Knowledge Evolution in Neural Networks
Appendix

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

A. Appendix

The following appendix-sections extend their corresponding sections in the paper manuscript. For instance, the appendix related-work A extends the related-work section in the paper manuscript.

A. Appendix: Related Work

The proposed kernel-level convolution-aware splitting (KELS) technique enables the knowledge evolution (KE) approach to learn a slim network with a small inference cost. This signals KE+KELS as a pruning approach. In this section, we compare KE+KELS with the pruning literature. We categorize the pruning approaches by their pruning-granularity: weights *vs*. channels *vs*. filters.

Weight-pruning [12, 7, 6, 4]: These approaches prune network weights with small absolute magnitude (less salient [12]). Weight-pruning reduces the network size, which in turn reduces both DRAM access and energy consumption on mobile devices [6]. However, weight-pruning does not reduce the computational costs due to the irregular sparsity after pruning. Accordingly, a weight-pruned network requires sparse BLAS libraries or specialized hardware [3]. WELS can be regarded as a weight-pruning technique. However, WELS can be tweaked to reduce both the network size and the computational cost. For instance, we tweaked WELS to propose KELS for CNNs. For a fully connected network (FCN), WELS can split the weights into two independent halves with regular sparsity. With a regular sparsity, KE delivers a slim, not sparse, FCN.

Channel-pruning [14, 18, 10]: Given the limitation of weight-pruning and the complexity of filter-pruning, channel-pruning provides a nice tradeoff between flexibility and ease of implementation. Yet, channel pruning approaches make assumptions. For instance, Liu *et al*. [14] require a scaling layer or a batch norm layer; Huang *et al*. [10] require group convolution support [11]. Accordingly, these [14, 10] are CNN-specific approaches. Furthermore, some channel-pruning approaches (*e.g*., [18]) are applied after training a network. Thus, they do not introduce any performance improvements.

Filter-pruning [13, 22, 15]: KELS belongs to the filter-pruning category. It is easy to identify unimportant filters, Li *et al*. [13] quantify filters' importance using L1-Norm. By removing – or splitting – unimportant filters, filter-pruning reduces both the computational cost and the number of parameters. Thus, a filter-pruned network needs neither sparse BLAS libraries nor specialized hardware. These advantages make filter-pruning appealing. Unfortunately, it is challenging to remove the unimportant filters while maintaining valid network connectivity. For instance, Li *et al*. [13] apply filter-pruning on vanilla CNNs (*e.g*., VGG), but require projection-shortcuts to support Res-Nets, and require further modification to support concatenation operations (*e.g*., GoogLeNet). Similarly, ThiNet [15] suffers on Res-Nets and does not prune the last convolutional layer in all residual blocks. In contrast, KELS supports both vanilla and residual CNNs without bells and whistles.

KE+KELS removes – or splits – entire filters. This saves both the number of operations (FLOPs) and parameters (memory). KELS imposes no constraints on the CNN architecture or the loss function. These are key advantages, but KE+KELS has limitations. For instance, KE re-trains a neural network for a large number of generations. This large training cost is not a hurdle for our paper because we tackle the following question: how to train a deep network on a relatively small dataset?

B. Appendix: Approach

The kernel-level convolutional-aware splitting (KELS) technique supports both vanilla and residual networks. However, KELS requires a simple modification to support the concatenation operations (concat-op) in GoogLeNet and DenseNet. Figures 1 and 2 illustrate how to handle concatenation in these networks. The main difference between Fig. 1 and 2 is whether the concat-op is followed by a convolution or a batch-norm. To handle both variants, we keep references to the preceding convolutional filters (*e.g*., F_1 and F_2 in Fig. 1). Using these references, we outline the fit-hypothesis in the convolutional and batch-norm layers. In this way, we split the network properly and make sure the fit-hypothesis is a slim independent network.

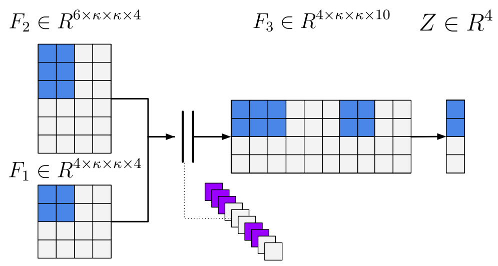
$F_2 \in R^{6\times\kappa\times\kappa\times4}$
$F_3 \in R^{4\times\kappa\times\kappa\times10}$
$Z \in R^4$
$F_1 \in R^{4\times\kappa\times\kappa\times4}$
$\|$

Figure 1. A Split-Net illustration on a toy feature concatenation operation using $s_r = 0.5$. $\|$ denotes a feature concatenation operation. The dotted line shows the dimension of the feature map after concatenation *i.e.*, it is not part of the network. In this example, feature concatenation is followed by a convolutional layer F_3 then a batch norm layer. This order of operations is employed in *both* GoogLeNet and DenseNet. To split F_3 properly (initialize its split-mask M_3), we keep references to the preceding convolutional filters (F_1 and F_2). Through these references, we determine for F_3 whether an input channel belongs to the fit-hypothesis or not.

