# Dreaming to Distill: Data-free Knowledge Transfer via DeepInversion

Hongxu Yin[1,2†*], Pavlo Molchanov[1*], Zhizhong Li[1,3†], Jose M. Alvarez[1],
Arun Mallya[1], Derek Hoiem[3], Niraj K. Jha[2], and Jan Kautz[1]

[1]NVIDIA, [2]Princeton University, [3]University of Illinois at Urbana-Champaign

{hongxuy, jha}@princeton.edu, {zli115, dhoiem}@illinois.edu,

{pmolchanov, josea, amallya, jkautz}@nvidia.com

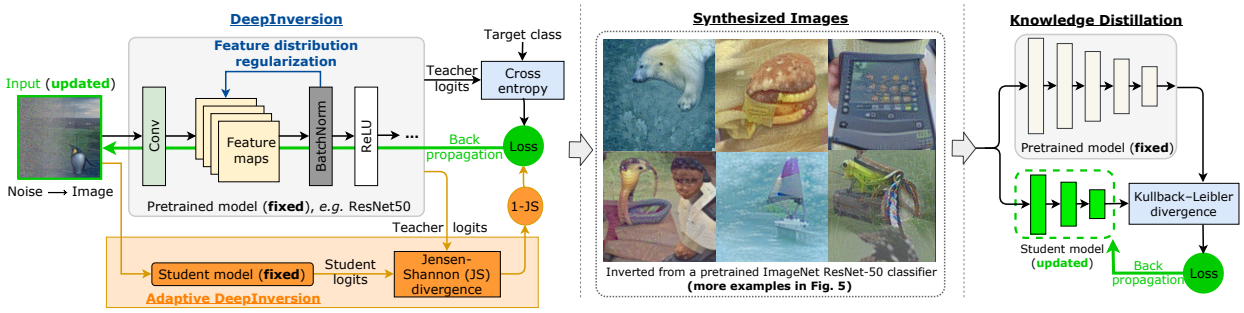



Figure 1: We introduce DeepInversion, a method that optimizes random noise into high-fidelity class-conditional images given just a pretrained CNN (teacher) in Sec. 3.2. Further, we introduce Adaptive DeepInversion (Sec. 3.3) which utilizes both the teacher and application-dependent student network to improve image diversity. We show how the synthesized images can be used to perform various kinds of data-free knowledge distillation [20], greatly improving upon prior work. Using DeepInversion, we enable data-free pruning (Sec. 4.3), introduce and address data-free knowledge transfer (Sec. 4.4), and improve upon data-free incremental learning (Sec. 4.5).

## Abstract

*We introduce DeepInversion, a new method for synthesizing images from the image distribution used to train a deep neural network. We "invert" a trained network (teacher) to synthesize class-conditional input images starting from random noise, without using any additional information about the training dataset. Keeping the teacher fixed, our method optimizes the input while regularizing the distribution of intermediate feature maps using information stored in the batch normalization layers of the teacher. Further, we improve the diversity of synthesized images using Adaptive DeepInversion, which maximizes the Jensen-Shannon divergence between the teacher and student network logits. The resulting synthesized images from networks trained on the CIFAR-10 and ImageNet datasets demonstrate high fidelity and degree of realism, and help enable a new breed of data-free applications – ones that do not require any real images or labeled data. We demonstrate the applicability of our proposed method to three tasks of immense practical importance – (i) data-free network pruning, (ii) data-free knowledge transfer, and (iii) data-free continual learning.*

## 1. Introduction

The ability to transfer learned knowledge from a trained neural network to a new one with properties desirable for the task at hand has many appealing applications. For example, one might want to use a smaller or more resource-efficient network architecture while deploying on edge inference devices [44, 66, 76], or to adapt the network to the inference hardware [9, 63, 71], or for continuously learning to classify new image classes [29, 34], *etc*. Most current methods that solve such knowledge transfer tasks are based on the concept of knowledge distillation [20], wherein the new network (student) is trained to match its outputs to that of a previously trained network (teacher). However, all such methods have a significant constraint – they assume that either the previously used training data is available [8, 29, 45, 57], or some real images representative of the distribution of the prior training dataset are available [25, 26, 34, 56]. Even methods not based on distillation, such as [27, 50, 74] assume that some additional statistics about prior training is made available by the pretrained model provider.

The requirement for prior training information can be very restrictive in practice. For example, suppose a very deep network such as ResNet-152 [18] was trained on datasets with millions [10] or even billions of images [36], and we wish to distill its knowledge to a lower latency model such

*Equal contribution. † Work done during an internship at NVIDIA.

as ResNet-18. In this case, we would need access to these datasets, which are not only large but difficult to store, transfer, and manage. Further, another emerging concern is that of data privacy. While entities might want to share their trained models, sharing the training data might not be desirable due to user privacy, security, proprietary concerns, or competitive advantage.

In the absence of prior data or metadata, an interesting question arises – can we somehow recover training data from the already trained model and use it for knowledge transfer? A few methods have attempted to visualize what a trained deep network expects to see in an image [3, 37, 46, 49]. The most popular and simple-to-use method is DeepDream [46]. It synthesizes or transforms an input image to yield high output responses for chosen classes in the output layer of a given classification model. This method optimizes the input (random noise or a natural image), possibly with some regularizers, while keeping the selected output activations fixed, but leaves intermediate representations constraint-free. The resulting "dreamed" images lack natural image statistics and can be quite easily identified as unnatural. These images are also not very useful for the purposes of transferring knowledge, as our extensive experiments in Section 4 show.

In this work, we make an important observation about deep networks that are widely used in practice – they all implicitly encode very rich information about prior training data. Almost all high-performing convolutional neural networks (CNNs) such as ResNets [18], DenseNets [22], or their variants, use the batch normalization layer [24]. These layers store running means and running variances of the activations at multiple layers. In essence, they store the history of previously seen data, at multiple levels of representation. By assuming that these intermediate activations follow a Gaussian distribution with mean and variance equal to the running statistics, we show that we can obtain "dreamed" images that are realistic and much closer to the distribution of the training dataset as compared to prior work in this area.

Our approach, visualized in Fig. 1, called *DeepInversion*, introduces a regularization term for intermediate layer activations of dreamed images based on just the two layer-wise statistics, mean and variance, which are directly available with trained models. As a result, we do not require any training data or metadata to perform training image synthesis. Our method is able to generate images with high fidelity and realism at a high resolution, as can be seen in the middle section of Fig. 1, and more samples in Fig. 5.

We also introduce an application-specific extension of *DeepInversion*, called *Adaptive DeepInversion*, which can enhance the diversity of the generated images. More specifically, it exploits disagreements between the pretrained teacher and the in-training student network to expand the coverage of the training set by the synthesized images. It does so by maximizing the Jensen-Shannon divergence between the responses of the two networks.

In order to show that our dataset synthesis method is useful in practice, we demonstrate its effectiveness on three different use cases. First, we show that the generated images support knowledge transfer between two networks using distillation, even with different architectures, with a minimal accuracy loss on the simple CIFAR-10 as well as the large and complex ImageNet dataset. Second, we show that we can prune the teacher network using the synthesized images to obtain a smaller student on the ImageNet dataset. Finally, we apply DeepInversion to incremental learning that enables the addition of new classes to a pretrained CNN without the need for any original data. Using our DeepInversion technique, we empower a new class of "data-free" applications of immense practical importance which need neither any natural image nor labeled data.

