DATASET DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

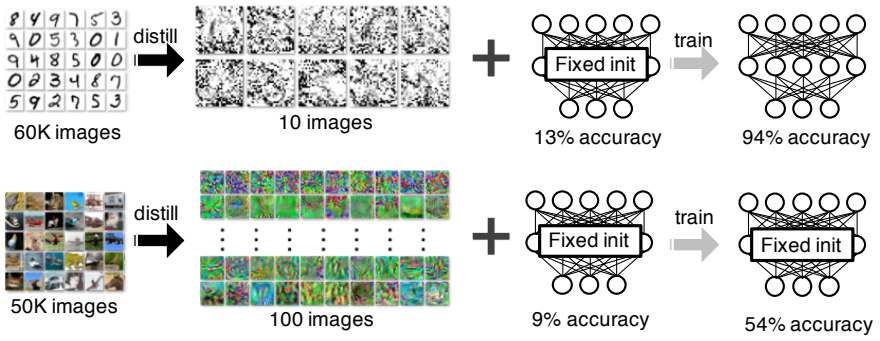
Model distillation aims to distill the knowledge of a complex model into a simpler one. In this paper, we consider an alternative formulation called *dataset distillation*: we keep the model fixed and instead attempt to distill the knowledge from a large training dataset into a small one. The idea is to *synthesize* a small number of data points that do not need to come from the correct data distribution, but will, when given to the learning algorithm as training data, approximate the model trained on the original data. For example, we show that it is possible to compress $60,000$ MNIST training images into just 10 synthetic *distilled images* (one per class) and achieve close to original performance with only a few steps of gradient descent, given a particular fixed network initialization. Apart from being an interesting new way to think about distillation, this approach could potentially open up several applications, such as fast domain adaptation and effective data poisoning attacks.

1 INTRODUCTION

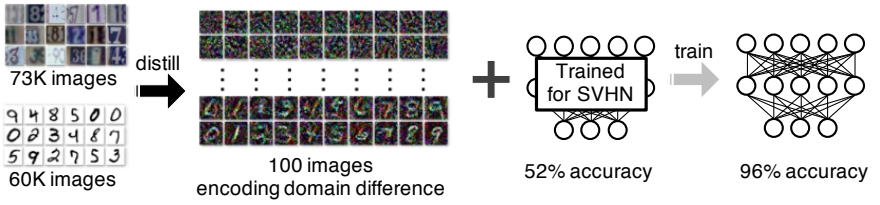
Hinton et al. (2015) proposed *network distillation* as a way to transfer the knowledge from an ensemble of many separately-trained networks into a single, typically compact network, performing a type of model compression. In this paper, we are considering a related but orthogonal task: rather than distilling the model, we propose to distill the dataset. Unlike network distillation, we keep the model fixed but encapsulate the knowledge of the entire training dataset, which typically contains thousands to millions of images, into a small number of *synthetic* training images. In fact, we show that we can go as low as *one* synthetic image per category, training the same model to reach surprisingly good performance on these synthetic images. For example in Fig. 1a, we compress $60,000$ training images of MNIST digit dataset into only 10 synthetic images (one per class), given a fixed network initialization. Training the standard LENET (LeCun et al., 1998) architecture on these 10 images yields test-time MNIST recognition performance of 94%, compared to 99% for the original task. For networks with unknown random weights, 100 synthetic images train to 77% with a few gradient descent steps. We name our method *Dataset Distillation* and these images *distilled images*.

But why is dataset distillation useful? First, there is the purely scientific question of how much data is really encoded in a given training set and how compressible it is? Second, we wish to know whether it is possible to "load up" a given network with an entire dataset-worth of knowledge by a few gradient descent steps on a handful of images. Third, on the practical side, if dataset distillation is successful, it may enable new applications such as fast domain adaptation (Fig. 1b). Finally, this technique could have repercussions for computer security, as it might allow for effective data poisoning attacks, e.g., presenting a few training images that will force the network to "forget" a category (Fig. 1c).

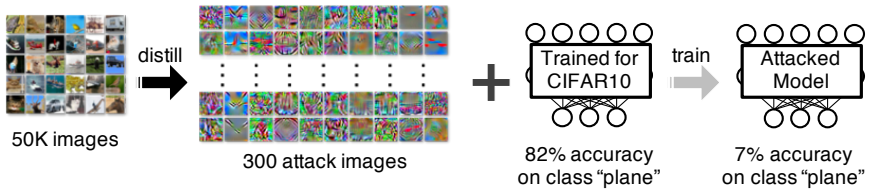
A key question is whether it is even possible to compress a dataset into a small set of synthetic data samples. For example, is it possible to train an image classification model on synthetic images that are not on the natural image manifold? Conventional wisdom would suggest that the answer is no, as the synthetic training data may not follow the same distribution as the real test data. Yet, in this work, we show that this is indeed possible. We present a new optimization algorithm for synthesizing a small number of synthetic data samples that not only capture much of the original training data but also are tailored explicitly to fast model training. To achieve our goal, we first derive the network weights as a differentiable function of our synthetic training data. Given this connection, instead of optimizing the network weights for a specific training objective, we can optimize the pixel values of our distilled images. However, this formulation requires access to the initial network weights. To relax this assumption, we develop a method for generating distilled images effective for networks with random initializations from a certain distribution. To further boost the performance, we propose



