Supplementary Materials

We provide more experimental details in the following sections. First, we elaborate on CIFAR-10 experiments, followed by additional details on ImageNet results. We then give details of experiments on data-free pruning (Section 4.3 of the main paper), data-free knowledge transfer (Section 4.4 of the main paper), and data-free continual learning (Section 4.5 of the main paper). Finally, we provide additional discussions.

A. CIFAR-10 Appendix

A.1. Implementation Details

We run each knowledge distillation experiment between the teacher and student networks for 250 epochs in all, with an initial learning rate of 0.1, decayed every 100 epochs with a multiplier of 0.1. One epoch corresponds to 195 gradient updates. Image generation runs 4 times per epoch, in steps of 50 batches when VGG-11-BN is used as a teacher, and 2 times per epoch for ResNet-34. The synthesized image batches are immediately used for network update (the instantaneous accuracy on these batches is shown in Fig. 4) and are added to previously generated batches. In between image generation steps, we randomly select previously generated batches for training.

A.2. BatchNorm vs. Set of Images for $\mathcal{R}_{\text{feature}}$

DeepInversion approximates feature statistics $\mathbb{E}\big(\mu_l(x)|\mathcal{X}\big)$ and $\mathbb{E}\big(\sigma_l^2(x)|\mathcal{X}\big)$ in $\mathcal{R}_{\text{feature}}$ (Eq. 4) using BN parameters. As an alternative, one may acquire the information by running a subset of original images through the network. We compare both approaches in Table 8. From the upper section of the table, we observe that feature statistics estimated from the image subset also support DeepInversion and Adaptive DeepInversion, hence advocate for another viable approach in the absence of BNs. It only requires 100 images to compute feature statistics for Adaptive DeepInversion to achieve almost the same accuracy as with BN statistics.

# Samples	DI Top-1 acc. (%)	ADI Top-1 acc. (%)
1	61.78	84.28
10	80.94	89.99
100	83.64	90.52
1000	84.53	90.62
BN statistics	84.44	90.68

Table 8: CIFAR-10 ablations given mean and variance estimates based on (i) up: calculations from randomly sampled original images, and (ii) bottom: BN running_mean and running_var parameters. The teacher is a VGG-11-BN model at 92.34% accuracy. The student is a freshly initialized VGG-11-BN. DI: DeepInversion; ADI: Adaptive DeepInversion.

B. ImageNet Appendix

B.1. DeepInversion Implementation

We provide additional implementation details of DeepInversion next. The total variance regularization $\mathcal{R}_{\text{TV}}$ in Eq. 3 is based on the sum of ℓ_2 norms between the base image and its shifted variants: (i) two diagonal shifts, (ii) one vertical shift, and (iii) one horizontal shift, all by one pixel. We apply random flipping and jitter (< 30 pixels) on the input before each forward pass. We use the Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1 \cdot 10^{-8}$ given a constant learning rate of 0.05. We speed up the training process using half-precision floating point (FP16) via the APEX library.

C. Data-free Pruning Appendix

C.1. Hardware-aware Loss

Our proposed pruning criterion considers actual latency on the target hardware for more efficient pruning. Characterized by Eq. 10, in iterative manner neurons are ranked according to $\mathcal{I}_{\mathcal{S}}(\mathbf{W})$ and the least important ones are removed. The new scheme leverages $\mathcal{I}_{\mathcal{S},err}$ and $\mathcal{I}_{\mathcal{S},lat}$ to focus on absolute changes in network error and inference latency, specifically, when the filter group $s \in \mathcal{S}$ is zeroed out from the set of neural network parameters $\mathbf{W}$.

Akin to [44], we approximate $\mathcal{I}_{\mathcal{S},err}$ as the sum of the first-order Taylor expansion of individual filters

$$\mathcal{I}_{\mathcal{S},err}(\mathbf{W}) \approx \sum_{s \in \mathcal{S}} \mathcal{I}_s^{(1)}(\mathbf{W}) = \sum_{s \in \mathcal{S}} (g_s w_s)^2, \qquad (12)$$

where g_s and w_s denote the gradient and parameters of a filter s, respectively. We implement this approximation via gate layers, as mentioned in the original paper.

The importance of a group of filters in reducing latency can be assessed by removing it and obtaining the latency change

$$\mathcal{I}_{\mathcal{S},lat} = \text{LAT}(\mathbf{W}|w_s = 0, s \in \mathcal{S}) - \text{LAT}(\mathbf{W}), \qquad (13)$$

where $\text{LAT}(\cdot)$ denotes the latency of an intermediate pruned model on the target hardware.