Our main contributions are as follows:

- We introduce DeepInversion, a new method for synthesizing class-conditional images from a CNN trained for image classification (Sec. 3.2). Further, we improve the synthesized image diversity by exploiting student-teacher disagreements via Adaptive DeepInversion (Sec. 3.3).
- We enable data-free and hardware-aware pruning that achieves performance comparable to the state-of-the-art methods that rely on the training dataset (Sec. 4.3).
- We introduce and address the task of data-free knowledge transfer between a teacher and a randomly initialized student network (Sec. 4.4).
- We improve upon prior work on data-free incremental learning and achieve results comparable with oracle methods given the original data (Sec. 4.5).

## 2. Related Work

**Knowledge distillation.** A long line of work has explored transferring knowledge from one model to another. Breiman and Shang first introduced this concept in learning a single decision tree to approximate the outputs of multiple decision trees [4]. Similar ideas are explored in neural networks by Bucilua *et al.* [6], Ba and Caruana [2], and Hinton *et al.* [20]. Of these, Hinton *et al.* formulated the problem as "knowledge distillation", where a compact student mimics the output distributions of expert teacher models [20]. Knowledge distillation and its improved variants [1, 53, 57, 67, 73] enable teaching students with goals such as quantization [42, 55], compact neural network architecture design [57], semantic segmentation [31], self-distillation [14], and un-/semi-supervised learning [34, 54, 70]. All these methods still rely on images from original or proxy datasets. More recent research has explored data-free knowledge distillation. Lopes *et al.* [33] synthesized inputs based on pre-stored auxiliary layer-wise

statistics of the teacher network. Chen *et al.* [7] trained a new generator network for image generation while treating the teacher network as a fixed discriminator. Despite remarkable insights, scaling to tasks such as the ImageNet classification remains difficult for these methods.

**Image synthesis.** GANs [16, 43, 47, 75] have been at the forefront of generative image modeling, and have yielded high fidelity images as recently shown by BigGAN proposed by Brock *et al*. [5]. Though adept at capturing image distribution, GAN training requires access to the original data to train its generator for future image generation.

An alternat line of work focuses on image synthesis from a single CNN. In work on security, Fredrikson *et al*. [13] propose the *model inversion* attack to reverse class-wise training images of a network through a gradient descent on the input. Follow-up works have refined and expanded the approach to new threat scenarios such as collaborative learning and membership detection [19, 64, 68]. These methods have only been demonstrated on shallow networks.

In vision, researchers have focused on visualizing neural networks and understanding factors behind their intrinsic properties. Mahendran *et al*. explored inversion, activation maximization, and caricaturization techniques to synthesize "natural pre-images" from a trained network [37, 38]. Nguyen *et al*. used a deep generator network as a GAN prior to inverse images from a trained CNN [48], and its followup Plug & Play approach further improved the image diversity and quality via latent code prior [47]. Along the same line, Bhardwaj *et al*. explored the training data cluster centroids to improve inversion efficacy [3]. These methods still rely on auxiliary dataset information or additional pretrained networks. Of particular relevance to this work is DeepDream [46] proposed by Mordvintsev *et al*., which has enabled the "dreaming" of new object features onto natural images given a single pretrained CNN. Despite notable progress, synthesizing high fidelity and high resolution natural images from a deep network remains challenging.

## 3. Method

We propose a new framework for data-free knowledge distillation. Our framework broadly consists of two steps: (i) model inversion, and (ii) application-specific knowledge distillation. In this section, we briefly discuss the background and notation, and then introduce our *DeepInversion* and *Adaptive DeepInversion* methods.

### 3.1. Background

**Knowledge Distillation.** Distillation [20] is a popular technique for knowledge transfer between two models. The simplest form of distillation consists of two steps. First, the teacher, a large single or ensemble of models, is trained. Second, a smaller model, the student, is trained to mimic the behavior of the teacher model by matching the temperature-scaled soft target distribution produced by the teacher on training data. Given a trained model $p_T$ and a dataset $\mathcal{X}$, the parameters of the student model, $\mathbf{W}_S$, can be learned by

$$\min_{\mathbf{W}_S} \sum_{x \in \mathcal{X}} \mathrm{KL}(p_T(x), p_S(x)), \tag{1}$$

where $\mathrm{KL}(\cdot)$ refers to the Kullback-Leibler divergence and $p_T(\cdot)$ and $p_S(\cdot)$ are the output distributions produced by the teacher and student model respectively, typically obtained using a high temperature on the softmax inputs [20].

Knowledge distillation also works using other images from the same domain as the original training data. Note that labels are not required for distillation. Despite its efficacy, the process still relies on real images from the same domain. Below, we focus on methods to synthesize a large set of images $\hat{x} \in \hat{\mathcal{X}}$ from noise that could replace $x \in \mathcal{X}$.

**DeepDream [46].** Mordvintsev *et al*. originally formulated DeepDream to derive artistic effects on natural images, but their method is also suitable for optimizing noise into images. Given a randomly initialized input ($\hat{x} \in \mathcal{R}^{H \times W \times C}$, $H, W, C$ being the height, width, and number of color channels) and an arbitrary target label $y$, the image synthesis process can be expressed as solving an optimization of the form

$$\min_{\hat{x}} \mathcal{L}(\hat{x}, y) + \mathcal{R}(\hat{x}), \tag{2}$$

where $\mathcal{L}(\cdot)$ is a classification loss such as the standard softmax cross-entropy, and $\mathcal{R}(\cdot)$ is a image regularization term to improve $\hat{x}$'s visual quality. DeepDream [46] proposes an image prior term [11, 37, 49, 61] to steer the generated images away from unrealistic images that are classified correctly but possess no discernible visual information:

$$\mathcal{R}_{\text{prior}}(\hat{x}) = \alpha_{\text{tv}} \mathcal{R}_{\text{TV}}(\hat{x}) + \alpha_{\ell_2} \mathcal{R}_{\ell_2}(\hat{x}), \tag{3}$$

where $R_{\text{TV}}$ and $R_{\ell_2}$ penalize the total variance and $\ell_2$ norm of $\hat{x}$, respectively, and $\alpha_{\text{tv}}$, $\alpha_{\ell_2}$ are scaling factors. As empirically observed in our experiments and prior work [37, 46, 49], image prior regularization provides more stable convergence to valid images. However, these images still lack an overlap in their distribution with natural (or original training) images and thus lead to unsatisfactory results for knowledge distillation.

### 3.2. DeepInversion (DI)

We improve DeepDream's image quality by extending the regularization with a new feature distribution regularization term. Given access to only $\hat{x} \in \hat{\mathcal{X}}$, the image prior regularization term defined in the previous section provides little guidance for obtaining an $\hat{x}$ that contains similar low- and high-level features as $x \in \mathcal{X}$. To effectively enforce feature similarities at all levels, we propose to minimize the

distance between feature map statistics for $\hat{x}$ and $x$. We assume that feature statistics follow the Gaussian distribution across batches and, therefore, can be defined by mean $\mu$ and variance $\sigma^2$. Then, the *feature distribution regularization* term can be formulated as:

$$\mathcal{R}_{\text{feature}}(\hat{x}) = \sum_l || \mu_l(\hat{x}) - \mathbb{E}(\mu_l(x)|\mathcal{X}) ||_2 + \sum_l || \sigma_l^2(\hat{x}) - \mathbb{E}(\sigma_l^2(x)|\mathcal{X}) ||_2, \tag{4}$$

where $\mu_l(\hat{x})$ and $\sigma_l^2(\hat{x})$ are the batch-wise mean and variance estimates of feature maps corresponding to the $l^{\text{th}}$ convolutional layer. The $\mathbb{E}(\cdot)$ and $||\cdot||_2$ operators denote the expected value and $\ell_2$ norm calculations, respectively.