(a) Dataset distillation on MNIST and CIFAR10



(b) Application: fast domain adaptation for SVHN → MNIST



(c) Application: fast data poisoning attack for CIFAR10

Figure 1: Dataset Distillation: we can distill the knowledge of tens of thousands of images into a few synthetic training images called distilled images. (a): On MNIST, 10 distilled images can train a standard LeNet with fixed initialization to 94% test accuracy (compared to 99% when fully trained). On CIFAR10, 100 distilled images can train a deep network with fixed initialization to 54% test accuracy (compared to 80% when fully trained). (b): Utilizing pre-trained networks for SVHN, we can distill the domain difference between two SVHN and MNIST into 100 distilled images. These images can be used to quickly adapt a network trained for SVHN to achieve high accuracy on MNIST. (c): Our formulation can be used to create adversarial attack images. If a well-optimized network is trained with these images for **one single** gradient step, it will catastrophically misclassify a particular targeted class.

an iterative version, where we obtain a sequence of distilled images to train a model and each distilled image can be trained with multiple passes. Finally, we study the case of a simple linear model, deriving a lower bound on the size of distilled data required to achieve the same performance as training on full dataset.

Experiments on multiple datasets demonstrate that a handful of distilled images can be used to train a model with a fixed initialization to achieve surprisingly high performance. For a network with random initializations (e.g., different weights pre-trained on the same task), we can still find distilled images for fast model fine-tuning with our iterative method. Built on this main result, we show two practical applications including fast domain adaptation and fast data poisoning attack. We systematically compare our method with existing baselines and alternative approaches. Our code and models will be available upon publication.

2 Related Work

Knowledge Distillation The main inspiration for this paper is network distillation (Hinton et al., 2015), a widely used technique in ensemble learning (Radosavovic et al., 2018) and model compression (Ba & Caruana, 2014; Romero et al., 2015; Howard et al., 2017). While network distillation

aims to distill the knowledge of multiple networks into a single model, our goal is to compress the knowledge of an entire dataset into a few synthetic training images. Another way to distill knowledge is to summarize the entire dataset by a small subset, either by only using the "valuable" data for model training (Angelova et al., 2005; Lapedriza et al., 2013; Felzenszwalb et al., 2010) or by only labeling the "valuable" data via active learning (Cohn et al., 1996; Tong & Koller, 2001). However, these algorithms require many more training examples per category than we do, in part because their "valuable" images have to be real, whereas our distilled images are exempt from this constraint.

Gradient-based Hyperparameter Optimization Our work bears similarity with the gradient-based hyperparameter optimization techniques, which compute the gradient of hyperparameter w.r.t. the final validation loss by reversing the entire training procedure (Bengio, 2000; Domke, 2012; Pedregosa, 2016; Maclaurin et al., 2015). We also backpropagate errors through optimization steps. However, we use only training set data and focus much more heavily on learning synthetic training data rather than training hyperparameters. To our knowledge, this direction has only been slightly touched on previously (Maclaurin et al., 2015). We explore it in much greater depth and demonstrate the idea of dataset distillation through various applications. More crucially, our distilled images can work well across random initialization weights, which cannot be achieved by any previous work.

Understanding Datasets Researchers have presented various approaches for understanding and visualzing learned models (Zeiler & Fergus, 2014; Zhou et al., 2015; Mahendran & Vedaldi, 2015; Bau et al., 2017; Koh & Liang, 2017). Unlike these approaches, we are interested in understanding the intrinsic properties of the training data rather than a specific trained model. Analyzing training datasets has, in the past, been mainly focused on the investigation of bias in datasets (Ponce et al., 2006; Torralba & Efros, 2011). For example, Torralba & Efros (2011) proposed to quantify the "value" of dataset samples using cross-dataset generalization. Our method offers a new perspective for understanding datasets. We often find that the number of distilled images required to achieve good performance is an informative indicator of the dataset diversity.

3 Approach

Given a model and a dataset, we aim to obtain a new, much-reduced *synthetic* dataset which performs almost as well as the original dataset. We first present our main optimization algorithm for training a network with a fixed initialization with one gradient descent (GD) step (Sec. 3.1). In Sec. 3.2, we derive the resolution to a more challenging case, where the initial weight is random rather than fixed. We also discuss the initial weights distribution where our method can work well. Furthermore, we study a linear network case to help the readers understand both the solution and limits of our method in Sec. 3.3. In Sec. 3.4, we extend our approach to more than one gradient descent steps and more than one passes. Finally, Sec. 3.5 demonstrates two applications enabled by our method.

Consider a training dataset $\mathbf{x} = \{x_i\}_{i=1}^N$, we parameterize our neural network as θ and denote $\ell(x_i, \theta)$ as the loss function that represents the loss of this network on a data point x_i. Our task is to find the minimizer of the empirical error over the entire training data:

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^N \ell(x_i, \theta) = \arg\min_\theta \ell(\mathbf{x}, \theta), \tag{1}$$

where for notation simplicity we overload the $\ell(\cdot)$ notation so that $\ell(\mathbf{x}, \theta)$ represents the average error of θ over the entire dataset $\mathbf{x} = \{x_i\}_{i=1}^N$. We make the mild assumptions that ℓ is twice-differentiable, which holds for majority of modern machine learning models (e.g., most neural networks) and tasks.