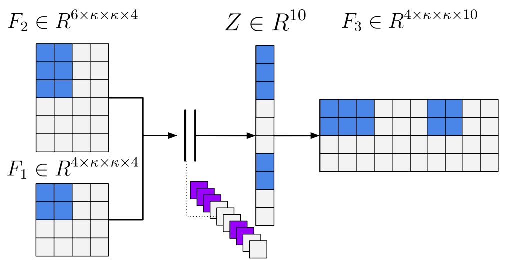
$F_2 \in R^{6\times\kappa\times\kappa\times4}$
$Z \in R^{10}$
$F_3 \in R^{4\times\kappa\times\kappa\times10}$
$F_1 \in R^{4\times\kappa\times\kappa\times4}$
$\|$

Figure 2. A Split-Net illustration on a toy feature concatenation operation. In this example, feature concatenation is followed by a batch norm layer then the convolutional layer F_3. This order of operations is employed in DenseNet.

Appendix Intuition #1: Dropout

In the paper, we have illustrated how Split-Nets resemble dropout, *i.e.*, both encourage neurons (subnetwork) to learn an independent representation. However, Split-Nets target a specific set of neurons (subnetwork). For instance, if a toy network layer has 10 neurons, dropout promotes an independent representation to all 10 neurons. In contrast, Split-Nets promote an independent representation to the neurons inside the fit-hypothesis $H^{\triangle}$ only. Thus, the split-mask M provides a finer level of control.

After highlighting the resemblance between KE and dropout, we want to emphasize that extending dropout for CNNs (channel-dropout) *seems* trivial, but it is not. Channel-dropout has been challenging because features in deep layers have great specificity [20, 17]. For an input image, a small fraction of channels is activated [21]. Thus, it is important to treat channels *unequally*, *i.e.*, uniform random dropping is deficient. Consequently, Hou and Wang [9] have proposed Weighted Channel Dropout (WCD). This approach adds three extra modules to a deep network: Global Average Pooling, Weighted Random Selection, and Random Number Generator. These three modules are added to multiple convolutional layers.

Similar to KE, WCD [9] is applied during training. However, WCD does not reduce the inference cost. In addition, Hou and Wang [9] apply WCD to certain –not all – convolutional layers (*e.g.*, *res5a* and *res5c* in ResNet-101). Thus, WCD requires tuning per architecture.

Appendix Intuition #2: Residual Network

It is challenging to train a deep network on a small dataset. This challenge stems from the large number of parameters in a deep network. While all parameters are required for a large dataset, they become redundant and enable overfitting on a small dataset. To mitigate overfitting, weight regularizers (*e.g.*, weight-decay) have been proposed. These regularizers reduce the network's complexity by suppressing the weights' magnitudes, *i.e.*, promote a *zero-mapping*.

A Res-Net splits a network into two branches: an identity shortcut and residual subnetwork. This network-splitting enables a *zero-mapping* in residual links since a default identity mapping already exists. From this perspective, Res-Nets resemble weight-decay in terms of favoring a simpler subnetwork (*e.g.*, $R(x) = 0; \forall x$). Yet, one difference is that a Res-Net can suppress the residual subnetworks while keeping the network's depth intact.

Similar to Res-Nets, a Split-Net splits a network into two branches: the fit-hypothesis $H^{\triangle}$ and the reset-hypothesis $H^{\triangledown}$. Split-Nets promote a zero mapping inside $H^{\triangledown}$ because, after the first generation, $H^{\triangle}$ is always closer to convergence. A zero-mapping inside $H^{\triangledown}$ reduces the number of active parameters, which in turn mitigates overfitting and reduces the burden for data collection. If all weights inside $H^{\triangledown}$ converge to zero, the network's depth remains intact, thanks to the fit-hypothesis $H^{\triangle}$.

C. Appendix: Experiments

C.1. Knowledge Evolution on Classification

We have used public implementations for our baselines: RePr[1], BANs[2], AdaCos[3], and CS-KD[4]. We leverage a public implementation[5] to profile the fit-hypothesis computational cost.

In the paper manuscript, Fig. 2 illustrates the KELS technique on a toy Res-Net. Table 1 uses the ResNet18 architecture and a split-rate $s_r = 0.5$ to present (1) the dimensions of both the dense network N and the slim fit-hypothesis

[1]https://github.com/siahuat0727/RePr
[2]https://github.com/nocotan/born_again_neuralnet
[3]https://github.com/4uiiurz1/pytorch-adacos
[4]https://github.com/alinlab/cs-kd
[5]https://github.com/mitchellnw/micro-net-dnw/blob/master/image_classification/model_profiling.py

Table 1. The dimensions of the ResNet18 N versus its fit-hypothesis $H^{\triangle}$ with split-rate $s_r = 0.5$. The last table-section compares N and $H^{\triangle}$ through the number of operations and parameters (millions). The fit-hypothesis $H^{\triangle}$ is a slim independent network with 102 logits (Flower-102).