Since the vast majority of computation stems from convolutional operators, we approximate the overall network latency as the sum of their run-times. This is generally appropriate for inference platforms like mobile GPU, DSP, and server GPU [9, 66]. We model the overall latency of a network as:

$$\text{LAT}(\mathbf{W}) = \sum_l \text{LAT}(o_l^{(\mathbf{W})}), \qquad (14)$$

where o_l refers to the operator at layer l. By benchmarking the run-time of each operator in hardware into a single look-up-table (LUT), we can easily estimate the latency of

any intermediate model based on its remaining filters. The technique of using a LUT for latency estimation has also been studied in the context of neural architecture search (NAS) [9, 66]. For pruning, the LUT consumes substantially less memory and profiling effort than NAS: instead of an entire architectural search space, it only needs to focus on the convolutional operations given *reduced numbers* of input and output filters against the original operating points of the pre-trained model.

C.2. Implementation Details

Our pruning setup on the ImageNet dataset follows the work in [44]. For knowledge distillation given varying image sources, we experiment with temperature τ from $\{5, 10, 15\}$ for each pruning case and report the highest validation accuracy observed. We profile and store latency values in the LUT in the following format:

$$key = [c_{in}, c_{out}, kernel^*, stride^*, fmap^*], \qquad (15)$$

where $c_{in}, c_{out}, kernel, stride,$ and $fmap$ denote input channel number, output channel number, kernel size, stride, and input feature map dimension of a Conv2D operator, respectively. Parameters with superscript * remain at their default values in the teacher model. We scan input and output channels to cover all combinations of integer values that are *less than* their initial values. Each key is individually profiled on a V100 GPU with a batch size 1 with CUDA 10.1 and cuDNN 7.6 over eight computation kernels, where the mean value of over 1000 computations for the fastest kernel is stored. Latency estimation through Eq. 14 demonstrates a high linear correlation with the real latency ($R^2 = 0.994$). For hardware-aware pruning, we use $\eta = 0.01$ for Eq. 10 to balance the importance of $\mathcal{I}_{\mathcal{S},err}$ and $\mathcal{I}_{\mathcal{S},lat}$, and prune away 32 filters each time in a group size of 16. We prune every 30 mini-batches until the pre-defined filter/latency threshold is met, and continue to fine-tune the network after that. We use a batch size of 256 for all our pruning experiments. To standardize input image dimensions, default ResNet pre-processing from PyTorch is applied to MS COCO and PASCAL VOC images.

C.3. Hardware-aware Loss Evaluation

As an ablation, we show the effectiveness of the hardware-aware loss (Eq. 10 in Section 4.3) by comparing it with the pruning baseline in Table 9. We base this analysis on the ground truth data to compare with prior art. Given the same latency constraints, the proposed loss improves the top-1 accuracy by 0.5%-14.8%.

C.4. Pruning without Labels

Taylor expansion for pruning estimates the change in loss induced by feature map removal. Originally, it was proposed for CE loss given ground-truth labels of input images. We argue that the same expansion can be applied to the knowledge distillation loss, particularly the KL divergence loss, computed between the teacher and student output distributions. We also use original data in this ablation for a fair comparison with prior work and show the results in Table 10. We can see that utilizing KL loss leads to only -0.7% to $+0.1\%$ absolute top-1 accuracy changes compared to the original CE-based approach, while completely removing the need for labels for Taylor-based pruning estimates.

V100 Lat. (ms)	Taylor [44] Top-1 acc. (%)	Hardware-aware Taylor (**Ours**) Top-1 acc. (%)
4.90 - baseline	76.1	76.1
4.78	76.0	76.5
4.24	74.9	75.9
4.00	73.2	73.8
3.63	69.2	71.6
3.33	55.2	70.0

Table 9: ImageNet ResNet-50 pruning ablation with and without latency-aware loss given original data. Latency measured on V100 GPU at batch size 1. Top-1 accuracy corresponds to the highest validation accuracy for 15 training epochs. Learning rate is initialized to 0.01, decayed at the 10^{th} epoch.

Filters pruned (%)	CE loss, w/ labels [44] Top-1 acc. (%)	KL Div., w/o labels (**Ours**) Top-1 acc. (%)
0 - baseline	76.1	76.1
10	72.1	72.0
20	58.0	58.1
30	37.1	36.4
40	17.2	16.6

Table 10: ImageNet ResNet-50 pruning ablation with and without labels given original images. CE: cross-entropy loss between predictions and ground truth labels; KL Div: KullBack-Leibler divergence loss between teacher-student output distributions. Learning rate is 0, hence no fine-tuning is done.