3.1 Optimizing Distilled Data

Standard training usually applies minibatch stochastic gradient descent (SGD) or its variants. At each step t, we sample a minibatch of training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^n$ and update the current parameters as

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \ell(\mathbf{x}_t, \theta_t),$$

where η is the learning rate. Such a training process often takes tens of thousands or even millions of above update steps to converge. Instead, we aim to learn a tiny set of synthetic distilled training data $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^M$ with $M \ll N$ and a corresponding learning rate $\tilde{\eta}$ so that a single GD step like

$$\theta_1 = \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0) \tag{2}$$

Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial parameters
Input: α: step size; n: batch size; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$
Input: T: the number of optimization iterations; M: the number of distilled data
1: Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
2: **for each** training step $t = 1$ to T **do**
3: Get a minibatch of real data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^{n}$
4: Sample a batch of initial parameters $\theta_0^{(j)} \sim p(\theta_0)$
5: **for each** sampled $\theta_0^{(j)}$ **do**
6: Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta}\nabla_{\theta_0^{(j)}}\ell(\tilde{\mathbf{x}}, \theta_0^{(j)})$
7: Evaluate on real data: compute $\ell(\mathbf{x}_t, \theta_1^{(j)})$
8: **end for**
9: Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha\nabla_{\tilde{\mathbf{x}}}\sum_j \ell(\mathbf{x}_t, \theta_1^{(j)})$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha\nabla_{\tilde{\eta}}\sum_j \ell(\mathbf{x}_t, \theta_1^{(j)})$
10: **end for**
Output: distilled data $\tilde{\mathbf{x}}$ and the optimized learning rate $\tilde{\eta}$

using these learned synthetic data $\tilde{\mathbf{x}}$ greatly boosts performance on the real training dataset.

Given an initialization θ_0, we obtain these synthetic data and $\tilde{\eta}$ that minimize the below objective $\mathcal{L}$:

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}},\tilde{\eta}} \mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0) = \arg\min_{\tilde{\mathbf{x}},\tilde{\eta}} \ell(\mathbf{x}, \theta_1) = \arg\min_{\tilde{\mathbf{x}},\tilde{\eta}} \ell(\mathbf{x}, \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0)), \tag{3}$$

where we derive the new weights θ_1 as a function of distilled images $\tilde{\mathbf{x}}$ and learning rate $\tilde{\eta}$ using Eqn. 2 and then evaluate the new weights over all the training images $\mathbf{x}$. Note that the loss $\mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0)$ is differentiable w.r.t. to $\tilde{\mathbf{x}}$ and $\tilde{\eta}$, and can thus be optimized using standard gradient-based algorithms. In many classification tasks, the data $\mathbf{x}$ may contain discrete parts, e.g., the class labels in data-label pairs. For such cases, we fix the discrete part rather than learn them.

3.2 Distilled Data for Random Initializations

Unfortunately, the above distilled data optimized for a given initialization do not generalize well to other initialization weights. The distilled data often look like random noise (e.g., in Fig. 2a) as it encodes the information of both training dataset $\mathbf{x}$ and a particular network initialization θ_0. To address the above issue, we turn to calculate a small number of distilled data that can work for networks with random initializations from a specific distribution. We formulate the optimization problem as follows:

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}},\tilde{\eta}} \mathbb{E}_{\theta_0\sim p(\theta_0)}\mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0), \tag{4}$$

where θ_0 is a randomly sampled network initialization from the distribution $p(\theta_0)$: e.g., different networks pre-trained on the same task. Algorithm 1 illustrates our main method. During optimization, the distilled data are optimized to work well for multiple networks whose initial weights are sampled from $p(\theta_0)$. In practice, we observe that the final distilled data generalize well to the unseen initializations. Besides, these distilled images usually look quite informative, encoding discriminative features of each category (Fig. 3).

For distilled data to be properly learned, it turns out to be crucial for $\ell(\mathbf{x}, \cdot)$ to share similar local conditions (e.g., output values, gradient magnitudes) over θ_0 sampled from $p(\theta_0)$. In the next section, we derive a lower bound on the amount of distilled data needed for a simple model with arbitrary initial θ_0, and discuss its implications on choosing $p(\theta_0)$.