Layers	ResNet18 N	Fit-hypothesis $H^{\triangle}$
conv1	$64 \times 7 \times 7 \times 3$	$32 \times 7 \times 7 \times 3$
bn1	64	32
layer1.0.conv1	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.0.bn1	64	32
layer1.0.conv2	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.0.bn2	64	32
layer1.1.conv1	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.1.bn1	64	32
layer1.1.conv2	$64 \times 3 \times 3 \times 64$	$32 \times 3 \times 3 \times 32$
layer1.1.bn2	64	32
layer2.0.conv1	$128 \times 3 \times 3 \times 64$	$64 \times 3 \times 3 \times 32$
layer2.0.bn1	128	64
layer2.0.conv2	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.0.bn2	128	64
layer2.0.downsample.0	$128 \times 1 \times 1 \times 64$	$64 \times 1 \times 1 \times 32$
layer2.0.downsample.1	128	64
layer2.1.conv1	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.1.bn1	128	64
layer2.1.conv2	$128 \times 3 \times 3 \times 128$	$64 \times 3 \times 3 \times 64$
layer2.1.bn2	128	64
layer3.0.conv1	$256 \times 3 \times 3 \times 128$	$128 \times 3 \times 3 \times 64$
layer3.0.bn1	256	128
layer3.0.conv2	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.0.bn2	256	128
layer3.0.downsample.0	$256 \times 1 \times 1 \times 128$	$128 \times 1 \times 1 \times 64$
layer3.0.downsample.1	256	128
layer3.1.conv1	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.1.bn1	256	128
layer3.1.conv2	$256 \times 3 \times 3 \times 256$	$128 \times 3 \times 3 \times 128$
layer3.1.bn2	256	128
layer4.0.conv1	$512 \times 3 \times 3 \times 256$	$256 \times 3 \times 3 \times 128$
layer4.0.bn1	512	256
layer4.0.conv2	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.0.bn2	512	256
layer4.0.downsample.0	$512 \times 1 \times 1 \times 256$	$256 \times 1 \times 1 \times 128$
layer4.0.downsample.1	512	256
layer4.1.conv1	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.1.bn1	512	256
layer4.1.conv2	$512 \times 3 \times 3 \times 512$	$256 \times 3 \times 3 \times 256$
layer4.1.bn2	512	256
fc	102×512	102×256
#Ops (G-Ops)	3.63	0.96
#Parameters	22.44	5.64

$H^{\triangle}$; (2) the computational cost of both N and $H^{\triangle}$. The paper manuscript evaluates KE on DenseNet169 using the WELS technique and a split-rate $s_r = 0.7$. Tables 2 and 3 present quantitative classification evaluations on DenseNet169 using KELS and WELS, respectively. Both WELS and KELS evaluations use $s_r = 0.8$.

In the paper manuscript, all experiments employ randomly initialized networks. Yet, pretrained networks achieve better performance on relatively small datasets. Table 4 highlights the performance gap between randomly initialized (CS-KD+KE) and ImageNet initialized (CE+ImageNet) networks. The CE+ImageNet baseline provides an upper bound. The CS-KD+KE baseline use KELS and $s_r = 0.8$ with ResNet18, and WELS and $s_r = 0.7$ with DenseNet169, *i.e.*, last rows in Tables 2 and 3. KE closes the performance gap between randomly initialized and ImageNet initialized networks significantly.

Table 2. Quantitative evaluation using DenseNet169 with KELS and $s_r = 0.8$, *i.e.*, $\approx 36\%$ sparsity.

Method	Flower	CUB	Aircraft	MIT	Dog
CE (N_1)	45.76	55.49	51.96	57.37	65.09
CE + KE-N_3 **(ours)**	50.50	57.73	56.34	60.64	66.08
CE + KE-N_{10} **(ours)**	**58.78**	**58.96**	**61.70**	**61.76**	**67.30**
Smth (N_1)	45.85	59.01	58.45	57.07	66.31
Smth + KE-N_3 **(ours)**	53.69	**62.38**	63.18	**59.52**	68.00
Smth + KE-N_{10} **(ours)**	**65.88**	60.57	**65.60**	59.15	**68.66**
CS-KD (N_1)	49.32	66.71	57.62	56.77	68.82
CS-KD + KE-N_3 **(ours)**	59.67	**69.63**	59.43	57.14	**70.66**
CS-KD + KE-N_{10} **(ours)**	**66.34**	69.35	**59.76**	**57.37**	70.59

Table 3. Quantitative evaluation using DenseNet169 with WELS and split-rate $s_r = 0.8$, *i.e.*, 20% sparsity.

Method	Flower	CUB	Aircraft	MIT	Dog
CE (N_1)	44.88	56.32	51.61	55.13	66.15
CE + KE-N_3 **(ours)**	50.23	**59.81**	56.25	**60.27**	66.44
CE + KE-N_{10} **(ours)**	**58.03**	59.38	**60.80**	59.45	**67.25**
Smth (N_1)	45.92	58.70	56.73	58.26	66.48
Smth + KE-N_3 **(ours)**	54.84	**62.41**	62.68	60.49	67.98
Smth + KE-N_{10} **(ours)**	**64.69**	60.36	**65.62**	**62.13**	**68.26**
CS-KD (N_1)	46.75	66.66	58.87	56.85	69.22
CS-KD + KE-N_3 **(ours)**	58.27	69.67	60.98	**57.51**	70.94
CS-KD + KE-N_{10} **(ours)**	**64.18**	**71.37**	**61.37**	57.22	**71.33**

Table 4. Comparative evaluation between pretrained (CE + ImageNet) and randomly initialized (CS-KD + KE) networks. The performance of CE + ImageNet provides an upper-bound for KE.

