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

[16] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*, 1993.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[21] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[23] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum. Mining on manifolds: Metric learning without labels. In *CVPR*, 2018.

[24] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, 2011.

[25] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV workshops*, 2013.

[26] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *NeurIPS*, 1990.

[27] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[28] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[29] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[30] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[31] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[32] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018.

[33] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[34] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *NeurIPS*, 2019.

[35] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *NeurIPS*, 2019.

[36] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[37] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[38] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

[39] Aaditya Prakash, James Storer, Dinei Florencio, and Cha Zhang. Repr: Improved training of convolutional filters. In *CVPR*, 2019.

[40] Siyuan Qiao, Zhe Lin, Jianming Zhang, and Alan L Yuille. Neural rejuvenation: Improving deep network training by enhancing computational resource utilization. In *CVPR*, 2019.

[41] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *CVPR*, 2009.

[42] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *CVPR*, 2020.

[43] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[46] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[48] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[49] Ahmed Taha, Yi-Ting Chen, Teruhisa Misu, Abhinav Shrivastava, and Larry Davis. Boosting standard classification architectures through a ranking regularizer. In *WACV*, 2020.

[50] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[51] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

[52] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[53] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

[54] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*, 2016.

[55] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018.

[56] Ting-Bing Xu and Cheng-Lin Liu. Data-distortion guided self-distillation for deep neural networks. In *AAAI*, 2019.

[57] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[58] Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In *CVPR*, 2020.

[59] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *ICCV*, 2019.

[60] Xiao Zhang, Rui Zhao, Yu Qiao, Xiaogang Wang, and Hongsheng Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. In *CVPR*, 2019.

[61] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *ECCV*, 2016.

[62] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*, 2019.

[63] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.