3.3 Analysis of a Simple Linear Case with Quadratic Loss

This section studies our formulation in a simple linear regression case. We derive the lower bound of the number of distilled images needed to achieve the same performance as training on full dataset for arbitrary initialization with one GD step. Consider a dataset $\mathbf{x}$ containing N data-target pairs $\{(d_i, t_i)\}_{i=1}^{N}$, where $d_i \in \mathbb{R}^D$ and $t_i \in \mathbb{R}$, which we represent as two matrices: an $N \times D$ data matrix $\mathbf{d}$ and an $N \times 1$ target matrix $\mathbf{t}$. Given the mean squared error and a $D \times 1$ weight matrix θ, we have

$$\ell(\mathbf{x}, \theta) = \ell((\mathbf{d}, \mathbf{t}), \theta) = \frac{1}{2N}\|\mathbf{d}\theta - \mathbf{t}\|^2. \tag{5}$$

We aim to learn M synthetic data-target pairs $\tilde{\mathbf{x}} = (\tilde{\mathbf{d}}, \tilde{\mathbf{t}})$, where $\tilde{\mathbf{d}}$ is an $M \times D$ matrix, $\tilde{\mathbf{t}}$ an $M \times 1$ matrix ($M \ll N$), and $\tilde{\eta}$ the learning rate, to minimize $\ell(\mathbf{x}, \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0))$. The updated weight matrix after one GD step with these distilled data is

$$\theta_1 = \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0) = \theta_0 - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T(\tilde{\mathbf{d}}\theta_0 - \tilde{\mathbf{t}}) = (\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}})\theta_0 + \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{t}}. \tag{6}$$

Suppose some learned distilled data $\tilde{\mathbf{x}}$ allows us to achieve the same performance as training on full dataset $\mathbf{x}$ (i.e., attaining the global minimum) for *any* initialization θ_0. For such models, global minimum is attained at any θ^* satisfying $\mathbf{d}^T\mathbf{d}\theta^* = \mathbf{d}^T\mathbf{t}$. Substituting Eqn. (6) in, we have

$$\mathbf{d}^T\mathbf{d}(\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}})\theta_0 + \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{t}} = \mathbf{d}^T\mathbf{t}. \tag{7}$$

Here we make the mild assumption that the feature columns of the data matrix $\mathbf{d}$ are independent (i.e., $\mathbf{d}^T\mathbf{d}$ has full rank). For a $\tilde{\mathbf{x}} = (\tilde{\mathbf{d}}, \tilde{\mathbf{t}})$ to satisfy the above equation for any θ_0, we must have

- $\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}} = \mathbf{0}$, which implies that $\tilde{\mathbf{d}}^T\tilde{\mathbf{d}}$ has full rank and $M \geq D$, and
- $\frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{t}} = \mathbf{d}^T\mathbf{t}$, where $\tilde{\mathbf{t}}$ projected onto $\tilde{\mathbf{d}}$ is equal to $\mathbf{t}$ projected onto $\mathbf{d}$, up to a factor.

Discussion The above analysis considers only a simple case, but it suggests that the distilled data seems to try to cover a (potentially transformed) subspace spanned by the training data. Moreover, we also showed that in this case, a small number of distilled data fails to generalize to arbitrary starting θ_0. This is intuitively expected as the optimization target $\ell(\mathbf{x}, \theta_1) = \ell(\mathbf{x}, \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0))$ depends on the local behavior of $\ell(\mathbf{x}, \cdot)$ around θ_0, which can be drastically different across various θ_0 values. We note that the lower bound $M \geq D$ is a quite restricting one, considering that real datasets often have thousands to even hundreds of thousands of dimensions (e.g., image classification). This motivates us to focus on $p(\theta)$ distributions that yield similar local conditions over the support, e.g., network weights initialized using methods that attempts to ensure gradient flow of constant magnitude (He et al., 2015; Glorot & Bengio, 2010), and network weights pre-trained on other tasks. Sec. 4.1 and Sec. 4.2 explore both options. Additionally, to address the limitation of using single GD step, we extend our method to multiple GD steps in the next section. In Sec. 4.1, we empirically verify that using multiple steps is much more effective than using a single step on deep convolutional networks, with total amount of distilled data fixed.

3.4 Multiple Gradient Descent Steps and Multiple Epochs

We can extend Algorithm 1 to more than one gradient descent steps by changing Line 6 to multiple sequential GD steps each on a different batch of distilled data and learning rate, i.e., each step i is

$$\theta_{i+1} = \theta_i - \tilde{\eta}_i\nabla_{\theta_i}\ell(\tilde{\mathbf{x}}_i, \theta_i), \tag{8}$$

and changing Line 9 to backpropagate through all steps. However, naively computing gradients is both memory-intensive and computationally-expensive. Therefore, we exploit a recent technique called *back-gradient optimization*, which allows for significantly faster gradient calculation of such updates in *reverse-mode differentiation* (i.e., backpropagation). Specifically, back-gradient optimization formulates the necesary second order terms into efficient Hessian-vector products (Pearlmutter, 1994), which can be easily calculated with modern automatic differentiation systems such as PyTorch (Paszke et al., 2017). For further algorithm details in this aspect, we refer readers to prior work (Domke, 2012; Maclaurin et al., 2015).

Multiple Epochs To further improve the performance, we can train the network with the same distilled images for multiple passes (epochs). For examples, for image data, we simply tie the image pixels for the same distilled images used in different epochs. Note that we do not tie the trained learning rates across epochs, since later epochs often use smaller learning rates.

3.5 Applications

Here we demonstrate two applications enabled by our main algorithm. We present different objective functions tailored to individual applications.