Method	Flower	CUB	Aircraft	MIT	Dog
	ResNet18				
CE + ImageNet	**88.83**	**74.46**	**61.01**	**72.84**	**74.29**
CS-KD + KE-N_{10}	69.88	73.39	59.08	57.96	70.81
	DenseNet169				
CE + ImageNet	**93.46**	**80.73**	**69.85**	**77.90**	**79.92**
CS-KD + KE-N_{10}	65.27	70.36	61.22	57.44	70.72

KE vs RePr

In the paper manuscript, we highlight two differences between KE and RePr. Yet, there are other worth noting differences. (I) RePr delivers a dense network only. (II) RePr's re-initialization step (QR decomposition) is computationally expensive. (III) During training, RePr prunes a different set of filters at different stages. If the pruned filters are regarded as a reset-hypothesis, then RePr changes the reset-hypothesis at different training stages. In contrast, KE outlines both fit and reset hypotheses using a single split-mask. This mask remains the same across all generations.

KE vs DSD

DSD is a prominent training approach. Han *et al.* [5] evaluated DSD using various tasks: image classification, caption generation, and speech recognition. Surprisingly, the DSD's intuition is never discussed in its paper [5].

We claim that DSD is a special case of KE. To support

Algorithm 2: Workflow of DSD from [5]. The $\lambda = S[k]$ denotes the k-th largest weight where $k = |W| * (1 - sparsity)$, and $|W|$ is the number of weights inside a network.

```
Result: $W^{(t)}$
1  $W^{(0)} \sim N(0, \Sigma)$;   // Randomly initialize $W^{(0)}$
2  while not converged do // Dense Phase
3      $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{t-1})$;
4      $t = t + 1$;
5  end
   // Sparse Phase
6  $S = sort(abs(W^{(t-1)}))$ ;   // descendingly
7  $\lambda = S[k]$;
8  Mask $= \mathbb{1}(abs(W^{(t-1)}) > \lambda)$;
9  while not converged do
10     $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{t-1})$;
11     $W^{(t)} = W^{(t)}$Mask;
12     $t = t + 1$;
13 end
14 while not converged do // Dense Phase
15     $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{t-1})$;
16     $t = t + 1$;
17 end
```

this claim, we first summarize the DSD training approach in Algorithm 2. In this algorithm, we focus on two steps: Step #8 and Step #11. In Step #8, DSD outlines the less important weights to be pruned using the binary variable $Mask$. This step is similar to our network-splitting step that outlines the fit and reset hypotheses through WELS. However, WELS splits a network N randomly while DSD splits N using a weight magnitude threshold.

Step #11 re-initializes the pruned weights to zero. Again, this step is similar to our reset-hypothesis re-initialization step. However, there are two differences. (1) We re-initialize the reset-hypothesis randomly instead of zero-values. If the re-initialization step is regarded as a noise injection process, then DSD injects noise with a zero standard deviation. In contrast, KE injects noise with a non-zero standard deviation. This difference is important because the DSD's noise (zero-values) is bad for KELS. KELS re-initializes entire filters in the reset-hypothesis, *i.e.*, a zero filter is an inferior initialization. (2) KE injects noise efficiently, *i.e.*, across generations only. In contrast, DSD executes Step #11 for every training mini-batch. Concretely, if we train a network on a dataset of size B, the re-initialization cost is $O(g \times L)$ for KE, and $O(g \times e \times \frac{B}{b} \times L)$ for DSD, where g is the number of generations, e is the number of epochs, L is the number of layers, and b is the mini-batch size. The vanilla DSD assumes $g = 1$, but this

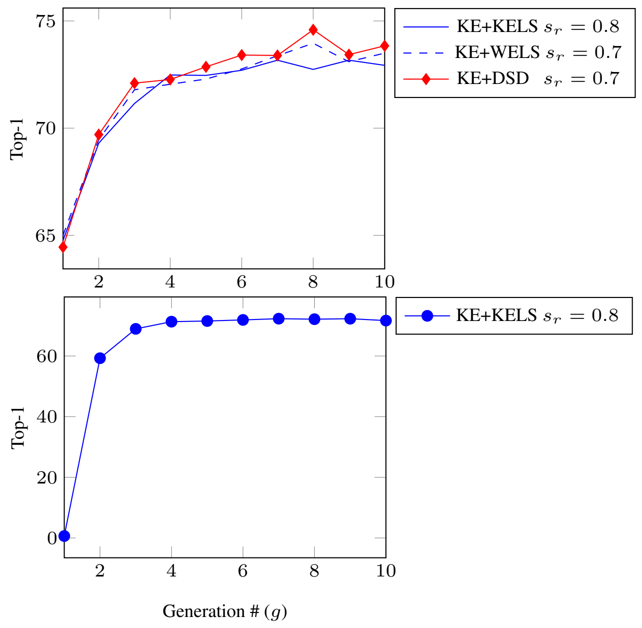

Figure 3. Quantitative comparison between KE and KE+DSD. (Top) The classification performance of the dense network N. (Bottom) The performance of the slim fit-hypothesis $H^{\triangle}$. Through KELS, $H^{\triangle}$ achieves 71.67% top-1 accuracy at $g = 10$.

is an inferior setting as we show next.