It might seem as though a set of training images would be required to obtain $\mathbb{E}(\mu_l(x)|\mathcal{X})$ and $\mathbb{E}(\sigma_l^2(x)|\mathcal{X})$, but the running average statistics stored in the widely-used batchnorm (BN) layers are more than sufficient. A BN layer normalizes the feature maps during training to alleviate covariate shifts [24]. It implicitly captures the channel-wise means and variances during training, hence allows for estimation of the expectations in Eq. 4 by:

$$\mathbb{E}\big(\mu_l(x)|\mathcal{X}\big) \simeq \text{BN}_l(\text{running\_mean}), \tag{5}$$

$$\mathbb{E}\big(\sigma_l^2(x)|\mathcal{X}\big) \simeq \text{BN}_l(\text{running\_variance}). \tag{6}$$

As we will show, this feature distribution regularization substantially improves the quality of the generated images. We refer to this model inversion method as *DeepInversion* - a generic approach that can be applied to any trained *deep* CNN classifier for the *inversion* of high-fidelity images. The regularization $R(\cdot)$ (corr. to Eq. 2) can thus be expressed as

$$\mathcal{R}_{\text{DI}}(\hat{x}) = \mathcal{R}_{\text{prior}}(\hat{x}) + \alpha_{\text{f}}\mathcal{R}_{\text{feature}}(\hat{x}). \tag{7}$$

### 3.3. Adaptive DeepInversion (ADI)

In addition to quality, diversity of the generated images also plays a crucial role in avoiding repeated but redundant image synthesis. Prior work on GANs has studied the problem and proposed various techniques, such as min-max training competition [15] and the truncation trick [5]. These methods rely on the joint training of two networks over original data and therefore are not applicable to our problem. We propose *Adaptive DeepInversion*, an enhanced image generation scheme based on a novel iterative competition scheme between the image generation process and the student network. The main idea is to encourage the synthesized images to cause student-teacher disagreement. For this purpose, we introduce an additional loss $\mathcal{R}_{\text{compete}}$ for image generation based on the Jensen-Shannon divergence that penalizes output distribution similarities,

$$\mathcal{R}_{\text{compete}}(\hat{x}) \quad = \quad 1 - \text{JS}(p_T(\hat{x}), p_S(\hat{x})), \tag{8}$$

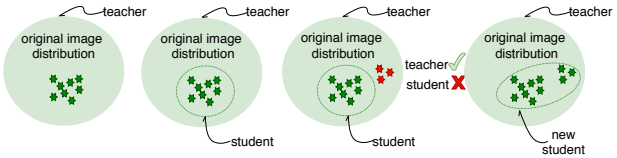



Figure 2: Illustration of the Adaptive DeepInversion competition scheme to improve image diversity. Given a set of generated images (shown as green stars), an intermediate student can learn to capture part of the original image distribution. Upon generating new images (shown as red stars), competition encourages new samples out of student's learned knowledge, improving distributional coverage and facilitating additional knowledge transfer. Best viewed in color.

$$\text{JS}(p_T(\hat{x}), p_S(\hat{x})) = \frac{1}{2}\Big(\text{KL}(p_T(\hat{x}), M) + \text{KL}(p_S(\hat{x}), M)\Big),$$

where $M = \frac{1}{2}\cdot\big(p_T(\hat{x}) + p_S(\hat{x})\big)$ is the average of the teacher and student distributions.

During optimization, this new term leads to new images the student cannot easily classify whereas the teacher can. As illustrated in Fig. 2, our proposal iteratively expands the distributional coverage of the image distribution during the learning process. With competition, the regularization $R(\cdot)$ from Eq. 7 is updated with an additional loss scaled by $\alpha_c$ as

$$\mathcal{R}_{\text{ADI}}(\hat{x}) = \mathcal{R}_{\text{DI}}(\hat{x}) + \alpha_c\mathcal{R}_{\text{compete}}(\hat{x}). \tag{9}$$

### 3.4. DeepInversion vs. Adaptive DeepInversion

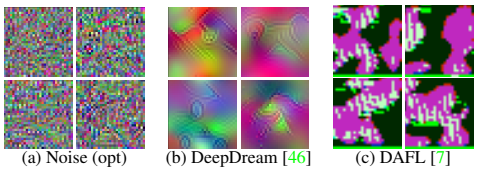
DeepInversion is a generic method that can be applied to any trained CNN classifier. For knowledge distillation, it enables an one-time synthesis of a large number of images given the teacher, to initiate knowledge transfer. Adaptive DeepInversion on the other hand, needs a student in the loop to enhance image diversity. Its competitive nature favors constantly-evolving students, which gradually force new image features to emerge. Interaction and feedback from another network enable the augmentation of DeepInversion, as we show in our experiments in the following section.

## 4. Experiments

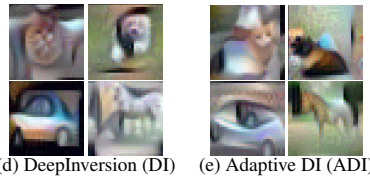
In this section, we demonstrate the effectiveness of our proposed inversion methods on datasets of increasing size and complexity. We perform a number of ablations to better understand the contributions of each component in our method on the simple CIFAR-10 dataset of $32 \times 32$ pixel images and 10 classes. Then, we focus on the complex ImageNet dataset that has 1000 classes and images of size $224 \times 224$. On ImageNet, we show the successful application of our inversion method on three different tasks under the data-free setting - (a) pruning, (b) knowledge transfer, and (c) continual/incremental learning. In all experiments, image pixels are initialized $i.i.d.$ from Gaussian noise of $\mu = 0$ and $\sigma = 1$ for DeepInversion (DI) and Adaptive DeepInversion (ADI).

| Teacher Network | VGG-11 | VGG-11 | ResNet-34 |
|---|---|---|---|
| Student Network | VGG-11 | ResNet-18 | ResNet-18 |
| Teacher accuracy | 92.34% | 92.34% | 95.42% |
| Noise ($\mathcal{L}$) | 13.55% | 13.45% | 13.61% |
| $+\mathcal{R}_{\text{prior}}$ (DeepDream [46]) | 36.59% | 39.67% | 29.98% |
| $+\mathcal{R}_{\text{feature}}$ (DeepInversion) | 84.16% | 83.82% | 91.43% |
| $+\mathcal{R}_{\text{compete}}$ (ADI) | **90.78%** | **90.36%** | **93.26%** |
| DAFL [7] | – | – | 92.22% |

Table 1: Data-free knowledge transfer to various students on CIFAR-10. For ADI, we generate one new batch of images every 50 KD iterations and merge the newly generated images into the existing set of generated images.



(a) Noise (opt) (b) DeepDream [46] (c) DAFL [7]



(d) DeepInversion (DI) (e) Adaptive DI (ADI)

Figure 3: Generated $32 \times 32$ images by inverting a ResNet-34 trained on CIFAR-10 with different methods. All images are correctly classified by the network, clockwise: cat, dog, horse, car.