C.5. Distribution Coverage Expansion

Adaptive DeepInversion aims at expanding the distribution coverage of the generated images in the feature space through competition between the teacher and the student networks. Results of its impact are illustrated in Fig. 7. As expected, the distribution coverage gradually expands, given the two sequential rounds of competition following the initial round of DeepInversion. From the two side bars in Fig. 7, we observe varying ranges and peaks after projection onto each principal component from the three image generation rounds.

To further visualize the diversity increase due to competition loss (Eq. 8), we compare the class of handheld computers generated with and without the competition scheme in Fig. 8. As learning continues, competition leads to the discovery of features for hands from the teacher's knowledge scope to challenge the student network. Moreover, generated images differ from their nearest neighbors, showing the

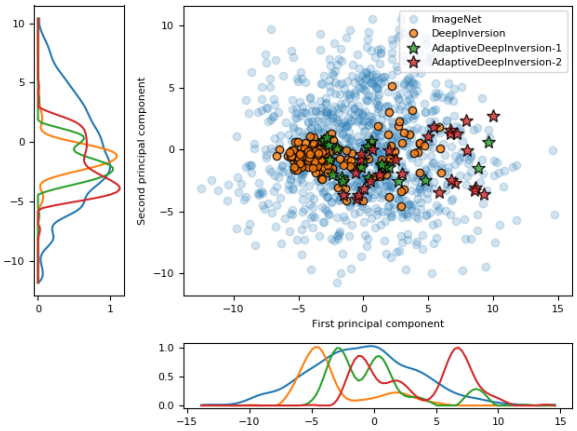

Figure 7: Projection onto the first two principal components of the ResNet-50-avgpool feature vectors of ImageNet class 'hand-held computer' training images. ADI-1/2 refers to additional images from round1/2 competition.

(a) DeepInversion (b) ADI

Figure 8: Nearest neighbors of the synthesized images in the ResNet-50-avgpool feature space for the ImageNet class 'hand-held computer' (a) without and (b) with the proposed competition scheme.

efficacy of the approach in capturing distribution as opposed to memorizing inputs.

D. Data-free Knowledge Transfer Appendix

We use the following parameters for the experiment on ResNet50v1.5: $\alpha_{\text{tv}} = 1 \cdot 10^{-4}$, $\alpha_f = 0.01$, and a learning rate of 0.2 for Adam. We generate images with an equal probability between the (i) multi-resolution scheme and (ii) the scheme described in Section 4.2 with 2k iterations only to further improve image diversity. We clip the synthesized image $\hat{x}$ using

$$\hat{x} \quad = \quad \min\big(\max(\hat{x}, -m/s), (1-m)/s\big), \qquad (16)$$

where $m = \{0.485, 0.456, 0.406\}$ and $s = \{0.229, 0.224, 0.225\}$.

E. Data-free Continual Learning Appendix

E.1. Implementation Details

Our DeepInversion setup for this application follows the descriptions in Section 4.2 and Appendix B.1 with minor modifications as follows. We use DeepInversion to generate $\{250, 64\}$ images of resolution 224×224 per existing class in the pretrained $\{$ResNet-18, VGG-16-BN$\}$. These images are generated afresh after adding each new dataset. For $\{$ResNet-18, VGG-16-BN$\}$, we use a learning rate of $\{0.2,\ 0.5\}$, optimize for 10k gradient updates in all, and decay the learning rate every 1.5k steps with a 0.3 multiplier. We use both ℓ_2 and ℓ_1 norms for total variance regularization at $\alpha_{\text{tv},\ell_2} = \{3 \cdot 10^{-5},\ 6 \cdot 10^{-5}\}$, $\alpha_{\text{tv},\ell_1} = \{1 \cdot 10^{-1}, 2 \cdot 10^{-1}\}$, jointly with $\alpha_{\ell_2} = 0$ and $\alpha_f = \{1 \cdot 10^{-1},\ 3 \cdot 10^{-2}\}$ for DeepInversion. These parameters are chosen such that all loss terms are of the same magnitude, and adjusted to optimize qualitative results.