To highlight the similarity between KE and DSD quantitatively, we modify the vanilla DSD training approach. We keep the masking and re-initialization steps (Steps #8 and #11), but change the phases into generations. The dense and sparse phases become the old and *even* generations, respectively. This modification means we do not *resume* the learning rate lr scheduler between phases, but *re-start* the lr scheduler across generations. Basically, we get rid of (1) the hard three-phases constraint, (2) the loss convergence criterion, and (3) the learning rate resumption across phases. We refer to this DSD variant as KE+DSD. Similar to KE, KE+DSD trains every generation for $e = 200$ epochs.

Fig. 3 compares KE with our proposed KE+DSD. We train GoogLeNet for $g = 10$ generations on CUB-200. We evaluate KE using both KELS and WELS. We use a split-rate $s_r = 0.8$ with KELS and $s_r = 0.7$ with WELS. For KE+DSD, we prune each layer to the default 30% sparsity. KE+DSD achieves comparable performance to the KE. Yet, we want to highlight one subtle difference between KE and KE+DSD. During training, KE allows all weights to change. However, KE+DSD freezes 30% of the weights to zero at the even generations – the original sparse phases – through Step #11. This form of strict regularization gives KE+DSD a marginal edge during even generations – the 8th and the 10th generations in Fig. 3.

To conclude, DSD is a special case of KE. However, one clear difference between DSD [5] and our paper is KELS.

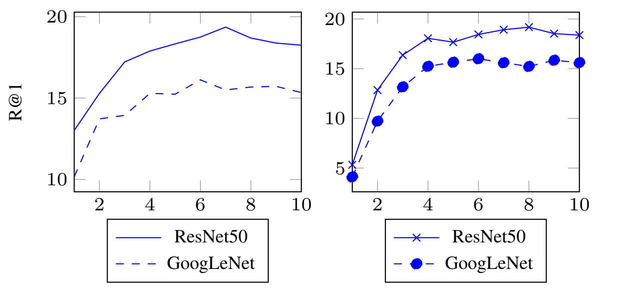

Figure 4. Quantitative retrieval evaluation using CUB-200 on both GoogLeNet and ResNet50. Both networks are trained for 10 generations. (Left) Recall@1 of the dense network N. (Right) Recall@1 of the slim fit-hypothesis $H^{\triangle}$.

Through KELS, we learn both slim and dense networks simultaneously. Having said that, the main contribution of our paper is how we present a deep network as a set of hypotheses. We introduce the idea of a fit-hypothesis to encapsulate a network's knowledge. Then, we show how to evolve this knowledge to boost performance on relatively small datasets.

C.2. Knowledge Evolution on Metric Learning

Evaluation Metrics: For metric learning evaluation, we leverage the **Recall@K** metric and **Normalized Mutual Info** (NMI) on the test split. The NMI score evaluates the quality of cluster alignments. NMI $= \frac{I(\Omega,C)}{\sqrt{H(\Omega)H(C)}}$, where $\Omega = \{\omega_1, .., \omega_n\}$, is the ground-truth clustering, while $C = \{c_1, ...c_n\}$ is a clustering assignment for the learned embedding. $I(\bullet,\bullet)$ and $H(\bullet)$ denote mutual information and entropy, respectively. We use K-means to compute C.
Results: In the paper, we report the retrieval performance using the dense network N. However, KELS delivers a slim $H^{\triangle}$ as well. Figures 4 and 5 present quantitative retrieval evaluation on CUB-200 and CARS196, respectively. Both figures leverage the R@1 metric for quantitative evaluation. We report the performance of both the dense network N and the slim fit-hypothesis $H^{\triangle}$. As the number of generations increases, the retrieval performance increases for both N and $H^{\triangle}$. Table 5 presents the fit-hypothesis $H^{\triangle}$ performance and inference cost. The fit-hypothesis $H^{\triangle}$ performance reaches the dense network N performance after $g = 10$ generations; yet, $H^{\triangle}$ achieves this performance at a significantly smaller inference cost.

D. Appendix: Ablation Study

In the paper manuscript, we have utilized VGG11_bn to monitor the development of the fit and reset hypotheses across generations. Fig. 6 shows the mean absolute values ($\widehat{H}^{\triangle}$ and $\widehat{H}^{\triangledown}$) inside the fit and reset hypotheses across all eight convolutional layers.

In this section, we present three ablation studies. We (1) evaluate the impact of changing the split-mask M across generations, (2) discuss why the improvement-margins of KE differ among datasets, and (3) evaluate KE on a large dataset, *i.e.*, ImageNet [2].

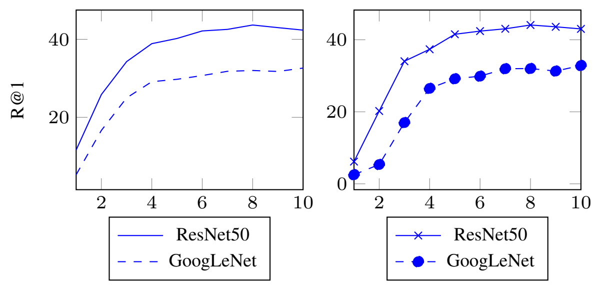

Figure 5. Quantitative retrieval evaluation using CARS196 on both GoogLeNet and ResNet50. (Left) Recall@1 of the dense network N. (Right) Recall@1 of the slim fit-hypothesis $H^{\triangle}$.