### 4.1. Results on CIFAR-10

We first perform experiments on the CIFAR-10 dataset to validate our design choices for image generation. We consider the task of data-free knowledge distillation, where we teach a randomly initialized student network from scratch.

**Implementation Details.** We use VGG-11-BN and ResNet-34 networks that are pretrained on the CIFAR-10 dataset as the teachers. For all image synthesis in this section, we use an Adam optimizer and a learning rate of 0.05. We generate $32 \times 32$ resolution images in batches of size 256. Each image batch requires 2k gradient updates. After a simple grid search optimizing for student accuracy, we found $\alpha_{\text{tv}} = 2.5 \cdot 10^{-5}, \alpha_{\ell_2} = 3 \cdot 10^{-8}$, and $\alpha_f = \{1.0, 5.0, 10.0, 100.0\}$ to work best for DI, and $\alpha_c = 10.0$ for ADI. Additional details are in the supplementary material.

**Baselines – Noise & DeepDream [46].** From Table 1, we observe that optimized noise, Noise ($\mathcal{L}$), does not provide any support for knowledge distillation - a drastic change in input distribution disrupts the teacher and impacts the validity of the transferred knowledge. Adding $R_{\text{prior}}$, like in DeepDream, slightly improves the student's accuracy.

**Effectiveness of DeepInversion ($R_{\text{feature}}$).** Upon adding $R_{\text{feature}}$, we immediately find large improvements in accuracy of $40\% - 69\%$ across all the teaching scenarios. As can be seen in Fig. 3, DeepInversion images (d) are vastly superior in realism, as compared to baselines (a) and (b).

**Effectiveness of Adaptive DeepInversion ($R_{\text{compete}}$).** Using competition-based inversion further improves accuracy by $1\% - 10\%$, bringing the student accuracy very close to that of the teacher trained on real images from the CIFAR-10 dataset (within $2\%$). The training curves from one of the runs are shown in Fig. 4. Encouraging teacher-student disagreements brings in additional "harder" images during training (shown in Fig. 3 (e)) that confuse the student but remain correct for the teacher, as seen in the left part of Fig. 4 which shows low student confidence during image generation.

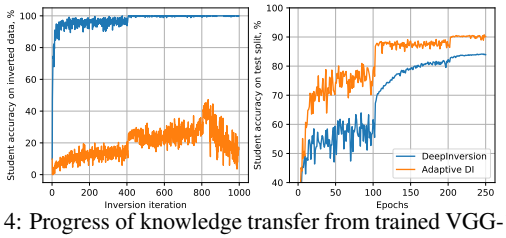



Figure 4: Progress of knowledge transfer from trained VGG-11-BN (92.34% acc.) to freshly initialized VGG-11-BN network (student) using inverted images. Plotted here are accuracies on generated (left) and real (right) images. Final student accuracies are in Table 1.

**Comparison with DAFL [7].** We further compare our method with [7], which trains a new generator network to convert noise into images while treating the teacher as a fixed discriminator. As seen in Fig. 3 (c), we notice that these images are reminiscent of "unrecognizable images", or "fooling images" [49]. Our method enables higher visual fidelity of images while cutting down on the training cost for an additional generator network. Our approach also exhibits higher student accuracy under the same experimental setup.

### 4.2. Results on ImageNet

After successfully demonstrating our method's abilities on the small CIFAR-10 dataset, we move on to examine its effectiveness on the large-scale ImageNet dataset [10]. We first run DeepInversion on networks trained on ImageNet, and perform quantitative and qualitative analysis. Then, we show the effectiveness of our synthesized images on 3 different tasks of immense importance: data-free pruning, data-free knowledge transfer, and data-free continual learning.

**Implementation Details.** For all experiments in this section, we use the publicly available pretrained ResNet-$\{18, 50\}$ from PyTorch as the fixed teacher network, with top-1 accuracy of $\{69.8\%, 76.1\%\}$. For image synthesis, we use an Adam optimizer and a learning rate of $0.05$. We set $\alpha_{\text{tv}} = 1 \cdot 10^{-4}, \alpha_{\ell_2} = \{0, 1 \cdot 10^{-2}\}, \alpha_f = 1 \cdot 10^{-2}$ for DI, and $\alpha_c = 0.2$ for ADI. We synthesize images of resolution $224 \times 224$ in a batch size of $1,216$ using 8 NVIDIA V100 GPUs and automatic-mixed precision (AMP) [41] acceleration. Each image batch consumes 20k updates over 2h.

Figure 5: Class-conditional $224 \times 224$ samples obtained by DeepInversion, given only a ResNet-50 classifier trained on ImageNet and no additional information. Note that the images depict classes in contextually correct backgrounds, in realistic scenarios. Best viewed in color.

| Model | DeepDream top-1 acc. (%) | DeepInversion top-1 acc. (%) |
|---|---|---|
| ResNet-50 | 100 | 100 |
| ResNet-18 | 28.0 | **94.4** |
| Inception-V3 | 27.6 | **92.7** |
| MobileNet-V2 | 13.9 | **90.9** |
| VGG-11 | 6.7 | **80.1** |

Table 2: Classification accuracy of ResNet-50 synthesized images by other ImageNet-trained CNNs.

| Method | Resolution | GAN | Inception Score |
|---|---|---|---|
| BigGAN [5] | 256 | ✓ | 178.0 / 202.6$^{+}$ |
| DeepInversion (**Ours**) | 224 | | 60.6 |
| SAGAN [75] | 128 | ✓ | 52.5 |
| SNGAN [43] | 128 | ✓ | 35.3 |
| WGAN-GP [16] | 128 | ✓ | 11.6 |
| DeepDream [46]* | 224 | | 6.2 |

Table 3: Inception Score (IS) obtained by images synthesized by various methods on ImageNet. SNGAN ImageNet score from [60]. *: our implementation. $^{+}$: BigGAN-deep.

#### 4.2.1 Analysis of DeepInversion Images

Fig. 5 shows a set of images generated by applying DeepInversion to an ImageNet-pretrained ResNet-50. Remarkably, given just the model, we observe that DeepInversion is able to generate images with high fidelity and resolution. It also produces detailed image features and textures around the target object, *e.g.*, clouds surrounding the target balloon, water around a catamaran, forest below the volcano, *etc*.

**Generalizability.** In order to verify that the generated images do not overfit to just the model used for inversion, we obtain predictions using 4 other networks trained on ImageNet. As seen in Table 2, images generated using a ResNet-50 generalize to other models and are correctly classified by a range of models. Further, DeepInversion outperforms DeepDream by a large margin. This indicates robustness of our generated images while transferring across networks.

**Inception Score (IS).** We also compare the IS [58] of our generated images with other methods in Table 3. Again, DeepInversion substantially outperforms DeepDream by an improvement of 54.2 on IS. Without sophisticated training, DeepInversion even beats multiple GAN baselines that have limited scalibility to high image resolutions.

### 4.3. Application I: Data-free Pruning

In this section, we focus on the application of data-free pruning. The goal of pruning is to remove individual weights or entire filters (neurons) from a network such that the metric of interest (*e.g.* accuracy, precision) does not drop significantly. Many techniques have been proposed to successfully compress neural networks including [17, 28, 32, 35, 44, 45, 69, 72]. All these methods require images from the original dataset to perform pruning. We build upon the pruning method of Molchanov *et al*. [44], which uses the Taylor approximation of the pruning loss for a global ranking of filter importance over all the layers. We extend this method by applying the Kullback-Leibler divergence loss, computed between the teacher and student output distributions. Filter importance is estimated from images inverted with DeepInversion and/or Adaptive DeepInversion, making such a pruning completely data-free.