Each method and dataset combination has individually-tuned learning rate and number of epochs obtained on a validation split using grid search, by optimizing the new dataset's performance while using the smallest learning rate and number of epochs possible to achieve this optimal performance. For each iteration, we use a batch of DeepInversion data $(\hat{x}, y_o(\hat{x}))$ and a batch of new class real data (x_k, y_k). The batch size is 128 for both kinds of data when training ResNet-18, and 64 for VGG-16-BN. Similar to prior work [56], we reduce the learning rate to 20% at $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$, and $\frac{5}{6}$ of the total number of epochs. We use stochastic gradient descent (SGD) with a momentum of 0.9 as the optimizer. We clip the gradient ℓ_2 magnitude to 0.1, and disable all updates in the BN layers. Gradient clipping and freezing BN do not affect the baseline LwF.MC [56] much ($\pm 2\%$ change in combined accuracy after hyperparameter tuning), but significantly improve the accuracy of our methods and the oracles. We start with the pretrained ImageNet models provided by PyTorch. LwF.MC [56] needs to use binary CE loss. Hence, we fine-tuned the model on ImageNet using binary CE with a small learning rate. The resulting ImageNet model is within $\pm 0.5\%$ top-1 error of the original model. We did not investigate the effect of the number of images synthesized on the performance.

E.2. VGG-16-BN Results

We show our data-free continual learning results on the VGG-16-BN network in Table 11. The proposed method outperforms prior art [56] by a large margin by enabling up to 42.6% absolute increase in the top-1 accuracy. We observe a small gap of $< 2\%$ combined error between our proposal and the best-performing oracle for this experimental setting, again showing DeepInversion's efficacy in replacing ImageNet images for the continual learning task.

E.3. Use Case and Novelty

The most significant departure from prior work such as EWC [27] is that our DeepInversion-based continual learning can operate on *any* regularly-trained model, given the widespread usage of BN layers. Our method eliminates the

Method	Top-1 acc. (%)			
	Combined	ImageNet	CUB	Flowers
ImageNet + CUB (1000 → 1200 outputs)				
LwF.MC [56]	47.43	64.38	30.48	–
DeepInversion (**Ours**)	**70.72**	**68.35**	**73.09**	–
Oracle (distill)	72.71	71.95	73.47	–
Oracle (classify)	72.03	71.20	72.85	–
ImageNet + Flowers (1000 → 1102 outputs)				
LwF.MC [56]	67.67	65.10	–	70.24
DeepInversion (**Ours**)	**82.47**	**72.11**	–	**92.83**
Oracle (distill)	83.07	72.84	–	93.30
Oracle (classify)	81.56	71.97	–	91.15

Table 11: Results on incrementally extending the network softmax output space by adding classes from a new dataset. All results are obtained using VGG-16-BN.

need for any collaboration from the model provider, even when the model provider (1) is unwilling to share any data, (2) is reluctant to train specialized models for continual learning, or (3) does not have the know-how to support a downstream continual learning application. This gives machine learning practitioners more freedom and expands their options when adapting existing models to new usage scenarios, especially when data access is restricted.

F. Limitation Discussion

- **Image synthesis time.** Generating 215K ImageNet samples of 224×224 resolution for a ResNet-50 takes 2.8K NVIDIA V100 GPU-hours, or 22 hours on 128 GPUs. This time scales linearly with the number of synthesized images. The multi-resolution scheme described in Section 4.4 reduces this time by 10.7× (0.26K GPU-hours / 4 hours on 64 GPUs).
- **Image color and background similarity.** We believe there are two possible reasons for this similarity. 1) The method uncovers and visualizes the unique discriminative characteristics of a CNN classifier, which can guide future work on neural network understanding and interpretation. Post-training, the network learns to capture only the informative visual representations to make a correct classification. For example, the key features of a target object are retained, *e.g.*, detailed bear heads in Fig. 6 or the fur color/patterns of penguins and birds in Fig. 5, whereas the background information is mostly simplified, *e.g.*, green for grass or blue for ocean. 2) For all the images synthesized in this work, we use a default Gaussian distribution with zero mean and unit variance to initialize all the pixels, which may lead to unimodal behaviors. We have also observed that the style varies with the choice of the optimization hyperparameters.
- **Continual learning class similarity.** We implemented DeepInversion on iCIFAR and iILSVRC [56] (two splits) and observed statistically equivalent or slightly worse performance compared to LwF.MC. We suspect that the synthesized images are more effective in replacing old class images that are *different* from the new images, compared to a case where the old and new images are similar (*e.g.*, random subsets of a pool of classes).