Table 5. Quantitative evaluation for KELS using the number of both operations (G-Ops) and parameters (millions). R1_g denotes the recall@1 performance at the g^{th} generation. $\blacktriangle_{\text{ops}}$ denotes the relative reduction in the number of operations. $\blacktriangle_{\text{r1}}$ denotes the absolute improvement margin on top of the dense baseline N_1.

	s_r	R1_1	R1_{10}	$\blacktriangle_{\text{r1}}$	#Ops	$\blacktriangle_{\text{ops}}$	#Param
			CUB on GoogLeNet				
N_g	0.8	10.16	15.34	5.1%	3.00	-	11.44
$H_g^{\triangle}$		4.12	15.61	5.4%	1.98	34.0%	7.43
			CUB on ResNet50				
N_g	0.8	13.01	18.25	5.2%	8.19	-	47.48
$H_g^{\triangle}$		5.33	18.38	5.3%	5.32	35.0%	30.55
			CARS on GoogLeNet				
N_g	0.8	5.29	32.63	27.3%	3.00	-	11.44
$H_g^{\triangle}$		2.53	32.85	27.5%	1.98	34.0%	7.43
			CARS on ResNet50				
N_g	0.8	11.63	42.36	30.7%	8.19	-	47.48
$H_g^{\triangle}$		6.17	43.02	31.3%	5.32	35.0%	30.55

(1) Changing the split-mask M across generations
In the paper manuscript, we split the network using a split-mark M. The *same* mask is used to re-initialize every generation. However, we also highlighted the similarity between KE and dropout. Dropout does not drop the *same* neurons during training. Thus, we investigate the impact of changing the split-mask M across generations. This is possible with the WELS technique. In this experiment, We use CUB-200, ResNet18, label smoothing regularizer, the WELS technique, and four split-rates $s_r = \{0.2, 0.3, 0.5, 0.8\}$. We train N for 10 generations. After each generation, we re-initialize M randomly, *i.e.*, as if we initialize it for the first time. We refer to this WELS variant as WELS-Rand.

Fig. 7 compares WELS against WELS-Rand. With small split-rates ($s_r = \{0.2, 0.3\}$), WELS is significantly superior to WELS-Rand. However, as the split-rate increases ($s_r = \{0.5, 0.8\}$), both WELS and WELS-Rand become

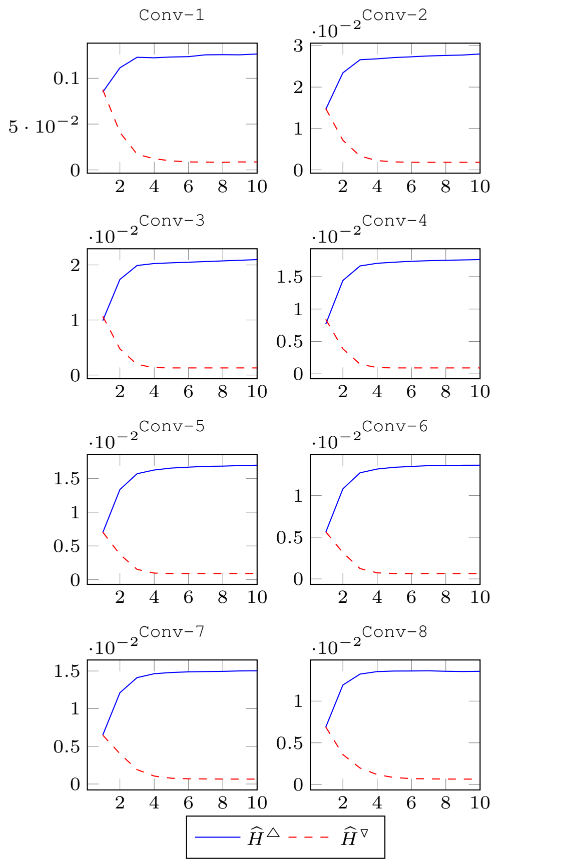

Figure 6. Quantitative evaluation using CUB-200 on VGG11_bn. The x-axis denotes the number of generations. $\widehat{H}^{\triangle}$ and $\widehat{H}^{\triangledown}$ denote the mean absolute value inside $H^{\triangle}$ and $H^{\triangledown}$, respectively.

comparable. This happens because different fit-hypotheses, in WELS+Rand, overlap partially. Given a split-rate s_r, a network-weight belongs to two consecutive fit-hypotheses with probability s_r^2. Accordingly, WELS-Rand with a small s_r flushes the entire knowledge of a parent network. In contrast, WELS-Rand with a large split-rate retains the parent-network's knowledge at least partially.

(2) Why the improvement margins $\blacktriangle_{\text{acc}}$ of KE differ?
In deep learning, we assume that more training data leads to better accuracy. However, the KE's improvement margins $\blacktriangle_{\text{acc}}$ contradict this assumption. For instance, Table 2 shows that $\blacktriangle_{\text{acc}}$ on Flower-102 is bigger than $\blacktriangle_{\text{acc}}$ on CUB-200, *i.e.*, 14.78 vs 5.68 after 10 generations with the CS-KD regularizer. Fig. 1 also emphasizes this behavior; Flower-102 is a much smaller dataset compared to CUB-200, yet $\blacktriangle_{\text{acc}}$ is over 20% for Flower-102 but less than 10% for CUB-200. We posit that $\blacktriangle_{\text{acc}}$ depends not only on the dataset size, but also on the dataset simplicity.