Fast Domain Adaptation Domain mismatch and dataset bias represent a challenging problem in machine learning today (Torralba & Efros, 2011). Extensive prior work has been proposed to adapt models to new tasks and datasets (Daume III, 2007; Saenko et al., 2010). Why does a model trained on one dataset perform poorly on the other dataset? To answer this question, we characterize the domain mismatch via distilled data. In particular, we optimize distilled data for models trained on one dataset so that the distilled data that can largely improve the model performance on the new dataset. Formally, given a source dataset and a target dataset, we optimize the following objective:

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_s \sim p(\theta_s)} \mathcal{L}^t(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_s), \tag{9}$$

where $p(\theta_s)$ is the distribution of pre-trained models on the source dataset and $\mathcal{L}^t$ denotes the training loss for the target dataset.

Fast Data Poisoning Attack Moreover, our approach can be used to construct data poisoning attack. To illustrate this idea, we consider the following scenario. When a single GD step is applied with our synthetic adversarial data, a well-behaved image classifier catastrophically forgets a category but still maintains high performance on other categories. Formally, given an attacked category K and a target category T, we want the classifier to misclassify images from category K to category T. To achieve this, we optimize the following objective:

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta \sim p(\theta)} \mathcal{L}_{K \to T}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta), \tag{10}$$

where $p(\theta)$ is the distribution of well-trained classifiers. The classification loss $\mathcal{L}_{K \to T}$ encourages the model to classify category K images mistakenly as category T while correctly predicting other images, e.g., a cross entropy loss with target labels of K modified to T.

Compared to prior data poisoning attacks (Biggio et al., 2012; Li et al., 2016; Muñoz-González et al., 2017; Koh & Liang, 2017), our approach crucially *does not* require the poisoned training data to be stored and trained on repeatedly. Instead, our method attacks the model training just in one iteration and with only a few data. This makes our method effective for many online training algorithms and useful for the case where malicious users hijack the data feeding pipeline for only one gradient step (e.g., one network transmission). In Sec. 4.2, we show that a single batch of distilled data applied in one step can successfully attack well-optimized neural network models.

Finally, we note that this application can be viewed as distilling dataset knowledge of a specific category into data.

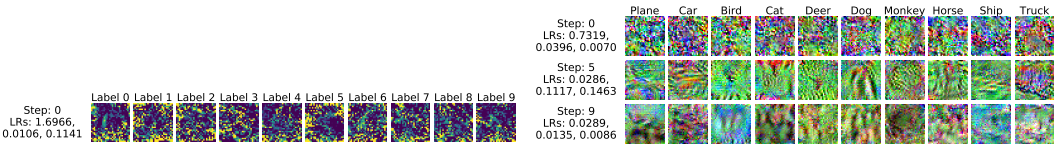
4 EXPERIMENTS

We present image classification results on MNIST (LeCun, 1998) and CIFAR10 (Krizhevsky & Hinton, 2009). For MNIST, distilled images are trained with LENET (LeCun et al., 1998), which can achieve about 99% test accuracy if fully trained. For CIFAR10, we use a network architecture following Krizhevsky (2012) (with top locally connected layers replaced by fully connected ones), which would achieve around 80% test accuracy if fully trained. For random initializations and baselines with randomness (e.g., randomly selecting real training images), we report means and standard deviations over 200 runs.

4.1 DATASET DISTILLATION

Fixed Initialization With access to initial network weights, distilled images can directly train this particular network to high performance. For example, 10 learned distilled images can boost the test accuracy of a neural network from 12.9% to 93.76% on MNIST (Fig. 2a) and 100 images can train a network from 8.82% to 54.03% test accuracy on CIFAR10 (Fig. 2b). This indicates that a few distilled images have enough capacity to include at least part of dataset knowledge.

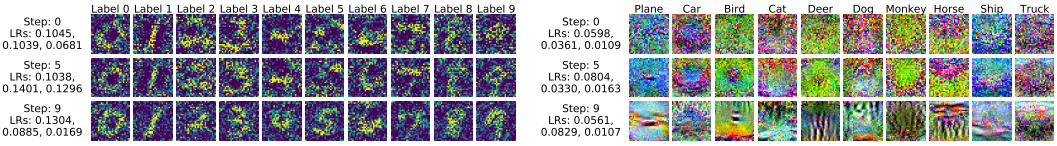
Random Initialization Trained with randomly sampled initializations using an approach proposed by Glorot & Bengio (2010) (known as "Xavier"), the learned distilled images do not need to encode information tailored for a particular starting point and thus can represent meaningful content independent of network initializations. In Fig. 3, we see that such distilled images reveal discriminative features of the corresponding categories: e.g., the `ship` image in Fig. 3b. These 100 images can



(a) MNIST. These images train networks with a particular initialization from 12.9% to 93.76% test accuracy.

(b) CIFAR10. These images train networks with a particular initialization from 8.82% to 54.03% test accuracy.

