

Introduction

> 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.

> 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 out of the manifold of natural images? Conventional wisdom would suggest that the answer is no, as the synthetic training data may not follow the same distribution of 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 not only **capturing much of the original training data** but also tailored explicitly for **fast model training** in only a few gradient steps.

> **our goal is to compress the knowledge of an entire dataset into a few synthetic training images.**

> 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 particular training objective, we optimize the pixel values of our distilled images.

Other methods

> Dataset pruning, core-set construction, and instance selection. 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; Felzenszwalb et al., 2010; Lapedriza et al., 2013) or by only labeling the “valuable” data via active learning (Cohn et al., 1996; Tong & Koller, 2001). Similarly, core-set construction (Tsang et al., 2005; Har-Peled & Kushal, 2007; Bachem et al., 2017; Sener & Savarese, 2018) and instance selection (Olvera-López et al., 2010) methods aim to select a subset of the entire training data.

> However, algorithms constructing these subsets 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.

Approach

> Algorithm 1

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 entire training data:

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

(1)

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

where η is the learning rate. Such a training process often takes tens of thousands or even millions of 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 such as

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

using these learned synthetic data $\tilde{\mathbf{x}}$ can greatly boost the performance on the real test set. Given an initial θ_0 , we obtain these synthetic data $\tilde{\mathbf{x}}$ and learning rate $\tilde{\eta}$ by minimizing the objective below \mathcal{L} :

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

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

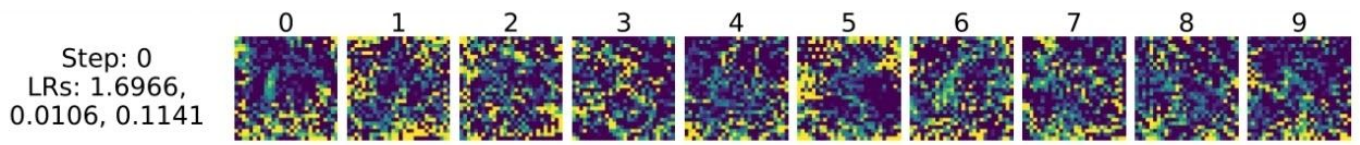
Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial weights; M : the number of distilled data

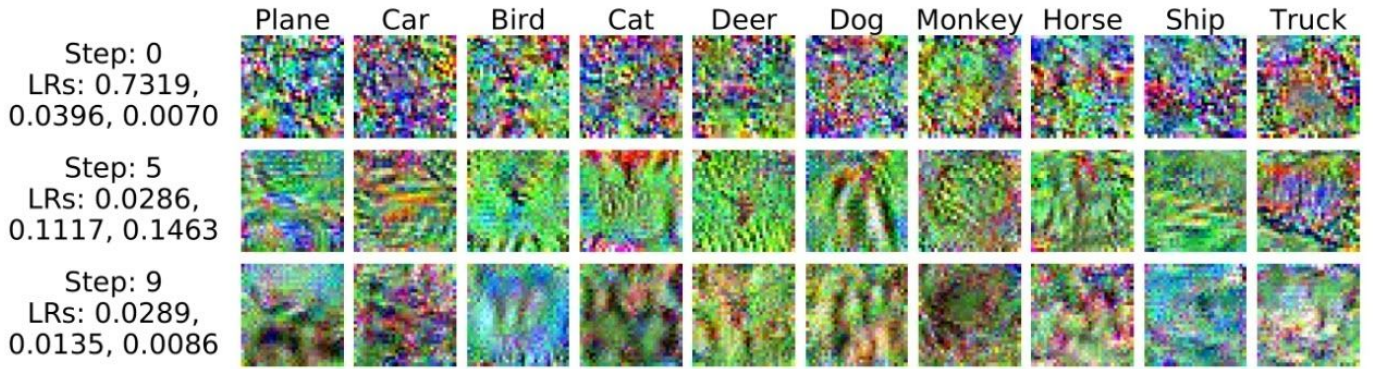
Input: α : step size; n : batch size; T : the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

- 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 training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^n$
- 4: Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
- 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 the objective function on real training data: $\mathcal{L}^{(j)} = \ell(\mathbf{x}_t, \theta_1^{(j)})$
- 8: **end for**
- 9: Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha \nabla_{\tilde{\mathbf{x}}} \sum_j \mathcal{L}^{(j)}$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \mathcal{L}^{(j)}$
- 10: **end for**

Output: distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$



(a) MNIST. These distilled images train a fixed initialization from 12.90% test accuracy to 93.76%.