**Hardware-aware Loss.** We further consider actual latency on the target hardware for a more efficient pruning. To achieve this goal, the importance ranking of filters needs to reflect not only accuracy but also latency. This impact can


(a) DeepInversion (b) ADI

Figure 6: 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.

be quantified by:

$$\mathcal{I}_{\mathcal{S}}(\mathbf{W}) = \mathcal{I}_{\mathcal{S},err}(\mathbf{W}) + \eta\, \mathcal{I}_{\mathcal{S},lat}(\mathbf{W}), \qquad (10)$$

where $\mathcal{I}_{\mathcal{S},err}$ and $\mathcal{I}_{\mathcal{S},lat}$ 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 $\mathbf{W}$. $\eta$ balances their importance. Pruning in layer-wise groups, *e.g.*, $2^n$ filters per group, conforms better to common digital architecture design routines. Akin to [44], $\mathcal{I}_{\mathcal{S},err}$ can be approximated by the sum of the first-order Taylor expansion of individual filters. We approximate latency reduction term, $\mathcal{I}_{\mathcal{S},lat}$, via precomputed hardware-aware look-up tables of operation costs. We include details of latency estimation in supplementary materials. In iterative manner neurons are ranked according to $\mathcal{I}_{\mathcal{S}}(\mathbf{W})$ and the least important ones are removed.

We next implement the method for ResNet-50 pruning. First, we study data-free pruning without the original dataset. Then, we demonstrate the effectiveness of latency-aware loss. Finally, we explore the joint benefits of both techniques.

**Data-free Pruning Evaluation.** For better insights, we study four image sources: **(i)** partial ImageNet with 0.1M original images; **(ii)** unlabeled images from proxy dataset, MS COCO [30] (127k images), and PASCAL VOC [12] (9.9k images) datasets; **(iii)** 100k generated images from the BigGAN-deep [5] model, and a data-free setup with the proposed methods: we first **(iv)** generate 165k images via DeepInversion, and then **(v)** add to the set an additional 24k/26k additional images through two competition rounds of ADI, given pruned students at 61.9%/73.0% top-1 acc.

Results of pruning and fine-tuning are summarized in Table 4. Our approach boosts the top-1 accuracy by more than 54% given inverted images. ADI performs relatively on par with BigGAN. Despite beating VOC, we still observe a gap between synthesized images (ADI and BigGAN) and natural images (MS COCO and ImageNet). Such gap is narrowed down as fewer filters are pruned.

**Impact of Competition.** To 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. 6. 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 efficacy of the approach in capturing distribution as opposed to memorizing inputs. The distribution coverage of the generated images with and without competition loss are shown in supplementary materials.

| Image Source | Top-1 acc. (%) -50% filters -71% FLOPs | Top-1 acc. (%) -20% filters -37% FLOPs |
|---|---|---|
| No finetune | 1.9 | 16.6 |
| Partial ImageNet | | |
| 0.1M images / 0 label | 69.8 | 74.9 |
| Proxy datasets | | |
| MS COCO | 66.0 | 73.8 |
| PASCAL VOC | 54.4 | 70.8 |
| GAN | | |
| Generator, BigGAN | 63.0 | 73.7 |
| Noise (**Ours**) | | |
| DeepInversion (DI) | 55.9 | 72.0 |
| Adaptive DeepInversion (ADI) | 60.7 | 73.3 |

Table 4: ImageNet ResNet-50 pruning results for the knowledge distillation setup, given different types of input images.

| Method | ImageNet data | GFLOPs | Lat. (ms) | Top-1 acc. (%) |
|---|---|---|---|---|
| Base model | ✓ | 4.1 | 4.90 | 76.1 |
| Taylor [44] | ✓ | 2.7 (1.5×) | 4.38 (1.1×) | 75.5 |
| SSS [23] | ✓ | 2.8 (1.5×) | - | 74.2 |
| ThiNet-70 [35] | ✓ | 2.6 (1.6×) | - | 72.0 |
| NISP-50-A [72] | ✓ | 3.0 (1.4×) | - | 72.8 |
| **Ours** | | | | |
| Hardware-Aware (HA) | ✓ | 3.1 (1.3×) | 4.24 (1.2×) | 76.1 |
| ADI (Data-free) | | 2.7 (1.5×) | 4.36 (1.1×) | 73.3 |
| ADI + HA | | 2.9 (1.4×) | 4.22 (1.2×) | 74.0 |

Table 5: ImageNet ResNet-50 pruning comparison with prior work.

**Comparisons with SOTA.** We compare data-free pruning against state-of-the-art methods in Table 5 for the setting of pruning away 20% of filters globally. We evaluate three setups for our approach: (i) individually applying the hardware-aware technique (HA), (ii) data-free pruning technique with DeepInversion and Adaptive DeepInversion (ADI), and (iii) jointly applying both (ADI+HA). First, we evaluate the hardware-aware loss on original dataset, and achieve a 16% faster inference with zero accuracy loss compared to the base model. We also observe simultaneous improvements in inference speed and accuracy over the state-of-the-art pruning baseline [44]. In a data-free setup, we achieve similar post-pruned model performances compared to prior work (which use the original dataset), while completely removing the need for any images/labels. The data-free setup demonstrates a 2.8% loss in accuracy with respect to the base model. A final combination of both data-free and hardware-aware techniques (ADI+HA) closes this gap to only 2.1%, while simultaneously improving the inference speed.

### 4.4. Application II: Data-free Knowledge Transfer

In this section, we show that we can distill information from a teacher network to a student network without using any real images at all. We apply DeepInversion to a ResNet-50v1.5 [52] trained on ImageNet to synthesize im-