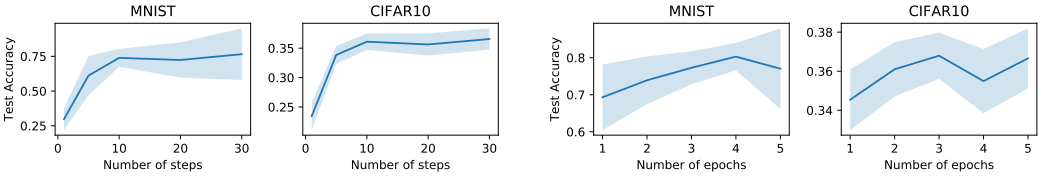
Figure 2: Learned distilled images given fixed initializations. Both experiments use 3 epochs and 10 images per GD step (one per category). MNIST distilled images use 1 GD step with 10 images in total. CIFAR10 distilled images use 10 GD steps with 100 images in total (only images for selected steps are shown). At left, we report the learning rates for 3 epochs.



(a) MNIST. These images train networks with unknown initialization to $77.28\% \pm 4.45\%$ test accuracy.

(b) CIFAR10. These images train networks with unknown initialization to $36.79\% \pm 1.18\%$ test accuracy.

Figure 3: Distilled images trained for 10 GD steps and 3 epochs with unknown random initializations. We show images from selected GD steps and corresponding trained learning rates (for all three epochs).



(a) Ablation study on number of steps

(b) Ablation study on number of epochs

Figure 4: Ablation studies: (a) average test accuracy w.r.t. the number of gradient descent steps. Each step contains 10 images (one per category). The number of epochs is fixed to be 2. (b) average test time accuracy with respect to the number of epochs. Each epoch uses the same set of distilled images, which has 10 GD steps each containing 10 images (one per category).

	N (#images per category)	Total #images ($N \times$ #categories)	Applied in 1 GD step	Applied in N GD steps
MNIST	1	10	$\mathbf{19.81\% \pm 5.83\%}$	$\mathbf{19.81\% \pm 5.83\%}$
	5	50	$21.19\% \pm 5.47\%$	$\mathbf{53.94\% \pm 9.45\%}$
	10	100	$21.45\% \pm 5.81\%$	$\mathbf{69.32\% \pm 8.86\%}$
	20	200	$21.49\% \pm 5.96\%$	$\mathbf{72.68\% \pm 12.44\%}$
	30	300	$21.57\% \pm 5.91\%$	$\mathbf{83.58\% \pm 5.08\%}$
CIFAR10	1	10	$\mathbf{17.95\% \pm 2.41\%}$	$\mathbf{17.95\% \pm 2.41\%}$
	5	50	$18.25\% \pm 2.51\%$	$\mathbf{32.49\% \pm 1.28\%}$
	10	100	$18.46\% \pm 2.58\%$	$\mathbf{34.54\% \pm 1.55\%}$
	20	200	$18.58\% \pm 2.58\%$	$\mathbf{37.84\% \pm 1.63\%}$
	30	300	$18.59\% \pm 2.56\%$	$\mathbf{38.20\% \pm 1.47\%}$

Table 1: Comparison between applying the same number of images in one GD step versus multiple steps, with the number of epochs fixed to 1. N denotes the total number of images per category. For multiple steps runs, each of the N steps applies one image per category. Using multiple steps drastically improves the performance.

train randomly initialized networks to 36.79% average test accuracy on CIFAR10. Similarly, for MNIST, the 100 distilled images shown in Fig. 3a can train randomly initialized networks to 77.28% test accuracy.

More Than One Gradient Descent Steps To improve the accuracy, we can train a model with a sequence of distilled images and multiple GD steps. In Fig. 3, we learn distilled images for 10 GD steps, leading to a total of 100 images (with each step containing one image per category). The early steps tend to look noisier, likely regularizing random weights to points easier for optimization and fine-tuning. In later steps, the images become increasingly resembling real data, which summarizes the discriminative features for these categories. Fig. 4a shows that using more steps significantly improves the results. Alternatively, we can train the model with one GD step but a big batch size. Sec. 3.3 showed some limitation of using only one step in a simple linear case. In Table 1, we empirically verify that with deep convolutional networks, using multiple steps drastically outperforms using only a single step, with the same number of distilled images.

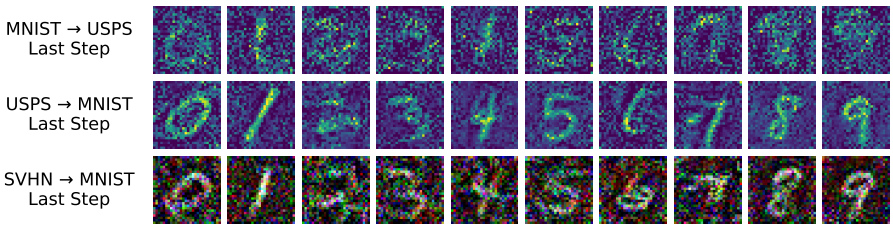


Figure 5: We train 100 distilled images applied in 10 steps and 3 epochs for each fast domain adaptation task. Images in last steps are visualized. From top to bottom: MNIST $\rightarrow$ USPS, USPS $\rightarrow$ MNIST, SVHN $\rightarrow$ MNIST.