(b) CIFAR10. These distilled images train a fixed initialization from 8.82% test accuracy to 54.03%.

Figure 2: Dataset distillation for fixed initializations: MNIST distilled images use one GD step and three epochs (10 images in total). CIFAR10 distilled images use ten GD steps and three epochs (100 images in total). For CIFAR10, only selected steps are shown. At left, we report the corresponding learning rates for all three epochs.

> Algorithm with random initialization

Unfortunately, the above distilled data optimized for a given initialization do not generalize well to other initializations. The distilled data often look like random noise (e.g., in Figure 2a) as it encodes the information of both training dataset \mathbf{x} and a particular network initialization θ_0 . To address this 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:

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

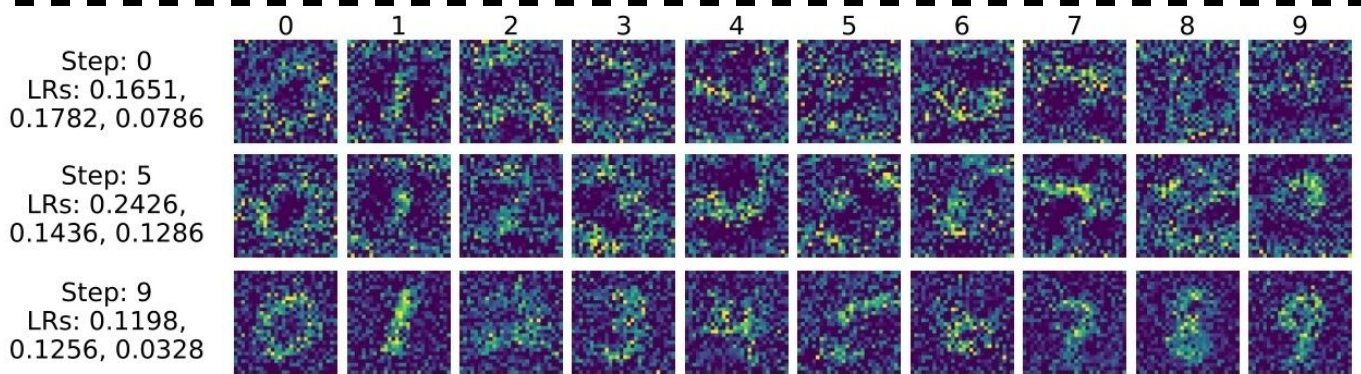
>> In practice, we observe that the final distilled data generalize well to unseen initializations. In addition, these distilled images often look quite informative, encoding the discriminative features of each category

>> it turns out crucial for $l(\mathbf{x}, \cdot)$ to share similar local conditions (e.g., output values, gradient magnitudes) over initializations θ_0 sampled from $p(\theta_0)$.

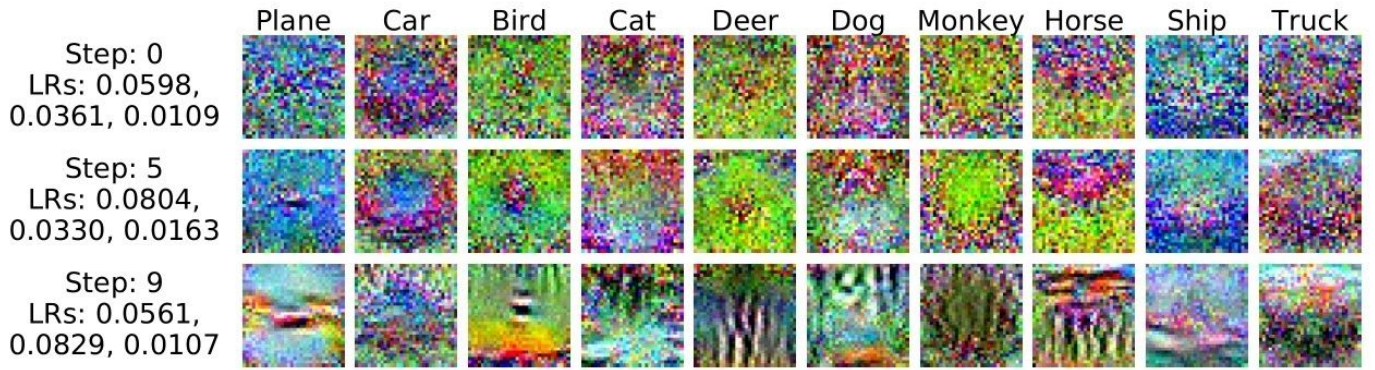
Inspired by the analysis of the simple linear case in Section 3.3, we aim to focus on initial weights distributions $p(\theta)$ that yield similar local conditions over the data distribution. In this work, we focus on the following four practical choices:

- **Random initialization:** Distribution over random initial weights, e.g., He Initialization (He et al., 2015) and Xavier Initialization (Glorot & Bengio, 2010) for neural networks.
- **Fixed initialization:** A particular fixed network initialized by the method above.
- **Random pre-trained weights:** Distribution over models pre-trained on other tasks or datasets e.g., ALEXNET (Krizhevsky et al., 2012) networks trained on ImageNet (Deng et al., 2009).
- **Fixed pre-trained weights:** A particular fixed network pre-trained on other tasks and datasets

Random initialization. Trained with randomly sampled initializations using Xavier Initialization (Glorot & Bengio, 2010), 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 Figure 3, we see that such distilled images reveal the discriminative features corresponding categories: e.g., the ship image in Figure 3b. These 100 images can train randomly initialized networks to 36.79% average test accuracy on CIFAR10. Similarly, for MNIST, the 100 distilled images shown in Figure 3a can train randomly initialized networks to 79.50% test accuracy.



(a) MNIST. These distilled images unknown random initializations to $79.50\% \pm 8.08\%$ test accuracy.



(b) CIFAR10. These distilled images unknown random initializations to $36.79\% \pm 1.18\%$ test accuracy.

Figure 3: Distilled images trained for *random initialization* with ten GD steps and three epochs (100 images in total). We show images from selected GD steps and the corresponding learning rates for all three epochs.

> Algorithm with multiple gradient steps and multiple epochs

We 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 rates, i.e., each step i is

$$\theta_{i+1} = \theta_i - \tilde{\eta}_i \nabla_{\theta_i} \ell(\tilde{\mathbf{x}}_i, \theta_i), \quad (9)$$

and changing Line 9 to backpropagate through all steps. However, naively computing gradients is memory 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. Specifically, back-gradient optimization formulates the necessary 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 algorithmic details, we refer readers to prior work (Domke, 2012; Maclaurin et al., 2015).

Multiple gradient descent steps and multiple epochs. In Figure 3, distilled images are learned for 10 GD steps applied in 3 epochs, leading to a total of 100 images (with each step containing one image per category). Images used in early steps tend to look noisier. However, in later steps, the distilled images gradually look like real data and share the discriminative features for these categories. Figure 4a shows that using more steps significantly improves the results. Figure 4b shows a similar but slower trend as the number of epochs increases. We observe that longer training (i.e., more epochs) can help the model learn all the knowledge from the distilled images, but the performance is eventually limited by the total number of images. Alternatively, we can train the model with one GD step but a big batch size. Section 3.3 has shown theoretical limitations of using only one step in a simple linear case. In Figure 5, we observe that using multiple steps drastically outperforms the single step method, given the same number of distilled images.