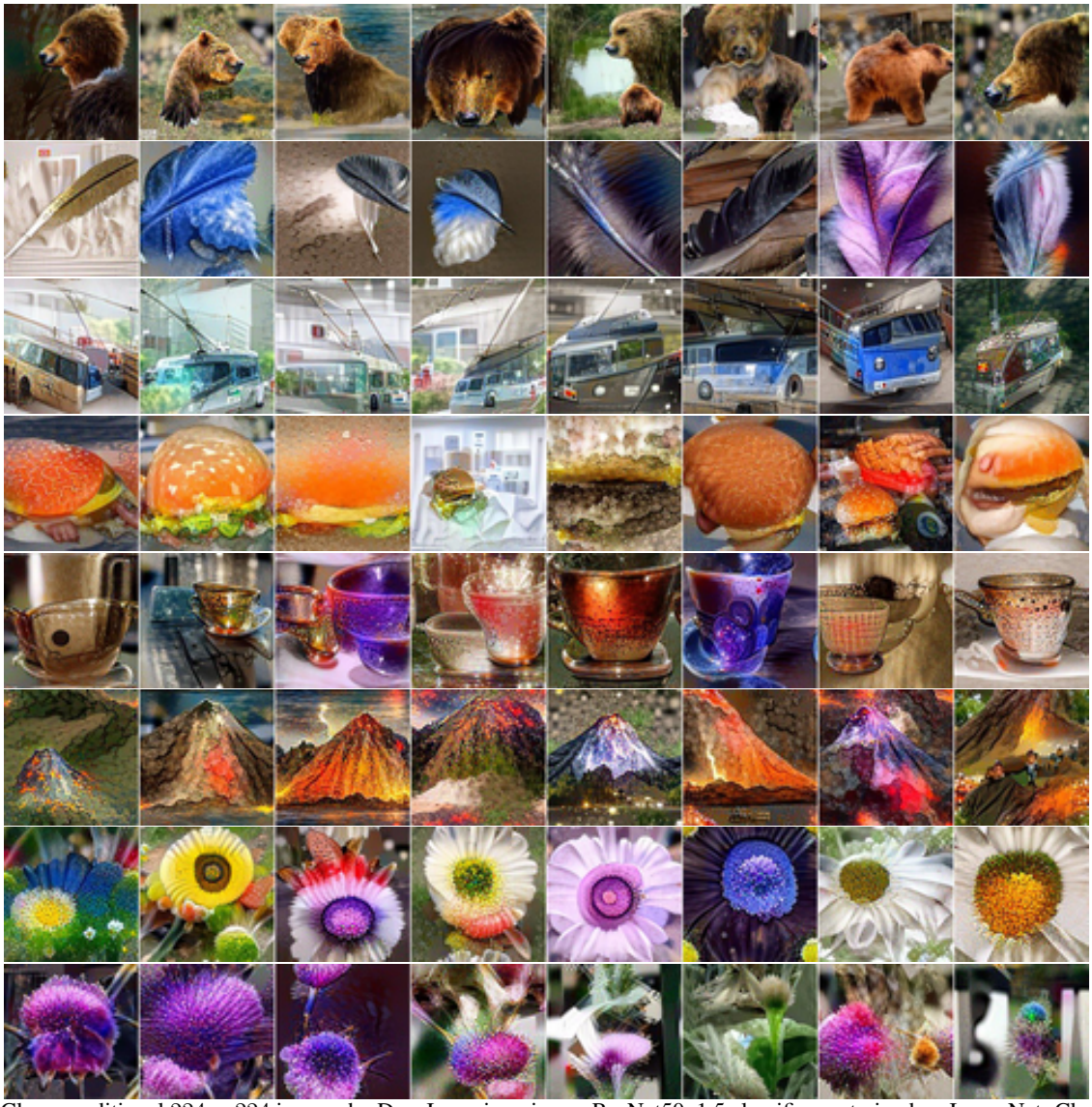


Figure 7: Class-conditional $224 \times 224$ images by DeepInversion given a ResNet50v1.5 classifier pretrained on ImageNet. Classes top to bottom: brown bear, quill, trolleybus, cheeseburger, cup, volcano, daisy, cardoon.

ages. Using these images, we then train another randomly initialized ResNet-50v1.5 from scratch. Below, we describe two practical considerations - a) image clipping, and b) multi-resolution synthesis, which we find can greatly help boost accuracy while reducing run-time.

**a) Image clipping.** We find that enforcing the synthesized images to conform to the mean and variance of the data preprocessing step helps improve accuracy. This intuitively makes sense as maintaining the correct pixel range is important so that the network receives a 'realistic image' as input. Note that commonly released networks, such as the publicly available PyTorch [52] model that we conduct experiments with, use means and variances computed on ImageNet. We would like to emphasize that our method does not require statistics of prior datasets, but merely use those provided with the model, such as the data preprocessing statistics. We clip the synthesized image $\hat{x}$ using

$$\hat{x} \quad = \quad \min\big(\max(\hat{x}, -m/s), (1-m)/s\big), \qquad (11)$$

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

**b) Multi-resolution synthesis.** We find that we can speed up DeepInversion by employing a multi-resolution optimization scheme. We first optimize the input of resolution $112 \times 112$ for 2k iterations. Then, we up-sample the image via nearest neighbour interpolation to $224 \times 224$, and then optimize for an additional 1k iterations. We use the following parameters for this experiment: $\alpha_{\text{tv}} = 1 \cdot 10^{-4}$, $\alpha_f = 0.01$, and a learning rate of $0.2$ for Adam. This implementation speeds up DeepInversion and enables the generation of $84$

| Image source | Real images | Data amount | Top-1 acc. |
|---|---|---|---|
| Base model | ✓ | 1.3M | 77.26% |
| Knowledge Transfer, 90 epochs | | | |
| ImageNet | ✓ | 215k | 76.1% |
| MS COCO | ✓ | 127k | 72.3% |
| Generator, BigGAN | | 215k | 64.0% |
| DeepInversion (DI) | | 140k | 68.0% |
| Knowledge Transfer, 250 epochs, with mixup | | | |
| DeepInversion (DI) | | 140k | 73.8% |

Table 6: Knowledge transfer from the trained ResNet-50v1.5 to the same network initialized from scratch.

images within 6 minutes on an NVIDIA V100 GPU. As a final setup, we generate images with an equal probability between this (i) multi-resolution scheme and (ii) the scheme described in Section 4.2 with 2k iterations only to further improve image diversity.

A set of images generated by DeepInversion on the pretrained ResNet-50v1.5 using the techniques described above are shown in Fig. 7. The images again demonstrate high fidelity and diversity. Clipping helps improve the contrast, while multi-resolution synthesis helps speed-up the process.

**Knowledge Transfer.** We synthesize 140k images via DeepInversion on ResNet50v1.5 [52] to train a student network with the same architecture from scratch. Our teacher is a pretrained ResNet-50v1.5 which achieves 77.26% top-1 accuracy. We apply knowledge distillation for 90/250 epochs, with temperature $\tau = 3$, initial learning rate of 1.024, batch size of 1024 split across eight V100 GPUs, and other settings same as in the original setup [52]. Results are summarized in Table 6. By the proposed method, given only the trained ResNet50v1.5 model, we can teach a new model from scratch to achieve a 73.8% accuracy, which is only 3.46% below the accuracy of the teacher.

### 4.5. Application III: Data-free Continual Learning

In this section, we focus on the task of data-free continual or incremental learning, another scenario that benefits from the image generation capability of DeepInversion. The main idea of incremental learning is to add new classification classes to a pretrained model without comprehensive access to its original training data. To the best of our knowledge, only LwF [29] and LwF.MC [56] achieve data-free incremental learning. Other methods require information that can only be obtained from the original dataset, *e.g.*, a subset of data (iCaRL [56]), parameter importance estimations (in the form of Fisher matrix in EWC [27], contribution to loss change in SI [74], posterior of network weights in VCL [50]), or training data representation (encoder [62], GAN [21, 59]). Some methods rely on network modifications *e.g.* Packnet [40] and Piggyback [39]. In comparison, DeepInversion does not need network modifications or the original (meta-)data, as BatchNorm statistics are inherent to neural networks.