	Ours with fixed initializations	Ours with random initializations	Random real images	No adaptation	Directly training with **much more** real images
MNIST $\rightarrow$ USPS	$\mathbf{97.90}\%$	$\mathbf{94.62}\% \pm \mathbf{1.47}\%$	$94.35\% \pm 1.49\%$	$67.54\% \pm 3.91\%$	$97.32\% \pm 0.27\%$
USPS $\rightarrow$ MNIST	$\mathbf{93.19}\%$	$\mathbf{92.74}\% \pm \mathbf{1.38}\%$	$84.40\% \pm 3.27\%$	$90.43\% \pm 2.97\%$	$98.60\% \pm 0.53\%$
SVHN $\rightarrow$ MNIST	$\mathbf{96.15}\%$	$\mathbf{85.21}\% \pm \mathbf{4.73}\%$	$81.68\% \pm 3.78\%$	$51.64\% \pm 2.77\%$	$98.60\% \pm 0.53\%$

Table 2: Domain adaptation performance for our method and baselines. Distilled images are trained for 10 GD steps (100 images in total). Baselines use the same numbers of randomly sampled real images from target dataset and also apply in 10 GD steps. We evaluate these real images using learned learning rate, and each of the fixed learning rate in $\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3\}$, and report the best result.

More Than One Epochs To further boost the model performance, we can train the model on the same distilled images but with multiple passes (epochs). Fig. 4b shows that using the same number of distilled images but with more than 1 epochs improves the performance. However, the improvement slows down as we increase the number of epochs. We believe that longer training (i.e., more epochs) can help the model learn all the knowledge from the distilled images, but can make the model training less stable and is eventually limited by the capacity of the images (i.e., number of total images).

4.2 Applications

Next, we show two applications built on our main algorithm. Both cases assume that the initial weights are random but pre-trained on the same dataset, which is different from the current dataset.

Fast Domain Adaptation We can optimize distilled images to quickly fine-tune a pretrained model for a new dataset. Additionally, the learned distilled images can visualize the domain gap between two datasets. Fig. 5 visualizes distilled images trained to increase the performance of pre-trained models on various domain adaptation tasks. Table 2 shows that our method is more effective compared to using the same number of random real images.

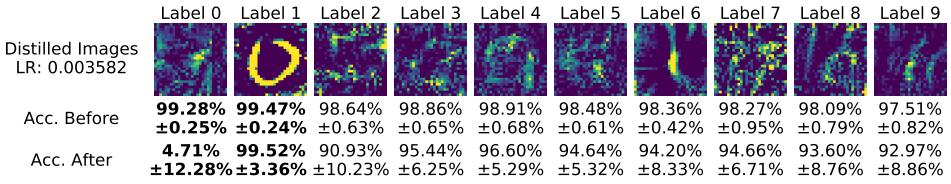
Fast Data Poisoning Attack Finally, our method can construct a new type of data poisoning, where the attacker can apply just one GD step with a few malicious data to manipulate a well-trained model. As shown in Fig. 6, we train distilled images to make trained neural networks to misclassify a particular attacked category to another target category with just one GD step. On MNIST with the attacked category 0 and the target category 1, we can see that the distilled images for many categories other than 0 show the digit 0 shape, with the effect most evident in distilled image for target category 1, i.e., the category we want the attacked model to predict 0 digits as. Table 3 shows that our method is able to attack well-optimized models with a single GD step on a small batch of images, and is much more effective than the attack baseline with randomly sampled real images with incorrect labels.

5 Discussion

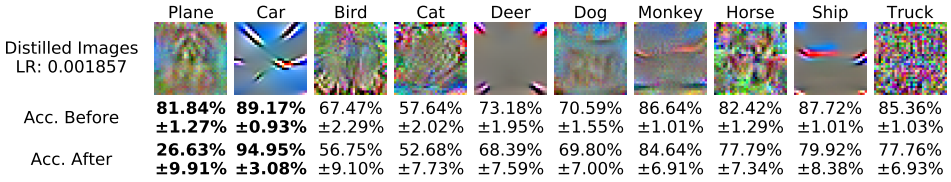
In this paper, we present dataset distillation for compressing the knowledge of entire training data into a few synthetic training images. We can train a network to reach high performance with small number of distilled images and several gradient descent steps. Finally, we demonstrate two applications including fast domain adaptation and effective data poisoning attack. In the future, we plan to extend our method to compress large-scale visual datasets such as ImageNet (Deng et al., 2009) and other types of data (e.g., audio and text). Also, our current method works well for random pre-trained weights, but performs worse for randomly initialized networks. We would like to investigate various initialization strategies, with which distilled images can work well.