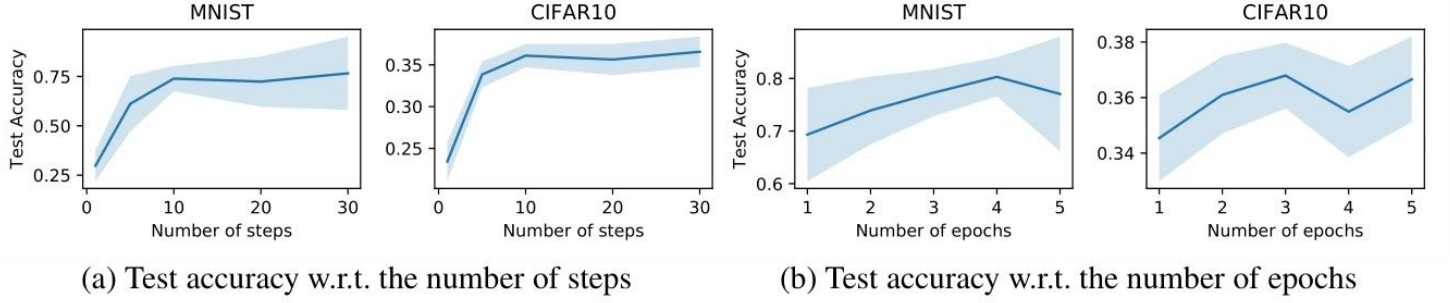


Figure 4: Hyper-parameter sensitivity studies: we evaluate models in *random initialization* settings. (a) Average test accuracy w.r.t. the number of GD steps. We use two epochs. (b) Average test accuracy w.r.t. the number of epochs. We use 10 steps, with each step containing 10 images.

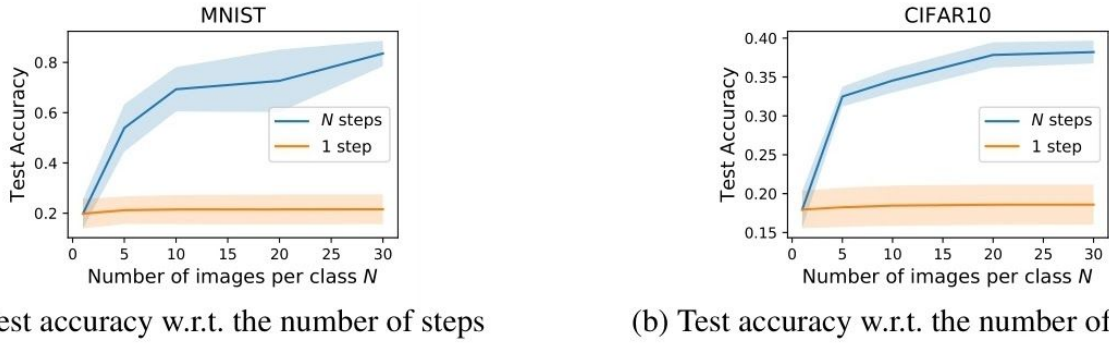


Figure 5: Comparison between one and multiple GD steps given the same number of images. We evaluate models in *random initialization* settings and use one epoch. N denotes the total number of images per category. For multiple steps runs, each step uses 10 images in total (one per category).

Theoretical analysis

> This section studies our formulation in a simple linear regression problem with quadratic loss. We derive the lower bound of the size of distilled data needed to achieve the same performance as training on the full dataset for arbitrary initialization with one GD step.

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

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

how small can the size of distilled data be? For such models, the global minimum is attained at any θ^* satisfying $\mathbf{d}^T \mathbf{d} \theta^* = \mathbf{d}^T \mathbf{t}$. Substituting Equation (6) in the condition above, we have

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

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

$$\mathbf{I} - \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T \tilde{\mathbf{d}} = \mathbf{0}, \quad (8)$$

which implies that $\tilde{\mathbf{d}}^T \tilde{\mathbf{d}}$ has full rank and $M \geq D$.

> The analysis above only considers a simple case but suggests that any small number of distilled data fail to generalize to arbitrary initial θ_0 .

> This analysis motivates us to focus on $p(\theta_0)$ distributions that yield similar local conditions.

Distillation for malicious data poisoning

Formally, given an attacked category K and a target category T , we minimize a new objective $\ell_{K \rightarrow T}(\mathbf{x}, \theta_1)$, which is a classification loss that encourages θ_1 to misclassify images of category K as category T while correctly predicting other images, e.g., a cross entropy loss with target labels of K modified to T . Then, we can obtain the malicious distilled images by optimizing

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg \min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}_{K \rightarrow T}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0), \quad (10)$$

where $p(\theta_0)$ is the distribution over *random weights* of well-optimized classifiers. Trained on a distribution of such classifiers, the distilled images *do not* require access to the exact model weights and thus can generalize to unseen models. In our experiments, the malicious distilled images are trained on 2000 well-optimized models and evaluated on 200 held-out ones.

> our approach crucially does not require the poisoned training data to be stored and trained on repeatedly. Instead, our method attacks the model training in one iteration and with only a few data. This advantage makes our method potentially effective for 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).

Experiments

> **Baselines**

Baselines. For each experiment, in addition to baselines specific to the setting, we generally compare our method against baselines trained with data derived or selected from real training images:

- **Random real images:** We randomly sample the same number of real images per category.
- **Optimized real images:** We sample different sets of random real images as above, and choose the top 20% best performing sets.
- **k -means:** We apply k -means clustering to each category, and use the cluster centroids as training images.
- **Average real images:** We compute the average image for each category, which is reused in different GD steps.