To evaluate our postulate, we quantify the simplicity of our five datasets (**F**lower, **C**UB, **A**ircraft, **M**IT, and **D**og). We create a new dataset, dubbed FCAMD, using the five

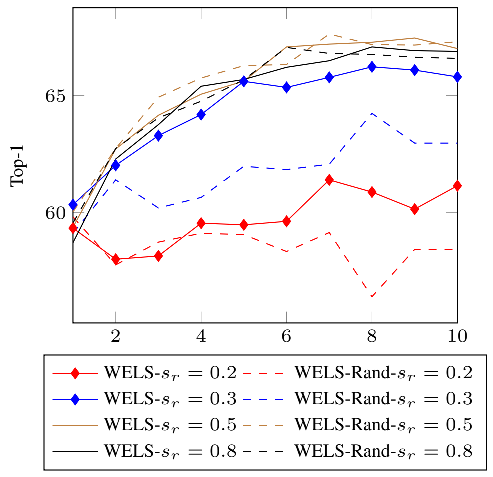

Figure 7. Comparative evaluation between WELS and WELS-Rand. WELS uses the same binary mask M across all generations. In contrast, WELS-Rand randomly re-initialize M after every generation. With a small split-rate, WELS-Rand flushes the parent-networks' knowledge.

Table 6. The KE's improvement margins $\blacktriangle_{\text{acc}}$ versus the FCAMD accuracies on each dataset. There is a strong positive Pearson correlation ($r = 0.9529$) between $\blacktriangle_{\text{acc}}$ and the datasets' simplicity (FCAMD's accuracies).

Datasets	$\blacktriangle_{\text{acc}}$	FCAMD Acc
Flower	14.78	63.06
CUB	5.68	19.60
Aircraft	0.93	15.80
MIT	0.59	19.20
Stanford Dogs	1.21	13.20
		$r = 0.952$

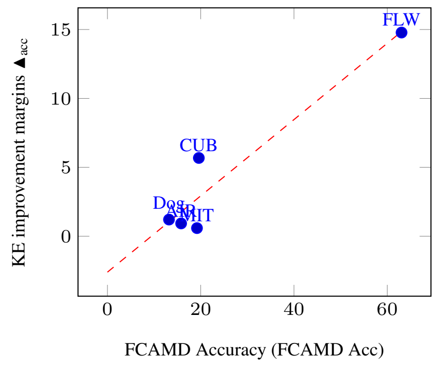

Figure 8. The average accuracy of the Flower (FLW), CUB, Aircraft (AIR), MIT, and Dog datasets inside the FCAMD dataset. The five datasets are equally represented inside FCAMD, *i.e.*, 50 classes each and 10 images per class. The accuracy metric reflects the simplicity of each dataset. The x-axis denotes the accuracy of a dataset inside FCAMD and the y-axis denotes the KE improvement margins. There is a strong positive correlation between the datasets' simplicity and the KE improvement margins.

datasets. We randomly sample 50 classes from each dataset. For each class, we randomly sample 10 training and 10 test-

Table 7. The KE's improvement margins $\blacktriangle_{\text{acc}}$ versus the accuracies of a *fine-tuned* ResNet18. There is a strong positive Pearson correlation ($r = 0.850$) between $\blacktriangle_{\text{acc}}$ and the datasets' simplicity (fine-tuned ResNet18 accuracies).

Datasets	$\blacktriangle_{\text{acc}}$	Fine-tuned ResNet18
Flower	14.78	88.83
CUB	5.68	74.46
Aircraft	0.93	61.01
MIT	0.59	72.84
Stanford Dogs	1.21	74.29
		$r = 0.850$

ing images. Thus, FCAMD has 2500 training and 2500 testing images, *i.e.*, 250 classes, 10 training images per class. We train a ResNet18 from scratch on FCAMD. To quantify the simplicity of each dataset, we measure the average accuracy of its 50 classes. Higher accuracy indicates a simpler dataset. There is a strong positive Pearson correlation ($r = 0.9529$) between the datasets' simplicity (from FCAMD's accuracies) and the KE improvement margins $\blacktriangle_{\text{acc}}$ as shown in Fig. 8 and Table 6. To compute the Pearson correlation, we use the KE improvement margins $\blacktriangle_{\text{acc}}$ achieved after 10 generations on top of the CS-KD [19] baseline, *i.e.*, $\blacktriangle_{\text{acc}}$ from the last section of Table 2. Even if we dismissed Flower-102 as an outlier, the correlation would become $r = 0.494$ for the remaining four datasets (CUB, AIR, MIT, and Dog).

Another way to quantify the simplicity of a dataset is through a pretrained network. A pretrained network contains the ImageNet's knowledge. This large knowledge mitigates the impact of both a small dataset size and a small number of samples per class. Thus, we fine-tune a pretrained ResNet18 on the five datasets as shown in Table 4. The accuracy of the fine-tuned ResNet18 reflects the simplicity of each dataset. Higher accuracy indicates a simpler dataset. Again, there is a strong positive Pearson correlation ($r = 0.850$) between $\blacktriangle_{\text{acc}}$ and the fine-tuned ResNet18 accuracies as shown in Fig. 9 and Table 7.

The FCAMD and fine-tuned ResNet18 experiments present an interesting finding. It seems that the dataset size is no longer the dominant factor that controls the performance of a randomly initialized network on relatively small datasets.