	#images	Ours			Random real images		
		Attacked class classified **correctly**	Attacked class classified **as target class**	Overall accuracy w.r.t. **incorrect** labels	Attacked class classified **correctly**	Attacked class classified **as target class**	Overall accuracy w.r.t. **incorrect** labels
MNIST: $0 \to 1$	10	$\mathbf{4.71\% \pm 12.31\%}$	$\mathbf{65.13\% \pm 30.44\%}$	$\mathbf{91.88\% \pm 4.25\%}$	$98.74\% \pm 0.38\%$	$0.02\% \pm 0.05\%$	$88.84\% \pm 0.52\%$
	50	$\mathbf{2.88\% \pm 8.35\%}$	$\mathbf{70.32\% \pm 31.16\%}$	$\mathbf{90.71\% \pm 5.59\%}$	$98.80\% \pm 0.33\%$	$0.02\% \pm 0.05\%$	$88.87\% \pm 0.50\%$
	100	$\mathbf{2.55\% \pm 8.90\%}$	$\mathbf{71.45\% \pm 29.68\%}$	$\mathbf{90.87\% \pm 5.28\%}$	$98.81\% \pm 0.33\%$	$0.02\% \pm 0.04\%$	$88.88\% \pm 0.50\%$
	200	$\mathbf{1.74\% \pm 7.28\%}$	$\mathbf{71.43\% \pm 30.74\%}$	$\mathbf{89.62\% \pm 6.88\%}$	$98.80\% \pm 0.33\%$	$0.02\% \pm 0.05\%$	$88.88\% \pm 0.50\%$
	300	$\mathbf{1.61\% \pm 7.66\%}$	$\mathbf{72.98\% \pm 30.78\%}$	$\mathbf{89.15\% \pm 7.71\%}$	$98.80\% \pm 0.34\%$	$0.02\% \pm .05\%$	$88.88\% \pm 0.50\%$
CIFAR10: `plane`$\to$`car`	10	$\mathbf{26.63\% \pm 9.94\%}$	$\mathbf{33.23\% \pm 14.37\%}$	$\mathbf{69.59\% \pm 2.03\%}$	$77.81\% \pm 4.45\%$	$3.05\% \pm 0.86\%$	$68.71\% \pm 1.67\%$
	50	$\mathbf{12.41\% \pm 8.15\%}$	$\mathbf{42.39\% \pm 17.13\%}$	$\mathbf{70.36\% \pm 2.12\%}$	$78.93\% \pm 2.23\%$	$3.01\% \pm 0.60\%$	$69.69\% \pm 1.09\%$
	100	$\mathbf{9.23\% \pm 6.69\%}$	$\mathbf{45.92\% \pm 18.11\%}$	$\mathbf{70.39\% \pm 2.39\%}$	$79.31\% \pm 1.59\%$	$3.01\% \pm 0.56\%$	$69.96\% \pm 1.03\%$
	200	$\mathbf{7.96\% \pm 6.58\%}$	$\mathbf{45.58\% \pm 19.05\%}$	$69.90\% \pm 3.20\%$	$79.57\% \pm 1.47\%$	$3.03\% \pm 0.51\%$	$\mathbf{70.08\% \pm 1.01\%}$
	300	$\mathbf{7.15\% \pm 6.26\%}$	$\mathbf{47.77\% \pm 18.43\%}$	$70.09\% \pm 2.76\%$	$79.69\% \pm 1.43\%$	$3.02\% \pm 0.49\%$	$\mathbf{70.11\% \pm 1.00\%}$

Table 3: Fast data poisoning attack performance for our method and baseline. Distilled images are trained for one GD step. Baselines uses the same numbers of randomly sampled real images with incorrect labels and also apply one GD step. We evaluate these real images using learned learning rate, and each of the fixed learning rate in $\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3\}$, and report the result that achieves the highest accuracy w.r.t. the incorrect labels. While the overall accuracies w.r.t. the incorrect labels are similar, our attack images are always better at changing the prediction for images of the attacked class.



(a) MNIST. Attacked category: 0. Target category: 1.



(b) CIFAR10. Attacked category: `plane`. Target category: `car`.

Figure 6: Malicious distilled images can manipulate well-trained neural network models to misclassify attacked category images as target category with one GD step and 10 images. These images are trained with 2000 well-optimized neural network models and evaluated on 200 held-out models. Better results with larger batch size can be found at Table 3.

REFERENCES

Anelia Angelova, Yaser Abu-Mostafam, and Pietro Perona. Pruning training sets for learning of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pp. 494–501. IEEE, 2005.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2654–2662, 2014.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3319–3327. IEEE, 2017.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

Hal Daume III. Frustratingly easy domain adaptation. In *Annual Meeting of the Association for Computational Linguistics*, 2007.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(9):1627–1645, 2010.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh (eds.), *International Conference on Machine Learning (ICML)*, volume 70. PMLR, 06–11 Aug 2017.

Alex Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. *Source code available at https://github. com/akrizhevsky/cuda-convnet2 [March, 2017]*, 2012.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Agata Lapedriza, Hamed Pirsiavash, Zoya Bylinskii, and Antonio Torralba. Are all training examples equally valuable? *arXiv preprint arXiv:1311.6510*, 2013.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning (ICML)*, 2015.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 27–38. ACM, 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, 2016.

Jean Ponce, Tamara L Berg, Mark Everingham, David A Forsyth, Martial Hebert, Svetlana Lazebnik, Marcin Marszalek, Cordelia Schmid, Bryan C Russell, Antonio Torralba, et al. Dataset issues in object recognition. In *Toward category-level object recognition*, pp. 29–48. Springer, 2006.

Ilija Radosavovic, Piotr Dollár, Ross Girshick, Georgia Gkioxari, and Kaiming He. Data distillation: Towards omni-supervised learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, 2010.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2(Nov):45–66, 2001.

Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1521–1528. IEEE, 2011.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pp. 818–833. Springer, 2014.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *International Conference on Learning Representations (ICLR)*, 2015.