For these baselines, we perform each evaluation on 200 held-out models with all combinations of learning rate $\in \{\text{learned learning rate with our method}, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3\}$ and #epochs $\in \{1, 3, 5\}$, and report results using the best performing combination. Please see the appendix Section S-1 for more details about training and baselines.

	Ours		Baselines					
	Fixed init.	Random init.	Used as training data in same number of GD steps				Used in K-NN classification	
			Random real	Optimized real	k -means	Average real	Random real	k -means
MNIST	96.6	79.5 \pm 8.1	68.6 \pm 9.8	73.0 \pm 7.6	76.4 \pm 9.5	77.1 \pm 2.7	71.5 \pm 2.1	92.2 \pm 0.1
CIFAR10	54.0	36.8 \pm 1.2	21.3 \pm 1.5	23.4 \pm 1.3	22.5 \pm 3.1	22.3 \pm 0.7	18.8 \pm 1.3	29.4 \pm 0.3

Table 1: Comparison between our method trained for ten GD steps and three epochs and various baselines. For baselines using K-Nearest Neighbor (K-NN), best result among all combinations of distance metric $\in \{l_1, l_2\}$ and $K \in \{1, 3\}$ is reported. In K-NN and k -means, K and k can have different values. All methods use ten images per category (100 in total), except for the average real images baseline, which reuses the same images in different GD steps.

> we can optimize distilled images to quickly fine-tune pre-trained models for a new dataset.

	Ours w/ fixed pre-trained	Ours w/ random pre-trained	Random real	Optimized real	k -means	Average real	Adaptation Motiian et al. (2017)	No adaptation	Train on full target dataset
$\mathcal{M} \rightarrow \mathcal{U}$	97.9	95.4 \pm 1.8	94.9 \pm 0.8	95.2 \pm 0.7	92.2 \pm 1.6	93.9 \pm 0.8	96.7 \pm 0.5	90.4 \pm 3.0	97.3 \pm 0.3
$\mathcal{U} \rightarrow \mathcal{M}$	93.2	92.7 \pm 1.4	87.1 \pm 2.9	87.6 \pm 2.1	85.6 \pm 3.1	78.4 \pm 5.0	89.2 \pm 2.4	67.5 \pm 3.9	98.6 \pm 0.5
$\mathcal{S} \rightarrow \mathcal{M}$	96.2	85.2 \pm 4.7	84.6 \pm 2.1	85.2 \pm 1.2	85.8 \pm 1.2	74.9 \pm 2.6	74.0 \pm 1.5	51.6 \pm 2.8	98.6 \pm 0.5

Table 2: Performance of our method and baselines in adapting models among MNIST (\mathcal{M}), USPS (\mathcal{U}), and SVHN (\mathcal{S}). 100 distilled images are trained for ten GD steps and three epochs. Our method outperforms few-shot domain adaptation (Motiian et al., 2017) and other baselines in most settings.

>In Table 3, we adapt a widely used ALEXNET model (Krizhevsky et al., 2012) pre-trained on ImageNet (Deng et al., 2009) to image classification on PASCAL-VOC (Everingham et al., 2010) and CUB-200 (Wah et al., 2011) datasets.