In this task setting, a network is initially trained on a dataset with classes $\mathcal{C}_o$, *e.g.* ImageNet [10]. Given new class

| Method | Top-1 acc. (%) Combined | ImageNet | CUB | Flowers |
|---|---|---|---|---|
| ImageNet + CUB (1000 → 1200 outputs) | | | | |
| LwF.MC [56] | 47.64 | 53.98 | 41.30 | – |
| DeepDream [46] | 63.00 | 56.02 | **69.97** | – |
| DeepInversion (**Ours**) | **67.61** | **65.54** | 69.68 | – |
| Oracle (distill) | 69.12 | 68.09 | 70.16 | – |
| Oracle (classify) | 68.17 | 67.18 | 69.16 | – |
| ImageNet + Flowers (1000 → 1102 outputs) | | | | |
| LwF.MC [56] | 67.23 | 55.62 | – | 78.84 |
| DeepDream [46] | 79.84 | 65.69 | – | **94.00** |
| DeepInversion (**Ours**) | **80.85** | **68.03** | – | 93.67 |
| Oracle (distill) | 80.71 | 68.73 | – | 92.70 |
| Oracle (classify) | 79.42 | 67.59 | – | 91.25 |
| ImageNet + CUB + Flowers (1000 → 1200 → 1302 outputs) | | | | |
| LwF.MC [56] | 41.72 | 40.51 | 26.63 | 58.01 |
| DeepInversion (**Ours**) | **74.61** | **64.10** | **66.57** | **93.17** |
| Oracle (distill) | 71.85 | 67.66 | 61.60 | 86.27 |
| Oracle (classify) | 76.94 | 66.25 | 77.80 | 86.76 |

Table 7: Continual learning results extending the network output space, adding new classes to ResNet-18. Accuracy over *combined* classes $\mathcal{C}_o \cup \mathcal{C}_k$ reported on individual datasets. Average over datasets also shown (datasets treated equally regardless of size, so ImageNet samples have less weight than CUB or Flowers samples).

data $(x_k, y_k)$, $y_k \in \mathcal{C}_k$, *e.g.* from the CUB [65] dataset, the pretrained model is now required to make predictions in a combined output space $\mathcal{C}_o \cup \mathcal{C}_k$. Similar to prior work, we start from a trained network $\big($denoted by $p_o(\cdot)$, effectively as a *teacher*$\big)$, make a copy $\big($denoted by $p_k(\cdot)$, effectively as a *student*$\big)$, and then add new randomly initialized neurons to $p_k(\cdot)$'s final layer to output logits for the new classes. We train $p_k(\cdot)$ to classify simultaneously over all classes, old and new, while network $p_o(\cdot)$ remains fixed.

**Incremental Learning Loss.** We formulate a new loss with DeepInversion images as follows. We use same-sized batches of DeepInversion data $(\hat{x}, p_o(\hat{x}))$ and new class real data $(x_k, y_k)$ for each training iteration. For $\hat{x}$, we use the original model to compute its soft labels $p_o(\hat{x})$, *i.e.* class probability among old classes, and then concatenate it with additional zeros as new class probabilities. We apply a KL-divergence loss between predictions $p_o(\hat{x})$ and $p_k(\hat{x})$ on DeepInversion images for prior memory, and a cross-entropy loss between one-hot $y_k$ and prediction $p_k(x_k)$ on new class images for emerging knowledge. Similar to prior work [29, 56], we also apply a third KL-divergence term between the new class images' old class predictions $p_k(x_k|y \in \mathcal{C}_o)$ and their original model predictions $p_o(x_k)$. This forms the loss

$$\begin{aligned}\mathcal{L}_{\text{IL}} =& \text{KL}\big(p_o(\hat{x}), p_k(\hat{x})\big) + \mathcal{L}_{xent}\big(y_k, p_k(x_k)\big) \\ &+ \text{KL}\big(p_o(x_k|y \in \mathcal{C}_o), p_k(x_k|y \in \mathcal{C}_o)\big).\end{aligned} \tag{12}$$

**Evaluation Results.** We add new classes from the CUB [65], Flowers [51], and both CUB and Flowers datasets to a ResNet-18 [18] classifier trained on ImageNet [10]. Prior to each step of addition of new classes, we generate 250 DeepInversion images per old category. We compare with prior work LwF.MC [56] as opposed to LwF [29] that

cannot distinguish between old and new classes. We further compare with oracle methods where we break the data-free constraint: we use the same amount of old class real images from old datasets in place of $\hat{x}$, jointly with either their ground truth label classification loss or their soft labels from $p_o(\cdot)$ for KL-divergence distillation loss. The third KL-divergence term in Eq. 12 is omitted in this case. Implementation details are in the appendix.

Results are shown in Table 7. Our method outperforms LwF.MC drastically in all cases and leads to consistent performance improvements over DeepDream in most scenarios. We are very close to the oracles (and occasionally outperforming them), showing DeepInversion's efficacy in replacing ImageNet images for incremental learning. We verify that our observations also hold for VGG-16 in the appendix.

## Conclusions

We have proposed DeepInversion for synthesizing training images with high resolution and fidelity given just a trained CNN. Further, by using student-in-the-loop, our Adaptive DeepInversion improves image diversity. Through extensive experiments, we have shown that our methods are generalizable, effective, and have empowered a new class of "data-free" tasks of immense practical significance.

## Acknowledgements

We would like to thank Arash Vahdat and Ming-Yu Liu for in-depth discussions and helpful comments.

## References

[1] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai. Variational information distillation for knowledge transfer. In *CVPR*, 2019. 2

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NeurIPS*, 2014. 2

[3] K. Bhardwaj, N. Suda, and R. Marculescu. Dream distillation: A data-independent model compression framework. In *ICML Workshop*, 2019. 2, 3

[4] L. Breiman and N. Shang. Born again trees. Technical report, UC Berkeley, 1996. 2

[5] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 3, 4, 6, 7

[6] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. In *SIGKDD*, 2006. 2

[7] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, and Q. Tian. Data-free learning of student networks. In *ICCV*, 2019. 2, 5

[8] T. Chen, I. Goodfellow, and J. Shlens. Net2net:Accelerating learning via knowledge transfer. In *ICLR*, 2016. 1

[9] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha. ChamNet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019. 1, 13, 14

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 5, 9

[11] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016. 3

[12] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *IJCV*, 2010. 7

[13] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCCS*, 2015. 3

[14] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born again neural networks. In *ICML*, 2018. 2

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 4

[16] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *NeurIPS*, 2017. 3, 6

[17] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *ICLR*, 2016. 6

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 9

[19] Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In *ACSAC*, 2019. 3

[20] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 2, 3

[21] W. Hu, Z. Lin, B. Liu, C. Tao, J. Tao, J. Ma, D. Zhao, and R. Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. In *ICLR*, 2019. 9

[22] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2

[23] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018. 7

[24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 2, 4

[25] A. Kimura, Z. Ghahramani, K. Takeuchi, T. Iwata, and N. Ueda. Few-shot learning of neural networks from scratch by pseudo example optimization. In *BMVC*, 2018. 1

[26] A. Kimura, Z. Ghahramani, K. Takeuchi, T. Iwata, and N. Ueda. Few-shot learning of neural networks from scratch by pseudo example optimization. In *BMVC*, 2018. 1

[27] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. In *Proc. National Academy Sciences*, 2016. 1, 9, 15

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *ICLR*, 2017. 6

[29] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2017. 1, 9

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 7

[31] Y. Liu, K. Chen, C. Liu, Z. Qin, Z. Luo, and J. Wang. Structured knowledge distillation for semantic segmentation. In *CVPR*, 2019. 2

[32] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *CVPR*, 2017. 6

[33] R. G. Lopes, S. Fenu, and T. Starner. Data-free knowledge distillation for deep neural networks. *arXiv preprint arXiv:1710.07535*, 2017. 2

[34] D. Lopez-Paz, L. Bottou, B. Schölkopf, and V. Vapnik. Unifying distillation and privileged information. In *ICLR*, 2016. 1, 2