(3) Evaluate KE on ImageNet

Our paper tackles the following question: how to train a deep network on a relatively small dataset? Answering this question will have a significant impact on both academia and industry. However, it is important to understand how KE behaves on a large dataset, *i.e.*, ImageNet. The goal of this experiment is *not* to boost performance on ImageNet; Stock *et al.* [16] and Beyer *et al.* [1] deliver strong arguments why boosting performance on ImageNet should no longer be an ultimate goal. While KE boosts performance on ImageNet, our goal is to monitor the performance of the

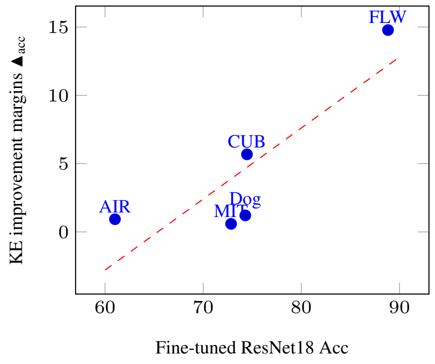

Figure 9. The accuracy of the Flower (FLW), CUB, Aircraft (AIR), MIT, and Dog datasets on a *fine-tuned* ResNet18. The accuracy metric reflects the simplicity of each dataset. The x-axis denotes the accuracy of a dataset on a *fine-tuned* ResNet18 and the y-axis denotes the KE improvement margins. There is a strong positive correlation between the datasets' simplicity and the KE improvement margins.

fit-hypothesis. We want to answer the following question: can KE evolve knowledge inside the fit-hypothesis even when presented with a large dataset?

Technical Details: We train a ResNet18 for 5 generations using KELS and a split-rate $s_r = 0.8$, *i.e.*, $\approx 36\%$ sparsity. Our implementation for ImageNet follows the practice in [8]. We use a batch size $b = 128$, and a step learning rate scheduler with a starting $lr = 0.1$. We train for $e = 150$ epochs per generation. Other parameters (*e.g.*, momentum, optimizer) are the same as those reported in the paper (Sec. 4.1).

Results: Fig. 10 presents a quantitative classification evaluation using ImageNet. KE boosts performance for both the dense network N and the slim fit-hypothesis $H^{\triangle}$. In the paper manuscript, we evaluate KE using relatively small datasets and large architectures. In contrast, this experiment evaluates KE using a large dataset and a small architecture. Accordingly, these improvement margins on ImageNet are a lower-bound on the potential of KE. As the architecture gets bigger, these improvement margins will increase. Accordingly, we conclude that KE can evolve knowledge inside the fit-hypothesis.

We further evaluate KE on two larger architectures. Table 8 presents quantitative classification evaluation using ResNet34 and ResNet50. We use the same technical details from the ResNet18 experiment. KE boosts performance on the fit-hypothesis $H^{\triangle}$ consistently. This confirms our finding that KE evolves knowledge in the fit-hypothesis $H^{\triangle}$.

References

[1] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

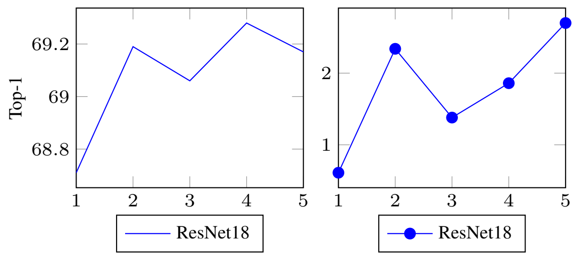

Figure 10. Quantitative classification evaluation using ImageNet on ResNet18 for 5 generations. (Left) The accuracy performance (Top-1 ↑) of the dense network N. (Right) The performance of the slim fit-hypothesis $H^{\triangle}$.

Table 8. Quantitative classification evaluation using both ResNet34 and ResNet50. N_g and $H_g^{\triangle}$ denote the performance of the dense network N and the fit-hypothesis $H^{\triangle}$ at the g^{th} generation. $\blacktriangle_H$ denotes the absolute improvement margin in the fit-hypothesis relative to the baseline $H_1^{\triangle}$

	ResNet34			ResNet50		
g	N_g	$H_g^{\triangle}$	$\blacktriangle_H$	N_g	$H_g^{\triangle}$	$\blacktriangle_H$
1	72.51	0.28	-	74.54	0.20	-
2 **(ours)**	**72.86**	1.25	0.97	74.78	3.44	3.24
3 **(ours)**	72.78	2.27	1.99	75.01	6.71	6.51
4 **(ours)**	**72.86**	1.96	1.68	75.15	4.63	4.43
5 **(ours)**	**72.86**	**4.49**	4.21	**75.27**	**13.81**	13.61

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] Saihui Hou and Zilei Wang. Weighted channel dropout for regularization of deep convolutional neural network. In *AAAI*, 2019.

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

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

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

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

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

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

[16] Pierre Stock and Moustapha Cisse. Convnets and imagenet beyond accuracy: Understanding mistakes and uncovering biases. In *ECCV*, 2018.

[17] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014.

[18] 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.

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

[20] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[21] Xiaopeng Zhang, Hongkai Xiong, Wengang Zhou, Weiyao Lin, and Qi Tian. Picking deep filter responses for fine-grained image recognition. In *CVPR*, 2016.

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