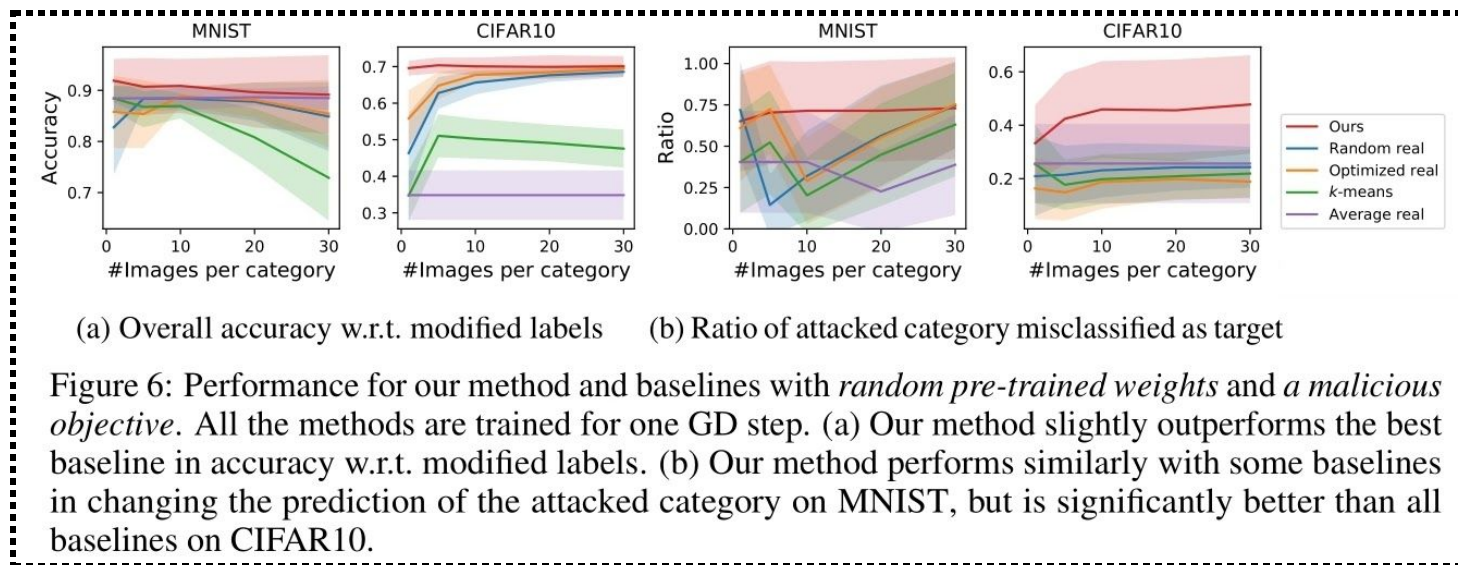
Target dataset	Ours	Random real	Optimized real	Average real	Fine-tune on full target dataset
PASCAL-VOC	70.75	19.41 \pm 3.73	23.82 \pm 3.66	9.94	75.57 \pm 0.18
CUB-200	38.76	7.11 \pm 0.66	7.23 \pm 0.78	2.88	41.21 \pm 0.51

Table 3: Performance of our method and baselines in adapting an ALEXNET pre-trained on ImageNet to PASCAL-VOC and CUB-200. We use one distilled image per category, with one GD step repeated for three epochs. Our method significantly outperforms the baselines. We report results over 10 runs.

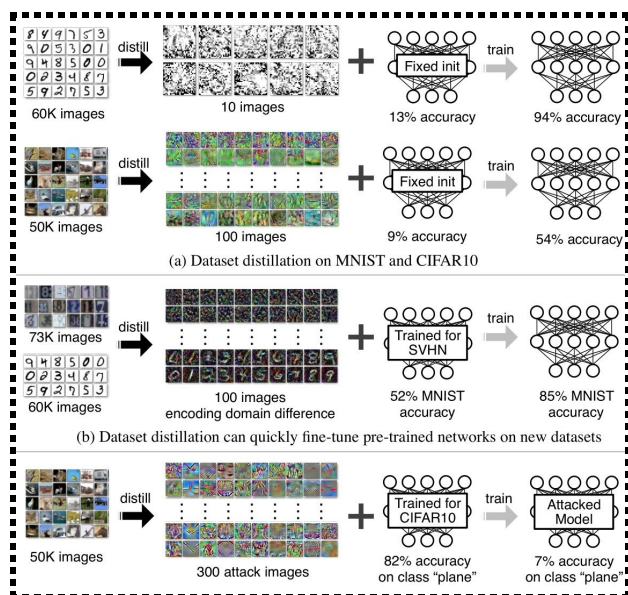
> our method can construct a new type of data poisoning, where an attacker can apply just one GD step with a few malicious data to manipulate a well-trained model.

>>Our method requires no access to the exact weights of the model

>>While some baselines perform similarly well as our method on MNIST, our method significantly outperforms all the baselines on CIFAR10.



Results



References

> 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).

Идеи:

> их идеи: In the future, we plan to **extend our method to compressing large-scale visual datasets such as ImageNet** and other types of data (e.g., audio and text). Also, our current method is sensitive to the distribution of initializations. We would like to **investigate other initialization strategies, with which dataset distillation can work well**.