[35] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017. 6, 7

[36] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018. 1

[37] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015. 2, 3

[38] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016. 3

[39] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, 2018. 9

[40] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018. 9

[41] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. In *ICLR*, 2018. 5

[42] A. Mishra and D. Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *ICLR*, 2018. 2

[43] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018. 3, 6

[44] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019. 1, 6, 7, 13, 14

[45] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. In *ICLR*, 2017. 1, 6

[46] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks, 2015. 2, 3, 5, 6, 9

[47] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *CVPR*, 2017. 3

[48] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *NeurIPS*, 2016. 3

[49] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, 2015. 2, 3, 5

[50] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. In *ICLR*, 2018. 1, 9

[51] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *ICCVGIP*, 2008. 9

[52] NVIDIA. ResNet50v1.5 training. https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/RN50v1.5, 2019. [Online; accessed 10-Nov-2019]. 7, 8, 9

[53] W. Park, D. Kim, Y. Lu, and M. Cho. Relational knowledge distillation. In *CVPR*, 2019. 2

[54] A. Pilzer, S. Lathuiliere, N. Sebe, and E. Ricci. Refine and distill: Exploiting cycle-inconsistency and knowledge distillation for unsupervised monocular depth estimation. In *CVPR*, 2019. 2

[55] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. In *ICLR*, 2018. 2

[56] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, 2017. 1, 9, 15

[57] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for thin deep nets. In *ICLR*, 2015. 1, 2

[58] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *NeurIPS*, 2016. 6

[59] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017. 9

[60] K. Shmelkov, C. Schmid, and K. Alahari. How good is my GAN? In *ECCV*, 2018. 6

[61] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 3

[62] A. Triki Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars. Encoder based lifelong learning. In *ICCV*, 2017. 9

[63] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019. 1

[64] Y. Wang, C. Si, and X. Wu. Regression model fitting under differential privacy and model inversion attack. In *IJCAI*, 2015. 3

[65] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical report, Caltech, 2010. 9

[66] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *CVPR*, 2019. 1, 13, 14

[67] Z. Xu, Y.-C. Hsu, and J. Huang. Training shallow and thin networks for acceleration via knowledge distillation with conditional adversarial networks. In *ICLR Workshop*, 2018. 2

[68] Z. Yang, E.-C. Chang, and Z. Liang. Adversarial neural network inversion via auxiliary knowledge alignment. *arXiv preprint arXiv:1902.08552*, 2019. 3

[69] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018. 6

[70] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*, 2017. 2

[71] H. Yin, G. Chen, Y. Li, S. Che, W. Zhang, and N. K. Jha. Hardware-guided symbiotic training for compact, accurate, yet execution-efficient LSTM. *arXiv preprint arXiv:1901.10997*, 2019. 1

[72] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018. 6, 7

[73] S. Zagoruyko and N. Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*, 2017. 2

[74] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017. 1, 9

[75] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *ICML*, 2019. 3, 6

[76] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *ICLR*, 2017. 1

## 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) and data-free incremental 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 BatchNorm 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 BatchNorms. It only requires 100 images to compute feature statistics for Adaptive DeepInversion to achieve almost the same accuracy as with BatchNorm 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 |
| BatchNorm 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: BatchNorm 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 $\ell_2$ 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, 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

$$\mathcal{I}_{\mathcal{S},err}(\mathbf{W}) \approx \sum_{s\in\mathcal{S}} \mathcal{I}_s^{(1)}(\mathbf{W}) = \sum_{s\in\mathcal{S}} (g_s w_s)^2, \qquad (13)$$

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

$$\mathcal{I}_{\mathcal{S},lat} = \text{LAT}(\mathbf{W}|w_s = 0, s \in \mathcal{S}) - \text{LAT}(\mathbf{W}), \qquad (14)$$

where $\text{LAT}(\cdot)$ measures 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:

$$\text{LAT}(\mathbf{W}) = \sum_l \text{LAT}(o_l^{(\mathbf{W})}), \qquad (15)$$

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 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 operations given *reduced numbers* of filters against 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 $\tau$ among $\{5, 10, 15\}$ for each pruning case and report the highest validation accuracy observed. We profile and store latency values in the LUT under the following format:

$$key = [c_{in}, c_{out}, kernel^*, stride^*, fmap^*], \qquad (16)$$

where $c_{in}, c_{out}, kernel, stride, fmap$ 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 one 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. 15 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 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, the default ResNet pre-processing from PyTorch is applied on MS COCO and PASCAL VOC images.

### C.3. Hard-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 same latency constraints, the proposed loss improves the top-1 accuracy by absolute $0.5\% - 14.8\%$.

### C.4. Pruning without Labels

Taylor expansion for pruning estimates the change in loss induced by feature map removal. Originally, it was proposed for cross-entropy (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 Kullback-Leibler (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 one. Top-1 accuracy corresponds to the highest validation accuracy for 15 training epochs. Learning rate is initialized to 0.01, decayed at the $10^{\text{th}}$ 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.

### C.5. Distribution Coverage Expansion

The proposed 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. 8. 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. 8, we observe varying ranges and peaks after projection onto each principal component from the three image generation rounds.

## D. Data-free Incremental Learning Appendix

### D.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 $224 \times 224$ per existing class in the pretrained {ResNet-18, VGG-16-BN}. These

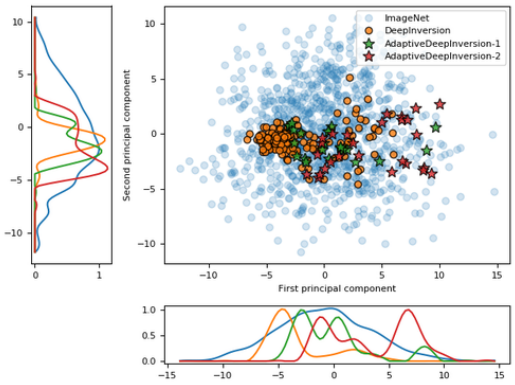



Figure 8: Projection onto the first two principal components of the ResNet-50-avgpool feature vectors of ImageNet class 'hand-held computer' training images. ADI-1/2 refers to additional images from round1/2 competition.

images are generated afresh after adding each new dataset. For {ResNet18, 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 $\ell_2$ and $\ell_1$ norms for total variance regularization at $\alpha_{\text{tv},\ell_2} = \{3\cdot10^{-5},\ 6\cdot10^{-5}\}$, $\alpha_{\text{tv},\ell_1} = \{1\cdot10^{-1}, 2\cdot10^{-1}\}$, joint with $\alpha_{\ell_2} = 0$ and $\alpha_f = \{1\cdot10^{-1},\ 3\cdot10^{-2}\}$ 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 ResNet18, 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 SGD with a momentum of 0.9 as the optimizer. We clip the gradient $\ell_2$ magnitude to 0.1, and disable all updates in the BatchNorm layers. Gradient clipping and freezing BatchNorm do not affect the baseline LwF.MC [56] much ($\pm2\%$ 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 cross entropy loss. Hence, we fine-tuned the model with a small learning rate. The resulting ImageNet model is within $\pm0.5\%$ top-1 error of the original model.

## D.2. VGG-16-BN Results

We show our data-free incremental 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 incremental learning task.

| 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.

## D.3. Use Case and Novelty

The most significant departure from prior work such as EWC [27] is that our DeepInversion-based incremental learning can operate on *any* regularly-trained model, given the widespread usage of BatchNorm layers. Our method eliminates the 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 incremental learning, or (3) does not have the know-how to support a downstream incremental 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.